Distributed (Correlation) Samplers

A Study on the Round Complexity of Cryptographic Sampling Protocols

Damiano Abram

PhD Dissertation

Department of Computer Science
Aarhus University
Denmark

Distributed (Correlation) Samplers

A Study on the Round Complexity of Cryptographic Sampling Protocols

A Dissertation
Presented to the Faculty of Natural Sciences
of Aarhus University
in Partial Fulfillment of the Requirements
for the PhD Degree

by
Damiano Abram
March 24, 2024

Abstract

This thesis presents a theoretical analysis of round complexity in cryptographic sampling protocols. We study the following problem: how can n participants agree on a random sample from the distribution $\mathcal{D}$ without anybody knowing any secret information hidden in the chosen sample? How many rounds of interactions are needed in any such protocol? Which assumptions are required?

These questions are particularly interesting in the context of multiparty computation (MPC) due to the extensive reliance (often unavoidable) on trusted setups such as common reference strings (CRSs). These are samples from known distributions made public before the beginning of the protocol. Security proofs often require CRSs to hide trapdoors. However, if these trapdoors get leaked, security is often fatally compromised. Since trusted setups are inherently single points of failure, it is desirable to generate CRSs in a distributed fashion using MPC protocols: as long as at least one of the participants remains honest, any trapdoor hidden in the CRS is guaranteed to remain secret.

Previous research (Garg et al. TCC '14) had shown that, in the setting of semi-honest security with dishonest majority, any functionality can be implemented in two rounds. In this thesis, we show that, using strong primitives such as indistinguishability obfuscation (iO), it is possible to build one-round sampling protocols. We call these constructions *distributed samplers*. We show that these protocols can be used to securely generate large amounts of correlated randomness with sublinear communication in a single round.

We study different ways to upgrade our semi-honest construction to active security. First, we try to circumvent Cleve's impossibility (STOC '86) by allowing the adversary limited influence on the output of the protocol: we consider the functionality $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ that proposes a polynomial number of different outputs to the adversary, letting it choose its favourite one ($\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ can still be used to generate CRSs). We show how to implement $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ in one round by relying on a programmable random oracle.

In subsequent work, we show that random oracles are necessary: any actively secure distributed samplers implementing $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ needs to rely on a CRS that is non-reusable, as large as a sample from $\mathcal{D}$ and structured (unless $\mathcal{D}$ is an obliviously samplable distribution). This suggests that, without random oracles, simulation-based, actively secure distributed samplers provide no advantage over trusted dealers.

We get around this impossibility by proposing new game-based definitions for actively secure distributed samplers. *Hardness-preserving distributed samplers* allow the removal of CRSs from MPC protocols preserving at the same time the hardness of search problems. *Indistinguishability-preserving distributed samplers*, on the other hand, substitute the trusted setup while preserving the functionality, however, the protocol needs to satisfy particular conditions. We show how to build distributed samplers with the defined properties by relying on extremely lossy functions, the subexponential security of iO and other primitives.

Next, in the context of active security with dishonest majority, we study the feasibility of protocols producing unbiased samples *with guaranteed output*: to circumvent Cleve's impossibility, we assume the existence of an auxiliary functionality that, upon invocation, publishes a random , short k-bit string. We first study the question when $\mathcal{D}$ is the uniform distribution over $\{0,1\}^m$ where $m \gg k$. This problem is called *coin tossing extension* (CTE) (Bellare et al. PODC '96). We show the existence of $O(1)$-round CTE constructions producing an arbitrarily large quantity of unbiased randomness based on various assumptions (OWFs, DDH, QR and DCR, class groups). Our most important result is an LWE-based, one-round, UC-secure, n-party CTE protocol withstanding adaptive corruption. The protocol can be viewed as an *explainable* randomness extractor. By relying on distributed samplers and the auxiliary functionality, we build *one-round* unbiased sampling protocols for any distribution. Finally, we present a lower bound for statistically secure CTE with black-box simulation: an R-round protocol can generate at most $k + R \cdot O(\log \lambda)$ bits of randomness.

Resumé

Denne afhandling præsenterer en teoretisk analyse af runde kompleksiteten for kryptografiske sampling protokoller. Vi studerer følgende problem: hvordan kan n deltagere blive enige om et udfald fra en fordeling $\mathcal{D}$, uden at nogen af dem kender den hemmelige information skjult i det udfald? Hvor mange runder kræver sådan en protokol? Og hvilke antagelser er påkrævede?

Disse spørgsmål er af speciel interesse i konteksten af sikre flerpartsberegninger (fork. MPC fra det engelske multiparty computation) grundet den udbredte (og tit uundgåelige) afhængighed på betroet opsætning, så som de såkaldte common reference strings (CRSer). CRSer er uddrag fra kendte fordelinger som offentliggøres før starten af protokollen. Sikkerhedsbeviser kræver ofte at CRSer gemmer på en faldlem. Hvis en faldlem bliver lækket, bliver sikkerhed dog ofte fatalt kompromitteret. Da betroet opsætning udgør et "single point of failure", er det ønskeligt at generere CRSer på en distribueret manér ved brug af MPC-protokoller: så længe mindst en af parterne forbliver ærlig, vil enhver faldlem i CRSen forblive hemmelig.

Tidligere resultater (Garg et al. TCC '14) har vist at enhver funktionalitet kan opnå semi-honest sikkerhed i to runder med et uærligt flertal. I denne afhandling vil vi vise at det er muligt at bygge sampling protokoller i en enkelt runde ved brug af stærke primitiver, som indistinguishability obfuscation (iO). Vi kalder disse konstruktioner *distributed samplers*. Vi viser at disse protokoller kan bruges til at generere store mængder korreleret tilfældighed med sublineær kommunikation i en enkelt runde.

Vi udforsker forskellige måder at opgradere vores konstruktion fra semi-honest til aktiv sikkerhed. Først prøver vi at omgå Cleve's umulighedsresultat (STOC '86) ved at give fjenden begrænset indflydelse på protokollens output: vi betragter funktionaliteten $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ som tilbyder fjenden et polynomielt antal udfald, og lader den vælge sit foretrukken resultat ($\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ kan stadig bruges til at producere CRSer). Vi viser hvordan $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ kan implementeres I en runde ved brug af et programmable random oracle.

I efterfølgende arbejde viser vi at random oracles er nødvendige: enhver distributed sampler som implementerer $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ og opnår aktiv sikkerhed skall bruge en CRS som ikke er genanvendelig med størrelsen på et udfald fra $\mathcal{D}$ og er struktureret (medmindre $\mathcal{D}$ er en obliviously samplable fordeling). Dette foreslår at, uden random oracles, tilbyder simulation-based aktivt sikre distributed samplere ingen fordel over en betroet fordeler.

Vi undgår denne umulighed ved at foreslå nye spil-baserede definitioner for aktivt sikre distributed samplers. *Hardness-preserving distributed samplers* tillader at fjerne CRSer fra MPC protokoller mens sværheden af søge problemer vedligeholdes. *Indistinguishability-preserving distributed samplers* erstatter derimod betroet opsætning mens funktionaliteten vedligeholdes, dette kræver dog at den underliggende protokol lever op til visse krav. Vi viser hvordan distributed samplers med de definerede egenskaber kan bygges ved extremely lossy functions, subeksponentiel security of iO og andre primitiver.

Næst, ser vi på gennemførligheden af protokoller der producerer udfald uden bias med garanteret oputput, i konteksten af aktiv sikkerhed med et uærligt flertal: for at omgå Cleve's umulighedsresultat, antager vi eksistensen af en hjælper funktionalitet, som udgiver en tilfældig kort k-bit streng når den bliver kaldt. Først betragter vi tilfældet hvor $\mathcal{D}$ er den uniforme fordeling over $\{0,1\}^m$ hvor $m \gg k$. Dette problem er kendt som *coin tossing extension* (CTE) (Bellare et al. PODC '96). Vi viser at $O(1)$-runde CTE konstruktioner der producere arbitrært store mængder tilfældighed eksisterer under forskellige antagelser (OWFer, DDH, QR and DCR, class groups). Vores vigtigeste resultat er en LWE-baseret, enkelt-runde, UC-sikker, n-parts CTE prokotol, der modstår adaptive korruptioner. Ved at afhænge på distributed samplers og hjælper funktionaliteten bygger vi *et-rundes* unbiased sampling protokoller for enhver fordeling. Til sidst præsenterer vi et lower bound for statistisk sikker CTE med black-box simulering: en R-rundes protokol kan højest genereret $k + R \cdot O(\log \lambda)$ bits af tilfældighed.

Acknowledgements

Although it is often an expression of one of the most open-minded and progressive parts of our society, academia carries on, in a somewhat contradictory way, the old ideology upon which it was founded: an ideology of elitism, individualism, competition, heroes, cult of personality. Its effects ripple silently throughout the community, resurfacing in various forms: impostor syndrome, anxiety, alienation, passivity. The first victims have often no voice: these are the people that dreamt an academic career but fell into anonymity. The voices and the stories we hear are most often those of the winners, the heroes, those who succeeded, those that are usually happy with how the academic world is today (or that just dismissed the dreams and beliefs they had when they were students as unachievable utopian perspectives).

This narrative inevitable seeps also in this thesis: it is tradition to find a name written under the title. This is portrayed as some sort of "hero of the thesis", the person that alone (or with the help of secondary characters called supervisors and coauthors), armed only with their sharp wit, won against ignorance and brought the light of progress (and of distributed samplers) upon society. Really?

What I see is something different. I see a society that believed in a child, feeding him with bread, cheese, adventure, math and (especially) care. Older faceless generations infused part of their (genetic or cultural) inheritance in the person I am today. This is what created this thesis! This work is just an inheritance of ideas born hundreds of years ago, who knows in what form, reshaped by later generations and resurfacing today as distributed samplers. I'm just the writer. Is this progress? No, I don't think so, probably in a hundred years or so everybody will have forgotten distributed samplers and me. But my inheritance may still live on: it sinks again under the surface of the sea and maybe it will resurface sometime and somewhere in the future in the mind of a person that will not even be aware of my existence or that of distributed samplers. And perhaps, the words I am writing in this page will be as much of a legacy as all 300 pages on distributed samplers.

An unstructured and somewhat anonymised list of thanks (have fun putting pieces together)! This work is dedicated to all faces whose inheritance shaped this thesis: to whom conceived me and raised me, sacrificing life and career for their children. To whom conceived and raised them and to the other generations before them. To the siblings with whom I shared my childhood experiences, playing, laughing, fighting and supporting each other, continuing to these days (I'm extremely proud of both of you). To the friends living in middle-of-nowhere mountains in the Italian Alps: to whom played with me in the woods, building dams, catching lizards, frogs and newts, teaching geese and ducks how to fly, fishing in the neighbours' ponds and later in golf courses (ending up being chased by angry golfers on carts) and proper lakes (fish are very grateful for my horrible skills). To whom met at "the rock", to whom played tag until we were so sweaty that we looked like we had just got out of the shower (except for the smell, I guess) and to whom built with me our secret base in an abandoned house. To whom accompanied me on crazy mountain-bike trips, exploring unknown valleys until we got lost. To whom introduced me to climbing in an always deserted, tiny gym ten years or so ago. To whom introduced me to climbing on real rock and hosted two children on their camper van, taking them on bouldering trips in beautiful granitic valleys and at the Arco Rock Master. To the hippie-Jesus hybrid with whom I shared the passion for slacklining, together with deep philosophical thoughts in the shade of the woods. To whom shared their love for nature and the simple things. To whom introduced me to hiking in the mountains, to whom organised Campeggio every summer (ended up trapped in a hut with 30 children for a week without showers) and later allowed me to organise it. To whom took me for the first time to Rifugio Brentei, and to whom welcomed me there every time I visited in the following years. To whom ended up isolated with me at 2000 m altitude in a closed alpine hut, helping with storing all material brought by the helicopter for the opening of the season and, in the free time, explored with me the stunning nature around it. To the friends that introduced me to multi-pitch climbing, taking me with them on adventures, exploring climbing crags around the world, getting stuck on walls on too hot days or in the middle of a storm in Val d'Ambiez. To whom died under an avalanche. To all people living in Sarnonico. I often hear people depicting remote rural areas as the homeland of conservatism: true, but it is only half of the story. Their inhabitants are often loving, honest, generous, kind and, in some cases, surprisingly open-minded. After all, the conservative views are often fed by the abandonment felt in these places.

I thank all people that invested their energies in my education and all my school companions: from a tiny primary school and kindergarten in the middle of nowhere, to a slightly bigger middle school and an even larger high school. These were amazing years: school was not just a place of knowledge, it was also a place with amazing friends. I especially thank my math teacher who, more than anybody else, nurtured my passion for math and my physics teacher (truly, thanks!). I thank all professors at the math department at University of Padova, especially who supervised me and who ended up passive-aggressively fighting with an arrogant student at an exam (you are not aware how much you helped that student in his life). I thank who played cards with me at every break and who helped me get rid of stress at the Intellighenzia project. I thank who followed me in crazy, unorganised, hiking-climbing expeditions in our backyard during my years at University of Trento. Who walked with me with heavy camping gear on their shoulders towards Arco. Who later came to Cima Capi and to mountain film festivals. Who explored Marzola with me, sometimes at day, sometimes at night. Who studied with me outside Rifugio Maranza (and who let me stay there). Who followed me at Sanbapolis and who later joined at Uva e Menta. To the professors that taught me who I don't want to become.

Thanks to whom gave an opportunity to a lost, scared, ignorant Italian student and hosted him in Germany, introducing him to the world of MPC and to the people who later on supervised his PhD. I will always be extremely grateful for that! Thanks to the people that helped me feel less lonely and alienated in that year (I'm truly sorry!).

To whom saw something in a student, believed in him and invited him to apply for a PhD in Aarhus (it took me three months to convince myself that's what I wanted, my original plan was actually to retreat in the mountains, perhaps teaching in some forgotten school). To this day, I still don't know what you saw in me, but, really, thanks Ivan! To whom supervised my work as a PhD student. I'm sorry for not sharing the same passion for practical cryptography, but thanks for everything Peter, especially your kindness! Thanks to whom hosted me around the world supervising my work and believing in an unknown student. I'm really grateful for the opportunity the both of you gave me, Mark and Yuval. Thanks to whom is going to host me in the following years! I hope I will keep collaborating with all of you in the future.

To the Aarhus crypto group, the staff at the department and the examiners of this thesis. To all the friends I made in these years: to whom shared with me amazing adventures around the world, always ready to laugh! To whom ended up eating prime quality Italian cuisine in Norway (beans and mackerel, pasta and butter). To whom saw me running after a suitcase over a check-in conveyor at four in the morning (with airport personal shouting in the background). To whom played tennis with me. To whom hosted Bill and Sandrine and to whom helped in their conception before a PKC talk (and ate chocolate ice cream after). To whom tolerates a single-threaded, daydreaming brother. To whom was always ready for a laughter at the Friday bar, at the kicker table, at the pub quiz, at Boulders, at the board game café, at the Klatreklub. To whom ended up freezing in a Croatian bungalow. To whom hosted a shower in their office bonded by a shared secret. To whom rolled down a hill into a wall of mud. To whom found the best tacos in a zombie land. To whom enjoyed my amazing voice at karaoke. To whom forgot the pesto. To whom helped me feel less lonely staying with me on the phone for hours. To whom is not scared of dreaming a better world. To whom trudged in snow or walked on long, steep hikes repeatedly hearing a frustrating "we are almost there". To whom was brave enough to travel to the other side of the world. To whom went fishing with me, but got home with bags of mushrooms. To whom did not know how to drive a bob sledge. To whom preferred a Rema 1000 bag. To "don't forget your gloves, love!" To whom sewed my constantly ripped trousers. To whom got dehydrated in the hot Californian sun. To whom enjoyed Castle Rock and hosted

me at a Stanford's pool. To Willy the Chilly. To my post person. To whom supported my parents when I was away. To whom walked the Via Francigena with me (tolerating my hunger and my inability in ordering ice creams). To whom fed me carrots, to whom ended up in the back of a taxi surrounded by backpacks and crashpads. To whom does not know how to spell Fjälkinge (die Ärzte). To the macaques. To the amazing Kombardo band. To whom texted "good morning". To whom scared me with a hand. To whom asked if I was eating enough and loves Mongolian legumes. To whom will soon retire. To whom baked cakes and biscuits. To carpe diem 2012-2013! To whom likes cold naked swims. To whom kept forgetting that "ikke salat!" To whom knows already what I'm going to order. To whom shared my feelings of alienation in big cities and the contradictions with the life we grew up with. To the tourists that took pictures of a naked Italian running in the big waves. To whom gets affected by the moon. To whom went "a far la sort". To whom explored the Danish west coast on a bike (thanks Ummagamma), bumping heads, running down from dunes, eating gnocchi, pesto and sand and exploring German architecture! To whom died in every trip. To whom coded before sleeping in the shelter. To whom enjoyed sunsets on the dunes. To whom went slacklining in Marselisborg. To whom taught me about environmentalism. To whom went for walks in Risskov. To whom taught me about Lohikäärme. To whom invited me to barbecue parties. To whom learnt how to fix a flat tyre. To piglet. To whom hosted me when I was locked out of my home. To whom discovered that I have an Italian face and that Italians are "angry and ugly". To whom discovered that in Mikladalur people die. To whom discovered that Ipads sometimes save lives. To whom prevented me from climbing buildings. To whom discovered that some Italians have debatable eating tastes and habits. To whom walked and will walk the Camino. To whom enjoyed sunsets jumping from rock to rock on a river and whom bathed like a starfish. To whom fell asleep looking at the stars and whom tolerated the sound of a bear-box opening at sunrise. To whom enjoyed a sunrise on a mountain over the desert. To whom helped me move to Aarhus. To whom forgot the lemons. To whom knows that a zebra is not a horse and whom considered adopting turtles. To the amazing pain au chocolat and bread and olives. To whom climbed in the dark with headlamps. To the pesto princess and to the alter ego. To whom asked why I did not go home, buy some goats and live a simple and happy life (I think there is also some wisdom in this). To whom got scared by a zombie on a Halloween night. To whom organised secret Santa and pakkeleg. To Cipherella's non-vibing prince. To whom loves random oracles. To whom heard a scared Italian shouting "Serpente!" in the woods. To Carlo. To whom smuggled Vodka. To whom liked Two Socks. To whom liked La Cabra, Davidbreadhead and Jumbo. To whom went to South Indian. To alien. To whom enjoyed pushing me off an SUP. To whom tried wake boarding. To whom teased me. To whom went winter bathing with me and to the sauna. To whom took care of me when I fainted. To whom worked from Lynfabrikken. To whom ranted about Incuba canteen, to whom enjoyed it. To whom came to the royal library (especially on the Saturday brunch). To OWF and PRG. To whom does not like t-shirts, sun cream, water bottles and proper equipment, but carries family-sized shampoo, is too generous and muscular and likes cooking robots and egg on top of pasta and pesto. To the walrus that dresses in orange and combs his moustache. To whom "climbing club" sounds like other types of clubs. To whom prepared chai tea, to whom took me surfing. To whom made me discover night trains. To whom went on a bike trip in the desert and snorkelled in a coral reef (and tolerated a suffering fool that got stung by corals). To Licious and Tuna. To whom sent tasty food from Italy. To whom is scared of buses. To whom camped by Moesgaard beach after an evening bonfire. To Giorgio. To whom enjoyed deer and hedgehogs. To spanna. To Mishmish and Søren. To whom tolerated my philosophical rants and who enjoyed them. To whom got me into woodworking. To whom threw away amazing wood on the streets of Aarhus. To fools and dreamers. To LaTeX. To whom gave me a mirror for free. To whom inspired me with their drawings. To whom shared deep thoughts at Domen, at Den Uendelige Bro and under a wind mill. To my favourite investigator. To whom called me a surf boy. To whom hates raisins. To whom slept in an aquarium. To whom joined for the bonfire night. To Alessandro Barbero. To whom played scopa at an Italian dinner. To whom made limoncello and to whom didn't know about it (as well as most places in Aarhus, despite living there for 9 years). To whom loves my pimples. To whom bakes cakes that are better than harbour water. To whom made Lennart drunk with a babà. To radio antanna. To whom pointed at mushrooms and brought the most useful paper in computer science (sorry for bringing an empty gas canister, but the barbecue pasta was super tasty!) To whom enjoys a shiny breakfast. To whom (did not) like in-n-out burritos, whom brought a trolley on a camping trip, whom got lost. To whom fell into a river while trying to cross on a fallen tree, to whom took amazing pictures. To whom tolerated a frustrated Italian chef turned influencer on a boat (especially when there was no Moussaka). To whom got beaten in the eye. To whom broke up in front of an Edeka. To my thick moustache. To whom I thought they hated me. To whom contributed to the soundtrack of my research. To Bruce. To Anna Tsing, Donna Haraway, Naomi Klein, Thomas Piketty (and whom made me discover them). To whom had to run in the mountains to avoid a storm, whom got sick after eating too much cheese but had sheep for groupies. To the person I stalked on a bus (your kidneys are safe). To whom convinced me to go on a three-day ferry trip. To whom discovered amazing view on forgotten paths at the edges of cliffs, to whom got attacked by nasty birds. To whom almost missed a ferry (sorry!) To whom woke up with the sound of crunched nuts in the ears and loved the amazing art of plastic bag origami at night. To whom caused lanceslides and fell down a slope magically saving all eggs in the backpack. To whom gets excited about cryptography more than anybody else. To whom survived a crazy rally with Mozart in the background and to the founders of bread-and-cheese town. To a goat. To whom taught me crocheting, and to the friend named after a swamp. To the crow that flew away from the oak. Finally, to the mushroom at the end of the world and the invisible hyphae connection between us. Thank you, I've never felt so seen, so understood and so happy. You made me forget what loneliness is. I hope to be able to bring in your life as much light as you brought in mine.

How to read this thesis

This thesis is split in two parts. Part I presents the research question we study, explaining its importance and embedding it into a broader cryptographic perspective. We also give a summary of the main results, presenting them in relation to other works produced by the community. Part I terminates with an overview of the techniques and mathematical arguments used in our research.

Part II lists the full versions of four manuscripts (three of them currently published in major IACR conferences). These are where the results in this thesis were first presented and formalised. Except for minor adjustments (summarised in Section 1.1.1), these manuscripts have not been edited and can be found online on the IACR ePrint Archive.

All readers are invited to start from Chapter 1 and refer to the manuscripts when additional details or more formal argumentations are desired (note that, sometimes, the notation in Chapter 1 differs from the one in the manuscripts in Part II). Inexperienced readers may struggle to switch from Section 1.1 to Section 1.1.1 due to the increasing use of cryptographic jargon. For these cases, in Section 1.2.1, the thesis provides a short, informal discussion of notation, preliminaries and basic cryptographic notions and techniques. If this would still not suffice, we recommend checking Oded Goldreich's *Foundations of Cryptography*[1].

[1]Goldreich O. *Foundations of Cryptography.* Cambridge University Press; 2001.

Contents

Part I

Overview

Chapter 1

Sampling Common Randomness in a Single Round

1.1 The Struggle for Secure Randomness

Randomness lies at the core of cryptography. Without this, basic concepts such as privacy and authentication would be unachievable. Encryption can be viewed as the art of hiding messages using randomness (taking the form of cryptographic keys): an encryptor merges a message and randomness into a tight, inseparable union, a new unpredictable object called a ciphertext. In this way, randomness protects the privacy of the message from any curious observer receiving the ciphertext: recovering the message is possible only if we possess (partial) knowledge of the randomness used by the encryptor. Authentication, on the other hand, can be viewed as the art of using randomness (again, taking the form of a cryptographic key) to design codes[1] whose elements are hard to find: we can discover codewords only if we possess (partial) knowledge of the randomness that produced the code.

Due to this tight and intrinsic connection to the world of probability and the fact that randomness is often non-reusable, cryptographic primitives make use of large amounts of randomness in the form of long sequences of uniform, independent, binary random variables. Generating true randomness, however, comes at a price: commonly, using specialised hardware that measures local physical phenomena subject to persistent, unpredictable variations. The randomness produced in this way is, however, raw: the outcomes of the measurements are often far from the independent, uniformly distributed bits required by cryptographic primitives. The data therefore undergoes further processing using algorithms called randomness extractors, which condense the unpredictability of the measurements into short, uniformly random strings of bits. The cost of this is high enough to motivate researchers into finding new, more efficient ways of generating randomness. This led to the introduction of cryptographic primitives such as pseudorandom generators (PRGs) and pseudorandom functions (PRFs): a short, truly random string of bits (called a seed or key) is deterministically expanded at a low price into a longer *pseudo-random* string behaving at all effects as if it was truly random[2].

Common random strings and common reference strings. Although PRGs and PRFs are a significant step forward in the study of randomness generation, they do not provide a definitive solution to the problem: in cryptography and, in particular, in multiparty computation (MPC), randomness assumes a large variety of shapes and only a small part of them can be reduced to individual entities locally sampling uniformly random strings independently of their external environment. Among the most common forms are *common random strings* and *common reference strings* (CRS). These consist of unpredictable binary strings made public at an unspecified point in time, before all participants (often called *parties* or *players*) start running

[1] A code consists of a subsets of binary strings of a given length.

[2] No further information about the seed or key must be revealed.

their cryptographic protocols. In the case of common random strings, the CRS is unstructured: it consists of a uniformly random string of bits of a set length. Common reference strings are, on the other hand, structured: they are described by arbitrary distributions. Typical examples are large RSA moduli [RSA78], i.e. integers obtained by multiplying two random prime numbers of a set bit-length, or KZG-like CRSs [KZG10], i.e. tuples $(g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^n})$ where g is a high order element in a multiplicative group G and α is an integer uniformly sampled over a finite interval.

CRSs are fundamental in cryptographic protocols, especially in the context of malicious security in the dishonest majority case[3]: without them, rigorous, modular and well-established security notions such as *universal composability* [Can01] would often be unachievable[4] [CKL03], and the list of unknown impossibilities keeps growing if we consider stronger security definitions such as adaptive corruption [IKOS10]. In simulation-based security, CRSs are often used to backdoor the protocol execution, allowing an algorithm called *simulator* to better understand and control how misbehaving participants act: we say that the protocol is secure if the simulator (essentially) always manages to manipulate all malicious parties into respecting the "ideal" behaviour of the computation without being noticed[5]. In other words, since the behaviour of the simulator goes unnoticed and the misbehaving parties (essentially) always fail at disrupting the protocol, it must be that the attacks of the malicious parties (essentially) always fail even when the simulator is not active. To summarise, CRSs and the backdoors (or trapdoors) hidden in them have allowed us to study secure computations in a modular and scalable way for many years, sensibly contributing to the development of cryptography in the academic and the real world. However, a natural question arises:

What happens if the CRS backdoor ends up in the wrong hands?

This usually is fatal for protocol security: backdoors often grant full control to the protocol execution. Malicious participants would be able to learn the inputs of all honest players and deviate the protocol execution into providing wrong outputs. Even worse, oftentimes the attacks go undetected as the simulator was. In other words, security is often a castle built on the foundation that no ill-willed entity is able to put its hands on the CRS trapdoor. When the foundation turns out to be wrong, the fortress crumbles.

Therefore, how do we generate CRSs in a secure way?

In the world of theoretical cryptography, the generation of a CRS is almost always entrusted to an idealised entity called the *trusted dealer* or the *trusted third party* (TTP): an honest-by-definition entity that generates the CRS sampling it according to the expected distribution, delivers it to all protocol participants and keeps all additional information including any hidden backdoor secret, abstaining from using it in any context. The issue is that this entity is very likely to not exist in the real world: aside from the fact that the common good-and-bad dichotomy may be just a illusory human construct, we live in a world where a part of the society believes (at conscious or unconscious level) that "maximising-personal-interests" leads to a good approximation of "collective-good". So, what happens if a real naïve-but-trusted dealer starts to use CRS trapdoors for (perhaps perceived-as-good) personal interests? We have seen this many times: consider the Snowden files or every time we hear about new mass-surveillance programs advertised as solutions against criminality. This is especially scary if we consider that in these examples the trusted-but-not-so-trustworthy dealer were public agencies and autocratic winds are currently on the rise all over the globe. On the other hand, we cannot expect private companies or individuals to behave any more ethically. On the contrary, due to the larger freedom they benefit from, the more centralised internal organisation and the limitedness of controls, the misuse of CRS backdoors is only more likely to go unnoticed. Finally, even if a completely honest entity existed, what would happen in case of any security breach leading to the theft of its CRS

[3]Malicious security refers to protocols that are secure even if a subset of participants deviates from the expected course of the computation and starts misbehaving, possibly colluding with other participants. Dishonest majority refers to protocols that are secure even if more than half of the participants misbehave.

[4]The universal composability model (UC) is an expressive security model introduced by Canetti. Unlike other common models such as standalone security, UC-secure protocols are guaranteed to be secure even if they are run in parallel with other computations. This is an extremely desirable property.

[5]The simulator achieves this by tampering with all information received by the misbehaving participants and by reading and modifying the communications they send.

trapdoors? This kind of attacks are becoming more and more common and they are especially concerning due to their growing warfare usage.

So, how can we generate CRSs without relying on a single dealer?

CRSs cannot be generated using PRGs and PRFs. First of all, the outputs of PRGs and PRFs are somewhat unstructured, whereas, as we have seen, CRSs are often structured. Even if we are dealing with common random strings, PRGs and PRFs do not constitute a solution. In the first place, it is unclear how to generate the public seed we would like to expand, secondly, even if we were able to magically obtain it, the output of a PRG expansion would not look random: these primitives guarantee security only as long as the seed remains secret (a similar argument applies to the case of PRFs). When the CRS is unstructured, applications often rely on idealised settings such as the *(programmable) random oracle model* (RO)[6] or on heuristics such as using a subsequence of digits of π. Although these are considerably better solutions, they are also far from perfect: the digits of π are clearly not random, how would we be able to compute them otherwise? Random oracles do not exist in the real world, so they are often heuristically instantiated by relying on particular families of (usually keyless) hash functions: Who samples the key of the hash function if it is not keyless? Who guarantees that the hash function and its key are not backdoored? Finally, if the hash function is keyless, as for the digits of π, we end up with a non-random CRS: we have a deterministic algorithm that allows to compute it.

Generating structured CRSs is an even harder task. As for PRGs and PRFs, the digits of π and hash functions lead to somewhat unstructured strings of bits. Moreover, often we cannot boil down the problem to generating an unstructured random string and then using the latter as randomness for the distribution describing the structured reference string. Indeed, revealing the randomness that produced the CRS often leaks the hidden backdoor. For example, in the case of RSA moduli, the randomness used by the sampling procedure would immediately provide the factorisation of the output; in the case of a KZG-like CRS, the randomness would leak the value of α. If the randomness is kept secret, computing the factorisation (or the value of α) from the CRS is considered an infeasible task. For this reason (and also their particularly nice algebraic properties), they have been extensively used as CRS trapdoors.

Multiparty computation. A possible solution to the problems described above is *multiparty computation*: instead of delegating the generation of CRSs to trusted, perhaps-trustworthy entities, the CRS is sampled by a cryptographic protocol between a larger number of participants, possibly including (some of) the parties that will later on make use of produced output. In this way, it is possible to generate CRSs that are secure even if all but one of the participants of the sampling protocol misbehave: as long as this condition is satisfied, all backdoors will remain secret. To summarise, we do not put all eggs in one basket.

This solution raises, however, a new question.

How can we design sampling protocols that guarantee the desired level of security at a minimal cost?

This is a particularly important question also in light of the fact that some CRSs in MPC constructions are *non-reusable*: for each protocol execution, the parties would need to rerun the sampling protocol to obtain a fresh CRS. While the multiparty generation of unstructured CRSs can be easily handled in a commit-then-reveal fashion, the situation for structured CRSs if usually much more involved. Practical solutions are often complex, require several rounds of interaction, high communication and expensive computations. Typical examples of this can be drawn from the vast literature on secure sampling of RSA moduli [BF97, FMY98, PS98, Gil99, HMRT12, FLOP18, CCD$^+$20, dMRT21, CHI$^+$21]. It is true that the cost of sampling protocols can often be amortised on the long run, however, this argument is based on the assumption that the parties *need* to perform intense computations in the first place. This is often true for the kind of commercial applications of MPC we often imagine: a small group of servers around the world is delegated the majority of (secure) computations on earth. What if we instead consider more decentralised applications? What if the parties are just end-users, normal people around the globe engaged in a network of secure information

[6]A random oracle consists of an entity which can be accessed from any place and at any time. On input any query, the oracle responds with a random binary string of set length. Repeated queries are answered consistently.

exchange? We cannot assume that people are interested in performing intense computations with the same subsets of players on an every-day scale and on the long run. Maybe they make use of MPC protocol rarely and every time with a different subset of participants! What if none of the end-users are happy with just relying on the CRS produced by some external entity they have no control upon? They would need to regenerate the CRS at each occasion! To summarise my point, expensive and slow sampling protocols hinder the responsiveness, the fluidity, the flexibility of secure computations: before the parties can even start computing their output, they need to get involved in a slow and costly setup protocol. This burdens the democracy of secure computation and information over the Internet, which is then inherited in the real world (and inequalities seem to feed on pre-existing inequalities).

The works presented in this thesis try to make a small step forward in understanding the *theoretical* efficiency of sampling protocols, focusing especially on their *round complexity* relative to their security guarantees. Before our research was carried out, the best known solutions were drawn from the study of 2-round MPC [GGHR14, MW16, GS17, BL18, GS18b, BL20], which trivially implies the existence of 2-round semi-honest sampling of any CRS. The starting point of the thesis will therefore be the following question:

Is it possible to design ONE-ROUND sampling protocols?

Sampling correlated randomness. Another form of randomness commonly used in cryptography is *correlated randomness*: each participant of the MPC protocol is provided with a secret random value correlated to those received by the other parties. Locally sampled randomness and CRSs can be viewed as extreme cases of correlated randomness: the first one corresponds to randomness with no correlation at all (the samples received by the parties are all independent), the other one to randomness that is fully correlated (the sample received by any party fully determines the samples received by everybody else).

Correlated randomness is not only fundamental in one-time trusted setups such as *public-key infrastructures* (PKI)[7], it has also become extremely popular in *MPC with preprocessing* [Bea92, BDOZ11, DPSZ12, NNOB12], leading the way to the rapid growth of multiparty computation in the last decade. These protocols are composed of two phases: an input-dependent (usually lightweight) secure computation requiring correlated randomness (usually in large amounts) and a (usually expensive) preprocessing phase where the correlated randomness is generated in a distributed way. The former is often called the *online phase*, the latter is referred as the *offline phase*. The type of correlated randomness required by these protocols is usually limited in its usage: it can be used at most once. Furthermore, in many cases, MPC protocols requires amounts of correlated randomness scaling proportionally to the size of the circuits describing the computation (e.g. one tuple of correlated material per multiplication gate). This grows the amounts of necessary preprocessed material to significant levels. For this reason, production and storage of the correlated material are often the bottleneck of MPC protocols with preprocessing, increasing the urge for new, more efficient solutions.

Of course, if a trusted dealer existed in the real world, we could delegate it with the task of generating and distributing correlated randomness. This, however, not only leads to the same problems we discussed for CRSs, it also requires significant effort at the dealer side, requiring it to constantly be online, ready to answer the relentless requests for more correlated material from the protocol participants. Once again, it is therefore fundamental to rely on MPC protocols that guarantee the security of the generated correlated material even if a subset of participants misbehaves.

The pattern we saw for structured CRSs repeats again: known MPC protocols for correlated randomness generation are often complex, require multiple rounds of interaction and high costs, both in terms of computations and communication [DPSZ12, KOS16, KPR18]. This is not only problematic in terms of amortised efficiency, it also affects negatively the responsiveness, the fluidity, the flexibility, the democracy of MPC protocols: before the parties can even start to compute their output, they need to get involved in an slow and costly setup protocol.

A recent break-through on the topic was the introduction of *pseudorandom correlation generators* (PCGs) and *pseudorandom correlation functions* (PCFs) [BCG$^+$19b, BCG$^+$19a, BCG$^+$20b, BCG$^+$20a]: these are

[7]In a PKI, at an unspecified point in time, before the beginning of the protocol execution, the participants are provided with (possibly correlated) private key material.

primitives describing how to securely obtain large amounts of correlated material by locally expanding small correlated seeds given to the protocol participants. By relying on PCGs or PCFs it is possible to design low-communication, low-storage offline phases for MPC protocols; computation and round complexity in practical solutions remain, on the other hand, usually high. An exception are *public-key PCFs*: one-round protocols that allow to produce large amounts of correlated randomness with sublinear communication in the size of the outputs. At the time the works described in this thesis began, public-key PCFs were known only for particular types of correlation, namely *oblivious transfer* (OT) and *vector oblivious linear evaluation* (VOLE) [OSY21]. The second question we try to answer is therefore the following:

Is it possible to design PUBLIC-KEY PCFs for ANY correlation?

1.1.1 A Summary of the Main Results

This thesis encompasses the results presented in four papers:

- Damiano Abram, Peter Scholl, Sophia Yakoubov. *Distributed (Correlation) Samplers: How to Remove a Trusted Dealer in One Round.* (EUROCRYPT 2022) [ASY22a, ASY22b] (see §2)
- Damiano Abram, Maciej Obremski, Peter Scholl. *On the (Im)possibility of Distributed Samplers: Lower Bounds and Party-Dynamic Constructions.* (Not yet published manuscript) [AOS23] (see §3)
- Damiano Abram, Brent Waters, Mark Zhandry. *Security-Preserving Distributed Samplers: How to Generate any CRS in One Round without Random Oracles.* (CRYPTO 2023) [AWZ23a, AWZ23b] (see §4)
- Damiano Abram, Jack Doerner, Yuval Ishai, Varun Narayanan. *Constant-Round Simulation-Secure Coin Tossing Extension with Guaranteed Output.* (EUROCRYPT 2024) [ADIN24a, ADIN24b] (see §5)

The full versions of these works (available on the ePrint archive [epr]) have been listed in Part II of this thesis after applying minor adaptations (such as change of format, moving security proofs and preliminaries from the appendices to the main body and fixing references). Due to space constraints, some of the results of [AOS23] and [AWZ23a] were not included in the thesis. In particular, the omitted sections concern the study of *unbounded universal samplers*[8] [AOS23, Section 4], *party-dynamic distributed samplers*[9] [AOS23, Section 5] and *CRS-less NIZKs with security against uniform adversaries* [AWZ23a, Sections 9-10]. At the base of this choice is only the will to narrate a more concise and linear story about sampling randomness in one round.

Sampling CRSs in one round: semi-honest security. Our story starts by studying one-round sampling protocols in the easiest setting: *semi-honest security*[10]. We consider a particular type of one-round sampling protocols in which the output can be publicly derived from the transcript. We call this a *distributed sampler*. More formally, a distributed sampler for a distribution $\mathcal{D}(1^\lambda)$ consists of a pair $(\mathsf{Gen}, \mathsf{Sample})$ where Gen is an algorithm used by all parties to generate the message they send in the only round of interaction and

[8]An unbounded universal sampler can be view as a cryptographic object that, on input the description of any distribution, produces a sample without leaking any additional information. The construction is unbounded in the sense that there exists no bound on the set of input distributions. In particular, the sampler can produce secure samples from distributions that are arbitrarily bigger than the sampler itself.

[9]A party-dynamic distributed sampler consists of a one-round sampling protocol in which the set of participants is dynamic: the messages exchanged in these one-round protocols are independent on the identities and the number of other players. This gives rise to sampling protocols in which the participants do not need to be online: it is sufficient for them to publish a single message on a bulletin board (e.g. blockchain). To generate a sample, it is sufficient to recover the messages published by an arbitrary subset of trusted players without them having to return online. The output is secure as long as one of the trusted parties behaves honestly.

[10]A semi-honest (or passive) protocol guarantees security only if all parties follow the protocol description. Notice that this security notion is not trivial, as a protocol may leak problematic information (e.g. CRS trapdoors) even if all parties behave honestly.

Ideal Functionality for Semi-Honest Distributed Samplers $\mathcal{F}_{\mathcal{D}}$

On input Sample from all parties, compute $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$ and output it to all participants and the adversary.

Figure 1.1: Ideal functionality for semi-honest distributed samplers. The functionality remains unvaried in the non-rushing semi-malicious case.

Sample is a deterministic procedure that, on input the exchanged messages, returns the output of the computation. We formalise the security of semi-honest distributed samplers using a simulation-based definition: we require the obtained protocol to implement the ideal functionality in Figure 1.1.

In [ASY22a], we studied semi-honest distributed samplers for any efficient distribution $\mathcal{D}(1^\lambda)$ and any (polynomial) number of parties[11]. We restricted our security analysis to PPT adversaries statically corrupting any number of parties, but leaving at least one honest[12]. We show that, in this setting, under strong assumptions such as indistinguishability obfuscation (iO) [BGI+01, GGH+13, JLS21, JLS22] and multi-key fully homomorphic encryption (multi-key FHE) [LTV12, MW16, AJJM20], building distributed samplers is always possible. Furthermore, the construction we provide remains secure even if the adversary is allowed to maliciously choose the randomness of the corrupted players *before the protocol execution starts* (note: the adversary is not allowed to use rushing). As for the semi-honest case, the corrupted players are still required to follow the protocol. We call this stronger version of adversary *non-rushing semi-malicious.*

Theorem 1.1.1 (Informal version of Theorem 2.4.1). Let $n(\lambda)$ be a polynomial function in the security parameter. Assume the existence of (polynomially secure[13]) iO and (polynomially secure) multi-key FHE. Then, in the context of computational security, there exists an n-party non-rushing semi-malicious distributed sampler for any efficient distribution $\mathcal{D}(1^\lambda)$ withstanding up to $n-1$ static corruptions.

Sampling CRSs in one round: active security. The next setting we considered is *active security* (also called *malicious security*). Semi-honest security and similarly non-rushing semi-malicious security are based on strong assumptions: in the real world, it is implausible that corrupted players do not take full advantage of their opportunities and keep following the protocol description. It is therefore natural to study the security of distributed samplers in presence of maliciously (mis)behaving participants.

Unfortunately, we immediately run into an issue: due to a well-known result by Cleve [Cle86] on the impossibility of coin tossing, the functionality $\mathcal{F}_{\mathcal{D}}$ in Figure 1.1 cannot be implemented against active adversaries in the dishonest majority setting (notice that when $\mathcal{D}$ is the uniform distribution over $\{0, 1\}$, $\mathcal{F}_{\mathcal{D}}$ becomes the coin-tossing functionality). More in general, there exists a class of attacks that are inherent to all one-round MPC protocols. An active adversary can always use *rushing behaviour*: as soon as the honest parties speak, the adversary can rush to see their messages, choosing only at that point what to send on behalf of the corrupted players (all of this happens in the same round). In particular, the adversary can "grind" multiple choices of the corrupted messages, until it finds an output it likes. To summarise, maliciously secure distributed sampler protocol inevitably allow (limited) influence to the adversary: the adversary can try different (but polynomially many) protocol executions in its head (all of them with the same of choice of honest messages) and choose the one that appreciates the most. To take this into account, in the malicious setting, we modify the functionality $\mathcal{F}_{\mathcal{D}}$ as in Figure 1.2. Notice that if we generate the CRS of any secure MPC protocol using $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$, the protocol remains still secure.

[11]An efficient distribution consists of a distribution that is described by a polynomial-sized circuit mapping a sequence of uniformly random bits into a sample.

[12]Static corruption requires the adversary to choose the set of corrupted players before the beginning of the protocol execution, allowing no subsequent changes. If instead the adversary is free to corrupt the participants even after the protocol started running, we talk about adaptive corruption.

[13]We use the term polynomial security as opposed to subexponential security. A primitive is subexponentially secure if there exists a constant $\epsilon > 0$ such that, for every $O(2^{\lambda^\epsilon})$-time probabilistic adversaries, the construction retains its security properties with $O(2^{-\lambda^\epsilon})$ advantage. A primitive is polynomially secure if, for every PPT adversary, the advantage is negligible in λ.

Ideal Functionality for Active Distributed Samplers $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$

Initialisation: On input Init from all parties set $Q \leftarrow \emptyset$. If all parties are honest, output $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$.
Query: On input $(\mathsf{Query}, \mathsf{id})$ from the adversary where id is a not-yet-queried label, sample $R_{\mathsf{id}} \xleftarrow{\$} \mathcal{D}(1^\lambda)$, set $Q \leftarrow Q \cup \{(\mathsf{id}, R_{\mathsf{id}})\}$ and provide R_{id} to the adversary.
Output: On input $(\mathsf{Output}, \widehat{\mathsf{id}})$ from the adversary, retrieve the pair $(\widehat{\mathsf{id}}, R_{\widehat{\mathsf{id}}})$ from Q. If there exists at least one corrupted party, output $R_{\widehat{\mathsf{id}}}$ and halt.

Figure 1.2: Ideal functionality for active distributed samplers. The functionality remains unvaried in the rushing semi-malicious case.

The study of active distributed samplers began in [ASY22a]. In the work, we proposed a compiler capable of converting any one-round, *inputless* protocol with security against non-rushing semi-malicious adversaries into a one-round *maliciously secure* protocol implementing (a slightly modified version of) the same functionality. Such compiler, named *anti-rusher*, is build from (polynomially secure) iO and non-interactive zero-knowledge (NIZK) in the *programmable random oracle model.* By applying the anti-rusher compiler on the non-rushing semi-malicious distributed samplers of Theorem 1.1.1, we obtain the following result.

Theorem 1.1.2 (Informal version of Theorem 2.5.3). Let $n(\lambda)$ be a polynomial function in the security parameter. Assume the existence of (polynomially secure) iO and (polynomially secure) multi-key FHE and (polynomially secure) NIZKs for NP. Then, in the context of computational security in the UC model with *programmable random oracle*, there exists an n-party maliciously secure distributed sampler for any efficient distribution $\mathcal{D}(1^\lambda)$ withstanding up to $n-1$ static corruptions.

Active security and random oracles. The result in Theorem 1.1.2 suffers from an important disadvantage: it relies on a programmable random oracle. Random oracles are indeed an idealised model that cannot be instantiated in the real world, and programmable random oracles are an even stronger flavour of the setting. In [AOS23], we tried to answer the question of whether the functionality in Figure 1.2 can be implemented without random oracles in the actively secure setting. To simplify the task we considered also distributed samplers that rely on CRSs. At first this might seem strange: what is the point of using distributed samplers to generate CRSs if the latter need CRSs too? We argue that if the CRSs they rely upon is reusable or particularly easy to generate (for instance because short or unstructured), distributed samplers of this type can still be interesting. At least for the case of universal composability (UC) [Can01], we showed that none of this is possible. The lower-bound is based on the following result.

Theorem 1.1.3 (Informal version of Theorem 3.4.1). Suppose that there exists a active distributed sampler for the distribution $\mathcal{D}(1^\lambda)$ withstanding any number of static corruptions in the UC model. Suppose that the protocol relies on a CRS σ, then the Shannon entropy[14] of the output R conditioned on σ is

$$\mathsf{H}(R|\sigma) = O(\log \lambda).$$

The theorem is essentially stating that the CRS σ almost determines the output of the distributed sampler. This has three negative repercussions: the CRS is non-reusable (see Corollary 1.1.4), it cannot be short (see Corollary 1.1.5) and it cannot be unstructured unless $\mathcal{D}(1^\lambda)$ was obliviously samplable to begin with (in other words, there existed a way to deterministically convert public, uniformly random coins into a secure sample from $\mathcal{D}$. See Corollary 1.1.6). To summarise, without random oracles, actively secure distributed samplers that implement the functionality $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ essentially provide no advantage over the trusted dealer that generates a public sample from $\mathcal{D}(1^\lambda)$!

Corollary 1.1.4 (Informal version of Corollary 3.4.2). Two executions of a UC-secure, active distributed sampler reusing the same CRS produce the same output with inverse polynomial probability.

[14]The conditional Shannon entropy $\mathsf{H}(R|\sigma)$ measures how unpredictable R is once σ is known.

Corollary 1.1.5 (Informal version of Corollary 3.4.3). In a UC-secure active distributed sampler, the CRS size is at least

$$|\sigma| \geq \mathsf{H}_{\mathsf{Yao}}(\mathcal{D}) - O(\log \lambda),$$

where $\mathsf{H}_{\mathsf{Yao}}(\mathcal{D})$ denotes the Yao entropy [Yao82] of the distribution $\mathcal{D}$[15].

Corollary 1.1.6 (Informal version of Corollary 3.4.4). A UC-secure, active distributed sampler can have an unstructured CRS if and only if the underlying distribution $\mathcal{D}(1^\lambda)$ is *obliviously samplable*, i.e., there exists a way to generate secure samples from $\mathcal{D}$ using public random coins and no interaction.

All the results we described above were proved to hold for an even weaker class of attacks: *rushing semi-malicious adversaries*, which are forced to follow the protocol instructions, but are allowed to maliciously choose the randomness of the corrupted players after seeing the messages of the honest parties. This makes our lower bounds even stronger.

Working around the impossibility: security-preserving distributed samplers. With the lower bounds we just described, we abandoned all hopes of obtaining UC-secure, active distributed sampler in the dishonest majority setting. In [AWZ23a], we tried to work around the impossibilities. After failing in different security settings such as *superpolynomial simulation*, *honest majority* and *standalone security*, we decided to try a different path: instead of aiming for simulation-security, we considered new game-based definitions. This led to the introduction of two new notions: *hardness-preserving distributed samplers* and *indistinguishability-preserving distributed samplers*.

Hardness-preserving distributed samplers. Hardness-preserving distributed samplers preserve the hardness of search problems: suppose that retrieving a trapdoor T hidden in a sample $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$ is infeasible for all PPT adversaries. Then, if R is generated by executing a hardness-preserving distributed sampler, allowing the adversary to maliciously control up to $n-1$ parties, retrieving the trapdoor T remains still infeasible. More in general, hardness-preserving distributed samplers can be used to remove CRSs from MPC protocols while guaranteeing a weak form of security against active adversaries: all attacks that failed with *overwhelming probability* against the protocol in the CRS model, still fail with *overwhelming probability* even if the CRS is generated by an hardness-preserving distributed sampler (again the adversary is allowed to control a proper subset of parties).

In [AWZ23a], we showed how to build a hardness-preserving distributed sampler (relying on a reusable, unstructured λ-bit CRS) from subexponentially secure iO, subexponentially secure multi-key FHE, extremely lossy functions (ELFs) [Zha16] and particular NIZKs we called *almost-everywhere extractable*. The latter consist of witness-extractable NIZKs for which finding accepting proofs that make the extractor fail is particularly hard. Indeed, so hard that it is possible to apply the trick of [BCP14][16] to argue that the obfuscations (using iO) of the following programs are indistinguishable despite the existence of differing inputs: both programs receive an input and an almost everywhere extractable NIZK proving its welformedness. The programs perform the same operations on the provided input conditioned on the successful verification of the NIZK (in case of a failure, the programs output $\perp$). The second program, however, tries also to extract the witness from the proof, outputting $\perp$ in case of a failure.

Theorem 1.1.7 (Informal version of Theorem 4.8.1). Assume the existence of subexponentially secure iO, subexponentially secure multi-key FHE, ELFs and almost everywhere extractable NIZKs for NP. Then, there exists an n-party hardness-preserving distributed sampler for any efficient distribution $\mathcal{D}(1^\lambda)$.

Theorem 1.1.8 (Informal version of Theorem 4.4.6). Assume the existence of perfectly correct identity based encryption (IBE), perfectly binding non-interactive commitments, subexponentially secure injective one-way

[15]The Yao entropy measures the degree to which the samples of a distribution can be efficiently compressed without losing information. In other words, $\mathsf{H}_{\mathsf{Yao}}(\mathcal{D})$ describe the size of the most compact encoding of the samples produced by $\mathcal{D}$.

[16]In [BCP14], Boyle, Chung and Pass show that if we can distinguish between two obfuscated program having a polynomial number of differing inputs, then we can find one of these differing inputs in polynomial time. The strategy is to reobfuscate the programs fixing some of their input bits and perform some sort of binary search: if the resulting obfuscations are still distinguishable, it must be that the input bits we fixed are consistent with at least one differing input.

functions (OWFs) and perfectly sound witness-indistinguishable proofs for NP (NIWIs). Then, there exist almost everywhere extractable NIZKs for NP.

Indistinguishability-preserving distributed samplers. Indistinguishability-preserving distributed samplers preserve the functionality of the protocols they compile if particular conditions are satisfied: suppose that we deal with a protocol Π relying on CRS. Suppose also that Π implements a functionality $\mathcal{F}$ and, more crucially, its simulator produces the simulated CRS *before* interacting with $\mathcal{F}$ (in other words, the simulated CRS is independent of any information held by the functionality). Then, if we generate the CRS of Π using an indistinguishability-preserving distributed samplers, the resulting protocol still implements $\mathcal{F}$.

In [AWZ23a], we show that our hardness-preserving distributed sampler is also indistinguishability preserving. The construction relies, however, on a short, reusable, unstructured CRS (this is unavoidable, otherwise we would be able to obtain 3-round actively secure OT in the plain model [PVW08, HV16]).

Theorem 1.1.9 (Informal version of Theorem 4.8.6)*.* Assume the existence of subexponentially secure iO, subexponentially secure multi-key FHE, ELFs and almost everywhere extractable NIZKs for NP. Then, there exists an n-party indistinguishability-preserving distributed sampler for any efficient distribution $\mathcal{D}(1^\lambda)$.

Unbiased sampling with active security: coin tossing extension. In the last paper presented in this thesis [ADIN24a], we go back to the question of implementing the functionality in Figure 1.1 *with guaranteed output* in a presence of an active adversary corrupting a dishonest majority of participants. As we have already explained, such functionality cannot be implemented in the plain model (nor if we rely on CRSs or random oracles) due to Cleve's impossibility [Cle86]. What happens however if we assume that the parties have access to a copy of $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$ where $\mathcal{U}_k$ denotes the uniform distribution over $\{0,1\}^k$ for a small value $k(\lambda) = \omega(\log\lambda)$? Does the impossibility still hold? The question is interesting as it is not hard to imagine ways $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$ can be implemented: for instance, we can use the randomness produced by a beacon on a blockchain, or we could rely on some physical device.

We started by considering the problem for $\mathcal{D} = \mathcal{U}_m$, where $m(\lambda) > k(\lambda)$ is any polynomial function and $\mathcal{U}_m$ denotes the uniform distribution over $\{0,1\}^m$. We are essentially asking whether there is a way to extend $k(\lambda)$ bits of unbiased randomness with security against active adversaries corrupting a dishonest majority of participants. It turns out that this problem has already been studied; it is called *coin tossing extension* (CTE) [BGR96, HMU06]. Clearly, the question becomes trivial in the programmable random oracle model. In the plain model, however, the problem turns out to be surprisingly interesting and elegant (we recall that, as argued in the introduction, we cannot obtain a CTE protocol by simply expanding the string produced by $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$ using a PRG).

In [HMU06], Hofheinz, Müller-Quade and Unruh presented a statistically secure, 1-round CTE protocol in the standalone model (the authors considered only CTE with abort, not realising that their construction has actually guaranteed output). The protocol achieves only $O(\log\lambda)$ stretch, meaning that in each protocol execution, we extend the $k(\lambda)$ bits provided by $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$ to $k(\lambda) + O(\log\lambda)$ bits of unbiased randomness. If we would like to produce $\omega(\log\lambda)$ randomness, we would therefore need to repeat $\omega(\log\lambda)$ sequential executions of the protocol.

Is it possible to design $O(1)$-round CTE protocols with $\omega(\log\lambda)$ stretch?

Coin tossing extension: positive results. In [ADIN24a], we showed that, if we restrict our security analysis to efficient adversaries, the answer to the above question is (likely) yes. Our most important result is the following.

Theorem 1.1.10 (Informal version of Theorem 5.5.3)*.* Let $m(\lambda)$ and $n(\lambda)$ be any polynomial functions. Under the hardness of learning with errors (LWE) with subexponential modulus-to-noise ratio [Reg05], there exists a one-round, n-party CTE protocol with stretch $m(\lambda)$. The protocol requires no CRS and a single call to $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$. Furthermore, it is secure against adaptive corruption in the UC model.

As our second result, we introduce an algebraic framework where it is possible to build 1-round UC-secure coin-tossing extension with arbitrary polynomial stretch. The framework, called *hidden subgroup framework*, consists of a group G hiding a smaller subgroup H (we require that the uniform distribution over G and H are indistinguishable) and can be instantiated based on Paillier, on class groups, or on decisional Diffie-Hellman (DDH). The resulting CTE protocol makes however use of a CRS.

Theorem 1.1.11 (Informal version of Theorem 5.6.4)*.* Let $m(\lambda)$ and $n(\lambda)$ be any polynomial functions. Assume the existence of simulation-extractable NIZKs for NP. Under the hardness of one of the following

- DDH over prime order groups [DH76];
- quadratic residuosity (QR) and decisional composite residuosity (DCR) over the Paillier group [Pai99];
- hard subgroup membership (HSM) over class groups [CL15];

there exists a one-round, UC-secure, n-party CTE protocol with stretch $m(\lambda)$. The protocol requires a (reusable) CRS and a single call to $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$.

The final positive result we present is an n-party CTE protocol with arbitrary stretch based on one-way functions (OWFs). The constructions, however, requires $O(n)$ rounds and is only standalone secure.

Theorem 1.1.12 (Informal version of Theorem 5.7.3)*.* Let $m(\lambda)$ and $n(\lambda)$ be any polynomial functions. Under the existence of OWFs, there exists a $O(n)$-round, n-party CTE protocol with stretch $m(\lambda)$. The protocol requires a single call to $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$.

Coin tossing extension: negative results. In [ADIN24a], we prove also a lower-bound for statistical security in the standalone model with black-box simulation.

Theorem 1.1.13 (Informal version of Theorem 5.2.6)*.* Any R-round CTE protocol with statistical security in the standalone model with black-box simulation has $O(R \cdot \log \lambda)$ stretch.

The result therefore proves that the CTE construction in [HMU06] is optimal.

On the relation between coin tossing extension and randomness extractors. Coin tossing extension protocols are tightly related to *seeded randomness extractors.* A randomness extractor consists of a deterministic function that transforms, with the aid of a short, uniformly random seed, any sufficiently-high entropy material into a long (essentially) random string of bits. More specifically, we can regard a CTE protocol as a randomness extractor for the class of entropy sources outputting the transcript of a CTE execution where at least one party behaves honestly. The seed consists in the output of the last call to the auxiliary functionality $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$. Randomness extractors necessitate the seed to be independent of the material produced by the entropy source. This property is ensured by the following result: in a CTE protocol, any subsequent round of interaction after the last call to $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$ is useless.

Theorem 1.1.14 (Informal version of Theorem 5.4.3)*.* Let Π be a coin tossing extension protocol. Let Π' be the protocol in which all parties behave as Π until the last call to $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$, after which all parties stop. Then Π' is still a secure coin tossing extension protocol.

We observe that the extractors we obtain from CTE protocols satisfy an interesting, somewhat surprising property: given a description of the entropy source and a random sample r in the output space, we are able to simulate an extractor execution producing r. We call extractors of this kind *explainable extractors.* By applying this observation on the construction of Theorem 1.1.10, we obtain the following corollary.

Corollary 1.1.15 (Informal version of Corollary 5.5.7)*.* Consider the class of entropy sources $\mathcal{S}$ producing the transcript of a protocol where all parties simultaneously broadcast a unformly random string and a PPT adversary can maliciously corrupt a dishonest majority of participants (but not their totality).

Under the hardness of LWE with subexponential modulus-to-noise ratio, there exists an explainable extractor for $\mathcal{S}$.

Unbiased sampling from any distribution. Finally, in [ADIN24a], we abandoned the study of coin tossing and we reverted back to the more general case of securely sampling from any arbitrary distribution $\mathcal{D}(1^\lambda)$. We have seen that, with the help of $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$, it is possible to circumvent Cleve's impossibility [Cle86], producing randomness free from any form of adversarial influence. We ask:

Is it possible to implement the functionality $\mathcal{F}_\mathcal{D}$ in Figure 1.1 with the help of $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$?

In other words, if we keep relying on the help of $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$, are we able to securely produce samples from any distribution $\mathcal{D}(1^\lambda)$ leaving no influence to the adversary? By combining the techniques we used to build 1-round CTE with indistinguishability-preserving distributed samplers, we show that the answer is yes.

Theorem 1.1.16 (Informal version of Theorem 5.9.5)*.* Assume the existence of indistinguishability-preserving distributed samplers, iO and injective, length-doubling PRGs. Then, there exists a one-round, n-party protocol securely realising the functionality $\mathcal{F}_\mathcal{D}$ (see Figure 1.1) in the $\mathcal{F}_{\mathcal{U}_\mathsf{k}}$-hybrid model. The protocol guarantees UC security against an active PPT adversary statically corrupting up to $n-1$ parties.

Sampling correlated randomness in one round: public-key PCFs. We now move to the question of generating correlated randomness in one round. In other words, we study public-key PCFs: one round protocols for the generation of large amount of correlated randomness with sublinear communication in the size of the outputs.

In [ASY22a], we showed how to build public-key PCFs from distributed samplers, iO and public-key encryption. We considered two definitions for public-key PCFs. The first one is based on the concept of reverse samplability [BCG$^+$19b]: we say that a form of correlated randomness is reverse-sampleable if, for any subset of corrupted parties, it is possible to simulate the correlated material of the honest parties from that of the corrupted players. Such simulator is called the *reverse-sampler.* Following this blueprint, we defined the security of the first family of public-key PCFs asking that the correlated material they produce for the honest parties is indistinguishable from the one we obtain by reverse-sampling the output of the corrupted players. In other words, we are implementing the functionality that lets the adversary choose the correlated material obtained by the corrupted players and then uses it to reverse-sample the output of the honest parties. For many applications of correlated randomness, this definition is often enough to guarantee security.

Theorem 1.1.17 (Informal version of Theorems 2.6.8, 2.6.9 and 2.6.10)*.* Assume the existence of non-rushing semi-malicious, n-party distributed samplers for any distribution, (polynomially secure) iO and public-key encryption (PKE). Then, there exits a non-rushing semi-malicious, n-party, public-key PCF for any reverse-samplable correlation.

The construction can be upgraded to active security by additional assuming the existence of NIZKs for NP and moving to the programmable random oracle model. Furthermore, in case the correlation is reverse-samplable with subexponential security, we can obtain actively secure public-key PFCs without random oracles by relying on subexponentially secure distributed samplers, subexponentially secure iO, subexponentially secure PKE and polynomially secure NIZKs.

In all these constructions, the communication is logarithmic in the size of correlated material they produce.

The second class of public-key PCFs we studied in [ASY22a] guarantees a stronger form of security: we implement the functionality that samples the correlated material and distributes it to all participants leaving no influence to the adversary. In other words, the correlated material obtained by the parties, including that of the corrupted players, looks like as if it was generated by the targeted correlation function. Unfortunately, this type of public-key PCFs can achieve sublinear communication in the size of the outputs only in the random oracle model [BCG$^+$19b]. If the oracle is programmable, however, it is possible to design public-key PCFs that are tailored to no specific correlation: the latter can be chosen, after the parties have sent their messages. We call this primitive an *ideal public-key PCF.*

Theorem 1.1.18 (Informal version of Theorem 2.7.8)*.* Assume the existence of active, n-party distributed samplers for any distribution (implementing the functionality in Figure 1.2), adaptive universal samplers

[HJK$^+$16] and public-key encryption (PKE). Then, there exits an active n-party, ideal public-key PCF in the programmable random oracle model.

1.1.2 Personal Contribution

In this section, I will summarise, as truthfully as I can, my contribution to the works that are part of this thesis. As a disclaimer, however, the perspective that the PhD school forces me to take here is far from my personal and moral view on the matter: I am not sure I believe in the concept of merit, and I am even more sceptical of the fact that merit can be quantified, added and divided as if it was a material thing. I prefer to view "my" research as the result of the work that the whole society carried out over ages, instead of just three-four names listed under the title of the work.

I believe that focusing on the researcher instead of the research (as I am forced to do below) promotes misleading and harmful perspectives of individualism and self-narration. These views then ripple throughout the world of research in many different forms, such as impostor syndrome, anxiety, objectification (researchers are more than their mere abilities), passivity.

If you believe that this society needs fewer stories of heroes and more stories of kind collaboration, as Ursula Le Guin argues in "The Carrier Bag Theory of Fiction" [Gui19], you are invited to skip the following paragraphs and read about who contributed to this work in the acknowledgement section.

A summary of personal contributions. All papers were written entirely by me with minor corrections from the other authors. The exceptions are all introduction sections and some of the technical overviews: Section 2.1 [ASY22a, Section 1] (written by Sophia Yakoubov), Sections 1, 2.2 and 2.3 of [AOS23] (written mostly by Peter Scholl with minor contributions from Maciej Obremski and myself), Section 1 (written by Mark Zhandry) and Section 2 (equal contribution between Mark Zhandry and myself) of [AWZ23a], Section 1 of [ADIN24a] (written by Jack Doerner).

The work in [ASY22a] comes from an intuition of Peter Scholl and Sophia Yakoubov about using iO and multi-key FHE to build public-key PCFs. The introduction of distributed samplers, their definition, the semi-honest construction and its security proof are all due to me. The same holds for the introduction, definition, construction and security proof of anti-rusher compilers. The ideas on how to define public-key PCFs and build them from distributed samplers were an equal contribution between Peter Scholl and I. The relative security proofs were my contribution.

The intuition behind the lower bounds presented in [AOS23] is due to me, whereas the question of building party-dynamic distributed samplers was proposed by Peter Scholl. The impossibility results for active distributed samplers were proved by myself with the support of Maciej Obremski. The introduction, the definition and construction of unbounded distributed samplers are due to myself (unfortunately, we discovered that some of the techniques had already been used in [GS18a]). The same holds for the definition and construction of party-dynamic distributed samplers.

In [AWZ23a], the introduction and definition of hardness-preserving distributed samplers is due to Mark Zhandry. The same was for the intuition that the primitive was connected to extremely lossy functions, and almost-everywhere extractable NIZKs could be built by relying on the trick of [BCP14]. The definition of indistinguishability-preserving distributed samplers, lossy distributed samplers and all constructions and proofs are my own contribution. The question of building CRS-less NIZKs with security against uniform adversaries, as well as the basic blueprint was proposed by Brent Waters. The rest of constructions and security proofs were my contribution.

In [ADIN24a], the study of coin-tossing extension was proposed by Yuval Ishai. All constructions, lower bounds and proofs were my own contribution. The only exception was the intuition behind explainable extractors and their relationship with CTE, which is due to Yuval Ishai. Lemma 5.8.1 and its proof is due to Varun Narayanan.

1.1.3 Related Work

We now give an overview of related work.

Non-interactive MPC. Non-interactive MPC (NIMPC) [BGI+14a, HIJ+17] allows the evaluation of a deterministic function f on the inputs provided by the parties using a single round of interaction. It could seem that using this primitive, we can immediately obtain distributed samplers: why not to evaluate the function that XORs the strings input by the parties and then uses the result as randomness for the distribution $\mathcal{D}(1^\lambda)$? This idea does not work as non-interactive MPC provides too weak security guarantees. This is due to an attack the cannot be prevented in any one-round protocol: *the residual function attack*. Specifically, an adversary can always rerun the protocol in its head changing the inputs of the corrupted parties, while keeping the messages of the remaining participants unvaried. Since the protocol requires a single round, at the end of this imagined execution, the adversary learns the output of the modified evaluation. In other words, by repeatedly applying this attack on the naïve sampling protocol, the adversary can obtain many correlated samples. For some distributions $\mathcal{D}(1^\lambda)$, this could be a serious problem. One could therefore try to find a more clever way to generate the randomness we feed into $\mathcal{D}(1^\lambda)$. Instead of XORing the inputs of the parties, we could perform more complex operations so that even if the adversary reruns the protocol in its head, the randomness we feed into $\mathcal{D}(1^\lambda)$ looks independent of the one used in the previous imaginary executions. There is however another problem: non-interactive MPC relies on a PKI even in the case of semi-honest security. The latter ensures that in the imaginary executions, the adversary can only regenerate the messages of the corrupted players. Since we are interested in building semi-honest distributed samplers in the plain model, we need to try something new.

Non-interactive key exchange. Non-interactive key exchange (NIKE) [DH76, KRS15] allows a set of parties to agree on a secret random key using a single round of interaction. The primitive differs from distributed samplers from three points of view. The first one is the distribution of the produced sample: while NIKEs generate a uniformly random string of bits, distributed samplers tackles generic distributions. The second big difference is that distributed samplers produce public samples: their output is computable also by external entities that just happen to see the messages exchanged by the parties. Finally, while a NIKE guarantees that its output looks random only if all participants are honest, distributed samplers provide much stronger security guarantees: their output looks random even if there exists only one honest player.

Universal Samplers. A universal sampler can be viewed as a cryptographic object allowing the generation of samples from the distributions it receives as input. The primitive was introduced by Hofheinz et al. [HJK+16] building it from indistinguishability obfuscation. Universal samplers come in two flavours: selective universal samplers, which guarantee security only for a single distribution chosen before the generation of the sampler, and adaptive universal samplers, which allow the secure generation of arbitrarily many samples from adaptively chosen distributions (adaptive universal samplers can exist only in the random oracle model [HJK+16]).

There are two major differences between universal samplers and distributed samplers: the first one is that universal samplers guarantee security only if they are generated by a trusted setup, whereas distributed samplers are, as the name suggest, distributed, ensuring the security of their outputs as long as at least one party is honest. The second main difference is that universal samplers are indeed universal: they are not tailored to any specific distribution. Distributed samplers on the other hand are interesting even if the distribution of the samples is fixed. Of course, we can also consider a stronger definition of distributed samplers in which the parties can sample elements from any distribution. This primitive exists, it is called *distributed universal samplers* and it can be trivially obtained by using a distributed sampler to produce a universal sampler.

Spooky encryption. In [DHRW16], Dodis et al. introduced *spooky encryption*. This corresponds to a multiparty version of FHE in which the parties can obtain the output of the function evaluation immediately after receiving the encryption of the inputs. Specifically, the primitives allows to perform homomorphic operations on ciphertexts generated under independent public keys. Then, using their secret keys and without the need for any interaction, the parties can retrieve their output.

Due to the one-round nature of the primitive and the issue with residual function attacks, spooky encryption only supports the evaluation of a restricted class of randomised functions: in order to preserve semantic security, it is fundamental that the outputs of any subset of parties leak no information about the inputs of the remaining players.

Currently, we know how to build spooky encryption for functions outputting random additively secret-shared values depending on the inputs. This construction is based on LWE with subexponential modulus-to-noise ratio and relies on an unstructured CRS [DHRW16]. Furthermore, in the two party setting, we know how to build a more generic form of spooky encryption based on (among other primitives) subexponentially secure iO [DHRW16].

One could imagine to build distributed samplers from spooky encryption by evaluating the inputless function described by $\mathcal{D}(1^\lambda)$. This however does not necessarily give a distributed sampler: spooky encryption only guarantees privacy of the inputs, but nothing prevents it from revealing information about the randomness input in $\mathcal{D}(1^\lambda)$.

Pseudorandom correlation generators and pseudorandom correlation functions. Pseudorandom correlation generators (PCGs) [BCG+19b, BCG+19a, BCG+20b] and pseudorandom correlation functions (PCFs) [BCG+20a] allow n parties to securely generate large amounts of correlated randomness by locally expanding small seeds. In the case of PCGs the expansion of the seed occurs all at once, producing a polynomial amount of correlated material. In the case of PCFs, the expansion takes place in a one by one fashion, similarly to PRFs. As a consequence, PCFs may have no polynomial bound on the amount of produced material.

PCGs and PCFs can be easily transformed into secure, sublinear communication MPC protocols for the generation of large amounts of correlated material: it is sufficient to design MPC protocols that generate and distribute the correlated seeds with linear communication in their size. Since the size of the seeds is sublinear in the amount of material they produce, we obtain the protocol we desire.

Setting up the seeds in a single round is however tricky. When this is possible we obtain *a public-key PCFs*. The name is due to the fact that the messages exchanged in the only round of interaction act as a public key, whereas the randomness used for their generation acts as the private counterpart. At the time our research began, public-key PCFs were known only for OT[17] and VOLE[18] correlation based on Paillier [OSY21], and for more general additively shared correlation based on spooky encryption [DHRW16][19]. In this thesis, we wonder whether public-key PCFs can be build for more generic forms of correlation.

Coin tossing and coin tossing extension. A coin tossing protocol [Blu82] allows a set of parties to agree on a uniformly random string of bits. Due to the large use of public randomness in cryptography, the primitive has captured the attentions of researchers for many years.

The question becomes particularly interesting in the malicious setting (semi-honest coin tossing is trivial). While the primitive is well understood in presence of an honest majority, the dishonest majority setting is burdened by Cleve's impossibility result [Cle86]: for every R-round coin tossing protocol, there exists a PPT adversary that biases the output by at least $O(1/R)$. For years, the community has therefore tried to find new ways to get around the impossibility, either by consider coin-tossing with abort [Blu82, Lin03] or by considering new models such as time-based cryptography [RSW00, BBBF18].

In this thesis, we study coin tossing extension (CTE), which tries to circumvent Cleve's impossibility by relying on an auxiliary resource providing all participants with a random, unbiased-but-short string of bits. This primitive was introduced by Bellare et al. [BGR96] and later studied by Hofheinz et al. [HMU06]. In this last work, the authors showed a series of lower bounds and upper bounds: they proved the impossibility of CTE whenever the auxiliary resource provides $O(\log \lambda)$ bits of randomness, the impossibility of perfectly

[17] In OT correlation a party obtains a random "shift" $\Delta \in \{0,1\}^k$ and random elements $K_1, \dots, K_L \in \{0,1\}^k$. The other party obtains random bits $b_1, \dots, b_L \in \{0,1\}$ and strings $M_1, \dots, M_L$ where $M_i = K_i \oplus b_i \cdot \Delta$ for every $i \in [L]$.

[18] In VOLE correlation a party obtains a random "shift" α in a ring R and random elements $a_1, \dots, a_L \in R$. The other party obtains random elements $x_1, \dots, x_L \in R$ and values $b_1, \dots, b_L$ where $b_i = a_i \oplus x_i \cdot \alpha$ for every $i \in [L]$.

[19] Additively shared correlation consists of any form of correlation in which the parties obtain random additive secret-sharing of correlated values.

secure CTE in the standalone model and of statistical CTE with UC security. On the positive side, Hofheinz et al. present a statistically secure, 1-round CTE protocol with $O(\log \lambda)$ stretch in the standalone model [HMU06].

In this thesis, we try to study coin tossing extension protocols with $\omega(\log \lambda)$ stretch and $O(1)$ rounds. Moreover, we study whether the auxiliary functionality can be used to produce secure unbiased sample from arbitrary distributions.

Indistinguishability obfuscation. Indistinguishability obfuscation [BGI+01, GGH+13, JLS21] is one of the strongest tools used in cryptography today. The primitive specifies how to "scramble" any given circuit into a new circuit (often called program) computing exactly the same function. The resulting object is however so muddled that it is infeasible to recognise the circuit it was originated from.

Distributed samplers are strongly related to obfuscation: all the constructions we present will make heavy use of this primitive. And this is no coincidence: we show that by using semi-honest distributed samplers along with LWE, we are able to build indistinguishability obfuscation. This (never published) result is obtained following the blueprint of Wee and Wichs [WW21]. We sketch our argument in Section 1.2.2.

1.2 Technical Overview

We now present a technical overview of the results presented in this thesis.

1.2.1 An Informal Discussion of Preliminaries

In this section, we present an informal description of cryptographic notions and primitives we used in our work. We recall that λ denotes the security parameter[20]. All protocols, primitives and random variables will be parametrised by λ. We say that a function $\epsilon(\lambda)$ is negligible if $\epsilon(\lambda) = \lambda^{-\omega(1)}$. For any $\ell \in \mathbb{N}$, we use $[\ell]$ to denote the set $\{1, 2, \dots, \ell\}$. All logarithms will be in base 2. We use 1^λ to denote the security parameter in unary notation (i.e. a string of λ bits all set to 1). We use $x \leftarrow y$ or $x \leftarrow \mathcal{A}(y)$ to assign y or the output of the *deterministic* algorithm $\mathcal{A}(y)$ to the random variable x. We use instead $x \overset{\$}{\leftarrow} X$ or $x \overset{\$}{\leftarrow} \mathcal{A}(y)$ to assign a uniformly random sample from a set X or the output of the *randomised* algorithm $\mathcal{A}(y)$.

Security games and advantage. The security of cryptographic primitives is often formalised by relying on security games (parametrised by the security parameter λ) between an "honest" entity called *the challenger* and an "evil" entity called *the adversary*. Typically, the goal of the game is for the adversary to find an object hidden in a large domain or to perform an almost impossibile task (search game) or to guess a bit chosen at random by the adversary (decision game). We measure the success of an adversary using the notion of *advantage*. In the case of search games, the advantage is defined as the probability that the adversary finds one of the hidden objects or manages to perform the infeasible task. In the case of decision games, the advantage is defined as the distance between the probability of the adversary guessing the bit sampled by the challenger and 1/2. Typically, we say that a game is hard if the advantage is negligible in λ for every adversary in the considered class.

Indistinguishability and hybrid arguments. We say that two random variables X_0 and X_1 are indistinguishable, if it is impossible to distinguish between them with non-negligible advantage. Formally, we consider the decision game in which the challenger samples a random bit $b \overset{\$}{\leftarrow} \{0, 1\}$ and provides the adversary with a sample from X_b. The two variables are statistically indistinguishable if all adversaries (also computationally unbounded ones) guess b with negligible advantage. We say that the variables are *computationally indistinguishable* if the advantage is negligible for all probabilistic, polynomial time (PPT) adversaries.

[20] A security parameter describes the computational power of the protocol participants and the adversary.

Indistinguishability is often at the base of hybrid arguments: suppose that we deal with a construction that makes use of a sample from X_0. We can consider the "hybrid" construction in which we substitute the sample from X_0 with a sample from X_1. Since X_0 and X_1 are indistinguishable, the hybrid construction will behave equivalently to the original one. By repeating this procedure multiple times in a cascade of hybrids, we can slowly switch the pieces of the construction until we reach a version for which it is easier to analyse the security properties. The distinguishability advantage between the starting point and the final point will be roughly the sum of all distinguishing advantages between pairs of subsequent hybrids, so as long as their number is polynomial, the final version of the construction will be indistinguishable from the original one (this is because, for any constant c, $\lambda^c \cdot \lambda^{-\omega(1)} = \lambda^{-\omega(1)}$). If instead the number of hybrids is superpolynomial, we need to require the average advantage between subsequent hybrids to be smaller than negligible: if there are $L(\lambda)$ hybrids, we need the average advantage to be $L(\lambda)^{-1} \cdot \lambda^{-\omega(1)}$. This is why, in this thesis, we often rely on subexponentially secure primitives, i.e., there exists a constant $\epsilon > 0$ for which the advantage of the primitive against $O(2^{\lambda^\epsilon})$-time adversaries is at most $2^{-\lambda^\epsilon}$.

MPC protocols and security with black-box simulation. In all multiparty computation protocols we describe in this chapter, we denote the number of participants by n. We denote the i-th party by $\mathcal{P}_i$.

The security of MPC protocols is usually defined using a real world/ideal world paradigm: in the real world, we consider the interaction of the protocol with an adversary. The adversary has the ability to corrupt a subset of participants and learn all their secrets. Furthermore, if the adversary is active (in contrast with semi-honest adversaries), the adversary can also control the operations of the corrupted parties, making them misbehave according to its will. The adversary always gets to choose the inputs of the honest parties (but not their actions), moreover it gets to see their output. We denote the set of corrupted parties by C and the set of honest parties by H.

In the ideal world, the adversary interacts with a functionality and a simulator. The functionality models the ideal execution of the protocol, controlling the outputs of the honest parties and what information gets leaked to the adversary. For instance, if the protocol computes a function f, the functionality could gather the inputs of all parties $x_1, \dots, x_n$ and output $f(x_1, \dots, x_n)$ to all participants without revealing any additional information. The simulator mediates the communications between the adversary and the functionality, trying to simulate the real execution of the protocol. Moreover, in case the simulator does not like how the interaction is developing, it can "rewind" the adversary to a previous state and replay the part of the protocol until it is satisfied with the result.

Commonly, security with black-box simulation is defined by asking for the existence of a polynomial-time simulator for which the information output by the adversary after interacting with the ideal world is indistinguishable from the one it outputs after interacting with the real world. In stronger security models, such as UC security, we require the simulation to be straightline: no rewinding is allowed!

With this indistinguishability-based definition, we are essentially saying that any attack the adversary succeeds in performing against the protocol, can be converted into an attack against the ideal functionality, a much simpler object whose security properties are significantly easier to analyse. If any vulnerability were to be found in the protocol, it would be an intrinsic vulnerability: it would not be that the protocol is not secure enough, the issue would be that what we are trying to compute is insecure in the first place!

In this thesis, we mostly deal with protocols in which the adversary is computationally bounded: it must run in polynomial time. We talk about computational security when the ideal world and the real world are only computationally indistinguishable. We talk about statistical security if instead the outputs of the adversary in the two worlds are statistically indistinguishable. We talk about superpolynomial simulation if the simulator is allowed to run in time $O(T)$ where T is a superpolynomial function of λ.

Computational assumptions. In this thesis, we sometimes rely on well-known cryptographic assumptions such as *learning with errors* (LWE) [Reg05], *decisional Diffie-Hellman* (DDH) [DH76] and *quadratic residuosity* (QR) and *decisional composite residuosity* (DCR) over the Paillier group [Pai99].

- *Learning with errors.* Let q, K, M, B be positive integers where M can be greater than K. Let χ a distribution over $\mathbb{Z}^M$ where its samples have norm at most B. The LWE assumption takes place over a

K-dimensional lattice over $\mathbb{Z}_q^M$ described by a matrix $A \xleftarrow{\$} \mathbb{Z}_q^{M\times K}$. In particular, the assumption states that, for $s \xleftarrow{\$} \mathbb{Z}_q^K$ and $e \xleftarrow{\$} \chi$, the value $A \cdot s + e$ looks indistinguishable from the uniform distribution over $\mathbb{Z}_q^M$, even when A is public. The low-norm distribution χ is usually instantiated using a discrete Gaussian. We refer to the quotient q/B as the modulus-to-noise ratio.

Given a modulus $p \leq q$ and an element x in $\mathbb{Z}_q$, we sometimes "round down" x to the modulus p. This means computing the value $c \in \mathbb{Z}_p$ that minimises $|x - c \cdot q/p|$.

- *Decisional Diffie-Hellman.* Suppose that G is a large, abelian, multiplicative group generated by an element g of order p (usually p is a prime). We say that DDH holds over G if, for random $a, b, c \xleftarrow{\$} [p]$, the tuples $(g, g^a, g^b, g^{a\cdot b})$ and (g, g^a, g^b, g^c) are computationally indistinguishable.
- *Quadratic residuosity and decisional composite residuosity over the Paillier group.* Let N be the product of two random, unknown, large primes p and q. The Paillier group is defined as $\mathbb{Z}_{N^2}^*$. The QR assumption states that, for a random $x \xleftarrow{\$} \mathbb{Z}_{N^2}^*$, it is hard to distinguish between x and x^2. The DCR assumption, in a similar way, states that it is hard to distinguish between x and x^N.

  It is easy to see that the order of the Paillier group is $N \cdot \phi(N)$ where $\phi(N) = (p-1)\cdot(q-1)$ denotes Euler's totient function. In other words, the order of the group is always a multiple of $2N$. From this we understand that QR and DCR can hold only against computationally bounded adversaries: the squares of $\mathbb{Z}_{N^2}^*$ form a proper subgroup and so do the N-th powers. Finally, we observe that if p and q are random safe primes, i.e., $p = 2p' + 1$ and $q = 2q' + 1$ where p' and q' are themselves primes, the Paillier group is isomorphic to the direct product of *additive* groups $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_N \times \mathbb{Z}_{p'\cdot q'}$. The subgroup of $2N$-th powers is therefore isomorphic to $\mathbb{Z}_{p'\cdot q'}$, which is cyclic. Under both QR and DCR, a random $2N$-th power is computationally indistinguishable from $x \xleftarrow{\$} \mathbb{Z}_{N^2}^*$.

One-way functions. A one-way function (OWF) consists of a function f that is hard to invert on random instances in its image: given a random y, no PPT (probabilistic, polynomial time) adversary is able to retrieve any x such that $f(x) = y$ (except with negligible probability).

Puncturable PRFs. A puncturable PRF [KPTZ13, BW13, BGI14b] consists of a PRF where we can erase all information concerning the evaluation at a particular point from the key. The key obtained is this way is called a *punctured key* and it allows the correct evaluation of the PRF at all positions except the punctured one. Moreover, even given the punctured key, the output of the original PRF at the punctured position looks random.

Hash functions. An hash function is a keyed function H_{hk} that maps large inputs into short digests (or hashes). The primitive guarantees that, if hk is sampled at random, it is infeasible that a PPT adversary manages to find a *collision*, i.e., different strings $x_1 \neq x_2$ such that $H_{\mathsf{hk}}(x_1) = H_{\mathsf{hk}}(x_2)$.

Non-interactive commitments. Non-interactive commitments allow a protocol participant to commit to a value without revealing it to anybody else (hiding property of the commitment). This is done by broadcasting an object called *the commitment*. At a later point in time, the party can reveal the chosen value along with a proof (often called *opening*). This proof can be validated along with the commitment by any bystander. The procedure always succeed, if the committer behaved honestly. Furthermore, if the committer misbehaves and broadcasts a value different from the one it chose earlier, the validation always fails, even if the commitment was maliciously generated (biding property of the commitment).

Public-key encryption. Public-key encryption (PKE) [RSA78, DH76] consists of an encryption scheme where the key (i.e. the information that allows encryption and decryption) is split in two parts: a public key pk and a secret key sk (also called private key). The encryption procedure requires only the knowledge of the public key pk. Decryption instead requires using sk. This primitive guarantees the privacy of the encrypted communication even if the public key becomes public.

Fully homomorphic encryption. Fully homomorphic encryption (FHE) [Gen09, GSW13] consists of a public-key encryption scheme in which it is possible to apply homomorphic operations on the ciphertexts: if c is an encryption of x, by homomorphically applying a function f on the ciphertext, we obtain an encryption of $f(x)$, without having to know x or the secret key.

Multi-key FHE. Multi-key FHE [LTV12, MW16, AJJM20] consists of a form of FHE that allows homomorphic operations on ciphertexts encrypted under different public keys: given ciphertexts $c_1, \ldots, c_n$ under public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, where c_i hides a value x_i, we are able to obtain a "joint ciphertext" encrypting $f(x_1, \ldots, x_n)$. The decryption of the joint ciphertext is performed in two phases: first each participant performs a partial decryption using their secret key, then the partial plaintexts are pooled together to reconstruct the output.

Identity-based encryption. Identity-based encryption (IBE) [Sha84, ABB10] is a form of PKE where the messages are encrypted under the identity of the recipient. Specifically, for every identity id, there exists a secret key $\mathsf{sk}_{\mathsf{id}}$ that allows the decryption of any ciphertext c produced under id. The knowledge of secret keys for any other identities does not provide any help in decrypting c.

Non-interactive zero-knowledge proof. A non-interactive zero-knowledge proof (NIZK) is an object that allows proving any statement to an external entity without revealing any additional information. Formally, the primitive is tailored to an NP relation $\mathcal{R}$[21] and always relies on a CRS. A prover knowing a pair $(x, w) \in \mathcal{R}$ can generate a proof π for the statement x. Any other entity (called *verifier*) can check the validity of the proof π. If the procedure succeeds, the verifier can be sure of the existence of a witness w for x, i.e., $(x, w) \in \mathcal{R}$ (soundness of the NIZK). The primitive also guarantees *zero-knowledge*: the proof π leaks no information about the witness w, nor anything else that cannot be computed directly from x. This property is formalised by relying on a simulator: we ask that no PPT adversary can distinguish between proofs that are generated following the protocol and simulated proofs that are produced using a trapdoor hidden in the CRS but no witness.

We say that a NIZK is extractable if it is possible to use the trapdoor hidden in the CRS to extract a witness from any valid proof produced by the adversary. In other words, given a valid proof π for a statement x, we are able to extract a witness w such that $(x, w) \in \mathcal{R}$. We say that a NIZK is simulation-extractable if this property holds even if the adversary is helped in its task by an oracle providing simulated proofs.

Non-interactive witness indistinguishability. A non-interactive witness-indistinguishable proof (NIWI) can be viewed as a NIZK satisfying soundness and a weaker form of zero-knowledge: we require that for any statement x having multiple witnesses, it is infeasible to tell which witness was used for the generation of the proof. Notice that if the witness w is unique, it is totally fine for a NIWI to just leak w. Unlike NIZKs, it is possible to build NIWIs that do not use any CRS [BOV03, GOS06a, GOS06b, BP15].

Shannon entropy, strong chain rule and mutual information. Entropy measures the amount of information contained in a random variable. The notion is tightly connected to the unpredictability of the value the variable assumes. There exist many different definitions of entropy. The most famous is perhaps Shannon's entropy [Sha48]:

$$\mathsf{H}(X) := -\sum_x \Pr[X = x] \cdot \log(\Pr[X = x]).$$

It is possible to prove that the Shannon entropy $\mathsf{H}(X)$ expresses the size of the optimal representation for X. In particular, if X is a distribution over a set of cardinality t, $\mathsf{H}(X) \leq \log t$.

[21] An NP relation $\mathcal{R}$ consists of a set of pairs (x, w), for which there exists a polynomial-time algorithm that correctly classifies whether $(x, w) \in \mathcal{R}$ or not. We call x *the statement* and w *the witness*.

Given random variables X and Y, the conditional Shannon entropy $\mathsf{H}(X|Y)$ measures the amount of information contained in X but not in Y. The quantity is defined as

$$\mathsf{H}(X|Y) := -\sum_y \sum_x \Pr[X = x, Y = y] \cdot \log(\Pr[X = x|Y = y]).$$

Shannon's entropy and conditional Shannon entropy are linked by an important property called *the strong chain rule*: for any variables X and Y, $\mathsf{H}(X, Y) = \mathsf{H}(X|Y) + \mathsf{H}(Y)$[22].

This fundamental equality led the way to the introduction of new information measures. For instance, the mutual information $\mathsf{I}(X; Y) := \mathsf{H}(X) - \mathsf{H}(X|Y)$ measures the amount of information contained in both X and Y. We can also define the conditional mutual information $\mathsf{I}(X; Y|W) := \mathsf{H}(X|W) - \mathsf{H}(X|Y, W)$: it measures the amount of information contained in both X and Y but not in W. Even more generally, we can define $\mathsf{I}(X; Y; Z|W) := \mathsf{I}(X; Y|W) - \mathsf{I}(X; Y|Z, W)$. This can be viewed as the information simultaneously contained in all of X, Y and Z but not in W.

Min entropy. Min entropy provides another measure of unpredictability. Given a distribution X, we denote the min-entropy of X by $\mathsf{H}_\infty(X) := -\max_x \log(\Pr[X = x])$. In other words, if $\mathsf{H}_\infty(X) \geq t$, it means that $\max_x \Pr[X = x] \leq 2^{-t}$. It is possible to prove that, for any variable X, we have $\mathsf{H}_\infty(X) \leq \mathsf{H}(X)$.

Yao entropy. Yao entropy $\mathsf{H}_{\mathsf{Yao}}(X)$ [Yao82] measures how much a random variable can be compressed in polynomial time without losing information. The notion is formalised using a pair of deterministic polynomial-time algorithms (c, d). The first algorithm is called the compressor, takes as input a sample from the random variable X and outputs a compressed representation s. The second algorithm is called the decompressor, it takes as input s and its goal is to reconstruct the original sample from X.

As we did for Shannon's entropy, we can generalise the notion: the conditional Yao entropy $\mathsf{H}_{\mathsf{Yao}}(X|Y)$ is defined similarly to $\mathsf{H}_{\mathsf{Yao}}(X)$, however, this time the compressor and the decompressor are aided in their task by a sample from Y (which may be correlated to X).

Lossy trapdoor functions. A lossy trapdoor function [PW08] consists of a function with two indistinguishable modes of operation: injective mode and lossy mode. When the function is in injective mode, each element in its domain is mapped into a different value in the image. Moreover, using a trapdoor, it is possible to efficiently invert the function. Notice that in an injective mode trapdoor function, the image has the same size as the domain. When the trapdoor function is in lossy mode, this is no longer true: the image is contained in a significantly smaller set (superpolynomial in size).

Extremely lossy functions. An extremely lossy function (ELF) [Zha16] can be viewed as a lossy trapdoor function where the lossy mode is "extreme": the image contains a polynomial amount of different elements. The exact number depends on a polynomial $q(\lambda)$ parametrising the lossy mode.

Actually, in an ELF, the lossy mode and the injective mode are unavoidably distinguishable. However, security guarantees that the distinguishing advantage can be made an arbitrarily small inverse-polynomial function: for any polynomial $p(\lambda)$ and inverse-polynomial quantity $\delta(\lambda)$, there exists a polynomial $q(\lambda)$ such that no adversary running in time $p(\lambda)$ can distinguish between the injective mode and the lossy mode with advantage greater than $\delta(\lambda)$, when the lossy mode is parametrised by $q(\lambda)$.

1.2.2 Semi-Honest Distributed Samplers

We start by presenting an overview of the semi-honest (actually non-rushing semi-malicious) distributed sampler of [ASY22a] (see Section 2.4). Throughout the section, we will denote the number of parties by n and the distribution from which we want to sample by $\mathcal{D}(1^\lambda)$.

[22] $\mathsf{H}(X, Y)$ is the entropy of the joint distribution (X, Y).

Compressing two rounds into one. Our starting point is a 2-round protocol $\Pi_{\mathcal{D}}$ where each party $\mathcal{P}_i$ inputs a random string r_i and the output is obtained by feeding the XOR of all inputs in $\mathcal{D}(1^\lambda)$ in place of the randomness. We require the protocol to be secure against rushing, semi-malicious adversaries. In other words, the construction is secure even if the adversary maliciously chooses the randomness of the corrupted players (as in the semi-honest setting, however, the corrupted players cannot deviate from the protocol). For example, we can rely on the multi-key FHE construction of [AJJM20] or, as we did in [ASY22a], on a weaker primitive called multiparty homomorphic encryption with private evaluation [AJJM20].

Our goal will be to compress the two rounds of the protocol into a single round, while at the same time ensuring security against *residual functions attacks* (see the discussion on non-interactive MPC in Section 1.1.3). Given that we want to implement an inputless functionality (see Figure 1.1), there is still hope to achieve this: we just need to ensure that in every imaginary execution run in the head of the adversary, the randomness fed into $\mathcal{D}(1^\lambda)$ to produce the output looks independent of the randomness used in all previous executions. In other words, if the adversary changes even just one of the messages exchanged in the only round of interaction, the output that the protocol produces should look independent of all previous ones.

We compress the two rounds by relying on indistinguishability obfuscation (iO) [BGI+01, GGH+13, JLS21]. The message sent by each party $\mathcal{P}_i$ in the distributed sampler protocol will consist of two obfuscated programs hiding the same puncturable PRF key K_i: the first one is called the *encryption program* (or, using the terminology of [ASY22a], the key generation program), the second one is called the *decryption program* (or, using the terminology of [ASY22a], the evaluation program). The encryption programs will take care of generating the messages in the first round of $\Pi_{\mathcal{D}}$, the decryption programs will instead generate the messages in the second round. In this way, we obtain a *virtual* execution of $\Pi_{\mathcal{D}}$ using a single round.

The encryption program. More formally, the encryption program uses the PRF key K_i to generate the input r_i of party $\mathcal{P}_i$. Then, using r_i along with other randomness extracted from K_i, the program generates $\mathcal{P}_i$'s message for the first round of $\Pi_{\mathcal{D}}$ and outputs it. To ensure that different choices of the messages of the other players lead to independently looking values for r_i (therefore preventing residual function attacks), we would like the encryption program to generate all randomness it necessitates by inputting the encryption programs of the other players in its puncturable PRF (notice that the encryption programs of the other parties fix $Pi_{\mathcal{D}}$'s inputs of the other players). This, however, cannot occur due to a mere matter of sizes: how can we fit $n-1$ encryption programs in a program of exactly the same size? Instead of relying on iO for Turing machines [KLW15, GS18a], we feed the encryption program of party $\mathcal{P}_i$ with a hash of the encryption programs of the other players (we let $\mathcal{P}_i$ choose the hash key). The hash will then be input in the puncturable PRF for the generation of r_i and the rest of the necessary randomness. In order to make the security proof go through, we need particular hash functions that are compatible with iO, for instance we can rely on *somewhere statistically binding* hash functions (SSB) [HW15]. More in general, it would be sufficient to have a hash function that can be programmed to have no collisions at the places hidden in the key.

The decryption program. The decryption program of party $\mathcal{P}_i$ is fed with the encryption programs of all other parties. By evaluating them, the program is able to retrieve all messages exchanged in the fictitious first round of $\Pi_{\mathcal{D}}$. Moreover, by leveraging the knowledge of K_i, the program is able to retrieve any secret information computed by party $\mathcal{P}_i$ during the generation of its first-round message. Such information is usually necessary to compute the message in the second round of $\Pi_{\mathcal{D}}$. The decryption program will therefore proceed by producing $\mathcal{P}_i$'s message in the second round of $\Pi_{\mathcal{D}}$, outputting it (the operation might require additional randomness which can be extracted from K_i). To conclude, encryption programs and decryption programs allow all participants to obtain a consistent transcript of the protocol $\Pi_{\mathcal{D}}$ using a single round of interaction. If the output of $\Pi_{\mathcal{D}}$ can be computed from just the transcript (i.e. without having to leverage any internal information known to the participants), we obtain a distributed sampler for $\mathcal{D}(1^\lambda)$. If instead this is not the case, we obtain a distributed sampler where the output can be retrieved only by the protocol participants (party $\mathcal{P}_i$ can retrieve the output by leveraging the knowledge of K_i).

Why is the construction secure against non-rushing semi-malicious adversaries? The answer is rather simple: the distributed sampler we sketched above satisfies a form of *programmability*. Given the randomness used by the other parties and any sample $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$, we are able to generate the programs of party $\mathcal{P}_i$ so that, when used in conjunction with the messages of the other parties, the final output is R. We achieve this by relying on the security of iO and puncturable PRFs as done in [SW14]. In particular, we need to puncture the key K_i in the hash of the encryption programs of the other players. At that point, we just need to rely on the security of $\Pi_{\mathcal{D}}$ against rushing, semi-malicious adversaries.

On the relation between obfuscation and semi-honest distributed samplers. It is possible to prove that (subexponentially secure) distributed samplers for any distributions (in conjunction with the subexponential hardness of LWE with subexponential modulus-to-noise ratio) imply the existence of (subexponentially secure) indistinguishability obfuscation. This result (never published in a paper) is obtained following the blueprint of iO from oblivious LWE sampling proposed by Wee and Wichs [WW21]. In their work, the authors showed how to build iO from (subexponentially hard) LWE, assuming the ability to compress LWE samples without leaking their secrets. In other words, we would like to compress many samples of the form $b_i \leftarrow A \cdot s + e_i$ where A is a public random matrix, s_i is a secret uniformly random vector and e_i is a random low-norm vector (usually sampled from a discrete Gaussian distribution). We would like to maintain the secrecy of all s_i, while allowing a simulator holding a trapdoor to efficiently recover them.

We achieve this by relying on public-key encryption (PKE) and semi-honest, two-party distributed samplers. The compression consists of a pair (pk, U_1), where $(\mathsf{pk}, \mathsf{sk})$ is a PKE key and U_1 is a distributed sampler message for the distribution $\mathcal{D}(1^\lambda)$ outputting: (1) a random matrix A;(2) a batch of L LWE samples with respect to A; (3) an encryption under pk of a trapdoor for A. The latter allows to retrieve all the LWE secrets $(s_i)_{i\in[L]}$.

We can generate a large amount of secure LWE samples by generating multiple distributed sampler messages U_j (using deterministically produced randomness for $j = 2, \dots, M$) and computing the output of the distributed sampler protocol in which one party sent U_1 while the other sent U_j. As long as sk is kept secret, the adversary cannot retrieve any information about the secrets hidden in the LWE samples. On the other hand, by leveraging the knowledge of sk, a simulator can easily retrieve them.

This trick allows to compress M batches of L LWE samples into a string of $\mathsf{poly}(\lambda, L)$ length. Under the security of the distributed sampler, each of these batches looks as if it was generated by $\mathcal{D}(1^\lambda)$. Notice that each batch is generated using a different LWE matrix. Although Wee and Wichs showed how to build iO from an oblivious LWE sampler that produces all its samples using the same matrix A, it is not hard to adapt their construction to our situation.

1.2.3 Actively Secure Distributed Samplers in the Random Oracle Model

We now explain how it is possible to upgrade the construction we just described to active security in the programmable random oracle model. The operation is performed by relying on a compiler we called *anti-rusher*. The latter allows the conversion of any one-round, inputless protocol with security against non-rushing semi-malicious adversaries into a new one-round, actively secure protocol implementing a variation of the same functionality in the programmable random oracle model.

Formally, if the original protocol implemented a functionality $\mathcal{F}$, the compiled protocol implements the functionality $\mathcal{F}^{\mathsf{Active}}$ which allows the adversary to try multiple executions of $\mathcal{F}$ and then choose the one it likes the most among the proposed ones. Notice that the adversary has still a limited influence on the protocol: it can choose only among a polynomial number of different executions. Observe also that, since we are dealing with one-round, inputless protocols, in the non-rushing, semi-malicious model, the adversary was still allowed to replay multiple executions of the protocol in its head (after seeing the real execution). In other words, the leakage of $\mathcal{F}^{\mathsf{Active}}$ is the same as in $\mathcal{F}$, what changes is the influence of the adversary on the output! By compiling the sampling functionality $\mathcal{F}_{\mathcal{D}}$ (see Figure 1.1), we obtain the functionality $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ in Figure 1.2.

So, how do we build an anti-rusher compiler? Essentially, we rely on the techniques introduced by Hofheinz et al. in [HJK+16], however, we apply them in a different context: Hofheinz et al. used them to build universal samplers that guarantee security of all the samples they generate, even if they are produced using multiple distributions adaptively chosen by the adversary. We apply them instead to beat rushing behaviour in MPC protocols.

In order to compile a non-rushing semi-maliciously secure, one-round, inputless protocol Π, we let each party $\mathcal{P}_i$ broadcast an obfuscated program hiding a puncturable PRF key K_i. The obfuscated program will take care of generating the message of party $\mathcal{P}_i$ in Π using the randomness extracted from K_i. As for the distributed samplers we saw in the previous section, we would like that different choices of the messages of the corrupted players lead to independent-looking randomness used by the honest parties in the recreated execution of Π. In this way, we prevent the adversary from adaptively choosing the messages of the corrupted players in Π after seeing the messages of the honest parties. Once again, we ensure this property by generating the randomness used in $\mathcal{P}_i$'s program by feeding a hash of the exchanged messages into the puncturable PRF. The only difference is that now we model the hash function using a *programmable random oracle.*

How to simulate the protocol? Although the construction we described so far intuitively prevents rushing in Π, it is unclear how to simulate the protocol while ensuring consistency with the outputs of the functionality. In order to solve this problem, we modify the construction: first of all, we ensure the wellformedness of the obfuscated programs by relying on extractable NIZKs [GO07] (if the CRS of the NIZK is unstructured, we can generate it using the random oracle). Then, we exploit the random oracle to program the output of the obfuscated circuits broadcast by the honest parties.

We equip the programs with a trapdoor: $\mathcal{P}_i$'s program will hold a key K'_i for an authenticated encryption scheme. All ciphertexts produced under this key will be indistinguishable from random oracle responses, however, with overwhelming probability, truly-random oracle responses will not correspond to valid encryptions. Upon receiving a random oracle response, the obfuscated program of party $\mathcal{P}_i$ will try to decrypt its input using K'_i. When the operation fails (therefore with overwhelming probability, if the oracle response was truly random), the program behaves as we described above: it generates $\mathcal{P}_i$'s message in Π using the randomness extracted from K_i and outputs it. If the decryption succeeds, on the other hand, the program directly outputs the plaintext.

Proving security. Our proof strategy will be the following: whenever the adversary queries the oracle with obfuscated programs, the simulator will extract the PRF keys hidden in them from the associated NIZKs. From the keys, the simulator will be able to recover the randomness used by the programs of the corrupted players for the generation of their message in Π. At that point, the simulator can rely on the non-rushing, semi-honest security of Π: with the help of $\mathcal{F}^{\mathsf{Active}}$, it can simulate the messages of the honest players in Π. Finally, it crafts an oracle response so that the obfuscated programs of the honest parties output these simulated messages. In other words, for each oracle query made by the adversary, the simulator will start a new execution of $\mathcal{F}$ inside $\mathcal{F}^{\mathsf{Active}}$. Once the adversary selects the messages for the corrupted players, the simulator will instruct $\mathcal{F}^{\mathsf{Active}}$ to finalise the corresponding execution of $\mathcal{F}$. More details on anti-rushers compilers can be found in [ASY22a] (see Section 2.5).

1.2.4 Impossibility of Actively Secure Distributed Samplers in Absence of Random Oracles

The actively secure distributed samplers we obtain by compiling our non-rushing semi-malicious construction with an anti-rusher compiler is great, however, it also has a great disadvantage: it relies on a programmable random oracle. In [AOS23] (see Chapter 3), we tried to build actively secure distributed samplers without relying on idealised models: we ended up proving that, at least for UC security, this task is impossible.

Our main result proves that, if the min-entropy[23] of $\mathcal{D}$ is not small (i.e. $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log\lambda)$), in any one-round protocol implementing the functionality $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ against rushing semi-malicious adversaries, the Shannon entropy of the output R conditioned on the CRS σ is small, namely $\mathsf{H}(R|\sigma) = O(\log\lambda)$. Below, we explain the intuition (for more information, check Section 3.2.1 and Section 3.4).

An overview of the ideal world. We consider the ideal world execution of a UC-secure distributed sampler. We observe that, since we are dealing with a rushing adversary, the execution starts with the simulation of the CRS σ and the messages of the honest parties U_H. Only at that point, the simulator will discover the messages U_C chosen by the corrupted players. This fixes the output of the protocol R, however, the task of the simulator is not finished: it needs to instruct the functionality $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ to output R to all honest players.

Now, let Q be the set of samples provided by the functionality $\mathcal{F}_{\mathcal{D}}^{\mathsf{Active}}$ to the simulator before σ and U_H are delivered to the adversary. This set will have polynomial size. We have two cases: either R belongs to Q, or it does not! In the first situation, the simulator has an easy life: it can just tell the functionality to output the previously sampled element R. If instead R does not belong to Q, the only option for the simulator is to query the functionality for samples until it hits R (since we are in the UC model, we cannot rewind). The probability of this event, however, is negligible as $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log\lambda)$.

We conclude that in the ideal world execution of a UC-secure distributed sampler, the CRS and the messages of the honest parties restrict the output in a set of polynomial size. In particular, this implies that, in the ideal world, $\mathsf{H}(R|U_H, \sigma) = O(\log\lambda)$.

An overview of the real world. Consider the rushing adversary that behaves honestly: it generates the messages of the corrupted parties following the protocol and using true randomness, but reveals U_C only after seeing σ and U_H. We wonder whether $\mathsf{H}(R|U_H, \sigma) = O(\log\lambda)$ even in the real world. Notice that the Shannon entropy is usually not preserved under computationally indistinguishability[24]. Therefore, it may be that, in the real world, $\mathsf{H}(R|U_H, \sigma) = \omega(\log\lambda)$.

We prove that if this is the case, there exists an efficient distinguisher between the real world and the ideal world. The algorithm is rather simple: the distinguisher reruns the protocol in its head many times reusing the same CRS and honest messages U_H as in the actual execution, but regenerating every time the messages of the corrupted players. It then counts the number of different outputs it obtains in this way. We have seen that, in the ideal world, the CRS and the honest messages restrict the output in a set of polynomial size. Therefore, the number of different samples stops growing after a while. On the other hand, in the real world, if $\mathsf{H}(R|U_H, \sigma) = \omega(\log\lambda)$, we prove that the number of different samples keeps growing.

A look at the entropy diagram. We show that, in the real world, $\mathsf{H}(R|\sigma) = O(\log\lambda)$. We start by summarising what we know about the entropy of the random variables involved in the protocol:

- Since the output R is uniquely determined by σ, U_H and U_C, we conclude that $\mathsf{H}(R|U_H, U_C, \sigma) = 0$.
- Since we are considering a honest-but-rushing adversary, U_H and U_C are independent of each other, conditioned on the knowledge of σ. Using the language of mutual information, $\mathsf{I}(U_H; U_C|\sigma) = 0$.
- Due to what we argued in the previous paragraphs, $\mathsf{H}(R|U_H, \sigma) = O(\log\lambda)$.
- Since the adversary behaves as the honest parties, by symmetry, $\mathsf{H}(R|U_C, \sigma) = O(\log\lambda)$.

[23]Notice that asking that $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log\lambda)$ is a very weak requirement: we are essentially saying that any guess for the output of $\mathcal{D}$ is wrong with overwhelming probability. All CRSs of MPC protocols satisfy this property.

[24]Computationally indistinguishable random variables can have significantly different Shannon entropy.

We conclude the proof by relying on the strong chain rule of Shannon entropy:

$$\begin{aligned}\mathsf{H}(R|\sigma) &= \mathsf{H}(R|U_H,\sigma) + \mathsf{I}(R;U_H|\sigma)\\ &= \mathsf{H}(R|U_H,\sigma) + \mathsf{I}(R;U_H|U_C,\sigma) + \mathsf{I}(R;U_H;U_C|\sigma)\\ &= \mathsf{H}(R|U_H,\sigma) + \mathsf{H}(R|U_C,\sigma) - \underbrace{\mathsf{H}(R|U_C,U_H,\sigma)}_{0} + \mathsf{I}(R;U_H;U_C|\sigma)\\ &= \underbrace{\mathsf{H}(R|U_H,\sigma)}_{O(\log\lambda)} + \underbrace{\mathsf{H}(R|U_C,\sigma)}_{O(\log\lambda)} + \underbrace{\mathsf{I}(U_H;U_C|\sigma)}_{0} - \mathsf{I}(U_H;U_C|R,\sigma).\end{aligned}$$

Since the mutual information between two random variables is always non-negative, $\mathsf{I}(U_H;U_C|R,\sigma) \geq 0$. We conclude that

$$\mathsf{H}(R|\sigma) \leq \mathsf{H}(R|U_H,\sigma) + \mathsf{H}(R|U_C,\sigma) = O(\log\lambda).$$

Corollary: the CRS is not reusable. We discuss the first corollary of the main impossibility: when $\mathsf{H}(R|\sigma) = O(\log\lambda)$, the CRS of the distributed sampler is not reusable. If the CRS was reusable, we would expect that the outputs of independent distributed sampler executions reusing σ look like independent samples from $\mathcal{D}$. We prove however that this is not the case (for more information, check Section 3.2.1 and Section 3.4.1).

We rely on the *collision entropy* $\mathsf{H}_2(R|\sigma)$: the latter measures the probability that, if we rerun the distributed sampler twice reusing the same CRS σ, we obtain colliding outputs $R = R'$. Formally, $\mathsf{H}_2(R|\sigma) = -\log(\Pr[R = R'])$. A well-known result in information theory (an easy application of Jensen's inequality) shows that collision entropy is always smaller than Shannon's entropy. We therefore conclude that $\mathsf{H}_2(R|\sigma) \leq \mathsf{H}(R|\sigma) = O(\log\lambda)$. In other words, the probability of colliding outputs in two distributed sampler executions with the same CRS is inverse-polynomial. Notice that since we are considering a distribution $\mathcal{D}$ such that $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log\lambda)$, the probability that $\mathcal{D}$ generates collisions is negligible.

Corollary: the CRS cannot be short. We show that, when $\mathsf{H}(R|\sigma) = O(\log\lambda)$, the CRS has roughly the size of the output. More formally, $|\sigma| \geq \mathsf{H}_{\mathsf{Yao}}(\mathcal{D}) - O(\log\lambda)$. The Yao entropy [Yao82] $\mathsf{H}_{\mathsf{Yao}}(\mathcal{D})$ measures how much a sample from $\mathcal{D}$ can be compressed in polynomial time without losing information. In other words, $\mathsf{H}_{\mathsf{Yao}}(\mathcal{D})$ provides a good approximation how the most succinct representation of the samples produced by $\mathcal{D}$ (for more details, check Section 3.2.1 and Section 3.4.2).

We prove our result we start by showing that if $\mathsf{H}(R|\sigma) = O(\log\lambda)$, then $\mathsf{H}_{\mathsf{Yao}}(R|\sigma) = O(\log\lambda)$. To conclude our argument, we would like to rely on the chain rule for Yao entropy [KPW13, Appendix B], which states that $\mathsf{H}_{\mathsf{Yao}}(R|\sigma) \geq \mathsf{H}_{\mathsf{Yao}}(R) - |\sigma|$. At that point, we are done: it is sufficient to recall that Yao's entropy is preserved under computational indistinguishability. Since, in the ideal world, when all parties are honest, the distribution of R is exactly $\mathcal{D}$, we conclude that $\mathsf{H}_{\mathsf{Yao}}(R) = \mathsf{H}_{\mathsf{Yao}}(\mathcal{D})$.

There is however a problem: in [KPW13, Appendix B], the authors prove the chain rule for Yao entropy by building a compressor for R, given an efficient compressor for R conditioned on the correlated σ. The issue is that the compressor they build for R is not polynomial-time, its running time is exponential in the size of the CRS! Their idea is the following: let (c,d) be the compressor-decompressor pair for R conditioned on σ. To compress R, brute force for a σ such that $d(c(R,\sigma),\sigma) = R$, then output $c(R,\sigma),\sigma$.

In the context of distributed samplers, we observe that we do not need to perform a brute force search to find the correlated σ: it is sufficient to generate the CRS by feeding R into the simulator of the distributed sampler protocol (we simulate an execution in which all parties are honest). In this way, the compressor runs in polynomial time.

Corollary: the CRS cannot be nice. We show that when $\mathsf{H}(R|\sigma) = O(\log\lambda)$, the CRS of the distributed sampler cannot be unstructured unless $\mathcal{D}$ is *obliviously samplable*, i.e., we can generate secure samples from $\mathcal{D}$ by just relying on public random coins (no interaction is needed).

We show this by proving a slightly more general result: if $\mathsf{H}(R|\sigma) = O(\log\lambda)$, it is possible to generate secure samples for $\mathcal{D}$ with no interaction, given just sufficiently many CRSs for the distributed sampler and public random coins. The idea is the following: to sample from $\mathcal{D}$, take one of the given CRSs and run the distributed sampler protocol generating the messages of the parties using the public randomness. We will denote this distribution by $\mathcal{D}'$. The question is: why is the produced sample secure? We need to show the existence of a PPT simulator that, given the sample R, outputs a CRS and public random coins that produce R.

To simulate the CRS, we simply feed R in the simulator for the distributed sampler protocol (we simulate an execution in which all parties are honest). How can we simulate the public random coins, however? After all, recovering them from the simulated messages is infeasible! We solve the problem by relying on the first corollary we presented: if we rerun the distributed sampler protocol without changing the CRS, we end up with the same output with inverse-polynomial probability. Therefore, all we need to do is to retry running the protocol again and again with the same CRS until we obtain R.

There is an issue, however: the probability of colliding outputs in the first corollary is taken also over the randomness producing the sample R. In other words, the probability of successfully inverting a sample from $\mathcal{D}'$ is *on average* inverse-polynomial. That means that there could exist a *polynomial (but not overwhelming) fraction* of all samples R for which the probability of inverting is extremely low! Our simulation strategy succeeds only when we are not dealing with one of these bad samples. Luckily, we are easily able to tell apart good samples from bad samples: we simply try to invert them! If we succeed sufficiently often, we are dealing with a good sample, otherwise not. To conclude, we modify the sampling algorithm $\mathcal{D}'$: the algorithm will be given many CRSs for the distributed sampler and public random coins. It will produce many samples by running the distributed sampler once for every CRS, generating the messages using the public randomness. Then, it will test the results and output the first good sample it finds. In other words, $\mathcal{D}'$ will produce secure good samples. Notice that we can use good samples in place of any CRS generated according to $\mathcal{D}$. This is because a sample from $\mathcal{D}$ is good with inverse-polynomial probability[25]. For more details, we refer to Section 3.2.1 and Section 3.4.3.

On the generality of the impossibility. The main disadvantage of the lower bounds we just sketched is that our argumentations hold only in the universal composability model [Can01]. We believe however that the results should generalise to security with black-box simulation. Furthermore, also super-polynomial simulation and honest-majority do not seem to help us get around the impossibility. Below, we sketch the ideas at the base of our intuition:

- *Super-polynomial simulation.* Consider the ideal world execution of the protocol and suppose that we are in the UC model, but the simulator runs in super-polynomial time $O(T)$. Suppose also that $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log T)$ (otherwise, the trivial distributed sampler in which all parties output the most likely element in the support of $\mathcal{D}$ would (almost) be secure). We argue that the distributed sampler output chosen by the adversary must belong in the set Q of samples received by the simulator before sending U_H and σ. If that was not the case, the only possibility left for the simulator is to keep asking for samples until the functionality hits the expected output. Since $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log T)$, the probability of this event is negligible. Therefore, as for polynomial-time simulation, U_H and σ restrict the output in Q with overwhelming probability. Actually, we argue something stronger: U_H and σ restrict the output in a subset of Q having *polynomial size* (the size of Q is $O(T)$). The reason is simple: U_H and σ can be viewed as an encoding of the subset of outputs in Q the adversary can get out of them. Since U_H and σ have polynomial size, the entropy (and therefore the size) of this subset should be polynomial! Given this observation, we can prove all our lower bounds as we did for polynomial-time simulation.

- *Honest majority.* Suppose we are once again in the UC model, the simulator runs in polynomial time, but the adversary can corrupt at most t parties out of n. If we retrace our impossibility for the dishonest majority setting, we observe that the argument fails only towards the end: the fact that $\mathsf{H}(R|U_H, \sigma) = O(\log \lambda)$ does not imply that $\mathsf{H}(R|U_C, \sigma) = O(\log \lambda)$. This is because we cannot consider

[25]If a protocol was insecure when the CRS is a good sample, the protocol would be insecure with inverse-polynomial probability when the CRS comes from $\mathcal{D}$.

the protocol execution in which the roles of honest and corrupted parties are switched: we would pass from an honest majority to a dishonest majority. There is however another way to get around the problem: suppose that we manage to prove that there exists a constant $\tau > 0$ such that there always exist a subset H of at least $n - t + 1$ parties such that $\mathsf{H}(R|U_H, \sigma) \geq \mathsf{H}(R|\sigma)/\tau$, then we immediately obtain that $\mathsf{H}(R|\sigma) \leq O(\log \lambda)$.

I conjecture that $\tau = n/t$. To support this argument, observe that if the corruption threshold is particularly low, we can build distributed samplers that do not need any CRS. For instance, if $n = \lambda$ and $t = \log \lambda$ (therefore $n/t = \omega(1)$), we can design a standalone-secure distributed sampler for the uniform distribution over $\{0, 1\}^\lambda$: each party $\mathcal{P}_i$ chooses the i-th bit of the output at random and broadcasts its choice. The parties output the concatenation of the broadcast bits (for simulation, we need to rewind).

- *Standalone security with black-box simulation.* Suppose that we are again in presence of an adversary corrupting a dishonest majority of participants. The simulator still runs in polynomial time with black-box access to the adversary, however, we let it rewind it. If we retrace our impossibility proof for the UC model, we observe that our arguments fail quite soon: it is unclear whether U_H and σ restrict the output in Q with overwhelming probability! What we know is that they do it with at least inverse-polynomial probability. Indeed, if the output chosen by the adversary does not belong to Q, the simulator can rewind and retry.

  Now, consider the set Ω of all outputs an honest adversary can get once fixed σ and U_H. For any inverse-polynomial q, we can consider two subsets of this space: elements that the adversary obtains with probability greater than q (call it Ω_q^1) and elements that the adversary obtains with probability at most $q/2$ (call it Ω_q^2). Observe that we can tell if a given element belongs to Ω_q^1 or Ω_q^2 by simply estimating its probability by running some trials (apply the Chernoff bound).

  Now, if the probability of the output belonging to Ω_q^2 is negligible, it means that σ and U_H restrict the output in a set of polynomial size: the complementary of Ω_q^2 (it contain at most $2/q$ elements). So, we can reprove all the impossibilities we saw for the UC model.

  If instead the probability of the output belonging to Ω_q^2 is non-negligible, intuitively, we can obtain a distinguisher between the real world and the ideal world by checking whether the output of the sampler belongs to Ω_q^1 or not. Since the simulator ensures that the output belongs to Q and a good portion of elements in Q occurs with relatively high inverse-polynomial frequency, the probability of the output belonging in Ω_q^1 in intuitively larger in the ideal world.

1.2.5 New Definitions for Actively Secure Distributed Samplers

In [AWZ23a], we tried to get around the impossibilities we just finished describing. Our work ended up introducing two new game-based security definition: *hardness-preserving distributed samplers* and *indistinguishability-preserving distributed samplers* (collectively referred to as *security-preserving distributed samplers*).

Hardness-preserving distributed samplers. Hardness-preserving distributed samplers preserve the hardness of search games based on the sample they produce, even in presence of an adversary maliciously corrupting a proper subset of their participants. More formally, imagine a search game in which an adversary is provided with a sample R from a distribution $\mathcal{D}(1^\lambda)$ and needs to find some value T (for instance a trapdoor) related to R. Suppose that the game is hard: any PPT adversary will fail in finding T with overwhelming probability. Then, we would like that if we generate R using a distributed sampler, the search game still remains hard, even if a subset of participants in the protocol is maliciously corrupted. Hardness-preserving distributed samplers ensure exactly this.

Formally, the primitive is defined by relying on a real world/ideal world paradigm. In the real world, we let a PPT adversary run the distributed sampler, maliciously controlling a subset of participants. At the end of the game, we provide the adversary with the output of the protocol execution R (notice that the adversary would be able to compute R by itself). In the ideal world, on the other hand, the adversary interacts with a simulator, which is being provided an ideal sample $R' \xleftarrow{\$} \mathcal{D}(1^\lambda)$. At the end of the protocol, the adversary is given R' (which may or may not coincide with the output of the distributed sampler).

Importantly, *we do not ask the real world to be indistinguishable from the ideal world!* The two worlds will be clearly distinguishable: in the ideal world, there is high probability that R' will not coincide with the output of the distributed sampler! We ask instead that, *if the adversary outputs 1 with non-negligible probability in the real world, so it will in the ideal world.* Notice that this implies that, in the ideal world, R' coincides with the output of the protocol with non-negligible probability!

So, why does this security definition suffice to preserve the hardness of search games? The idea is simple: suppose this is not the case. In particular, that means that there exists a PPT adversary $\mathcal{A}$ that wins the search game with non-negligible probability when the sample R is produced by the distributed sampler. We can derive a new adversary $\mathcal{A}'$ which, after running the distributed sampler, runs the search game with $\mathcal{A}$ on the produced sample R and outputs 1 if $\mathcal{A}$ succeeds. This adversary outputs 1 with non-negligible probability in the real world execution of the hardness-preserving distributed sampler, so it must do the same even in the ideal world execution. We however reach a contradiction: in the ideal world, the search game is played on R', a true sample from $\mathcal{D}(1^\lambda)$! The probability of winning the game in this case is negligible! For more details on the definition of hardness-preserving distributed samplers, we refer to Section 4.2.1 and Section 4.5.1.

Indistinguishability-preserving distributed samplers. Hardness-preserving distributed samplers do not necessarily preserve the functionality of the protocols they compile. Indeed, consider a protocol Π relying on a CRS $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$. Suppose that Π implements a functionality $\mathcal{F}$ against active adversaries and let $\mathcal{S}$ be the corresponding simulator. If the protocol compiled using the distributed sampler Π' still implemented $\mathcal{F}$, how could we build a simulator for it? It is natural to try by relying on $\mathcal{S}$. The main issue, however, is that it is common for simulators to embed trapdoors in the CRSs they produce, which they later leverage to simulate the rest of the MPC construction: $\mathcal{S}$ can behave exactly in this way! In these cases, if we want to rely on $\mathcal{S}$, we would need to simulate the execution of the distributed sampler so that, whichever sample gets chosen by the adversary, we are able to recover the relative trapdoor for $\mathcal{S}$ to complete the simulation. In a hardness-preserving distributed sampler, it is not only unclear how to achieve this, it could also be that the produced samples do not hide any trapdoor the simulator can use! For instance, the sampler could generate random tuples of group elements (g_1, g_2, g_3, g_4), whereas the simulator could require a CRS of the form $(g_1, g_2, g_1^\alpha, g_2^\alpha)$ and the relative trapdoor α: under DDH the distributions are indistinguishable, but only one of them hides the desired trapdoor.

To solve these problems, we introduced a new notion: *indistinguishability-preserving distributed samplers.* This primitive allows compiling Π while preserving the functionality, *if particular conditions are satisfied.* Formally, we require that $\mathcal{S}$ provides the simulated CRS to the adversary *before interacting with the functionality* (in other words, the simulated CRS is independent of any information known to $\mathcal{F}$ but not to $\mathcal{S}$). This is a property satisfied by many MPC constructions (for instance, consider the HSS protocol of [OSY21]).

Notice that if we want to preserve the functionality, it is intrinsic to require additional properties: we cannot hope indistinguishability-preserving distributed samplers to work for any protocol. If that was the case, by compiling the trivial protocol where the all the parties output the CRS $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$, we would obtain an actively secure, one-round construction implementing the functionality $\mathcal{F}_\mathcal{D}$ in Figure 1.1. This not only violates all the lower bounds we discussed in Section 1.2.4, it even contradicts Cleve's result [Cle86].

Indistinguishability-preserving distributed samplers are defined relatively to a pair of computationally indistinguishable distributions $(\mathcal{D}, \mathcal{D}')$, where $\mathcal{D}'$ produces a trapdoor along with its sample. For intuition, regard them as the distribution of the CRS in Π and the distribution of the CRS simulated by $\mathcal{S}$, respectively. The distributed sampler will have two indistinguishable modes of operation: in the real mode, it will produce samples from $\mathcal{D}$, while, in the simulated mode, it will generate samples from $\mathcal{D}'$. Furthermore, in simulated mode, by leveraging a backdoor, we can efficiently retrieve the trapdoors embedded in the samples chosen by the adversary. To summarise, we can build a simulator for the compiled protocol Π' by running the indistinguishability-preserving distributed sampler in simulated mode followed by an execution of $\mathcal{S}$ (after providing it with the trapdoor embedded in the sample chosen by the adversary). For more details on the definition of hardness-preserving distributed samplers, we refer to Section 4.2.1 and Section 4.5.2.

Remark. Indistinguishability-preserving distributed samplers always need a CRS. If that was not the case, we could obtain an actively secure, 3-round OT protocol with black-box simulation in the plain model by compiling the 2-round construction by Peikert, Vaikuntanathan and Waters [PVW08] (actively secure, 3-round OT with black-box simulation in the plain model is known to be impossible [HV16]). Given this result, our goal is therefore to construct indistinguishability-preserving distributed samplers relying on a the simplest CRS possible: reusable, unstructured and of length independent of $\mathcal{D}$, $\mathcal{D}'$ and the number of participants.

1.2.6 Building Security-Preserving Distributed Samplers

In [AWZ23a], we build security-preserving distributed samplers by relying on a new type of distributed samplers we called *lossy distributed samplers*.

Lossy distributed samplers. Lossy distributed samplers have two modes of operation: *standard mode* and *lossy mode*. When the distributed sampler is in standard mode, the set of possible outputs Ω an adversary can obtain once the CRS (if this exists) and messages of the honest parties are fixed, is *large*, superpolynomial in size (Ω is often called the *support*). From an entropy perspective, $\mathsf{H}(R|U_H, \sigma) = \omega(\log \lambda)$.

When the distributed sampler is in lossy mode, instead, the support Ω is small: its size is upper bounded by a polynomial q parametrising the lossy mode. Again, from an entropy perspective $\mathsf{H}(R|U_H, \sigma) = O(\log \lambda)$.

Most importantly, the two modes of operations *are not* indistinguishable! If the sampler is in lossy mode, an adversary that keeps rerunning the protocol in its head changing the messages of the corrupted parties but not the rest, will soon start seeing a lot of repeated outputs (because Ω is small). This, on the other hand, will not happen if the sampler is in standard mode (Ω is big). Instead of asking for indistinguishability, we define security similarly to what Zhandry did for ELFs [Zha16]: we ask that, by parametrising the lossy mode with a sufficiently large polynomial q, we can make the distinguishability advantage between the two modes an arbitrarily small inverse-polynomial function. Formally, for every polynomial $p(\lambda)$ and inverse-polynomial quantity $\delta(\lambda)$, there should exist a polynomial $q(\lambda)$ for which no adversary running in time at most $p(\lambda)$ can distinguish the two modes of operation, where the lossy mode is parametrised by $q(\lambda)$, with advantage greater than $\delta(\lambda)$. This similarity between lossy distributed samplers and ELFs is not a coincidence: ELFs will be the base of our construction.

We are interested in lossy distributed samplers that satisfy an additional property: *programmability*. In a programmable distributed sampler set in lossy mode, it is possible to hide an ideal sample[26] $R \overset{\$}{\leftarrow} \mathcal{D}(1^\lambda)$ in the support Ω. In particular, due to the small size of the support in lossy mode, there exists an inverse-polynomial probability that the adversary ends up choosing the ideal sample R as the output of the protocol without even realising! For more details, we refer to Section 4.2.1 and Section 4.6.

From lossy to hardness-preserving distributed samplers. Any programmable, lossy distributed sampler is also hardness-preserving. Proving it is rather simple: in the real world, the sampler will be run in standard mode; in the ideal world, on the other hand, the sampler will be set in lossy mode and we rely on programmability to hide the ideal sample R' (the one given to the simulator) in the support.

The proof relies on a hybrid argument. Suppose we deal with a PPT adversary $\mathcal{A}$ that, after interacting with the standard mode of the sampler and being provided with the output of the protocol, outputs 1 with non-negligible probability.

- We start by switching the distributed sampler to lossy mode: all of the sudden the size of the support shrinks to polynomial, however, the distinguishing advantage between this hybrid and the starting point is non-negligible. This is not an issue: by choosing sufficiently large parameters of the lossy mode, we can still ensure that $\mathcal{A}$ outputs 1 with non-negligible probability.

[26]We use the term "ideal sample" to denote a sample produced by $\mathcal{D}(1^\lambda)$ in contrast to an object that is computationally indistinguishable from it.

- Next, we rely on programmability and we hide an ideal sample $R' \xleftarrow{\$} \mathcal{D}(1^\lambda)$ in the support of the lossy-mode distributed sampler. This hybrid is indistinguishable from the previous one, so $\mathcal{A}$ still outputs 1 with non-negligible probability. Furthermore, with inverse-polynomial probability, the adversary ends up choosing R' as output of the protocol without even realising.

- In the final hybrid, corresponding to the ideal world execution of the hardness-preserving game, instead of providing the adversary with the output of the sampler, we provide it with R'. With inverse-polynomial probability, this is still the correct output and, conditioned on this event, the adversary still outputs 1 with non-negligible probability.

We conclude with two observations: the first one is that the simulator for the hardness-preserving distributed sampler will depend on the adversary due to the parameters of the lossy mode. The second observation is that our hardness-preserving distributed sampler suffers from a polynomial security loss. This is unavoidable because, in all hardness-preserving distributed samplers, the adversary can always perform the attack we saw in our impossibilities: it tries a polynomial number of executions by regenerating the messages of the corrupted players and then chooses the one with the outcome it likes the most. For more details, we refer to Section 4.2.1 and Section 4.8.

Building lossy distributed samplers: the construction. Our idea is to rely on ELFs [Zha16]. We would like to modify our semi-honest distributed samplers (sketched in Section 1.2.2) by equipping the construction with an ELF. When the ELF is in injective mode, the distributed sampler will have a large support (superpolynomial in size); when the ELF is in lossy mode, the support will be small.

Recall that our semi-honest (actually, non-rushing semi-malicious) construction was obtained by compressing a 2-round sampling protocol $\Pi_{\mathcal{D}}$ with security against rushing semi-malicious adversaries into a single round (we call this the *virtual execution of* $\Pi_{\mathcal{D}}$). We did this by relying on two obfuscated programs: the encryption program, which took care of generating the messages in the first round of $\Pi_{\mathcal{D}}$, and the decryption program, which took care of the second round of $\Pi_{\mathcal{D}}$. We also recall that the construction satisfied a particular property we called *programmability* (not the same as for lossy distributed samplers): if we know in advance the randomness chosen by the other parties, we can ensure that the encryption and decryption programs of party $\mathcal{P}_i$ produce their outputs using the simulator for $\Pi_{\mathcal{D}}$ (let it be $\mathcal{S}_{\mathcal{D}}$). In other words, knowing the randomness of the corrupted players in advance allows us to control the output of the protocol. This was sufficient to achieve security against non-rushing semi-malicious adversaries.

So, where shall we put the ELFs? Our idea is to let them appear only when the distributed sampler is set in lossy mode: we pick an honest party $\mathcal{P}_i$ and we modify its programs so that $\mathcal{P}_i$'s messages in the virtual execution of $\Pi_{\mathcal{D}}$ are always generated using the simulator $\mathcal{S}_{\mathcal{D}}$. Notice that in order to simulate the second round in $\Pi_{\mathcal{D}}$, $\mathcal{S}_{\mathcal{D}}$ needs to receive a sample $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$ from the functionality. This will be the output of the lossy distributed sampler! We let $\mathcal{P}_i$'s decryption program generate this samples using the randomness produced by a (puncturable) PRF key $\widehat{K}$ and the ELF: first we apply the ELF on the concatenation of the encryption programs, then we feed the result in the puncturable PRF.

Three observations: (1) if the ELF is in injective mode, the samples provided to $\mathcal{S}_{\mathcal{D}}$ will look random and independent of those generated in the other executions of the decryption program; (2) if the ELF is in lossy mode, the support of the lossy distributed sampler will be polynomial in size (at most one possible output per element in the image of the ELF); (3) using standard techniques based on iO and puncturable PRFs [SW14], we can hide an ideal sample in the support (just pick a random element in the image of the ELF and puncture the PRF in that point). The third property will give us programmability[27]!

There is a problem: in order for $\mathcal{S}_{\mathcal{D}}$ to succeed, it also needs to know the randomness used by the other parties in the virtual execution of $\Pi_{\mathcal{D}}$ (this is because $\Pi_{\mathcal{D}}$ is only semi-maliciously secure). How do we allow the decryption program of party $\mathcal{P}_i$ to recover them? We observe that the randomness used by the other parties in the virtual execution of $\Pi_{\mathcal{D}}$ is determined by the PRF key $(K_j)_{j\neq i}$ hardcoded in their encryption programs. The decryption program of party $\mathcal{P}_i$ receives all these programs as input, so it is just a matter of

[27]We need to require the ELF to be *regular* [Zha16]. In a regular ELF, the uniform distribution over the inputs is mapped statistically close to the uniform distribution over the image.

extracting the hidden keys from them. We do this by relying on extractable NIZKs: the parties will prove the wellformedness of their encryption programs using these NIZKs. Along with the encryption programs of the other parties, each decryption program will also obtain the associated NIZKs. If the sampler is in standard mode, the decryption programs will simply verify the NIZKs, outputting $\perp$ whenever the procedure fails. If instead, the sampler is in lossy mode, $\mathcal{P}_i$'s decryption program will try to extract the witness from the proofs. If the procedure fails, the program will output $\perp$, otherwise, it will use the extracted PRF keys to run $\mathcal{S}_{\mathcal{D}}$. For more details, we refer to Section 4.2.2 and Section 4.7.

Building lossy distributed samplers: proving security. We want to prove that, if we set the ELF is injective mode, $\mathcal{P}_i$'s message in the lossy-mode distributed sampler is indistinguishable from the one sent in the standard-mode execution. This would be sufficient to prove the security of our construction (to prove that the distinguishing advantage between lossy mode and standard mode can be made an arbitrarily small inverse-polynomial amount, we can just apply th security properties of the ELF). We do this by relying on the subexponential security of iO and of the protocol $\Pi_{\mathcal{D}}$ (for instance, if we build $\Pi_{\mathcal{D}}$ using the multi-key FHE construction of [AJJM20], we achieve subexponential security). Specifically, by repeating the same hybrid argument for every choice of the randomness of the other parties (for instance, as done in [HIJ$^+$17]), we can slowly transition from the programs that generate the messages in the virtual execution of $\Pi_{\mathcal{D}}$ honestly, to programs that always rely on $\mathcal{S}_{\mathcal{D}}$.

There is however another issue: there are too many choices for the randomness of the other parties. So many, that it is impossible to choose the security parameters of the primitives we used without making $\mathcal{P}_i$'s programs explode: the size of $\mathcal{P}_i$'s encryption program would need to be greater than the size of the other parties' programs! Obviously, this is a problem: the message of any party should look independent on whether the party is honest or not!

We solve the issue by requiring each party to generate the randomness they use by expanding a λ-bit PRG seed! The NIZKs will enforce this behaviour to all parties: the total entropy in the messages becomes therefore $n \cdot \lambda$ (n denotes the number of participants). When the sampler is in lossy mode, $\mathcal{P}_i$ will leverage a trapdoor in the NIZK CRS to deviate from the protocol without being detected: it will generate its message using randomness of entropy significantly greater than $n \cdot \lambda$, increasing the security of $\mathcal{P}_i$'s programs without making their size explode! For more details, we refer to Section 4.2.2 and Section 4.7.

Almost everywhere extractable NIZKs. There is still a technical issue in our security proof: when we move from the standard mode to the lossy mode of the distributed sampler, we switch from a program that verifies the provided NIZKs to a program that tries to extract the witnesses from them. An adversary can therefore try to distinguish between the two worlds by trying to generate a proof that verifies, but does not allow the extraction of the witness. This task is hard, so intuitively, obfuscation prevents it from distinguishing the two types of programs. If we look a little bit more in detail, however, we notice an issue: indistinguishability obfuscation does not seem to suffice to argue this, we would need to rely on stronger forms of obfuscation such as ideal and virtual black-box obfuscation (known to not exist [BGI$^+$01]) or differing-input obfuscation (diO) [BGI$^+$01] (for which different results suggest its impossibility [GGHW14, BSW16]).

Instead of walking that road, we instead rely on a stronger form of extractable NIZKs that is compatible with indistinguishability obfuscation. We called this new primitive *almost-everywhere extractable NIZKs*.

The idea is to rely on the result of Boyle, Chung and Pass [BCP14]: they proved that if two programs have only polynomially-many differing inputs and these are all hard to find, then indistinguishability obfuscation is sufficient to make the two programs indistinguishable. We observe that the result can sometimes be generalised to a superpolynmial number of differing inputs by relying on subexponentially secure iO: suppose that all prefixes of differing inputs lie in a set of superpolynomial size L, and finding one of these prefixes is hard even for adversaries running in $O(L)$ time, then subexponentially secure iO is enough to make the two programs indistinguishable. To leverage this trick, in an almost everywhere extractable NIZK, all proofs that verify without allowing the extraction of the witness will have a prefix in a set of size L. All elements in this set will be hard to find even for adversaries running in time $O(L)$.

We build these NIZKs by relying on a subexponentially secure injective one-way function (OWF). The CRS of the construction will consists of a OWF challenge and a PKE public key pk: all NIZKs that verify but do not allow the extraction of the witness will begin with a (perfectly binding) commitment to the preimage of the challenge in the CRS. In other words, the number of possible prefixes of these proofs will be at most 2^ℓ where ℓ denotes the length of the randomness needed by the commitment scheme. We can choose the security parameter of the one-way function so that it remains secure even against adversaries running in time $O(2^\ell)$.

Along with the commitment, each proof will include also an encryption under pk and a perfectly sound witness-indistinguishable proof (NIWI) [BOV03, GOS06a, GOS06b, BP15]. The NIWI will prove that either the commitment hides a preimage to the OWF challenge in the CRS or the ciphertext hides a valid witness for the statement. In other words, to extract the witness it is sufficient to decrypt the ciphertext using a trapdoor: the secret counterpart of pk. Notice that we need to rely on a perfectly correct public key encryption scheme, otherwise the number of prefixes of NIZKs that verify without allowing extraction would increase significantly. Notice that the CRS of our scheme is reusable and depends only on the security parameter λ. Furthermore, we can instantiate the construction so that the CRS becomes unstructured.

Identity-based almost everywhere extractable NIZKs. There is still one last issue: ideally, we would like to build a distributed sampler in which the CRS is reusable across different sets of participants. It is therefore tempting to rely on a single instance of the almost everywhere extractable NIZK and let all the parties prove the wellformedness of their message with respect to the same small CRS. However, how can the two variants of decryption program (the one that simply verifies the NIZKs and the one that instead extracts the witnesses) be indistinguishable if we simulate the NIZKs of the honest parties? If all parties prove security with respect to the same CRS, the simulated NIZKs can be easily converted into a differing input! The issue is slightly more general: our NIZK construction is not necessarily simulation-almost-everywhere-extractable. In other words, in the security proof of our distributed samplers, we can switch from the hybrid in which $\mathcal{P}_i$'s decryption program simply verifies the NIZKs to the hybrid in which the program extracts the witnesses, only as long as we do not provide the adversary with any simulated NIZK! The issue is that, once we have modified the decryption program, we cannot switch to a simulated NIZK anymore: in order for the modified decryption programs to extract the witnesses, they need to use a trapdoor. We can hope to rely on the zero-knowledge properties of the NIZK only as long as the trapdoor remains secret, however, in our protocol, we provide it to the adversary in obfuscated form as part of the decryption program of party $\mathcal{P}_i$. It is unclear whether this is a real security threat, however, it prevents our security proof from going through.

We solve the problem by making the construction identity-based. Suppose that all participants are associated with a unique identity. We substitute the public-key encryption scheme in our construction with (perfectly correct) identity based encryption (IBE) [Sha84, ABB10]: the parties will generate the ciphertexts in their proofs by encrypting their witnesses under their own identity. With this modification, we solve the issue we described above: the decryption program of party $\mathcal{P}_i$ does not need to know the IBE master secret key, it just needs to know the decryption keys associated with the identity of the other participants! Notice that even if these trapdoors are leaked, the ciphertexts in $\mathcal{P}_i$'s proofs cannot be decrypted (by the security of IBE), so we can still rely on zero-knowledge. For more details on almost-everywhere extractable NIZKs, we refer to Section 4.2.6 and Section 4.4.

Remark. Another way to deal with NIZKs that verify but do not allow the extraction of their witness without giving up on indistinguishability obfuscation, is to rely on *statistical simulation extractable NIZKs* as done in [HIJ+17]. In these NIZKs, we can hide a statement in the CRS. The construction guarantees that we can generate simulated proofs only for the statement hidden in the CRS. This solution has however a disadvantage compared to almost everywhere extractable NIZKs: their CRS is as big as the statements and is not reusable!

Building indistinguishability-preserving distributed samplers. It turns out that our lossy distributed sampler is also indistinguishability-preserving. The proof is based on a hybrid argument we sketch below. Suppose we want to compile a protocol Π that relies on a CRS $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$. Let $\mathcal{F}$ be the functionality it implements and let $\mathcal{S}$ be the associated simulator. Let $\mathcal{D}'(1^\lambda)$ be the trapdoored distribution of the simulated CRSs. We start from the real world execution of the compiled protocol Π'.

- In the first hybrid, we switch the distributed sampler to lossy mode. All of a sudden, the support becomes of polynomial size. However, the distinguishing advantage between this hybrid and the original protocol execution is an inverse-polynomial quantity $\epsilon_1(\lambda)$. We will see why this not an issue.

- In the next hybrids, we gradually change the distribution of the samples that the lossy-mode decryption program feeds into $\mathcal{S}_\mathcal{D}$. Recall that these samples will be the outputs of the distributed sampler. More formally, for each element in the image of the ELF, we use the programmability properties of the sampler to switch from $\mathcal{D}(1^\lambda)$ to the trapdoored distribution $\mathcal{D}'(1^\lambda)$. Notice that in the intermediate hybrids, only a fraction of the support of the sampler will be distributed according to $\mathcal{D}'(1^\lambda)$. Whenever the adversary ends up picking one of these samples, we can recover the associated trapdoor by using the PRF key $\widehat{K}$ hidden in the lossy-mode decryption program. We then use the retrieved trapdoor to simulate the rest of Π' using $\mathcal{S}$ and $\mathcal{F}$. If instead the adversary chooses a sample from $\mathcal{D}(1^\lambda)$, we simply keep running Π'. Notice that we do not need to require the subexponential indistinguishability between $\mathcal{D}(1^\lambda)$ and $\mathcal{D}'(1^\lambda)$! Thanks to the ELF, we can transition from one distribution to the other using a polynomial number of hybrids.

- In the last hybrid, corresponding to the ideal world, we switch the ELF in the lossy-mode decryption program back to injective mode. All of a sudden, the support of the sampler becomes again of superpolynomial size, however, all the samples are still produced using $\mathcal{D}'(1^\lambda)$ and, independently of which one gets chosen by the adversary, all subsequent communication is simulated using $\mathcal{S}$. Once again, on the other hand, the distinguishing advantage between this hybrid and the previous one is an inverse-polynomial quantity $\epsilon_2(\lambda)$.

So, how can we prove that the ideal world is indistinguishable from Π'? Suppose it is not, i.e., the distinguishing advantage is greater than some non-negligible function $\epsilon(\lambda)$. It is possible to choose the parameters of the lossy-mode ELF so that $\epsilon_1(\lambda)+\epsilon_2(\lambda) \leq \epsilon(\lambda)/2$. We have just proven that the distinguishing advantage between real world and ideal world is strictly smaller than $\epsilon(\lambda)$. This is a contradiction! For more details on the construction of indistinguishability-preserving distributed samplers, we refer to Section 4.2.5 and Section 4.8.1.

On the relationship between our techniques and non-interactive MPC. The techniques we used to build our lossy distributed sampler can be generalised to build a new version of one-round MPC that does not need public-key infrastructures: suppose we want to evaluate a function F with n inputs (one for each party). Start from a 2-round protocol for F with subexponential security against rushing semi-malicious adversaries (for instance we can build it in the plain model using multi-key FHE [AJJM20]). Compress the two rounds into one as we did in our lossy distributed sampler.

Clearly, we cannot prevent residual function attacks (see Section 1.1.3), however, we can guarantee that if, for any (i, x_1, x_2), the residual functions

$$F(\cdots \underbrace{x_1}_{i} \cdots), \qquad F(\cdots \underbrace{x_2}_{i} \cdots)$$

coincide, then the adversary cannot tell whether $\mathcal{P}_i$'s input is x_1 or x_2.

As Yuval Ishai made me notice, this is some form of *multiparty indistinguishability obfuscation*. The security properties of our NIMPC protocol seem extremely weak, almost useless, but the same could have been said (wrongly) for indistinguishability obfuscation: why should we care about a primitive that guarantees the indistinguishability between obfuscated programs only when they compute the same function? The relationship between the two primitives becomes clearer if we consider a function F_{iO} that on input a circuit C and a value x, outputs $C(x)$. That immediately gives an indistinguishability obfuscator.

1.2.7 Constant-Round Coin Tossing Extension with Large Stretch

We now focus on *coin tossing extension* (CTE). We recall that this is the study of protocols that implement the functionality $\mathcal{F}_{\mathcal{U}_\mathsf{m}}$ (see Figure 1.1, $\mathcal{U}_m$ denotes the uniform distribution over $\{0,1\}^{m(\lambda)}$) given access

Ideal Functionality for Coin Tossing with Identifiable Abort $\mathcal{F}_{\mathsf{IA}}$

On input Sample from all parties, compute $w \xleftarrow{\$} \{0,1\}^{m(\lambda)}$ and give it to the adversary. If the adversary answers with OK, output w to all honest parties. Otherwise, wait for a *non-empty* subset S of corrupted players and output $(\perp, S)$ to all honest parties.

Figure 1.3: Ideal functionality for coin tossing with identifiable abort.

to a functionality $\mathcal{F}_{\mathcal{U}_{\mathsf{k}}}$ for some small polynomial function $k(\lambda)$[28]. We are interested in achieving security against active adversaries corrupting an arbitrary number of participants. We often call the functionality $\mathcal{F}_{\mathcal{U}_{\mathsf{k}}}$ *the magic button*: whenever the parties press the button, $\mathcal{F}_{\mathcal{U}_{\mathsf{k}}}$ spits out $k(\lambda)$ uniformly random coins. Throughout the section, we use r to denote the output of the protocol and of $\mathcal{F}_{\mathcal{U}_{\mathsf{m}}}$, we use s to denote the sample produced by the magic button. Recall that a result by Hofheinz et al. [HMU06] shows that coin tossing extension protocols require $k(\lambda) = \omega(\log \lambda)$.

In [ADIN24a], we presented several coin tossing extension constructions with $\omega(\log \lambda)$ stretch and $O(1)$-round complexity. All of them achieve only computational security. We describe them below.

Coin tossing extension from coin tossing with identifiable abort. The starting point of our first construction is a coin tossing protocol with identifiable abort Π_{IA}. Formally, Π_{IA} implements the functionality $\mathcal{F}_{\mathsf{IA}}$ in Figure 1.3. Notice that Π_{IA} is not a secure coin tossing because the adversary can adaptively choose to abort after seeing the output of the protocol. For instance, it can ensure that the honest parties never obtain an output where the first bit is 0. It is true that, when an abort occurs, the honest parties discover the identity of at least one corrupted party, so they can kick that player out and restart the protocol with the guarantee that, after restarting at most $n-1$ times, they will obtain an output. However, that output will not be uniformly random, our adversary can still sensibly bias the distribution, making the first bit much more likely to be 1.

We solve this issue by "encrypting" the output of the protocol under the randomness produced by the magic button. Specifically, in our CTE protocol, the parties will run Π_{IA} restarting and kicking out corrupted players as we described above. Once Π_{IA} successfully terminates, providing all honest parties with a sample w, the participants will call the magic button and output $r \leftarrow w \oplus \mathsf{PRG}(s)$, where s is the string produced by $\mathcal{F}_{\mathcal{U}_{\mathsf{k}}}$. In other words, w can be viewed as an encryption of the final output r under the PRG seed s. The adversary has still the ability to abort after seeing w, however, since at the moment of the decision s is still secret, w leaks no information about the final output. To summarise, the adversary can force Π_{IA} to restart, but in this way it will not bias any any way the output of the CTE protocol.

Finally, we observe that coin tossing protocols with identifiable abort can be easily build using $O(1)$-rounds. So, if we rely on one of these constructions, we obtain an $O(n)$-round CTE protocol with arbitrarily large polynomial stretch. Notably, in [GLOV12], Goyal et al. show that $O(1)$-round coin tossing with identifiable abort and standalone security can be built from one-way functions. For more details, we refer to Section 5.2.3 and Section 5.7.

Coin tossing extension in the hidden subgroup framework. The round complexity of the CTE protocol we just described scales with the number of participants. We wonder: is there a truly $O(1)$-round CTE protocol with large stretch? For instance, is there a 1-round solution?

If we want to build a 1-round CTE protocol with $\omega(\log \lambda)$ stretch, we need to find a way to decrease the influence of the adversary! Indeed, if we consider the entropy diagram of the protocol in the ideal world, we notice that $\mathsf{H}(r|s) = \omega(\log \lambda)$. In other words, s does not contain enough entropy to determine the output entirely on its own, we need to rely on the messages exchanged by the parties! Since the output is

[28]In [ADIN24a] and Chapter 5, we use a different notation: instead of $k(\lambda)$, we use $n(\lambda)$. The number of parties is instead denoted by N.

independent of anything the adversary tries to do, the $\omega(\log \lambda)$ missing entropy should be taken care of by just the honest messages.

Our idea is to rely on an algebraic framework inspired by another work of mine in collaboration with Ivan Damgård, Claudio Orlandi and Peter Scholl [ADOS22]. We called it *the hidden subgroup framework*. We work over a large, abelian, multiplicative group G containing a cyclic subgroup H of much smaller size. Let h denote a generator for H. The subgroup H is *hidden*: we require that the uniform distributions over G and over H are computationally indistinguishable.

Our protocol is really simple: each party $\mathcal{P}_i$ broadcasts a random element $h_i \xleftarrow{\$} H$ along with a simulation-extractable NIZK π_i proving that h_i belongs indeed to H. After this unique round of interaction, the parties call the magic button and output $r \leftarrow \prod_{j\in[n]} h_j \cdot h^s$, where s is the randomness produced by $\mathcal{F}_{\mathcal{U}_k}$. We claim that r looks like a random element in G. The stretch is therefore positive: r contains $\log|G|$ bits of pseudoentropy, whereas s just needs $\log|H|$ bits of entropy.

We elaborate on why r looks random in G. We observe that a simulator can embed a trapdoor in the NIZK CRS. In this way, it can substitute the element h_i broadcast by an honest party $\mathcal{P}_i$ with a uniformly random element $g \xleftarrow{\$} G$. The corresponding NIZK π_i will be simulated using the trapdoor hidden in the CRS. Observe that the adversary can use rushing against the honest parties, so it will see g before committing to its choice of messages for the corrupted players. It will therefore try to bias the product $g \cdot \prod_{j\neq i} h_j$. For instance, it may make sure that its bit representation always starts with 1. Its influence is however limited: since the adversary does not know the NIZK trapdoor, the elements broadcast by the corrupted parties all belong to H. In other words, the adversary can only move the product within the coset gH! Whatever bias it manages to create will be erased by the random correction term h^s!

It is possible to prove that the protocol we just sketched is UC secure. That means that we can obtain arbitrary large stretch without affecting the round complexity: we run multiple executions of the protocol in parallel, we call the magic button only once and we use part of the produced randomness as a "virtual" magic button output for the next parallel execution.

Remark. If we want to obtain a CTE protocol that produces uniformly random bits instead of a random group element, we need to rely on an explainable conversion function $\mathsf{Convert}$. In other words, $\mathsf{Convert}$ should map the uniform distribution over G into a distribution over $\{0,1\}^{m(\lambda)}$ that is indistinguishable from random. Furthermore, given a random element $r \in \{0,1\}^{m(\lambda)}$ we should be able to simulate a random $g \in G$ such that $r = \mathsf{Convert}(g)$. For some groups (e.g. class groups), it is unclear whether this conversion function exists.

Instantiations of the hidden subgroup framework. We propose three instantiations of the framework, the first one is based on DDH, the second one on the Paillier group, the third one on class groups.

- *Decisional Diffie-Hellman.* In our first instantiation, we rely on a cyclic group G' of large prime order p. We set G to the direct product $G' \times G'$. The subgroup H will instead be generated by a random pair $(g_1, g_2) \in G$. Under DDH, a random element in H is indistinguishable from a random element in G.

- *Paillier group.* In the second instantiation, the large group G will be $\mathbb{Z}^*_{N^2}$ where N is the product of two random, large safe-primes p and q[29]. The hidden subgroup H will consist of all $2N$-th powers in G. Given that p and q are safe primes, this subgroup is cyclic of order $p' \cdot q'$. Furthermore, a random $2N$-th power in G looks like a uniformly distributed element under the QR and DCR assumptions.

- *Class groups.* A class group G' can be rewritten as the direct product of a cyclic subgroup F of known prime order q and another subgroup H' of unknown order. The subgroup H' is not necessarily cyclic, however, it is possible to generate an element $h \in H'$ such that $f^s \cdot h^r$ is computationally indistinguishable from h^r for $s \xleftarrow{\$} [q]$ and $r \xleftarrow{\$} [\ell]$. Here, ℓ is a public parameter of the class group that guarantees that h^r is statistically close to uniform in $\langle h \rangle$. The computational assumption we just described is called the *hidden subgroup membership* assumption (HSM) [CL15]. To instantiate the hidden subgroup framework, we will set $H := \langle h \rangle$ and $G := F \times H$.

[29] A safe prime is a prime number of the form $p = 2p' + 1$ where p' is also prime.

Notice that for some instantiations of the framework (specifically, DDH and class groups), the CRS of the CTE protocol becomes unstructured. In these cases, we can implement the setup using additional calls to the magic button. Notice that the CRS of these CTE protocols is always reusable. For more information on CTE from the hidden subgroup framework, we refer to Section 5.2.3 and Section 5.6.

One-round coin tossing extension without CRS. We wonder whether it is possible to build a one-round CTE protocol without any CRS. This is tricky: what allowed us to reduce the influence of the adversary in the construction we described above was indeed the NIZK trapdoor hidden in the CRS! If we remove all setups, the only hope we have to restrict the influence of the adversary is the output of the magic button. Without it, the adversary would have at least as much power as the simulator: what would happen, indeed, if the adversary runs a copy of the simulator?

We solve our problems by relying on a sort of *lossy trapdoor functions* [PW08]. Suppose that there exists a scheme that outputs elements over the group G of the hidden subgroup framework. Moreover, assume that when the function is in lossy mode the image is contained in the hidden subgroup H. We can build a one-round CTE protocol without any setup as follows: we let all parties broadcast a random element in the domain of the trapdoor function scheme. Let x_i be the message of party $\mathcal{P}_i$. After this only round of interaction, the parties call the magic button. Suppose that it provides the description of n lossy trapdoor functions $f_1, \dots, f_n$, one for each player, along with a correction term s, similarly to the protocol we saw in the previous paragraph. The output of our protocol will be $\prod_{j \in [n]} f_j(x_j) \cdot h^s$.

We can argue that the output of this protocol looks random in G: a simulator can generate all trapdoor functions in lossy mode except for one, addressed to an honest party $\mathcal{P}_i$. In this way, $g := f_i(x_i)$ will be uniformly distributed over G, whereas, for all $j \neq i$, $f_j(x_j)$ will belong to H. Once again, the influence of the adversary is restricted inside the coset gH. So, any bias that the adversary manages to introduce will get erased by the random correction term h^s!

Constructing lossy trapdoor functions. Unfortunately, we did not manage to build lossy trapdoor functions with the property we desire in the hidden subgroup framework. We succeeded however using lattice-based cryptography.

Our lossy trapdoor functions will be represented by a fat matrix over a ring $\mathbb{Z}_q$. Let M be the number of columns. We split the matrix corresponding to the trapdoor function f_j in two submatrices A_j and B_j: A_j will consist of the first K rows, B_j will consist of the remaining V rows. The message x_j broadcast by party $\mathcal{P}_j$ in the CTE protocol will be a low-norm vector of dimension M produced by a discrete Gaussian distribution. To compute $f_j(x_j)$, we simply perform a matrix multiplication, i.e., $f_j(x_j) = (y_j^1, y_j^2)$ where $y_j^1 = A_j \cdot x_j$ and $y_j^2 = B_j \cdot x_j$.

Two observations: (1) we choose the dimensions of A_j and B_j so that, when the latter are chosen at random, $f_j(x_j)$ will be statistically close to uniform by the leftover hash lemma [ILL89]; (2) it is possible to sample f_j along with a trapdoor T that allows to efficiently compute preimages [Ajt99, GPV08, MP12]. In other words, given any $y_j = (y_j^1, y_j^2)$, using T, we can derive a discrete Gaussian vector x_j such that $f_j(x_j) = y_j$.

To generate f_j in lossy mode, we simply sample A_j at random and we set $B_j^\intercal \leftarrow A_j^\intercal \cdot S + E_j$ where S is a random $K \times V$ matrix over $\mathbb{Z}_q$ and E_j is a low-norm $M \times V$ matrix sampled according to a discrete Gaussian distribution. Observe that under LWE, B_j looks random, so the injective (or better surjective) mode and the lossy mode look indistinguishable. Finally, we notice that, when f_j is in lossy mode, we have

$$y_j^2 = B_j \cdot x_j \approx (S^\intercal \cdot A_j) \cdot x_j = S^\intercal \cdot y_j^1.$$

To summarise, when f_j is in surjective mode, $f_j(x_j)$ is random over the vector space $G := \mathbb{Z}_q^{K+V}$. When f_j is instead in lossy mode $f_j(x_j)$ is close in norm to the subspace $H := \{(y_1, y_2) \in \mathbb{Z}_q^{K+V} | y_2 = S^\intercal \cdot y_1\}$.

Lattice-based coin tossing extension. If we use our lattice-based lossy trapdoor function in the blueprint we described earlier, we almost obtain a one-round CTE protocol. We just need to make some adjustments.

First of all, we need to modify the output: instead of multiplying the outputs of the trapdoor functions, we sum them, i.e., $\sum_{j\in[n]} f_j(x_j) + \gamma$, where γ is a correction term uniformly distributed in H. Observe that if we set all trapdoor functions of the corrupted parties in lossy mode using the same LWE secret S, the influence of the adversary is restricted to be close to H (the sum of vectors close to H will still be close to H). Therefore, any bias the adversary manages to introduce is almost entirely erased by the random correction term γ. *Almost!* Because γ cannot take care of the noise that prevents the outputs of the lossy-mode trapdoor functions from lying precisely in H. To deal with these small errors, we round down the last V entries of the output to a smaller modulus $p \ll q$ (we therefore need to rely on LWE with superpolynomial modulus-to-noise ratio).

There is another issue we need to take care of: currently our protocol has negative stretch! Notice indeed that the description of the trapdoor functions is larger than the output they produce. Furthermore, it is unclear how to represent the correction term γ in a succinct way. Solving the first problem is easy: we notice that the trapdoor functions are reusable. In other words, in the protocol, we let each party $\mathcal{P}_j$ broadcast L discrete Gaussian vectors $x_j^1, \dots, x_j^L$. We will produce L different outputs: the ℓ-th one will be obtained by rounding the last V entries of $\sum_{j\in[n]} f_j(x_j^\ell) + \gamma_j$. Notice that the number of correction terms has grown to L, however, for a sufficiently large L, the size of the output will now be greater than the description of the trapdoor functions.

We compress the correction terms $\gamma_1, \dots, \gamma_L$ as follows: we let the magic button generate two matrices $C \in \mathbb{Z}_q^{K\times W}$ and $D \in \mathbb{Z}_q^{V\times W}$ and L discrete Gaussian samples $e_1, \dots, e_L \in \mathbb{Z}_q^W$ (observe that discrete Guassians are explainable distributions[30] [AWY20]). We set $\gamma_\ell := (C \cdot e_\ell, D \cdot e_\ell)$. We choose W so that, if C is random, we can apply the leftover hash lemma to argue that $C \cdot e_\ell$ is statistically close to uniform over $\mathbb{Z}_q^K$. While in the real world execution of the protocol C and D will be uniformly random, in the simulation, we set $D^\intercal \leftarrow C^\intercal \cdot S + E$ where E is a $W \times V$ discrete Gaussian matrix (under LWE with superpolynomial modulus-to-noise ratio, D looks random). This ensures that

$$D \cdot e_\ell \approx S^\intercal \cdot C \cdot e_\ell.$$

In other words, γ_j will be close in norm to a random element in H. Notice that, by taking sufficiently large V and L, we can ensure that the description of $(C, D, e_1, \dots, e_L)$ is smaller than the amount of produced randomness. With these modifications, the stretch becomes positive.

Final adjustments: better complexity in the number of parties. The CTE protocol we have built so far can be proven secure in the UC model, so it can achieve arbitrarily large stretch. Furthermore, it is easy to prove that the protocol is secure even against adaptive corruption due to its one-round nature and the explainability of discrete Gaussian distributions [AWY20]: even if a party $\mathcal{P}_j$ gets corrupted after having sent its message, we can still simulate the randomness that produced the discrete Gaussian vectors it broadcast. There is only one annoying detail: the amount of randomness the magic button needs to produce scales linearly on the number of parties!

We show how to make the complexity scale logarithmically in n by relying on the techniques used by Gentry, Sahai and Waters in [GSW13] to build FHE. We recall that a GSW encryption of a bit b consists of $U \cdot R + b \cdot G$, where U is a fat matrix representing the public key, R is a random low-norm matrix and G is called the *gadget matrix*. In particular, there exists an efficient deterministic algorithm that, on input any matrix Y, produces a low-norm matrix $G^{-1}(Y)$ such that $G \cdot G^{-1}(Y) = Y$.

In our CTE protocol, instead of letting the magic button generate $f_1, \dots, f_n$, we generate $\lceil \log n \rceil + 1$ matrices $X_1, \dots, X_{\lceil \log n \rceil}, Y$. We view the first $\lceil \log n \rceil$ elements as GSW ciphertexts: for every $j \in [n]$ we derive f_j by computing $\mathsf{Eval}_{\mathsf{GSW}}(\delta_j, X_1, \dots, X_{\lceil \log n \rceil}) \cdot G^{-1}(Y)$, where $\mathsf{Eval}_{\mathsf{GSW}}$ denotes the function evaluation in the GSW FHE scheme and δ_j denotes the Kronecker delta function centred in j[31].

In the ideal world execution, the simulator will generate $X_1, \dots, X_{\lceil \log n \rceil}$ by encrypting the bit representation of the index of an honest party $\mathcal{P}_i$ under a public key U. In other words, f_j will become $U \cdot R_j + \delta_j(i) \cdot Y$, where R_j is a low-norm matrix. The public key U is not sampled at random: we split it into two parts

[30] In an explainable distribution, given a random sample, we can efficiently simulate the randomness that produced it.

[31] In other words, $\delta_j(i) = 1$ if and only if $j = i$, otherwise, $\delta_j(i) = 0$.

U_1, corresponding to the first K rows, and U_2 corresponding to the last V rows. We set $U_2^\intercal \leftarrow U_1^\intercal \cdot S + E'$ where E' is low-norm matrix distributed according to a discrete Gaussian (under LWE with subexponential modulus-to-noise ratio, U_2 looks random). This ensures that, whenever $j \neq i$, f_j is in lossy mode relative to S:

$$B_j = U_2 \cdot R_j \approx S^\intercal \cdot U_1 \cdot R_j = S^\intercal \cdot A_j.$$

On the other hand, f_i can be viewed as the sum of a lossy-mode function $U \cdot R_i$ and another function Y. To simulate the protocol, it is sufficient to generate Y in surjective mode (along with a lattice trapdoor T), this will make f_i surjective. For more details on one-round CTE from lattice-based cryptography, we refer to Section 5.2.3 and Section 5.5.

A lower bound for statistical coin tossing extension with black-box simulation. All the CTE constructions we presented so far achieve only security against computationally bounded adversaries. Since [HMU06], we know that statistically secure CTE is impossible in the UC model but possible if we rely on standalone security. Hofheinz et al. prove this by directly presenting a 1-round unconditionally secure CTE protocol. The construction, however, has only $O(\log \lambda)$ stretch. We ask: is it possible to build $O(1)$-round statistically secure protocols with larger stretch? We show that, if we restrict ourselves to black-box simulation, the answer is no: an R-round statistically secure protocol with black-box simulation and a single call to the magic button has at most $O(R \cdot \log \lambda)$ stretch.

Our proof follows the blueprint of our distributed sampler impossibility in [AOS23] (see Section 1.2.4). Dealing, however, with statistical security, we have a great advantage: conditional Shannon entropy is preserved under statistical closeness (the entropies differ at most by a negligible amount). In other words, the entropy diagrams in the real world and the ideal world will be essentially the same.

We start by proving that, in a CTE protocol in the dishonest majority setting (even a computationally secure one), any round after the last call to the magic button is useless! If all parties stop immediately after the last call and output the value they would return if everybody else suddenly halted, everybody would still agree on the output and the protocol would still be secure. To prove this lemma, we simply generalise the arguments of Cleve [Cle86] to n-party CTE protocols. Indeed, Cleve's result is essentially saying that if a bunch of parties manage to agree on a truly random m-bit string by purely interacting between each other (so without any use of external resources such as the magic button), then they must have already agreed on a truly random m-bit string before starting interacting.

Consider a statistically secure CTE protocol with R rounds. Without loss of generality, suppose that the protocol ends with the call to the magic button. For every $i \in [R]$, let U_H^i denote the list of messages sent until the i-th round by the honest parties. Define U_C^i similarly using the messages of the corrupted players. We analyse the entropy diagram of the construction considering the rushing adversary that simply follows the protocol. We start from the ideal world. By applying the strong chain rule of Shannon's entropy, we obtain that

$$\begin{aligned} m(\lambda) = \mathsf{H}(r) &= \mathsf{H}(r|U_H^R, U_C^R, s) + \mathsf{I}(r; (U_H^R, U_C^R, s)) \\ &= \mathsf{H}(r|U_H^R, U_C^R, s) + \mathsf{I}(r; s|U_H^R, U_C^R) + \mathsf{I}(r; (U_H^R, U_C^R)) \\ &= \mathsf{H}(r|U_H^R, U_C^R, s) + \mathsf{I}(r; s|U_H^R, U_C^R) + \sum_{i=1}^{R} \big(\mathsf{I}(r; U_H^i|U_H^{i-1}, U_C^{i-1}) + \mathsf{I}(r; U_C^i|U_H^i, U_C^{i-1})\big). \end{aligned} \tag{1.1}$$

It is easy to observe that

- since r is uniquely determined by U_H^R, U_C^R and s, we obtain $\mathsf{H}(r|U_H^R, U_C^R, s) = 0$;
- since s is a k-bit string, we obtain $\mathsf{I}(r; s|U_H^R, U_C^R) \leq \mathsf{H}(s) \leq k(\lambda)$.

Continuing with our analysis, we notice that if the simulation is straightline (i.e. we are not allow to rewind), for every $i \in [R]$, U_C^i is independent of r and U_H^i conditioned on (U_H^{i-1}, U_C^{i-1}). This is because r is sampled independently of anything else by the functionality and the adversary generates the messages of the corrupted parties independently of the communication of the honest players in the same round. Written using mutual information,

$$\mathsf{I}(U_C^i; (U_H^i, r) | U_H^{i-1}, U_C^{i-1}) = 0.$$

If we allow the simulator to rewind, however, this is no longer true: the simulator can replay the i-th round multiple times and choose an execution it likes, biasing the distribution of the transcript. Notice that if we denote the number of times we rewind in the i-th round by T_i, we still have

$$\mathsf{I}(U_C^i; (U_H^i, r) | U_H^{i-1}, U_C^{i-1}, T_i) = 0.$$

Moreover,

$$\mathsf{I}(U_C^i; (U_H^i, r) | U_H^{i-1}, U_C^{i-1}) = \mathsf{I}(U_C^i; (U_H^i, r); T_i | U_H^{i-1}, U_C^{i-1}) + \mathsf{I}(U_C^i; (U_H^i, r) | U_H^{i-1}, U_C^{i-1}, T_i) \leq \mathsf{H}(T_i).$$

Therefore, if we rewind at most $Q = \mathsf{poly}(\lambda)$ times, we obtain $\mathsf{I}(U_C^i; (U_H^i, r) | U_H^{i-1}, U_C^{i-1}) = O(\log \lambda)$.

Switching the roles of honest and corrupted parties (our adversary behaves honestly), we obtain the symmetric result $\mathsf{I}(U_H^i; (U_C^i, r) | U_H^{i-1}, U_C^{i-1}) = O(\log \lambda)$. We conclude with the following inequalities

$$\mathsf{I}(r; U_C^i | U_H^i, U_C^{i-1}) \leq \mathsf{I}(U_C^i; (U_H^i, r) | U_H^{i-1}, U_C^{i-1}) = O(\log \lambda),$$
$$\mathsf{I}(r; U_H^i | U_H^{i-1}, U_C^{i-1}) \leq \mathsf{I}(U_H^i; (U_C^i, r) | U_H^{i-1}, U_C^{i-1}) = O(\log \lambda).$$

Putting everything together in (1.1), we obtain $m(\lambda) \leq k(\lambda) + R \cdot O(\log \lambda)$. For more details on the lower bound, we refer to Section 5.2.4 and Section 5.8.

Remark. The lower bound holds even if we restrict the adversarial class to rushing, semi-malicious adversaries. Moreover, the impossibility can be generalised to coin tossing extension with abort, and therefore, coin tossing with abort (recall that we are considering statistical security with black-box simulation).

Unbiased sampling from any distribution. We have seen how the magic button functionality allows us to sample uniformly random strings of arbitrary length in a single round without leaving any influence to the adversary. Can we achieve the same if we want to produce samples from any generic distribution $\mathcal{D}(1^\lambda)$? We show that, using strong primitives such as indistinguishability obfuscation and indistinguishability-preserving distributed samplers, the answer is yes (but we need to rely on a small, reusable, unstructured CRS).

The idea is simple: we use the distributed sampler to compile the following zero-round protocol. The CRS of the construction consists of an obfuscated program that receives as input a λ-bit string s. The program feeds s in a puncturable PRF and uses the produced randomness to generate a sample from $\mathcal{D}(1^\lambda)$. This will be the output of its execution. In our zero-round protocol, the parties immediately call the magic button and feed the produced string s into the obfuscated program, outputting the result.

It is trivial to see that the zero-round protocol implements the functionality $\mathcal{F}_\mathcal{D}$ (see Figure 1.1): the simulator can just pick a random string s and program the obfuscated circuit to output the sample R when it receives s as input (R is provided by the functionality). The modified program looks indistinguishable from the original one even given s!

There is something weird however: why can we apply an indistinguishability-preserving distributed sampler? Earlier we saw that this primitive requires the simulator of the original protocol to generate the CRS without interacting with the functionality. Here, this property is not satisfied: the output of the functionality is obfuscated inside the CRS!

Well, it turns out that, earlier, we told a little lie: indistinguishability-preserving distributed samplers can be applied in a slightly more general setting. It is fine for the simulated CRS to depend on information known only to the functionality as long as this information remains hidden: e.g., in the zero-round protocol, even if we leak the ideal sample R, it is impossible to distinguish between an honestly generated CRS and one that is programmed to output R at a random point s (as long as s is kept secret). Essentially, it is as if the obfuscated program is an encryption of the output under s: the adversary is able to bias the result of the distributed sampler, however, this bias is independent of the output of the sampling protocol. Indeed, the output is encrypted under s and the adversary still does not know what the magic button is going to produce! It is the same trick as in our first CTE construction! For more details, we refer to Section 5.2.5 and Section 5.9.

Ideal Functionality $\mathcal{F}_{\mathcal{C}}$

On input Sample from all parties, for every $j \in [N]$, compute $(R_1^j, \dots, R_n^j) \xleftarrow{\$} \mathcal{C}(1^\lambda)$. Then, for every $i \in [n]$, output $(R_i^j)_{j \in [N]}$ to party $\mathcal{P}_i$.

Figure 1.4: Ideal functionality for the correlation function $\mathcal{C}$.

Ideal Functionality $\mathcal{F}_{\mathcal{C}}^{\mathsf{RSample}}$

On input Sample from all parties, for every $j \in [N]$, wait for $(R_i^j)_{i \in C}$ from the adversary, where C denotes the set of corrupted parties. Compute $(R_i^j)_{i \in H} \xleftarrow{\$} \mathsf{RSample}(1^\lambda, (R_i^j)_{i \in C})$, where H denotes the set of honest parties. Then, for every $i \in H$, output $(R_i^j)_{j \in [N]}$ to party $\mathcal{P}_i$.

Figure 1.5: Ideal functionality for the reverse-samplable correlation function $\mathcal{C}$.

1.2.8 Sampling Correlated Randomness in One Round

We finally explain how distributed samplers can be used to generate large amounts of correlated randomness in a single round and with sublinear communication in the size of the outputs. This is called a public-key PCF. Their relation with distributed samplers was studied in [ASY22a].

Public-key PCFs for reverse-samplable correlation. Suppose that we want to generate N samples from the correlation function $\mathcal{C}(1^\lambda)$ which produces n correlated values $R_1, \dots, R_n$, one for each party. Ideally, we would like the public-key PCF to implement the functionality $\mathcal{F}_{\mathcal{C}}$ in Figure 1.4. Unfortunately, if we do not want to give up on having sublinear communication in N, this is not possible: independently of the number of rounds we use, the communication in the protocol is lower bounded by the entropy of the outputs of the ideal functionality (which is linear in N).

This fact was already known since [BCG+19b]. In their paper, however, Boyle et al. proposed a new way to generate randomness with sublinear communication: we work with *reverse-samplable* correlation functions. These are correlation functions in which it is possible to efficiently simulate the outputs of the honest parties from the outputs of the corrupted players using an algorithm RSample. Notice that not all forms of correlation are reverse-samplable. For example, consider the two party correlation that outputs (x, y) where $y = f(x)$ for a one-way function f: we cannot simulate x given y!

As already shown in [BCG+19b], for any reverse-samplable correlation $\mathcal{C}$, it is possible to design MPC protocols that semi-honestly implement the functionality $\mathcal{F}_{\mathcal{C}}^{\mathsf{RSample}}$ (see Figure 1.5) with sublinear communication in N. Now, we ask if we can do this using a single round of interaction.

We notice that this is fairly easy: we start from a public-key PCF that relies on a CRS. The latter consists of an obfuscated program that takes as input a value $x \in \{0,1\}^L$ and n public keys $\mathsf{pk}_1, \dots, \mathsf{pk}_n$, one for each party. The program feeds the inputs in a puncturable PRF and uses part of the produced pseudorandomness to sample n correlated elements $R_1, \dots, R_n$ from $\mathcal{C}(1^\lambda)$. The rest of the output of the PRF is used to encrypt each R_i under the public key pk_i. The program outputs the n ciphertexts. In the PCF protocol, the parties simply generate PKE key pairs and broadcast the public counterpart. By feeding the public keys in the obfuscated program along with any $x \in \{0,1\}^L$, the parties can obtain correlated randomness without any interaction: party $\mathcal{P}_i$ just needs to decrypt the i-th ciphertext output by the obfuscated program using its secret key. Notice that we produce up to 2^L tuples of correlated material, one for each value of x.

To remove the CRS, we just use a distributed sampler. Notice that, despite we started from a one-round protocol with CRS, the compiled protocol still requires one round. This is because the communication in the original protocol was independent of the CRS: the parties can broadcast their public keys together with their distributed sampler message. It is possible to prove that, under the security of iO, public-key encryption and distributed samplers, the protocol implements $\mathcal{F}_{\mathcal{C}}^{\mathsf{RSample}}$ against semi-honest adversaries, with $\mathsf{poly}(\lambda, L)$ communication.

To upgrade the construction to active security, we have two solutions. If the public-key PCF we just described is secure against non-rushing semi-malicious adversaries (this is the case, for instance, if we used a distributed sampler that is secure against non-rushing semi-malicious adversaries), then, we can rely on the anti-rusher compiler (this would however require a programmable random oracle). The alternative is to rely on an indistinguishability-preserving distributed sampler to generate the obfuscated program in the public-key PCF construction[32]. Notice that indistinguishability-preserving distributed samplers are always compatible with $\mathcal{F}_{\mathcal{C}}^{\mathsf{RSample}}$ given that there is no communication from the functionality to the simulator. For more details, we refer to Section 2.6.

Ideal public-key PCFs. We observe that if we rely on a programmable random oracle, it is possible to get around the impossibility of [BCG+19b]: we design a public-key PCF that implements the functionality $\mathcal{F}_{\mathcal{C}}$ in Figure 1.4 with sublinear communication in N.

The idea is very simple: we use a distributed sampler to generate an adaptive universal sampler [HJK+16] (adaptive universal samplers can be built only in the programmable random oracle model). Simultaneously, the parties broadcast their public keys as they did in the public-key PCFs for reverse-samplable correlation. To generate large amounts of correlated randomness, the parties just pick an $x \in \{0,1\}^L$ and run the universal sampler on input the distribution that samples from $\mathcal{C}(1^\lambda)$, encrypts the produced correlated values under the public keys of the corresponding parties and outputs the n ciphertexts along with x. In this way, we can generate up to 2^L different samples.

Interestingly, in this protocol, the messages sent by the parties are independent of the correlation function $\mathcal{C}$. This leaves the possibility to the protocol participants to sample from multiple correlation functions adaptively chosen after the only round of interaction. We call this an *ideal public-key PCF*. For more details, we refer to Section 2.7.

Bibliography

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572. Springer, Heidelberg, May / June 2010.

[ADIN24a] Damiano Abram, Jack Doerner, Yuval Ishai, and Varun Narayanan. Constant-Round Simulation-Secure Coin Tossing Extension with Guaranteed Output. In Mark Joye and Gregor Leander, editors, *EUROCRYPT 2024*. Springer, May 2024.

[ADIN24b] Damiano Abram, Jack Doerner, Yuval Ishai, and Varun Narayanan. Constant-Round Simulation-Secure Coin Tossing Extension with Guaranteed Output. Cryptology ePrint Archive, Report 2024/?, 2024.

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452. Springer, Heidelberg, August 2022.

[AJJM20] Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 28–57. Springer, Heidelberg, November 2020.

[32]In [ASY22a], we proposed a third solution based on the subexponential security of obfuscation, of PKE and of non-rushing, semi-malicious distributed samplers. The techniques are similar to the one we used in our lossy distributed sampler, however, in order to succeed, they require the correlation function $\mathcal{C}$ to be subexponentially reverse-samplable.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP 99*, volume 1644 of *LNCS*, pages 1–9. Springer, Heidelberg, July 1999.

[AOS23] Damiano Abram, Maciej Obremski, and Peter Scholl. On the (Im)possibility of Distributed Samplers: Lower Bounds and Party-Dynamic Constructions. Cryptology ePrint Archive, Report 2023/863, 2023. https://eprint.iacr.org/2023/863.

[ASY22a] Damiano Abram, Peter Scholl, and Sophia Yakoubov. Distributed (correlation) samplers: How to remove a trusted dealer in one round. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 790–820. Springer, Heidelberg, May / June 2022.

[ASY22b] Damiano Abram, Peter Scholl, and Sophia Yakoubov. Distributed (correlation) samplers: How to remove a trusted dealer in one round. Cryptology ePrint Archive, Report 2022/535, 2022. https://eprint.iacr.org/2022/535.

[AWY20] Shweta Agrawal, Daniel Wichs, and Shota Yamada. Optimal broadcast encryption from LWE and pairings in the standard model. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 149–178. Springer, Heidelberg, November 2020.

[AWZ23a] Damiano Abram, Brent Waters, and Mark Zhandry. Security-Preserving Distributed Samplers: How to Generate Any CRS in One Round Without Random Oracles. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 489–514, Cham, 2023. Springer Nature Switzerland.

[AWZ23b] Damiano Abram, Brent Waters, and Mark Zhandry. Security-Preserving Distributed Samplers: How to Generate Any CRS in One Round Without Random Oracles. Cryptology ePrint Archive, Report 2023/860, 2023. https://eprint.iacr.org/2023/860.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.

[BCG$^+$19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

[BCG$^+$19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Heidelberg, August 2019.

[BCG$^+$20a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020.

[BCG$^+$20b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Heidelberg, August 2020.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 52–73. Springer, Heidelberg, February 2014.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Heidelberg, May 2011.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

[BF97] Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 425–439. Springer, Heidelberg, August 1997.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BGI$^+$14a] Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 387–404. Springer, Heidelberg, August 2014.

[BGI14b] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

[BGR96] Mihir Bellare, Juan A. Garay, and Tal Rabin. Distributed pseudo-random bit generators - a new way to speed-up shared coin tossing. In James E. Burns and Yoram Moses, editors, *15th ACM PODC*, pages 191–200. ACM, August 1996.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532. Springer, Heidelberg, April / May 2018.

[BL20] Fabrice Benhamouda and Huijia Lin. Mr NISC: Multiparty reusable non-interactive secure computation. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 349–378. Springer, Heidelberg, November 2020.

[Blu82] Manuel Blum. Coin flipping by telephone. In *Proc. IEEE Spring COMPCOM*, pages 133–137, 1982.

[BOV03] Boaz Barak, Shien Jin Ong, and Salil P. Vadhan. Derandomization in cryptography. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 299–315. Springer, Heidelberg, August 2003.

[BP15] Nir Bitansky and Omer Paneth. ZAPs and non-interactive witness indistinguishability from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 401–427. Springer, Heidelberg, March 2015.

[BSW16] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 792–821. Springer, Heidelberg, May 2016.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CCD+20] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and abhi shelat. Multiparty generation of an RSA modulus. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 64–93. Springer, Heidelberg, August 2020.

[CHI+21] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, abhi shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In *2021 IEEE Symposium on Security and Privacy*, pages 590–607. IEEE Computer Society Press, May 2021.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 68–86. Springer, Heidelberg, May 2003.

[CL15] Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In Kaisa Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 487–505. Springer, Heidelberg, April 2015.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Heidelberg, August 2016.

[dMRT21] Cyprien de Saint Guilhem, Eleftheria Makri, Dragos Rotaru, and Titouan Tanguy. The return of eratosthenes: Secure generation of RSA moduli using distributed sieving. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 594–609. ACM Press, November 2021.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[epr] Cryptology ePrint Archive. https://eprint.iacr.org.

[FLOP18] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 331–361. Springer, Heidelberg, August 2018.

[FMY98] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In *30th ACM STOC*, pages 663–672. ACM Press, May 1998.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 74–94. Springer, Heidelberg, February 2014.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 518–535. Springer, Heidelberg, August 2014.

[Gil99] Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 116–129. Springer, Heidelberg, August 1999.

[GLOV12] Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *53rd FOCS*, pages 51–60. IEEE Computer Society Press, October 2012.

[GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 323–341. Springer, Heidelberg, August 2007.

[GOS06a] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 97–111. Springer, Heidelberg, August 2006.

[GOS06b] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Heidelberg, May / June 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[GS17] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In Chris Umans, editor, *58th FOCS*, pages 588–599. IEEE Computer Society Press, October 2017.

[GS18a] Sanjam Garg and Akshayaram Srinivasan. A simple construction of iO for turing machines. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 425–454. Springer, Heidelberg, November 2018.

[GS18b] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499. Springer, Heidelberg, April / May 2018.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.

[Gui19] Ursula K. Le Guin. *The Carrier Bag Theory of Fiction.* Ignota Books, 2019.

[HIJ$^+$17] Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 181–211. Springer, Heidelberg, December 2017.

[HJK$^+$16] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 715–744. Springer, Heidelberg, December 2016.

[HMRT12] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient RSA key generation and threshold Paillier in the two-party setting. In Orr Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 313–331. Springer, Heidelberg, February / March 2012.

[HMU06] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. On the (im-)possibility of extending coin toss. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 504–521. Springer, Heidelberg, May / June 2006.

[HV16] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. What security can we achieve within 4 rounds? In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 486–505. Springer, Heidelberg, August / September 2016.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015.

[IKOS10] Yuval Ishai, Abishek Kumarasubramanian, Claudio Orlandi, and Amit Sahai. On invertible sampling and adaptive security. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 466–482. Springer, Heidelberg, December 2010.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st ACM STOC*, pages 12–24. ACM Press, May 1989.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Heidelberg, May / June 2022.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 419–428. ACM Press, June 2015.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 830–842. ACM Press, October 2016.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, Heidelberg, April / May 2018.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

[KPW13] Stephan Krenn, Krzysztof Pietrzak, and Akshay Wadia. A counterexample to the chain rule for conditional HILL entropy - and what deniable encryption has to do with it. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 23–39. Springer, Heidelberg, March 2013.

[KRS15] Dakshita Khurana, Vanishree Rao, and Amit Sahai. Multi-party key exchange for unbounded parties from indistinguishability obfuscation. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 52–75. Springer, Heidelberg, November / December 2015.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

[Lin03] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology*, 16(3):143–184, June 2003.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Heidelberg, October 2021.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

[PS98] Guillaume Poupard and Jacques Stern. Generation of shared RSA keys by two parties. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 11–24. Springer, Heidelberg, October 1998.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 187–196. ACM Press, May 2008.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[RSW00] Ronald Rivest, Adi Shamir, and David Wagner. Time-Lock Puzzles and Time-Released Crypto. Technical Report MIT/LCS/TR-684, Massachusetts Institute of Technology, Cambridge, USA, February 2000.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27:623–656, 1948.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer, Heidelberg, August 1984.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[WW21] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious LWE sampling. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 127–156. Springer, Heidelberg, October 2021.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd FOCS*, pages 80–91. IEEE Computer Society Press, November 1982.

[Zha16] Mark Zhandry. The magic of ELFs. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 479–508. Springer, Heidelberg, August 2016.

Part II

Manuscripts

Chapter 2

Distributed (Correlation) Samplers: How to Remove a Trusted Dealer in One Round

Damiano Abram, Peter Scholl, Sophia Yakoubov

Abstract. Structured random strings (SRSs) and correlated randomness are important for many cryptographic protocols. In settings where interaction is expensive, it is desirable to obtain such randomness in as few rounds of communication as possible; ideally, simply by exchanging one reusable round of messages which can be considered public keys.

In this paper, we describe how to generate any SRS or correlated randomness in such a single round of communication, using, among other things, indistinguishability obfuscation. We introduce what we call a *distributed sampler*, which enables n parties to sample a single public value (SRS) from any distribution. We construct a semi-malicious distributed sampler in the plain model, and use it to build a semi-malicious *public-key PCF* (Boyle *et al.*, FOCS 2020) in the plain model. A public-key PCF can be thought of as a distributed *correlation* sampler; instead of producing a public SRS, it gives each party a private random value (where the values satisfy some correlation).

We introduce a general technique called an *anti-rusher* which compiles any one-round protocol with semi-malicious security without inputs to a similar one-round protocol with active security by making use of a programmable random oracle. This gets us actively secure distributed samplers and public-key PCFs in the random oracle model.

Finally, we explore some tradeoffs. Our first PCF construction is limited to *reverse-sampleable* correlations (where the random outputs of honest parties must be simulatable given the random outputs of corrupt parties); we additionally show a different construction without this limitation, but which does not allow parties to hold secret parameters of the correlation. We also describe how to avoid the use of a random oracle at the cost of relying on sub-exponentially secure indistinguishability obfuscation.

2.1 Introduction

Randomness is crucial for many cryptographic protocols. Participants can generate some randomness locally (e.g. by flipping coins), but the generation of other forms of randomness is more involved. For instance, a *uniform reference string* (URS) must be produced in such a way that a coalition of corrupt protocol participants — controlled by the adversary — cannot bias it too much. Even more complex is the generation of a *structured* reference string (SRS, such as an RSA modulus), which can depend on secrets that should not be known to anyone. For instance, constructions such as cryptographic accumulators [Bd94] use an RSA modulus whose factorization is known to nobody, while many succinct zero-knowledge proofs such as SNARKs [BCCT12] require a more complex form of SRS.

In contrast to common reference strings, which are public, some protocols demand *correlated randomness*, where each participant holds a secret random value, but because the values must satisfy some relationship, they cannot be generated locally by the participants. An example of correlated randomness is random oblivious transfer, where one participant has a list of random strings, and another has one of those strings as well as its index in the list. Such correlated randomness often allows cryptographic protocols to run with a more efficient online phase.

Typically, in order to set up an SRS or correlated randomness without making additional trust assumptions, the parties must run a secure multi-party computation protocol, which takes several rounds of interaction. This is the case, for instance, in "setup ceremonies" [BGG19, BGM17] that have been designed to generate trusted SNARK parameters for applications. In this paper, we explore techniques that let parties sample *any* common reference string or correlation in just *one round* of interaction.

2.1.1 Related Work

There are a number of lines of work that can be used to generate randomness in different ways.

Universal samplers. A universal sampler [HJK$^+$16] is a kind of SRS which can be used to obliviously sample from any distribution that has an efficient sampling algorithm. That is, after a one-time trusted setup to generate the universal sampler, it can be used to generate arbitrary other SRSs. Hofheinz *et al.* [HJK$^+$16] show how to build universal samplers from indistinguishability obfuscation and a random oracle, while allowing an unbounded number of adaptive queries. They also show how to build weaker forms of universal sampler in the standard model, from single-key functional encryption [LZ17]. A universal sampler is a very powerful tool, but in many cases impractical, due to the need for a trusted setup.

Non-interactive multiparty computation (NIMPC). Non-interactive multiparty computation (NIMPC, [BGI$^+$14a]) is a kind of one-round protocol that allows n parties to compute any function of their secret inputs in just one round of communication. However, NIMPC requires that the parties know one another's public keys before that one round, so there is another implicit round of communication.[1] NIMPC for general functions can be constructed based on subexponentially-secure indistinguishability obfuscation [HIJ$^+$17].

Spooky encryption. Spooky encryption [DHRW16] is a kind of encryption which enables parties to learn joint functions of ciphertexts encrypted under independent public keys (given one of the corresponding secret keys). In order for semantic security to hold, what party i learns using her secret key should reveal nothing about the value encrypted to party j's public key; so, spooky encryption only supports the evaluation of *non-signaling* functions. An example of a non-signaling function is any function where the parties' outputs are an additive secret sharing. Dodis *et al.* [DHRW16] show how to build spooky encryption for any such additive function from the LWE assumption with a URS (this also implies multi-party homomorphic secret sharing for general functions). In the two-party setting, they also show how to build spooky encryption for a larger class of non-signaling functions from (among other things) sub-exponentially hard indistinguishability obfuscation.

Pseudorandom Correlation Generators and Functions (PCGs and PCFs). Pseudorandom correlation generators [BCG$^+$19a, BCG$^+$19b, BCG$^+$20b] and functions [BCG$^+$20a, OSY21] let parties take a small

[1]This requirement is inherent; otherwise, an adversary would be able to take the message an honest party sent, and recompute the function with that party's input while varying the other inputs. NIMPC does allow similar recomputation attacks, but only with *all* honest party inputs fixed, which a PKI can be used to enforce.

Primitive	Distribution	Output
Distributed Sampler (DS, Definition 2.3.1)	fixed	public
Reusable Distributed Universal Sampler (Definition 2.7.6)	any	public
Public-key PCF (pk-PCF, [OSY21])	fixed, reverse-samplable	private
Ideal pk-PCF (Definition 2.7.2)	any	private

Figure 2.1: In this table we describe one-round n-party primitives that can be used for sampling randomness. They differ in terms of whether a given execution enables sampling from *any* distribution (or just a fixed one), and in terms of whether they only output public randomness (in the form of a URS or SRS) or also return private correlated randomness to the parties.

amount of specially correlated randomness (called the *seed* randomness) and expand it non-interactively, obtaining a large sample from a target correlation. Pseudorandom correlation generators (PCGs) support only a fixed, polynomial expansion; pseudorandom correlation functions (PCFs) allow the parties to produce exponentially many instances of the correlation (via evaluation of the function on any of exponentially many inputs).

PCGs and PCFs can be built for any *additively secret shared* correlation (where the parties obtain additive shares of a sample from some distribution) using LWE-based spooky encryption mentioned above. Similarly, with two parties, we can build PCGs and PCFs for more general *reverse-samplable* correlations by relying on spooky encryption from subexponentially secure iO. PCGs and PCFs with better concrete efficiency can be obtained under different flavours of the LPN assumption, for simpler correlations such as vector oblivious linear evaluation [BCGI18], oblivious transfer [BCG$^+$19b] and others [BCG$^+$20b, BCG$^+$20a].

Of course, in order to use PCGs or PCFs, the parties must somehow get the correlated seed randomness. *Public-key* PCGs and PCFs allow the parties to instead derive outputs using their independently generated public keys, which can be published in a single round of communication. The above, spooky encryption-based PCGs and PCFs are public-key, while the LPN-based ones are not. Public-key PCFs for OT and vector-OLE were recently built based on DCR and QR [OSY21]; however, these require a structured reference string consisting of a public RSA modulus with hidden factorization.

2.1.2 Our Contributions

In this paper, we leverage indistinguishability obfuscation to build public-key PCFs for *any* correlation. On the way to realizing this, we define several other primitives, described in Figure 2.1. One of these primitives is a *distributed sampler*, which is a weaker form of public-key PCF which only allows the sampling of public randomness. (A public-key PCF can be thought of as a distributed *correlation* sampler.) Our constructions, and the assumptions they use, are mapped out in Figure 2.2. We pay particular attention to avoiding the use of sub-exponentially secure primitives where possible (which rules out strong tools such as probabilistic iO [CLTV15]).

We begin by exploring constructions secure against semi-malicious adversaries, where corrupt parties are assumed to follow the protocol other than in their choice of random coins. We build a semi-malicious distributed sampler, and use it to build a semi-malicious public-key PCF. We then compile those protocols to be secure against active adversaries. This leads to a public-key PCF that requires a random oracle, and supports the broad class of *reverse-sampleable* correlations (where, given only corrupt parties' values in a given sample, honest parties' values can be simulated in such a way that they are indistinguishable from the ones in the original sample).

We also show two other routes to public-key PCFs with active security. One of these avoids the use of a random oracle, but requires sub-exponentially secure building blocks. The other requires a random oracle, but can support general correlations, not just reverse-sampleable ones. (The downside is that it does not support correlations *with master secrets*, which allow parties to have secret input parameters to the correlation.)

These are valuable trade-offs, as described below.

On the importance of realizing general correlations: There are many valuable correlation that are

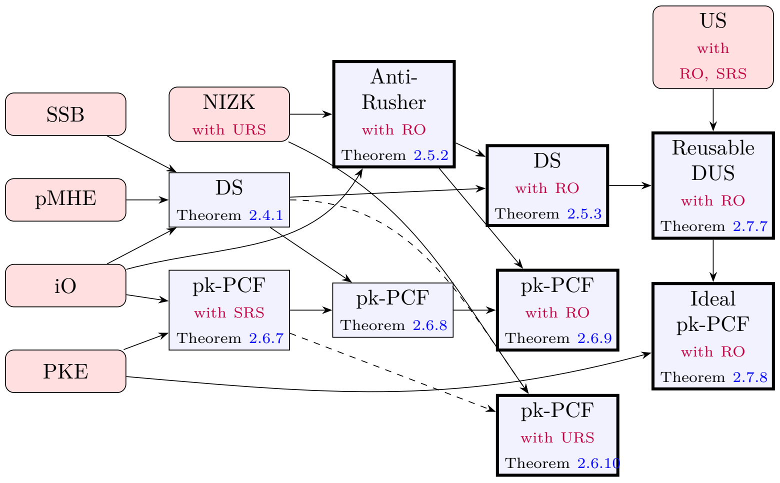

SSB
NIZK with URS
Anti-Rusher with RO Theorem 2.5.2
US with RO, SRS
pMHE
DS Theorem 2.4.1
DS with RO Theorem 2.5.3
Reusable DUS with RO Theorem 2.7.7
iO
pk-PCF with SRS Theorem 2.6.7
pk-PCF Theorem 2.6.8
pk-PCF with RO Theorem 2.6.9
Ideal pk-PCF with RO Theorem 2.7.8
PKE
pk-PCF with URS Theorem 2.6.10

Figure 2.2: In this table we describe the constructions in this paper. In pink are assumptions: they include somewhere statistically binding hash functions (SSB), multiparty homomorphic encryption with private evaluation (pMHE [AJJM20], a weaker form of multi-key FHE), indistinguishability obfuscation (iO), non-interactive zero knowledge proofs (NIZK), and universal samplers (US). In blue are constructions of distributed samplers (DS, Definition 2.3.1), reusable distributed universal samplers (reusable DUS, Definition 2.7.6) and public-key pseudorandom correlation functions (pk-PCFs, [OSY21]). Constructions with bold outlines are secure against active adversaries; the rest are secure against semi-malicious adversaries. In magenta are necessary setup assumptions. (Note that the availability of a random oracle (RO) immediately implies the additional availability of a URS.) Dashed lines denote the use of sub-exponentially secure tools.

not reverse-sampleable. An example of such a correlation gives a garbled circuit to one party, and all the wire labels to another. Since the labels cannot be reverse-sampled from the circuit (without violating the security properties of the garbling scheme), such a correlation is not reverse-sampleable; however, it is a valuable form of pre-processing for secure two-party computation.

On the importance of avoiding a random oracle: Random oracles are an idealized assumption with no known realization; there is a large gap between a random oracle and a hash function, which is often substituted for a random oracle in practice.

On the importance of avoiding sub-exponential assumptions: It may seem strange to want to avoid sub-exponentially secure primitives,[2] when many candidates for indistinguishability obfuscation itself are based on subexponential assumptions [JLS21]. However, despite informal arguments [LZ17], this is not known to be inherent: earlier iO candidates are based on polynomial hardness [GGH+13] (albeit for an exponential family of assumptions), and in future we may obtain iO from a single, polynomial hardness assumption. In general, it is always preferable to require a weaker form of security from a primitive, and this also leads to better parameters in practice. The problem of removing sub-exponential assumptions from iO, or applications of iO, has been studied previously in various settings [GPSZ17, LZ17].

2.1.3 Technical Overview

Distributed Samplers

We start by introducing a new tool called a *distributed sampler* (DS, Section 2.3). A distributed sampler allows n parties to sample a single, public output from an efficiently sampleable distribution $\mathcal{D}$ with just one

[2]By sub-exponential security, we mean that no PPT adversary cannot break the security of that primitive with probability better than $2^{-\lambda^c}$ for a constant c.

round of communication (which is modelled by the exchange of public keys).

Semi-malicious distributed samplers. We use multiparty homomorphic encryption with private evaluation (pMHE [AJJM20], a weaker, setup-free version of multi-key FHE) and indistinguishability obfuscation to build semi-malicious distributed samplers in the plain model (Section 2.4). In our distributed sampler construction, all parties can compute an encryption of the sample from everyones' public keys (using, among other things, the homomorphic properties of the encryption scheme), and then use an obfuscated program in party i's public key to get party i's partial decryption of the sample. The partial decryptions can then be combined to recover the sample itself. The tricky thing is that, in the proof, we must ensure that we can replace the real sample with an ideal sample. To do this, we must remove all information about the real sample from the public keys. However, pMHE secret keys are not *puncturable*; that is, there is no way to ensure that they do not reveal any information about the contents of one ciphertext, while correctly decrypting all others. We could, in different hybrids, hardcode the correct partial decryption for each of the exponentially many possible ciphertexts, but this would blow up the size of the obfuscated program. Therefore, instead of directly including a pMHE ciphertext in each party's DS public key, we have each party obfuscate an additional program which produces a new pMHE ciphertext each time it is used. This way, when we need to remove all information about a given sample, we can remove the entire corresponding secret key (via the appropriate use of puncturable PRFs and hardcoded values). This technique may be useful for other primitives, such as NIMPC [BGI+14a] and probabilistic iO [CLTV15], to avoid the use of an exponential number of hybrids.

Achieving active security with a random oracle. Upgrading to active security is challenging because we need to protect against two types of attacks: malformed messages, and rushing adversaries, who wait for honest parties' messages before sending their own. We protect against the former using non-interactive zero knowledge proofs. (This requires a URS which, though it is a form of setup, is much weaker than an SRS.) We protect against the latter via a generic transformation that we call an *anti-rusher* (Section 2.5.1). To use our anti-rusher, each party includes in her public key an obfuscated program which takes as input a hash (i.e. a random oracle output) of all parties' public keys. It then samples *new* (DS) public keys, using this hash as a PRF nonce. This ensures that even an adversary who selects her public keys after seeing the honest party public keys cannot influence the selected sample other than by re-sampling polynomially many times.

Public-key PCFs

We start by building a public-key PCF that requires an SRS (Section 2.6.3). The SRS consists of an obfuscated program that, given a nonce and n parties' public encryption keys, uses a PRF to generate correlated randomness, and encrypts each party's random output to its public key. We can then eliminate the need for a pre-distributed SRS by instead using a distributed sampler to sample it (Section 2.6.4).

Public-key PCFs without random oracles. The proofs of security for the constructions sketched above only require polynomially many hybrids, roughly speaking because the random oracle allows the simulator to predict and control the inputs to the obfuscated programs. We can avoid the use of the random oracle, at the cost of going through exponentially many hybrids in the proof of security, and thus requiring sub-exponentially secure primitives.

Public-key PCFs for any correlation with a random oracle. Boyle *et al.* [BCG+19b] prove that a public-key PCF in the plain model that can handle *any* correlation (not just reverse-sampleable ones) must have keys at least as large as all the correlated randomness it yields. We observe that we can use a random oracle to sidestep this lower bound by deriving additional randomness from the oracle.

As a stepping stone, we introduce a different flavour of the distributed sampler, which we call the *reusable distributed universal sampler* (reusable DUS). It is *reusable* because it can be queried multiple times (without the need for additional communication), and it is *universal* because each query can produce a sample from a different distribution (specified by the querier). We build a reusable distributed universal sampler from a universal sampler, a random oracle and a distributed sampler (by using the distributed sampler to produce the universal sampler). Our last public-key PCF (Section 2.7) then uses the reusable distributed universal sampler to sample from a distribution that first picks the correlated randomness and then encrypts each party's share under her public key.

2.2 Preliminaries

Notation. We denote the security parameter by λ and the set $\{1, 2, \dots, m\}$ by $[m]$. Our constructions are designed for an ordered group of n parties $P_1, P_2, \dots, P_n$. We will denote the set of (indexes of) corrupted parties by C, whereas its complementary, the set of honest players, is H.

We indicate the probability of an event E by $\mathbb{P}[E]$. We use the term *noticeable* to refer to a non-negligible quantity. A probability p is instead *overwhelming* if $1-p$ is negligible. We say that a cryptographic primitive is sub-exponentially secure, if the advantage of the adversary is bounded by $2^{-\lambda^c}$ for some constant $c > 0$. When the advantage is negligible, we say that it is polynomially secure.

We use the simple arrow $\leftarrow$ to assign the output of a deterministic algorithm $\mathsf{Alg}(x)$ or a specific value a to a variable y, i.e. $y \leftarrow \mathsf{Alg}(x)$ or $y \leftarrow a$. If Alg is instead probabilistic, we write $y \overset{\$}{\leftarrow} \mathsf{Alg}(x)$ and we assume that the random tape is sampled uniformly. If the latter is set to a particular value r, we write however $y \leftarrow \mathsf{Alg}(x; r)$. We use $\overset{\$}{\leftarrow}$ also if we sample the value of y uniformly over a set X, i.e. $y \overset{\$}{\leftarrow} X$. Finally, we refer to algorithms having no input as distributions. The latter are in most cases parametrised by λ. The terms *circuit* and *program* are used interchangeably.

2.2.1 Indistinguishability Obfuscation

We recall the formal definition of indistinguishability obfuscation (iO) [BGI+01, GGH+13]. Informally speaking an obfuscator is an efficient algorithm that "scrambles" any given circuit Cr until it is impossible to extract any information about Cr except its original input-output behaviour. Furthermore, the result of the operation is another program computing exactly the same function as Cr. We provide a formal definition.

Definition 2.2.1 (Indistinguishability Obfuscator). Let $(\mathcal{L}_\lambda)_{\lambda\in\mathbb{N}}$ be a class of circuits such that every $\mathsf{Cr} \in \mathcal{L}_\lambda$ maps a $\mathsf{inp}(\lambda)$-bit input into a $\mathsf{out}(\lambda)$-bit output.

An indistinguishability obfuscator for $(\mathcal{L}_\lambda)_{\lambda\in\mathbb{N}}$ is a PPT algorithm iO with the following properties.

- **Correctness.** For every $\lambda \in \mathbb{N}$, circuit $\mathsf{Cr} \in \mathcal{L}_\lambda$ and input $x \in \{0,1\}^{\mathsf{inp}(\lambda)}$

$$\mathbb{P}\big[\,\mathsf{Cr}'(x) = \mathsf{Cr}(x) \mid \mathsf{Cr}' \overset{\$}{\leftarrow} \mathsf{iO}(1^\lambda, \mathsf{Cr})\,\big] = 1$$

- **Security.** For every pair of circuits $\mathsf{Cr}_0, \mathsf{Cr}_1 \in \mathcal{L}_\lambda$ such that $\mathsf{Cr}_0(x) = \mathsf{Cr}_1(x)$ for each $x \in \{0,1\}^{\mathsf{inp}(\lambda)}$, the distributions given by $\mathsf{iO}(1^\lambda, \mathsf{Cr}_0)$ and $\mathsf{iO}(1^\lambda, \mathsf{Cr}_1)$ are computationally indistinguishable.

Observe that the obfuscator is tailored to a specific class of circuits. The latter often affects the size of the obfuscated programs, increasing it as the class becomes larger.

The first candidate indistinguishability obfuscator was presented in 2013 by Garg *et al.* [GGH+13]. The construction relies on subexponentially secure primitives. Subsequent work has more and more weakened the requirements under which it is possible to build obfuscation. However, all known constructions still rely on sub-exponentially secure primitives or exponentially-many assumptions. Indistinguishability obfuscators not suffering from these problems are still purpose of research.

2.2.2 Puncturable PRFs

We recall the formal definition of puncturable PRF [KPTZ13, BW13, BGI14b]. A punturable PRF is a PRF construction F in which the keys K have an additional property: we can puncture them in any position x, obtaining a possibly longer key containing no information about $F_K(x)$ but still allowing computing $F_K(y)$ for every $y \neq x$.

Definition 2.2.2 (Puncturable PRF). A puncturable PRF with output space $(\mathcal{X}_\lambda)_{\lambda\in\mathbb{N}}$ is a pair of PPT algorithms (F, Punct) with the following properties.

- **Correctness.** For every $\lambda \in \mathbb{N}$, key $K \in \{0,1\}^\lambda$ and values $x, y \in \{0,1\}^*$ with $x \neq y$
$$\mathbb{P}\big[\, F_K(y) = F_{\hat{K}}(y) \mid \hat{K} \leftarrow \mathsf{Punct}(K, x)\,\big] = 1$$
- **Security.** For every value $x \in \{0,1\}^\lambda$, the following two distributions are indistinguishable.
$$\left\{ (\hat{K}, r) \;\middle|\; \begin{array}{l} K \xleftarrow{\$} \{0,1\}^\lambda \\ \hat{K} \leftarrow \mathsf{Punct}(K, x) \\ r \leftarrow F_K(x) \end{array} \right\} \qquad \left\{ (\hat{K}, r) \;\middle|\; \begin{array}{l} K \xleftarrow{\$} \{0,1\}^\lambda \\ \hat{K} \leftarrow \mathsf{Punct}(K, x) \\ r \xleftarrow{\$} \mathcal{X}_\lambda \end{array} \right\}$$

As noticed in [BW13], it easy to build puncturable PRFs by relying on the GGM construction [GGM86].

2.2.3 Simulation-Extractable NIZKs

We recall the formal definition of simulation-extractable NIZK [GO07]. Let $\mathcal{R}$ be an efficiently computable relation consisting of pairs (x, w), where x is called statement and w witness. As widely known, a NIZK for $\mathcal{R}$ is a construction that allows a party to prove the knowledge of a witness w for a statement x with only one message and without revealing any information about w itself (zero-knowledge). The procedure relies on a CRS, which, depending on the construction, can be structured or unstructured. Zero-knowledge is formulated by requiring the existence of two PPT simulators: the first one generates a fake CRS embedding a trapdoor τ into it, the second one leverages the knowledge of τ to produce valid proofs for various statements without needing the corresponding witnesses.

A simulation-extractable NIZK has also an additional property: there exists a PPT algorithm that, in conjunction with the two simulators, is able to extract the witness from any valid proof generated by the adversary. Such algorithm is called extractor and exploits the knowledge of the trapdoor in the CRS. We now formalise the syntax and the security properties of what we have just described.

Definition 2.2.3 (Non-Interactive Proof). A non-interactive proof for a relation $\mathcal{R}$ is a triple of PPT algorithms $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax.

- Gen takes as input the security parameter 1^λ and outputs a CRS.
- Prove takes as input the security parameter 1^λ, the CRS, a statement x and a witness w. The output is a proof π for x.
- Verify takes as input a CRS, a proof π and a statement x. The output is a bit indicating whether the proof has been accepted or not.

Definition 2.2.4 (Simulation-Extractable NIZK). A non-interactive proof $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ for the relation $\mathcal{R}$ is a simulation extractable NIZK if it satisfies the following properties.

- **Completeness.** For every $(x, w) \in \mathcal{R}$,
$$\mathbb{P}\left[\mathsf{Verify}(\mathsf{crs}, \pi, x) = 1 \;\middle|\; \begin{array}{l} \mathsf{crs} \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \\ \pi \xleftarrow{\$} \mathsf{Prove}(1^\lambda, \mathsf{crs}, x, w) \end{array} \right] = 1$$
- **Multi-Theorem Zero Knowledge.** There exist PPT simulators Sim_1 and Sim_2 such that, for every set of pairs $\{(x_i, w_i)\}_{i\in[m]}$ in $\mathcal{R}$, no PPT adversary is able to distinguish between the following distributions.
$$\left\{ \big(\mathsf{crs}, (x_i, \pi_i)_{i\in[m]}\big) \;\middle|\; \begin{array}{ll} \mathsf{crs} \xleftarrow{\$} \mathsf{Gen}(1^\lambda) & \\ \forall i \in [m]: & \pi_i \xleftarrow{\$} \mathsf{Prove}(1^\lambda, \mathsf{crs}, x_i, w_i) \end{array} \right\}$$
$$\left\{ \big(\mathsf{crs}, (x_i, \pi_i)_{i\in[m]}\big) \;\middle|\; \begin{array}{ll} (\mathsf{crs}, \tau) \xleftarrow{\$} \mathsf{Sim}_1(1^\lambda) & \\ \forall i \in [m]: & \pi_i \xleftarrow{\$} \mathsf{Sim}_2(\mathsf{crs}, \tau, x_i) \end{array} \right\}$$

- **Simulation-Extractability.** There exists a PPT extractor $\mathsf{Extract}$ such that, for every set of statements $\{x_i\}_{i\in[m]}$ and PPT adversary $\mathcal{A}$, we have that

$$\mathbb{P}\left[\begin{array}{c|l} \forall i \in [m] : \ (\pi', x') \neq (\pi_i, x_i) & (\mathsf{crs}, \tau) \xleftarrow{\$} \mathsf{Sim}_1(1^\lambda) \\ \mathsf{Verify}(\mathsf{crs}, \pi', x') = 1 & \forall i \in [m] : \quad \pi_i \xleftarrow{\$} \mathsf{Sim}_2(\mathsf{crs}, \tau, x_i) \\ (x', w') \notin \mathcal{R} & (x', \pi') \xleftarrow{\$} \mathcal{A}\big(1^\lambda, \mathsf{crs}, (x_i, \pi_i)_{i\in[m]}\big) \\ & w' \leftarrow \mathsf{Extract}(\mathsf{crs}, \tau, x', \pi') \end{array}\right] = \mathsf{negl}(\lambda)$$

Observe that simulation-extractability and zero-knowledge imply soundness.

NIZKs can be classified based on the type of CRS. When the latter is a uniform string of bits, we talk about NIZK with URS (uniform reference string). When the CRS is instead structured and possibly depends on secrets that must not be revealed to the parties, we talk about NIZKs with SRS (structured reference string).

2.2.4 MHE with Private Evaluation

We recall the definition of MHE with private evaluation [AJJM20]. This is a construction that permits any party P_i to encrypt a value x_i obtaining a ciphertext c_i and the corresponding partial decryption key sk_i. Once given the ciphertexts of the other parties, P_i can apply a circuit Cr on $(c_j)_{j\in[n]}$, obtaining the i-th partial decryption d_i by means of pk_i. By pooling the partial plaintexts $(d_j)_{j\in[n]}$, it is then possible to obtain $\mathsf{Cr}(x_1, \dots, x_n)$ without learning any additional information on the inputs. The construction differs from multi-key FHE as there actually exists no public key but only a private one that changes from ciphertext to ciphertext. Furthermore, the encryption algorithm needs to be provided with the parameters (input size, output size and depth) of the circuits we are going to apply on the ciphertexts. The final decryption needs instead to know the exact circuit that was used to produce the partial plaintext it is provided with. It is possible to build MHE schemes with private evaluation from polynomially secure LWE [AJJM20].

Definition 2.2.5 (Multiparty Homomorphic Encryption with Private Evaluation). A multiparty homomorphic encryption scheme with private evaluation (pMHE) is a triple of PPT algorithms $(\mathsf{Enc}, \mathsf{PrivEval}, \mathsf{FinDec})$ with the following syntax.

- Enc is a randomised algorithm that takes as input the security parameter 1^λ, the parameters of a circuit (input size, output size and depth), an index i and a message x_i. The output is a ciphertext c_i and a partial decryption key sk_i.

- $\mathsf{PrivEval}$ is a randomised algorithm. It takes as input a partial decryption key sk_i, the description of a circuit Cr mapping n inputs into a single output and n ciphertexts $c_1, c_2, \dots, c_n$. The output is a partial plaintext d_i.

- FinDec is a deterministic algorithm taking as input a circuit Cr and the corresponding n partial decryptions $d_1, d_2, \dots, d_n$. The output is a plaintext d.

Definition 2.2.6 (Reusable Semi-Malicious Security). We say that a pMHE scheme satisfies *reusable semi-malicious security* if it satisfies the following properties.

- **Correctness.** For every security parameter $\lambda \in \mathbb{N}$, circuit Cr mapping n inputs into one output and inputs $x_1, x_2, \dots, x_n$, we have

$$\mathbb{P}\left[\begin{array}{c|l} & \forall i \in [n] : \quad (c_i, \mathsf{sk}_i) \xleftarrow{\$} \mathsf{Enc}(1^\lambda, \mathsf{Cr.params}, i, x_i) \\ d = \mathsf{res} & \forall i \in [n] : \quad d_i \xleftarrow{\$} \mathsf{Eval}(\mathsf{sk}_i, \mathsf{Cr}, c_1, c_2, \dots, c_n) \\ & d \leftarrow \mathsf{FinDec}(\mathsf{Cr}, d_1, d_2, \dots, d_n) \\ & \mathsf{res} \leftarrow \mathsf{Cr}(x_1, x_2, \dots, x_n) \end{array}\right] = 1$$

$\mathcal{G}^{C,\mathsf{Cr},(x_h)_{h\in H}}_{\text{pMHE}}(\lambda)$

Initialisation.

1. $b \xleftarrow{\$} \{0,1\}$
2. $\forall h \in H: \quad (c_h^0, \mathsf{sk}_h^0) \xleftarrow{\$} \mathsf{Enc}(1^\lambda, \mathsf{Cr.params}, h, x_h)$
3. $\big(\tau, (c_h^1)_{h\in H}\big) \xleftarrow{\$} \mathsf{Sim}_1(1^\lambda, H, \mathsf{Cr.params})$
4. Activate the adversary with input $\big(1^\lambda, (c_h^b)_{h\in H}\big)$.
5. Receive the inputs $(x_i)_{i\in C}$ and the randomness $(r_i)_{i\in C}$ of the corrupted parties from the adversary.
6. $\forall i \in C: \quad (c_i^0, \mathsf{sk}_i^0) \leftarrow \mathsf{Enc}(1^\lambda, \mathsf{Cr.params}, i, x_i; r_i)$

Decryption Queries. On input $(\mathsf{Decrypt}, \mathsf{Cr}')$ where $\mathsf{Cr}'.\mathsf{params} = \mathsf{Cr.params}$.

1. $\forall h \in H: \quad d_h^0 \xleftarrow{\$} \mathsf{PrivEval}(\mathsf{sk}_h, \mathsf{Cr}', c_1^0, c_2^0, \ldots, c_n^0)$
2. $\big(\tau, (d_h^1)_{h\in H}\big) \xleftarrow{\$} \mathsf{Sim}_2\big(\tau, \mathsf{Cr}', \mathsf{Cr}'(x_1, x_2, \ldots, x_n), (x_i, r_i)_{i\in C}\big)$
3. Reply with $(d_h^b)_{h\in H}$.

Output. The adversary wins if the final output is b.

Figure 2.3: The pMHE Security Game

- **Reusable Semi-Malicious Security.** There exist PPT simulators Sim_1 and Sim_2 such that, for every $n \in \mathbb{N}$, set of corrupted parties C, circuit Cr, inputs of the honest parties $(x_h)_{h\in H}$, no PPT adversary is able to win the game $\mathcal{G}^{C,\mathsf{Cr},(x_h)_{h\in H}}_{\text{pMHE}}(\lambda)$ (see Figure 2.3) with noticeable advantage.

Observe that correctness states that if we evaluate a circuit Cr over encrypted values $x_1, x_2, \ldots, x_n$ and we pool the partial decryptions, we obtain $C(x_1, x_2, \ldots, x_n)$. Security instead declares that if we publish the encryption of a value x_i and we later on disclose the partial plaintext corresponding to a certain circuit Cr, we reveal nothing more than the output of Cr.

2.2.5 Somewhere Statistically Binding Hash Functions

We recall the formal definition of somewhere statistically binding hashing (SSB) [HW15], which can be regarded as a more powerful, obfuscation-friendly version of hash functions. The notion was first introduced by Hubáček and Wichs in [HW15] and can be constructed from FHE [HW15].

The primitive is composed of two algorithms Gen and Hash. The first one is used to generate a key hiding a special index i. The second one instead produces a digest after receiving the key and a message as inputs. The interesting property of SSB hashing is that there exists no pair of messages with different i-th block but same digest. We provide now formal definitions.

Definition 2.2.7 (Somewhere Statistically Binding Hash Function)*.* A somewhere statistically binding hash function (SSB) with block alphabet Σ is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Hash})$ with the following syntax.

- Gen is a randomised procedure taking as input the security parameter 1^λ, the maximum number of blocks B and an index $l \in [B]$. The output is a hash key hk.
- Hash is a deterministic procedure taking as input a hash key hk and a message $x \in \Sigma^B$ where B is the maximum number of blocks supported by hk. The output is a digest y.

Definition 2.2.8 (Security of SSB Hashing)*.* A somewhere statistically binding hash function is secure if it satisfies the following properties.

- **Index Hiding.** For every $B \in \mathbb{N}$ and $i, j \in [B]$, no PPT adversary can distinguish between $\mathsf{Gen}(1^\lambda, B, i)$ and $\mathsf{Gen}(1^\lambda, B, j)$.

- **Somewhere Statistically Binding.** For every $B \in \mathbb{N}$ and $i \in [B]$,

$$\mathbb{P}\left[\begin{array}{c} \exists\, x, x' \in \Sigma^B \\ x_i \neq x'_i \\ \mathsf{Hash}(\mathsf{hk}, x) = \mathsf{Hash}(\mathsf{hk}, x') \end{array} \middle| \mathsf{hk} \xleftarrow{\$} \mathsf{Gen}(1^\lambda, B, i) \right] = \mathsf{negl}(\lambda).$$

Observe that the SSB property, in conjunction with index hiding, implies the collision resistance of the hash function.

The definition described above is actually weaker than the original one. In [HW15], an SSB hash function featured two additional algorithms Prove and Verify. The first one was used to generate short openings of the j-th preimage block of a digest, for any $j \in [B]$. The second one was used to verify them. These two algorithms are not used in this work.

2.2.6 Universal Samplers

We recall the definitions of universal sampler (US) [HJK+16], describing the syntax and the various security notions. Informally speaking, a US is a trusted-dealer-based construction that permits to deterministically derive samples from any distribution, learning no information about the corresponding randomness. This primitive is fundamental for the construction of our distributed universal samplers (see Section 2.7.1 and Section 2.7.1).

Definition 2.2.9 (Universal Sampler). Let $\ell(\lambda)$, $r(\lambda)$ and $t(\lambda)$ be polynomials. A universal sampler for (ℓ, r, t)-distributions is a pair of PPT algorithms $(\mathsf{Setup}, \mathsf{Sample})$ with the following syntax.

1. Setup is a randomised algorithm taking as input the security parameter 1^λ and outputting a sampler U.

2. Sample is a deterministic procedure possibly interacting with a random oracle $\mathcal{H}$. It takes as input a sampler U and the description $\mathcal{D}$ of an (ℓ, r, t)-distribution, outputting a sample R.

Observe that in the above definition, we ask that Setup does not interact with any random oracle. If that was not the case, it would be impossible to generate a universal sampler inside an obfuscated program. The original definition of US [HJK+16] was not as restrictive. However, all the constructions presented in [HJK+16] satisfy the property described above, including those on which we rely in this work.

One-time universal samplers. We now formalise the weaker security notion of the primitive. In particular, we require that the samples look random only for one specific distribution selected ahead of time. In [HJK+16], the authors present a construction satisfying this definition without random oracle.

Definition 2.2.10 (One-Time Universal Sampler). A universal sampler $(\mathsf{Setup}, \mathsf{Sample})$ for (ℓ, r, t)-distributions satisfies one-time security if there exists a PPT simulator Sim such that, for every (ℓ, r, t)-distribution $\mathcal{D}$, the following two distributions are computationally indistinguishable

$$\left\{ U, R \,\middle|\, \begin{array}{l} U \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ R \leftarrow \mathsf{Sample}(U, \mathcal{D}) \end{array} \right\} \qquad \left\{ U, R \,\middle|\, \begin{array}{l} R \xleftarrow{\$} \mathcal{D} \\ U \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \mathcal{D}, R) \end{array} \right\}$$

Notice that security states that one-time USs can be programmed to output ideal samples from the distribution $\mathcal{D}$, without the adversary noticing it.

Adaptively secure universal samplers. We present the stronger security notion of universal samplers. We now ask that the samples look random for every distribution adaptively chosen by the adversary. In particular, we do not care only about one distribution, but multiple ones. This definition can only be satisfied in the random oracle model. Hofheinz *et al.* presented an example of such construction in [HJK+16].

$\mathcal{G}_{\mathrm{US}}(\lambda)$

Initialisation. The challenger samples a random bit $b \xleftarrow{\$} \{0,1\}$ and computes $U_0 \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$. Then, it instantiates a random oracle $\mathcal{H}$ and a sampling oracle $\mathcal{O}$. Upon receiving an (ℓ, r, t)-distribution $\mathcal{D}$, the latter replies with $\mathcal{D}(F(\mathcal{D}))$ where F is a truly random function outputting $r(\lambda)$ bits. Finally, the challenger computes $(\tau, U_1) \xleftarrow{\$} \mathsf{SimGen}^{\mathcal{O}}(1^\lambda)$ and provides the adversary with U_b.
Oracle query. Upon receiving any query (RO, x) from the adversary, the challenger computes $r_0 \leftarrow \mathcal{H}(x)$ and $(\tau, r_1) \xleftarrow{\$} \mathsf{SimRO}^{\mathcal{O}}(1^\lambda, x, \tau)$. It replies with r_b.
Sample. Upon receiving any query $(\mathsf{Sample}, \mathcal{D})$, the challenger computes $R_0 \leftarrow \mathsf{Sample}^{\mathcal{H}}(U_0, \mathcal{D})$ and $R_1 \leftarrow \mathcal{O}(\mathcal{D})$. Then, it replies with R_b.
Output. The adversary wins if its output is equal to b.

Figure 2.4: The Universal Sampler Game

Definition 2.2.11 (Adaptively Secure Universal Sampler). A universal sampler $(\mathsf{Setup}, \mathsf{Sample})$ for (ℓ, r, t)-distributions satisfies adaptive security if there exist PPT simulators SimGen and SimRO such that no PPT adversary can win the game $\mathcal{G}_{\mathrm{US}}(\lambda)$ (see Figure 2.4) with noticeable advantage.

As for the one-time case, security states that the sampler U and the random oracle $\mathcal{H}$ could be programmed to output ideal samples, without the adversary noticing it. The only difference is that now the adversary can adaptively choose the distribution multiple times.

2.3 Defining Distributed Samplers

Informally speaking, a distributed sampler (DS) for the distribution $\mathcal{D}$ is a construction that allows n parties to obtain a random sample R from $\mathcal{D}$ with just one round of communication and without revealing any additional information about the randomness used for the generation of R. The output of the procedure can be derived given only the public transcript, so we do not aim to protect the privacy of the result against passive adversaries eavesdropping the communications between the parties.

If we assume an arbitrary trusted setup, building a DS becomes straightforward; we can consider the trivial setup that directly provides the parties with a random sample from $\mathcal{D}$. Obtaining solutions with a weaker (or no) trusted setup is much more challenging.

The structure and syntax of distributed samplers is formalised as follows. We then analyse different flavours of security definitions.

Definition 2.3.1 (n-party Distributed Sampler for the Distribution $\mathcal{D}$). An n-party distributed sampler for the distribution $\mathcal{D}$ is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Sample})$ with the following syntax:

1. Gen is a probabilistic algorithm taking as input the security parameter 1^λ and a party index $i \in [n]$ and outputting a sampler share U_i for party i. Let $\{0,1\}^{L(\lambda)}$ be the space from which the randomness of the algorithm is sampled.

2. Sample is a deterministic algorithm taking as input n shares of the sampler $U_1, U_2, \ldots, U_n$ and outputting a sample R.

In some of our security definitions, we will refer to the *one-round protocol* Π_{DS} that is induced by the distributed sampler $\mathsf{DS} = (\mathsf{Gen}, \mathsf{Sample})$. This is the natural protocol obtained from DS, where each party first broadcasts a message output by Gen, and then runs Sample on input all the parties' messages.

2.3.1 Security

In this section we formalise the definition of distributed samplers with relation to different security flavours, namely, semi-malicious and active. We always assume that we deal with a static adversary who can corrupt

$\mathcal{F}_{\mathcal{D}}$

Initialisation. On input Init from every honest party and the adversary, the functionality activates and sets $Q := \emptyset$. (Q will be used to keep track of queries.) If all the parties are honest, the functionality outputs $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$ to every honest party and sends R to the adversary, then it halts.

Query. On input Query from the adversary, the functionality samples $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$ and creates a fresh label id. It sends (id, R) to the adversary and adds the pair to Q.

Output. On input $(\mathsf{Output}, \widehat{\mathsf{id}})$ from the adversary, the functionality retrieves the only pair $(\mathsf{id}, R) \in Q$ with $\mathsf{id} = \widehat{\mathsf{id}}$. If such pair does not exist, the functionality does nothing. Otherwise, it outputs R to every honest party and terminates.

Abort. On input Abort from the adversary, the functionality outputs $\perp$ to every honest party and terminates.

Figure 2.5: Distributed Sampler Functionality

up to $n-1$ out of the n parties. We recall that a protocol has semi-malicious security if it remains secure even if the corrupt parties behave semi-honestly, but the adversary can select their random tapes.

Definition 2.3.2 (Distributed Sampler with Semi-Malicious Security). A distributed sampler $(\mathsf{Gen}, \mathsf{Sample})$ has *semi-malicious security* if there exists a PPT simulator Sim such that, for every set of corrupt parties $C \subsetneq [n]$ and corresponding randomness $(\rho_i)_{i \in C}$, the following two distributions are computationally indistinguishable:

$$\left\{ \begin{array}{c} (U_i)_{i\in[n]} \\ (\rho_i)_{i\in C}, R \end{array} \middle| \begin{array}{ll} \rho_i \xleftarrow{\$} \{0,1\}^{L(\lambda)} & \forall i \in H \\ U_i \leftarrow \mathsf{Gen}(1^\lambda, i;\, \rho_i) & \forall i \in [n] \\ R \leftarrow \mathsf{Sample}(U_1, U_2, \dots, U_n) & \end{array} \right\} \qquad \text{and}$$

$$\left\{ \begin{array}{c} (U_i)_{i\in[n]} \\ (\rho_i)_{i\in C}, R \end{array} \middle| \begin{array}{l} R \xleftarrow{\$} \mathcal{D}(1^\lambda) \\ (U_i)_{i\in H} \xleftarrow{\$} \mathsf{Sim}\big(1^\lambda, C, R, (\rho_i)_{i\in C}\big) \end{array} \right\}$$

Observe that this definition implies that, even in the simulation, the relation

$$R = \mathsf{Sample}(U_1, U_2, \dots, U_n)$$

holds with overwhelming probability. In other words, security requires that $(\mathsf{Gen}, \mathsf{Sample})$ securely implements the functionality that samples from $\mathcal{D}$ and outputs the result to all of the parties.

Observe that the previous definition can be adapted to passive security by simply sampling the randomness of the corrupted parties inside the game in the real world and generating it using the simulator in the ideal world.

We now define actively secure distributed samplers. Here, to handle the challenges introduced by a rushing adversary, we model security by defining an ideal functionality in the universal composability (UC) framework [Can01], and require that the protocol Π_{DS} securely implements this functionality.

Definition 2.3.3 (Distributed Sampler with Active Security). Let $\mathsf{DS} = (\mathsf{Gen}, \mathsf{Sample})$ be a distributed sampler for the distribution $\mathcal{D}$. We say that DS has *active security* if the one-round protocol Π_{DS} securely implements the functionality $\mathcal{F}_{\mathcal{D}}$ (see Figure 2.5) against a static and active adversary corrupting up to $n-1$ parties.

Remark 2.3.4 (Distributed Samplers with a CRS or Random Oracle). Our constructions with active security rely on a setup assumption in the form of a common reference string (CRS) and random oracle. For a CRS, we assume the algorithms $\mathsf{Gen}, \mathsf{Sample}$ are implicitly given the CRS as input, which is modelled as being sampled by an ideal setup functionality. As usual, the random oracle is modelled as an external oracle that may be queried by any algorithm or party, and programmed by the simulator in the security proof.

Observe that this definition allows the adversary to request several samples R from the functionality, and then select the one it likes the most. Our definition must allow this in order to deal with a rushing adversary who might wait for the messages $(U_i)_{i\in H}$ of all the honest parties and then locally re-generate the corrupt parties' messages $(U_i)_{i\in C}$, obtaining a wide range of possible outputs. Finally, it can broadcast the corrupt parties' messages that lead to the output it likes the most. This makes distributed samplers with active security rather useless when the distribution $\mathcal{D}$ has low entropy, i.e. when there exists a polynomial-size set S such that $\mathcal{D}(1^\lambda) \in S$ with overwhelming probability. Indeed, in such cases, the adversary is able to select its favourite element in the image of $\mathcal{D}$.

On the usefulness of distributed samplers with a CRS. Our distributed samplers with active security require a CRS for NIZK proofs. Since one of the main goals of the construction is avoid trusted setup in multiparty protocols, assuming the existence of a CRS, which itself is some form of setup, may seem wrong.

We highlight, however, that some types of CRS are much easier to generate than others. A CRS that depends on values which must remain secret (e.g. an RSA modulus with unknown factorization, or an obfuscated program which contains a secret key) is difficult to generate. However, assuming the security of trapdoor permutations [FLS90], bilinear maps [GOS06], learning with errors [PS19] or indistinguishability obfuscation [BP15], we can construct NIZK proofs where the CRS is just a random string of bits, i.e. a URS. In the random oracle model, such a CRS can even be generated without any interaction. So, the CRS required by our constructions is the simplest, weakest kind of CRS setup.

2.4 A Construction with Semi-Malicious Security

We now present the main construction of this paper: a distributed sampler with semi-malicious security based on polynomially secure MHE with private evaluation and indistinguishability obfuscation. In Section 2.5, we explain how to upgrade this construction to achieve active security.

The basic idea. Our goal is to generate a random sample R from the distribution $\mathcal{D}$. The natural way to do it is to produce a random bit string s and feed it into $\mathcal{D}$. We want to perform the operation in an encrypted way as we need to preserve the privacy of s. A DS implements the functionality that provides samples from the underlying distribution, but not the randomness used to obtain them, so no information about s can be leaked.

We guarantee that any adversary corrupting up to $n-1$ parties is not able to influence the choice of s by XORing n bit strings of the same length, the i-th one of which is independently sampled by the i-th party P_i. Observe that we are dealing with a semi-malicious adversary, so we do not need to worry about corrupted parties adaptively choosing their shares after seeing those of the honest players.

Preserving the privacy of the random string. To preserve the privacy of s, we rely on an MHE scheme with private evaluation $\mathsf{pMHE} = (\mathsf{Enc}, \mathsf{PrivEval}, \mathsf{FinDec})$. Each party P_i encrypts s_i, publishing the corresponding ciphertext c_i and keeping the private key sk_i secret. As long as the honest players do not reveal their partial decryption keys, the privacy of the random string s is preserved. Using the homomorphic properties of the MHE scheme, the parties are also able to obtain partial plaintexts of R without any interactions. However, we run into an issue: in order to finalise the decryption, the construction would require an additional round of communication where the partial plaintexts are broadcast.

Reverting to a one-round construction. We need to find a way to perform the final decryption without additional interaction, while at the same time preserving the privacy of the random string s. That means revealing a very limited amount of information about the private keys $\mathsf{sk}_1, \mathsf{sk}_2, \dots, \mathsf{sk}_n$, so that it is only possible to retrieve R, revealing nothing more.

Inspired by [HIJ+17], we build such a precise tool by relying on indistinguishability obfuscation: in the only round of interaction, each party P_i additionally publishes an obfuscated *evaluation program* EP_i containing the private key sk_i. When given the ciphertexts of the other parties, EP_i evaluates the circuit producing the final result R and outputs the partial decryption with relation to sk_i. Using the evaluation programs, the players are thus able to retrieve R by feeding the partial plaintexts into $\mathsf{pMHE.FinDec}$.

Dealing with the leakage about the secret keys. At first glance, the solution outlined in the previous paragraph seems to be secure. However, there are some sneaky issues we need to deal with.

In this warm-up construction, we aim to protect the privacy of the random string s by means of the reusable semi-malicious security of the MHE scheme with private evaluation. To rely on this assumption, no information on the secret keys must be leaked. However, this is not the case here, as the private keys are part of the evaluation programs.

In the security proof, we are therefore forced to proceed in two steps: first, we must remove the secret keys from the programs using obfuscation, and then we can apply reusable semi-malicious security. The first task is actually trickier than it may seem. iO states we cannot distinguish between the obfuscation of two equivalent programs. Finding a program with the same input-output behaviour as EP_i without it containing any information about sk_i is actually impossible, as any output of the program depends on the private key. We cannot even hard-code the partial decryptions under sk_i for all possible inputs into the obfuscated program as that would require storing an exponential amount of information, blowing up the size of EP_i.

In [HIJ+17], while constructing an NI-MPC protocol based on multi-key FHE and iO, the authors deal with an analogous issue by progressively changing the behaviour of the program input by input, first hard-coding the output corresponding to a specific input and then using the simulatability of partial decryptions to remove any dependency on the multi-key FHE secret key. Unfortunately, in our context, this approach raises additional problems. First of all, in contrast with some multi-key FHE definitions, MHE does not support simulatability of partial decryptions. Additionally, since the procedure of [HIJ+17] is applied input by input, the security proof would require exponentially many hybrids. In that case, security can be argued only if transitions between subsequent hybrids cause a subexponentially small increase in the adversary's advantage. In other words, we would need to rely on subexponentially secure primitives even if future research shows that iO does not. Finally, we would still allow the adversary to compute several outputs without changing the random strings $(s_h)_{h\in H}$ selected by the honest parties. Each of the obtained values leaks some additional information about the final output of the distributed sampler. In [HIJ+17], this fact did not constitute an issue as this type of leakage is intrinsically connected to the notion of NI-MPC.

Bounding the leakage: key generation programs. To avoid the problems described above, we introduce the idea of *key generation programs*. Each party P_i publishes an obfuscated program KGP_i which encrypts a freshly chosen string s_i, keeping the corresponding partial decryption key secret.

The randomness used by KGP_i is produced via a puncturable PRF F taking as a nonce the key generation programs of the other parties. In this way, any slight change in the programs of the other parties leads to a completely unrelated string s_i, ciphertext c_i and key sk_i. It is therefore possible to protect the privacy of s_i using a polynomial number of hybrids, as we need only worry about a single combination of inputs. Specifically, we can remove any information about sk_i from EP_i and hard-code the partial plaintext d_i corresponding to $(c_j)_{j\in[n]}$. At that point, we can rely on the reusable semi-malicious security of the MHE scheme with private evaluation, removing any information about s_i from c_i and d_i and programming the final output to be a random sample R from $\mathcal{D}$.

The introduction of the key generation programs requires minimal modifications to the evaluation programs. In order to retrieve the MHE private key, EP_i needs to know the same PRF key K_i used by KGP_i. Moreover, it now takes as input the key generation programs of the other parties, from which it will derive the MHE ciphertexts needed for the computation of R. Observe that EP_i will also contain KGP_i, which will be fed into the other key generation programs in a nested execution of obfuscated circuits.

Compressing the inputs. The only problem with the construction above, is that we now have a circularity issue: we cannot actually feed one key generation program as input to another key generation program, since the programs are of the same size. This holds even if we relied on obfuscation for Turing machines, since to prove security, we would need to puncture the PRF keys in the nonces, i.e. the key generation programs of the other parties. The point at which the i-th key is punctured, which is at least as big as the program itself, must be hard-coded into KGP_i, which is clearly too small.

Instead of feeding entire key generation programs into KGP_i, we can input their hash, which is much smaller. This of course means that there now exist different combinations of key generation programs leading to the same MHE ciphertext-key pair (c_i, sk_i), and the adversary could try to extract information about sk_i by looking for collisions. The security of the hash function should, however, prevent this attack. The only issue is that iO does not really get along with this kind of argument based on collision-resistant hashing. We instead rely on the more iO-friendly notion of a *somewhere statistically binding* hash function $\mathsf{Hash} = (\mathsf{Gen}, \mathsf{Hash})$ [HW15].

Final construction. We now present the formal description of our semi-maliciously secure DS. The algorithms Gen and Sample, as well as the unobfuscated key generation program $\mathcal{P}_{\mathsf{KG}}$ and evaluation program $\mathcal{P}_{\mathsf{Eval}}$, can be found in Figure 2.6. In the description, we assume that the puncturable PRF F outputs pseudorandom strings (r_1, r_2, r_3) where each of r_1, r_2 and r_3 is as long as the randomness needed by $\mathcal{D}$, $\mathsf{pMHE.Enc}$, and $\mathsf{HE.PrivEval}$ respectively. Moreover, we denote by B the maximum number of blocks in the messages fed into $\mathsf{Hash.Hash}$.

Theorem 2.4.1. If $\mathsf{Hash} = (\mathsf{Gen}, \mathsf{Hash})$ is a somewhere statistically binding hash function, $\mathsf{pMHE} = (\mathsf{Enc}, \mathsf{PrivEval}, \mathsf{FinDec})$ is a MHE scheme with private evaluation, iO is an indistinguishability obfuscator and (F, Punct) is a puncturable PRF, the construction in Figure 2.6 is an n-party distributed sampler with semi-malicious security for the distribution $\mathcal{D}$.

Proof. We prove the security of the construction in Figure 2.6 in a sequence of hybrids. In the initial hybrid (hybrid 0), we start with the real game; in the last hybrid (hybrid 6), we produce a simulated sampler share on behalf of one of the honest parties, which leads the parties to output an R sampled at random from $\mathcal{D}$. We choose one honest party h, and throughout hybrids $1 - 6$, we modify only how the sampler share U_h is produced. The rest of the honest parties continue to produce their sampler shares as per the Gen protocol in Figure 2.6.

Because the simulator has access to all parties' random tapes, it of course knows all their secrets. In the following, we refer to $U_i = (\mathsf{hk}_i, \mathsf{KGP}_i, \mathsf{EP}_i)$ as the sampler share produced by party i for $i \neq h$. We also refer to the secret keys K_i contained in those programs (also known to the simulator). For whatever values $U_h = (\mathsf{hk}_h, \mathsf{KGP}_h, \mathsf{EP}_h)$ the simulator produces on behalf of party h in a given hybrid, we let $\hat{s}_i$ denote the share of the randomness generated by KGP_i (on the appropriate nonce y), and $(\hat{c}_i, \hat{\mathsf{sk}}_i)$ denote the encryption of that randomness and the corresponding partial decryption key produced by KGP_i. Finally, we let $\hat{r}_2^i$ denote the random string input in $\mathsf{pMHE.Enc}$ by KGP_i for the generation of $\hat{c}_i$.

Hybrid 0: This is the initial hybrid, where the simulator, on behalf of every honest party i, generates a sampler share U_i as per the Gen algorithm in Figure 2.6.

Hybrid 1: In this hybrid, the simulator, on behalf of honest party h, punctures the key K_h at the relevant point (the hash of $(\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq h}$ produced by the other parties j), but programs the appropriate output at that point into the programs. By the correctness of F puncturing, the input-output behaviour of both programs is the same as it was in the previous hybrid. Therefore, by the security of iO, this hybrid is indistinguishable from the previous one.

More specifically, let $(\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq h}$ be the hash key and obfuscated key generation program of every other party. (The simulator knows these — even for corrupt parties — since she gets to see the randomness tape of corrupt parties in the definition of semi-malicious security.) The simulator does the following during Gen on behalf of party h, where the text in red indicates what changed since the previous hybrid:

1. $K_h \xleftarrow{\$} \{0,1\}^\lambda$
2. $\mathsf{hk}_h \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda, B, 0)$
3. $\hat{y}_h \leftarrow \mathsf{Hash.Hash}\big(\mathsf{hk}_h, (\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq h}\big)$

Distributed Sampler with Semi-Malicious Security

```
$\mathsf{Gen}(1^\lambda, i)$ :
    1. $K \xleftarrow{\$} \{0,1\}^\lambda$
    2. $\mathsf{hk} \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda, B, 0)$
    3. $\mathsf{KGP} \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}_{\mathsf{KG}}[K, i]\big)$
    4. $\mathsf{EP} \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}_{\mathsf{Eval}}[K, i, \mathsf{hk}, \mathsf{KGP}]\big)$
    5. Output $U := (\mathsf{hk}, \mathsf{KGP}, \mathsf{EP})$.
$\mathsf{Sample}\big((U_i = (\mathsf{hk}_i, \mathsf{KGP}_i, \mathsf{EP}_i))_{i \in [n]}\big)$ :
    1. $\forall i \in [n]: \quad d_i \leftarrow \mathsf{EP}_i\big((\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq i}\big)$
    2. Output $R \leftarrow \mathsf{pMHE.FinDec}\big(\tilde{\mathcal{D}}, (d_i)_{i \in [n]}\big)$
```

The algorithm $\tilde{\mathcal{D}}$.
Given a set of n random strings $s_1, s_2, \ldots, s_n$, perform the following operations.

```
    1. $s \leftarrow s_1 \oplus s_2 \oplus \cdots \oplus s_n$
    2. Output $R \leftarrow \mathcal{D}(1^\lambda; s)$
```

$\mathcal{P}_{\mathsf{KG}}[K, i]$: the key generation program

Hard-coded. The private key K and the index i of the party.
Input. A hash y.

```
    1. $(r_1, r_2, r_3) \leftarrow F_K(y)$
    2. $s \leftarrow r_1$
    3. $(c, \mathsf{sk}) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, \tilde{\mathcal{D}}.\mathsf{params}, i, s;\, r_2)$
    4. Output $c$.
```

$\mathcal{P}_{\mathsf{Eval}}[K, i, \mathsf{hk}_i, \mathsf{KGP}_i]$: the evaluation program

Hard-coded. The private key K, the index i of the party, the hash key hk_i, and the obfuscated key generation program KGP_i.
Input. A set of $n-1$ pairs $(\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq i}$ where the first element is a hash key and the second is an obfuscated key generation program.

```
    1. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash.Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{KGP}_l)_{l \neq j}\big)$
    2. $\forall j \neq i: \quad c_j \leftarrow \mathsf{KGP}_j(y_j)$
    3. $(r_1, r_2, r_3) \leftarrow F_K(y_i)$
    4. $s_i \leftarrow r_1$
    5. $(c_i, \mathsf{sk}_i) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, \tilde{\mathcal{D}}.\mathsf{params}, i, s_i;\, r_2)$
    6. $d_i \leftarrow \mathsf{pMHE.PrivEval}(\mathsf{sk}_i, \tilde{\mathcal{D}}, c_1, c_2, \ldots, c_n;\, r_3)$
    7. Output $d_i$.
```

Figure 2.6: A Distributed Sampler with Semi-Malicious Security

```
$\mathcal{P}^1_{\mathsf{KG}}[K, i, \hat{y}, \hat{c}]$

Hard-coded. The private key $K$, the index $i$ of the party, as well as a nonce $\hat{y}$, and a ciphertext $\hat{c}$.
Input. A nonce $y$.

1. If $y = \hat{y}$, output $\hat{c}$.
2. Otherwise, continue executing $\mathcal{P}_{\mathsf{KG}}[K]$:
3. $(r_1, r_2, r_3) \leftarrow F_K(y)$
4. $s \leftarrow r_1$
5. $(c, \mathsf{sk}) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, \tilde{\mathcal{D}}.\mathsf{params}, i, s;\ r_2)$
6. Output $c$.
```

Figure 2.7: The Key Generation Program

4. $\hat{K}_h \leftarrow \mathsf{Punct}(K_h, \hat{y}_h)$
5. $(\hat{s}_h, \hat{r}_2, \hat{r}_3) \leftarrow F_{K_h}(\hat{y}_h)$
6. $(\hat{c}_h, \hat{\mathsf{sk}}_h) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, \tilde{\mathcal{D}}.\mathsf{params}, h, \hat{s}_h;\ \hat{r}_2)$
7. $\mathsf{KGP}_h \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}^1_{\mathsf{KG}}[\hat{K}_h, h, \hat{y}_h, \hat{c}_h]\big)$ (see Figure 2.7)
8. $\mathsf{EP}_h \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}^1_{\mathsf{Eval}}[\hat{K}_h, h, \mathsf{hk}_h, \mathsf{KGP}_h, \hat{y}_h, \hat{c}_h, \hat{\mathsf{sk}}_h, \hat{r}_3]\big)$ (see Figure 2.8)
9. Output $U_h := (\mathsf{hk}_h, \mathsf{KGP}_h, \mathsf{EP}_h)$.

Next, for every l from 0 to the length of the input to $\mathsf{Hash.Hash}$, we proceed first to Hybrid 2.l.1, then to Hybrid 2.l.2.

Hybrid 2.l.1: In this hybrid, the simulator, on behalf of honest party h, makes the hash key hk_h statistically binding at index l (whereas before it was statistically binding at index $l-1$). This hybrid is indistinguishable from the previous one by the index hiding property of Hash.

Hybrid 2.l.2: In this hybrid, the simulator, on behalf of honest party h, changes the evaluation program EP_h to only use the hardcoded key and ciphertext if the first l blocks of the input coincide with a hardcoded reference input. (The simulated party h now obfuscates $\mathcal{P}^2_{\mathsf{Eval}}$ (Figure 2.9) instead of $\mathcal{P}^1_{\mathsf{Eval}}$ (Figure 2.8).) By the fact that $\mathsf{Hash.Hash}$ is statistically binding at l, the input-output behaviour of both programs is the same as it was in the previous hybrid. Therefore, by the security of iO, this hybrid is indistinguishable from the previous one.

More specifically, the simulator does the following during Gen on behalf of party h, where the text in red indicates what changed since the previous hybrid:

1. $K_h \xleftarrow{\$} \{0,1\}^\lambda$
2. $\mathsf{hk}_h \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda, B, l)$
3. $\hat{y}_h \leftarrow \mathsf{Hash.Hash}\big(\mathsf{hk}_h, (\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq h}\big)$
4. $\hat{K}_h \leftarrow \mathsf{Punct}(K_h, \hat{y}_h)$
5. $(\hat{s}_h, \hat{r}_2, \hat{r}_3) \leftarrow F_{K_h}(\hat{y}_h)$
6. $(\hat{c}_h, \hat{\mathsf{sk}}_h) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, \tilde{\mathcal{D}}.\mathsf{params}, h, \hat{s}_h;\ \hat{r}_2)$

$\mathcal{P}^1_{\mathsf{Eval}}[K, i, \mathsf{hk}_i, \mathsf{KGP}_i, \hat{y}, \hat{c}, \hat{\mathsf{sk}}, \hat{r}]$

Hard-coded. The private key K, the index i of the party, the hash key hk_i, the obfuscated key generation program KGP_i, as well as a nonce $\hat{y}$, a ciphertext $\hat{c}$, a secret key $\hat{\mathsf{sk}}$, and randomness $\hat{r}$.
Input. A set of $n-1$ pairs $(\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq i}$ where the first element is a hash key and the second is an obfuscated key generation program.

```
1. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash.Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{KGP}_l)_{l \neq j}\big)$
2. $\forall j \neq i: \quad c_j \leftarrow \mathsf{KGP}_j(y_j)$
3. If $y_i = \hat{y}$, set $\mathsf{sk}_i \leftarrow \hat{\mathsf{sk}}$, $c_i \leftarrow \hat{c}$ and $r_3 \leftarrow \hat{r}$.
4. Otherwise,
    (a) $(r_1, r_2, r_3) \leftarrow F_K(y_i)$
    (b) $s_i \leftarrow r_1$
    (c) $(c_i, \mathsf{sk}_i) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, \tilde{\mathcal{D}}.\mathsf{params}, i, s_i;\, r_2)$
5. $d_i \leftarrow \mathsf{pMHE.PrivEval}(\mathsf{sk}_i, \tilde{\mathcal{D}}, c_1, c_2, \ldots, c_n;\, r_3)$
6. Output $d_i$.
```

Figure 2.8: The Evaluation Program

7. $\mathsf{KGP}_h \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}^1_{\mathsf{KG}}[\hat{K}_h, h, \hat{y}_h, \hat{c}_h]\big)$ (see Figure 2.7)
8. $\hat{w} \leftarrow (\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq h}$
9. $\mathsf{EP}_h \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}^2_{\mathsf{Eval}}[\hat{K}_h, h, \mathsf{hk}_h, \mathsf{KGP}_h, \hat{y}_h, \hat{c}_h, \hat{\mathsf{sk}}_h, \hat{r}_3, \hat{w}]\big)$ (see Figure 2.9)
10. Output $U_h := (\mathsf{hk}_h, \mathsf{KGP}_h, \mathsf{EP}_h)$.

Hybrid 3: At this point, EP_h only uses the hardcoded key and ciphertext if the *entire* input matches the hardcoded reference input. Since that is the case, EP_h will only ever use the hardcoded decryption key on one ciphertext; so, we can remove the hardcoded decryption key entirely, and instead hardcode a partial decryption value. In this hybrid, the simulator, on behalf of party h, replaces $\mathcal{P}^2_{\mathsf{Eval}}$ (Figure 2.9) with $\mathcal{P}^3_{\mathsf{Eval}}$ (Figure 2.10) which does exactly this. This hybrid is indistinguishable from the previous one by the security of iO.

More specifically, the simulator does the following during Gen on behalf of party h, where the text in red indicates what changed since the previous hybrid:

1. $K_h \xleftarrow{\$} \{0,1\}^\lambda$
2. $\mathsf{hk}_h \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda, B, B)$
3. $\hat{y}_h \leftarrow \mathsf{Hash.Hash}\big(\mathsf{hk}_h, (\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq h}\big)$
4. $\hat{K}_h \leftarrow \mathsf{Punct}(K_h, \hat{y}_h)$
5. $(\hat{s}_h, \hat{r}_2, \hat{r}_3) \leftarrow F_{K_h}(\hat{y}_h)$
6. $(\hat{c}_h, \hat{\mathsf{sk}}_h) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, \tilde{\mathcal{D}}.\mathsf{params}, h, \hat{s}_h;\, \hat{r}_2)$
7. $\mathsf{KGP}_h \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}^1_{\mathsf{KG}}[\hat{K}_h, h, \hat{y}_h, \hat{c}_h]\big)$ (see Figure 2.7)

$\mathcal{P}^2_{\mathsf{Eval}}[K, i, \mathsf{hk}_i, \mathsf{KGP}_i, \hat{y}, \hat{\mathsf{pk}}, \hat{c}, \hat{\mathsf{sk}}, \hat{r}, \hat{w}]$

Hard-coded. The private key K, the index i of the party, the hash key hk_i, the obfuscated key generation program KGP_i, as well as a nonce $\hat{y}$, a ciphertext $\hat{c}$, a secret key $\hat{\mathsf{sk}}$, randomness $\hat{r}$, and a hardcoded input $\hat{w}$.
Input. A set of $n-1$ pairs $(\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq i}$ where the first element is a hash key and the second is an obfuscated key generation program.

```
1. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash.Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{KGP}_l)_{l \neq j}\big)$
2. $\forall j \neq i: \quad c_j \leftarrow \mathsf{KGP}_j(y_j)$
3. If $y_i = \hat{y}$ and the first $l$ blocks of $\hat{w}$ and $(\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq h}$ coincide, set $\mathsf{sk}_i \leftarrow \hat{\mathsf{sk}}$, $c_i \leftarrow \hat{c}$ and $r_3 \leftarrow \hat{r}$.
4. Otherwise,
   (a) $(r_1, r_2, r_3) \leftarrow F_K(y_i)$
   (b) $s_i \leftarrow r_1$
   (c) $(c_i, \mathsf{sk}_i) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, \tilde{\mathcal{D}}.\mathsf{params}, i, s_i;\ r_2)$
5. $d_i \leftarrow \mathsf{pMHE.PrivEval}(\mathsf{sk}_i, \tilde{\mathcal{D}}, c_1, c_2, \ldots, c_n;\ r_3)$
6. Output $d_i$.
```

Figure 2.9: The Evaluation Program

8. $\hat{d}_h \leftarrow \mathsf{pMHE.PrivEval}(\hat{\mathsf{sk}}_h, \tilde{\mathcal{D}}, \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_n;\ \hat{r}_3)$
9. $\hat{w} \leftarrow (\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq h}$
10. $\mathsf{EP}_h \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}^3_{\mathsf{Eval}}[\hat{K}_h, h, \mathsf{hk}_h, \mathsf{KGP}_h, \hat{w}, \hat{d}_h]\big)$ (see Figure 2.10)
11. Output $U_h := (\mathsf{hk}_h, \mathsf{KGP}_h, \mathsf{EP}_h)$.

Hybrid 4. In this hybrid, the simulator computes the final output R directly as $R \leftarrow \mathcal{D}(1^\lambda; s)$ where $s = \hat{s}_1 \oplus \hat{s}_2 \oplus \cdots \oplus \hat{s}_n$. (The simulator of course has all these values, as it has access to all parties' random tapes.) This hybrid is indistinguishable from the previous one by the correctness of obfuscation and MHE with private evaluation.

Hybrid 5. In this hybrid, the simulator, on behalf of party h, replaces the hardcoded output of F at $\hat{y}_h$ with a truly random one. This hybrid is indistinguishable from the previous one by the security of F.

More specifically, the simulator does the following during Gen on behalf of party h, where the text in red indicates what changed since the previous hybrid:

1. $K_h \xleftarrow{\$} \{0,1\}^\lambda$
2. $\mathsf{hk}_h \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda, B, B)$
3. $\hat{y}_h \leftarrow \mathsf{Hash.Hash}\big(\mathsf{hk}_h, (\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq h}\big)$
4. $\hat{K}_h \leftarrow \mathsf{Punct}(K_h, \hat{y}_h)$
5. Sample $(\hat{s}_h, \hat{r}_2, \hat{r}_3)$ at random from the appropriate space
6. $(\hat{c}_h, \hat{\mathsf{sk}}_h) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, \tilde{\mathcal{D}}.\mathsf{params}, h, \hat{s}_h;\ \hat{r}_2)$

$\mathcal{P}^3_{\mathsf{Eval}}[K, i, \mathsf{hk}_i, \mathsf{KGP}_i, \hat{w}, \hat{d}]$

Hard-coded. The private key K, the index i of the party, the hash key hk_i, the obfuscated key generation program KGP_i, as well as a hardcoded input $\hat{w}$ and a partial decryption $\hat{d}$.
Input. A set of $n-1$ pairs $(\mathsf{hk}_j, \mathsf{KGP}_j)_{j \neq i}$ where the first element is a hash key and the second is an obfuscated key generation program.

```
1. If $(\mathsf{hk}_j, \mathsf{KGP}_j)_{j\neq i} = \hat{w}$, output $\hat{d}$.
2. Otherwise, $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash.Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{KGP}_l)_{l\neq j}\big)$
3. $\forall j \neq i: \quad c_j \leftarrow \mathsf{KGP}_j(y_j)$
4. $(r_1, r_2, r_3) \leftarrow F_K(y_i)$
5. $s_i \leftarrow r_1$
6. $(c_i, \mathsf{sk}_i) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, \tilde{\mathcal{D}}.\mathsf{params}, i, s_i; r_2)$
7. $d_i \leftarrow \mathsf{pMHE.PrivEval}(\mathsf{sk}_i, \tilde{\mathcal{D}}, c_1, c_2, \ldots, c_n; r_3)$
8. Output $d_i$.
```

Figure 2.10: The Evaluation Program

7. $\mathsf{KGP}_h \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}^1_{\mathsf{KG}}[\hat{K}_h, h, \hat{y}_h, \hat{c}_h]\big)$ (see Figure 2.7)
8. $\hat{d}_h \leftarrow \mathsf{pMHE.PrivEval}(\hat{\mathsf{sk}}_h, \tilde{\mathcal{D}}, \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_n; \hat{r}_3)$
9. $\hat{w} \leftarrow (\mathsf{hk}_j, \mathsf{KGP}_j)_{j\neq h}$
10. $\mathsf{EP}_h \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}^3_{\mathsf{Eval}}[\hat{K}_h, h, \mathsf{hk}_h, \mathsf{KGP}_h, \hat{w}, \hat{d}_h]\big)$ (see Figure 2.10)
11. Output $U_h := (\mathsf{hk}_h, \mathsf{KGP}_h, \mathsf{EP}_h)$.

Hybrid 6. In this hybrid, the simulator replaces the real ciphertext $\hat{c}_h$ and partial decryption $\hat{d}_h$ with simulated ones. The production of this simulated values does not require party h's secret decryption key nor the plaintext $\hat{s}_h$; it forces the final decryption to output the value R. Since the view of the adversary contains now no information about $\hat{s}_h$, the simulator can sample R at random from $\mathcal{D}$. This hybrid is indistinguishable from the previous one by the reusable semi-malicious security of the MHE scheme with private evaluation.

More specifically, the simulator does the following during Gen on behalf of party h, where the text in red indicates what changed since the previous hybrid:

1. $K_h \xleftarrow{\$} \{0,1\}^\lambda$
2. $\mathsf{hk}_h \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda, B, B)$
3. $\hat{y}_h \leftarrow \mathsf{Hash.Hash}\big(\mathsf{hk}_h, (\mathsf{hk}_j, \mathsf{KGP}_j)_{j\neq h}\big)$
4. $\hat{K}_h \leftarrow \mathsf{Punct}(K_h, \hat{y}_h)$
5. $(\tau, \hat{c}_h) \xleftarrow{\$} \mathsf{pMHE.Sim}_1(1^\lambda, \{h\}, \tilde{\mathcal{D}}.\mathsf{params})$
6. $\mathsf{KGP}_h \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}^1_{\mathsf{KG}}[\hat{K}_h, h, \hat{y}_h, \hat{c}_h]\big)$ (see Figure 2.7)

7. $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$

8. Retrieve the values $(\hat{s}_j)_{j\neq h}$ and the randomness $(\hat{r}_2^j)_{j\neq h}$ used by the other parties for the generation of $(\hat{c}_j)_{j\neq h}$

9. $(\tau, \hat{d}_h) \xleftarrow{\$} \mathsf{pMHE.Sim}_2\big(\tau, \tilde{\mathcal{D}}, R, (\hat{s}_j, \hat{r}_2^j)_{j\neq h}\big)$

10. $\hat{w} \leftarrow (\mathsf{hk}_j, \mathsf{KGP}_j)_{j\neq h}$

11. $\mathsf{EP}_h \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}^3_{\mathsf{Eval}}[\hat{K}_h, h, \mathsf{hk}_h, \mathsf{KGP}_h, \hat{w}, \hat{d}_h]\big)$ (see Figure 2.10)

12. Output $U_h := (\mathsf{hk}_h, \mathsf{KGP}_h, \mathsf{EP}_h)$.

Since this hybrid gives us exactly the distribution we want our original Hybrid 0 to be indistinguishable from, this completes the proof.

□

Observe that a distributed sampler with semi-malicious security also has passive security.

2.5 Upgrading to Active Security

When moving from semi-malicious to active security, there are two main issues we need to tackle: corrupt parties publishing malformed shares of the sampler, and rushing adversaries. The former can be easily dealt with by adding NIZK proofs of well-formedness to the sampler shares (for this reason, our solution relies on a URS). Rushing adversaries are a more challenging problem, and to deal with this, we rely on a random oracle.

The problem of rushing. In the semi-maliciously secure construction described in Section 2.4, the randomness used to generate an honest party's MHE ciphertexts and private keys is output by a PRF, which takes as input a nonce that depends on the key generation programs of all parties (including the corrupt ones). To prove security, we need to puncture the PRF key at that nonce, erasing any correlation between the MHE ciphertext and the PRF key. This can be done in the semi-malicious case, as the simulator knows the programs of the corrupted parties before it must produce those of the honest parties. In the actively secure case, we run into an issue. The adversary is able to adaptively choose the programs of the corrupted parties after seeing those of the other players, in what is called *rushing behaviour*. In the security proof, we would therefore need to puncture a PRF key without knowing the actual position where puncturing is needed.

Although the issue we described above is very specific, dealing with rushing behaviour is a general problem. In a secure distributed sampler, we can program the shares of the honest parties to output an ideal sample when used in conjunction with the shares of the corrupted players. Since the latter are unknown upon generation of the honest players' shares, the immediate approach would be to program the outputs for every possible choice of the adversary. We run however into an incompressibility problem as we would need to store exponentially many ideal outputs in the polynomial-sized sampler shares.

2.5.1 Defeating Rushing

In this section, we present a compiler that allows us to deal with rushing behaviour without adding any additional rounds of interaction. This tool handles rushing behaviour not only for distributed samplers, but for a wide range of applications (including our public-key PCF in Section 2.6). Consider any single-round protocol with no private inputs, where $\mathsf{SendMsg}$ is the algorithm which party i runs to choose a message to send, and Output is an algorithm that determines each party's output (from party i's state and all the messages sent). More concretely, we can describe any such one-round protocol using the following syntax:

$\mathsf{SendMsg}(1^\lambda, i; r_i) \to \mathsf{g}_i$ generates party i's message g_i, and

$\mathcal{F}_{\mathsf{NoRush}}$

Initialisation. Upon receiving Init from every party and the adversary, the functionality activates and enters the querying phase.

Querying phase. Upon receiving the id-th Query from the adversary, the functionality waits for r_i from every corrupted party P_i. Then, for every $h \in H$, it samples $r_h \xleftarrow{\$} \{0,1\}^{L(\lambda)}$ and computes $\mathsf{g}_i \leftarrow \mathsf{SendMsg}(1^\lambda, i; r_i)$ for every $i \in [n]$. Finally, it stores $(r_i, \mathsf{g}_i)_{i\in[n]}$ along with the identity id and sends $(\mathsf{g}_h)_{h\in H}$ back to the adversary.

Output. Upon receiving Output from the adversary, the functionality waits for a value $\widehat{\mathsf{id}}$ from the adversary, and retrieves the corresponding tuple $(r_i, \mathsf{g}_i)_{i\in[n]}$ (or outputs $\perp$ if there is no such tuple). It then outputs r_h and $(\mathsf{g}_i)_{i\in[n]}$ to P_h for every $h \in H$.

Figure 2.11: The Anti-Rushing Functionality $\mathcal{F}_{\mathsf{NoRush}}$

$\mathsf{Output}(i, r_i, (\mathsf{g}_j)_{j\in[n]}) \to \mathsf{res}_i$ produces party i's output res_i.

(In the case of distributed samplers, $\mathsf{SendMsg}$ corresponds to Gen, and Output corresponds to Sample.)

We define modified algorithms ($\mathsf{ARMsg}, \mathsf{AROutput}$) such that the associated one-round protocol realizes an ideal functionality that first waits for the corrupted parties' randomness, and then generates the randomness and messages of the honest parties.

This functionality clearly denies the adversary the full power of rushing: the ability to choose corrupt parties' messages based on honest parties' messages. For this reason, we call it the *no-rush* functionality $\mathcal{F}_{\mathsf{NoRush}}$. However, we do allow the adversary a weaker form of rushing behaviour: *selective sampling.* The functionality allows the adversary to re-submit corrupt parties' messages as many times as it wants, and gives the adversary the honest parties' messages in response (while hiding the honest parties' randomness). At the end, the adversary can select which execution she likes the most.

Definition 2.5.1 (Anti-Rusher). Let ($\mathsf{SendMsg}, \mathsf{Output}$) be a one-round n-party protocol where $\mathsf{SendMsg}$ needs $L(\lambda)$ bits of randomness to generate a message. An anti-rusher for $\mathsf{SendMsg}$ is a one-round protocol ($\mathsf{ARMsg}, \mathsf{AROutput}$) implementing the functionality $\mathcal{F}_{\mathsf{NoRush}}$ (see Figure 2.11) for $\mathsf{SendMsg}$ against an active adversary.

If ($\mathsf{SendMsg}, \mathsf{Output}$) = ($\mathsf{Gen}, \mathsf{Sample}$) is a distributed sampler with semi-malicious security, applying this transformation gives a distributed sampler with active security.

Intuition Behind our Anti-Rushing Compiler.

We define ($\mathsf{ARMsg}, \mathsf{AROutput}$) as follows. When called by party i, ARMsg outputs an obfuscated program S_i; this program takes as input a response of the random oracle, and uses it as a nonce for a PRF F_{K_i}. The program then feeds the resulting pseudorandom string r into $\mathsf{SendMsg}$, and outputs whatever message $\mathsf{SendMsg}$ generates. Our techniques are inspired by the *delayed backdoor programming* technique of Hofheinz *et al.* [HJK+16], used for adaptively secure universal samplers.

The trapdoor. In order to prove that our compiler realizes $\mathcal{F}_{\mathsf{NoRush}}$ for $\mathsf{SendMsg}$, a simulator must be able to force the compiled protocol to return given outputs of $\mathsf{SendMsg}$, even *after* sending messages (outputs of ARMsg) on behalf of the honest parties.

Behind its usual innocent behaviour, the program S_i hides a trapdoor that allows it to secretly communicate with the random oracle. S_i owns a key k_i for a special authenticated encryption scheme based on puncturable PRFs. Every time it receives a random oracle response as input, S_i parses it as a ciphertext-nonce pair and tries to decrypt it. If decryption succeeds, S_i outputs the corresponding plaintext; otherwise, it resumes the usual innocent behaviour, and runs $\mathsf{SendMsg}$. (The encryption scheme guarantees that the decryption of random strings fails with overwhelming probability; this trapdoor is never used accidentally,

$\mathcal{P}_{\mathsf{AR}}[\mathsf{SendMsg}, k, K, i]$

Hard-coded. The algorithm $\mathsf{SendMsg}$, PRF keys k and K and the index i of the party.
Input. Oracle responses $(u, v) \in \{0,1\}^{\lambda \cdot m(\lambda)} \times \{0,1\}^{\lambda}$.

```
1. $(y_1^0, y_1^1, y_2^0, y_2^1, \ldots, y_m^0, y_m^1) \leftarrow F'_k(v)$
2. For every $j \in [m]$ set
       $x^j \leftarrow \begin{cases} 0 & \text{if } y_j^0 = u^j, \\ 1 & \text{if } y_j^1 = u^j, \\ \perp & \text{otherwise.} \end{cases}$
3. If $x^j \neq \perp$ for every $j \in [m]$, output $(x^1, x^2, \ldots, x^m)$.
4. Set $r \leftarrow F_K(u, v)$.
5. Output $\mathsf{g}_i \leftarrow \mathsf{SendMsg}(1^\lambda, i;\, r)$.
```

Figure 2.12: The Anti-Rushing Program

but it will play a crucial role in the proof.) Obfuscation conceals how the result has been computed as long as it is indistinguishable from a random $\mathsf{SendMsg}$ output.

The inputs fed into $(\mathsf{S}_i)_{i\in[n]}$ are generated by querying the random oracle with the programs themselves and NIZKs proving their well-formedness. The random oracle response consists of a random nonce v and additional n blocks $(u_i)_{i\in[n]}$, the i-th one of which is addressed to S_i. The input to S_i will be the pair (u_i, v). When the oracle tries to secretly communicate a message to S_i, u_i will be a ciphertext, whereas v will be the corresponding nonce.

Given a random oracle query, using the simulation-extractability of the NIZKs, the simulator can retrieve the secrets (in particular, the PRF keys) of the corrupted parties. It can then use this information to learn the randomness used to generate the corrupted parties' messages (i.e. their outputs of $\mathsf{SendMsg}$). The simulator then needs only to encrypt these messages received from $\mathcal{F}_{\mathsf{NoRush}}$ using $(k_i)_{i\in H}$, and include these ciphertexts in the oracle response.

Formal Description of our Anti-Rushing Compiler.

We now formalise the ideas we presented in the previous paragraphs. Our anti-rushing compiler is described in Figure 2.13. The unobfuscated program $\mathcal{P}_{\mathsf{AR}}$ is available in Figure 2.12. We assume that its obfuscation needs $M(\lambda)$ bits of randomness. Observe that $\mathcal{P}_{\mathsf{AR}}$ is based on two puncturable PRFs F and F', the first one of which is used to generate the randomness fed into $\mathsf{SendMsg}$.

The second puncturable PRF is part of the authenticated encryption scheme used in the trapdoor. We assume that its outputs are naturally split into $2m$ λ-bit blocks, where $m(\lambda)$ is the size of an output of $\mathsf{SendMsg}$ (after padding). To encrypt a plaintext $(x^1, \ldots, x^m) \in \{0,1\}^m$ using the key k and nonce $v \in \{0,1\}^\lambda$, we first expand v using F'_k. The ciphertext consists of m λ-bit blocks, the j-th one of which coincides with the $(2j+x^j)$-th block output by F'. Decryption is done by reversing these operations. For this reason, we assume that the values $(u_i)_{i\in[n]}$ in the oracle responses are naturally split into m λ-bit chunks. Observe that if the j-th block of the ciphertext is different from both the $2j$-th and the $(2j+1)$-th block output by the PRF, decryption fails.

Finally, let $\mathsf{NIZK}' = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Extract})$ be a simulation-extractable NIZK for the relation $\mathcal{R}$ describing the well-formedness of the obfuscated programs $(\mathsf{S}_i)_{i\in[n]}$. Formally, a statement consists of the pair (S_i, i), whereas the corresponding witness is the triple containing the PRF keys k_i and K_i hard-coded in S_i and the randomness used for the obfuscation of the latter.

Anti-Rushing Compiler Π_{NoRush}

URS. The protocol needs a URS $\mathsf{urs} \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Gen}(1^\lambda)$ for the NIZK proofs.

```
$\mathsf{ARMsg}(1^\lambda, i, \mathsf{urs})$:
    1. $k_i \xleftarrow{\$} \{0,1\}^\lambda$
    2. $K_i \xleftarrow{\$} \{0,1\}^\lambda$
    3. $w_i \xleftarrow{\$} \{0,1\}^{M(\lambda)}$
    4. $\mathsf{S}_i \leftarrow \mathsf{iO}\big(1^\lambda, \mathcal{P}_{\mathsf{AR}}[\mathsf{SendMsg}, k_i, K_i, i]; w_i\big)$ (see Figure 2.12)
    5. $\pi_i \xleftarrow{\$} \mathsf{Prove}\big(1^\lambda, \mathsf{urs}, (\mathsf{S}_i, i), (k_i, K_i, w_i)\big)$
    6. Output $\mathsf{armsg}_i := (\mathsf{S}_i, \pi_i)$.
$\mathsf{AROutput}\big(i, \big(\mathsf{armsg}_j = (\mathsf{S}_j, \pi_j)\big)_{j\in[n]}, \mathsf{urs}\big)$ :
    1. If there exists $j \in [n]$ such that $\mathsf{Verify}\big(\mathsf{urs}, \pi_j, (\mathsf{S}_j, j)\big) = 0$, output $\perp$.
    2. Query $(\mathsf{S}_j, \pi_j)_{j\in[n]}$ to the random oracle $\mathcal{H}$ to get $\big(v, (u_j)_{j\in[n]}\big)$.
    3. $\forall j \in [n]: \quad \mathsf{g}_j \leftarrow \mathsf{S}_j(u_j, v)$.
    4. Output $(\mathsf{g}_j)_{j\in[n]}$ and $F_{K_i}(u_i, v)$.
```

Figure 2.13: Anti-Rushing Compiler

Theorem 2.5.2. If $(\mathsf{SendMsg}, \mathsf{Output})$ is a one-round n-party protocol, $\mathsf{NIZK}' = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Extract})$ is a simulation-extractable NIZK with URS for the relation $\mathcal{R}$, iO is an indistinguishability obfuscator and (F, Punct) and (F', Punct') are two puncturable PRFs satisfying the properties described above, the protocol $\Pi_{\mathsf{NoRush}} = (\mathsf{ARMsg}, \mathsf{AROutput})$ described in Figure 2.13 realizes $\mathcal{F}_{\mathsf{NoRush}}$ for $\mathsf{SendMsg}$ in the random oracle model with a URS.

Proof. The techniques used in this proof are inspired by [HJK+16]. We prove the security of the protocol Π_{NoRush} described in Figure 2.13 by showing that it implements the functionality $\mathcal{F}_{\mathsf{NoRush}}$ (see Figure 2.11) in the UC-model [Can01]. We achieve this plan through a series of hybrids allowing us to transition from the real protocol Π_{NoRush} (Hybrid 0) to the composition of $\mathcal{F}_{\mathsf{NoRush}}$ with a PPT simulator (Hybrid 14). In all the stages, we assume that the keys k_h and K_h are uniformly sampled in $\{0,1\}^\lambda$ for every $h \in H$. Moreover, we assume, without loss of generality, that we deal with adversaries that always query the elements $(\mathsf{S}_i, \pi_i)_{i\in[n]}$ to the random oracle before broadcasting $(\mathsf{S}_i, \pi_i)_{i\in C}$ to the honest parties.

Hybrid 0. This is the initial stage, corresponding to the real world. The simulator generates the URS, the programs of the honest parties $(\mathsf{S}_h)_{h\in H}$, the corresponding NIZKs and the final outputs as per the protocol Π_{NoRush}. Moreover, it replies to the random oracle queries of the adversary sampling random strings. If any value is queried multiple times, the simulator takes care to answer always in the same way. Observe that with overwhelming probability, the final outputs are generated without using the trapdoor of the anti-rushing programs.

Hybrid 1. In this hybrid, we substitute the URS and the NIZKs proving the well-formedness of the programs of the honest parties with the outputs of the simulators $\mathsf{NIZK}'.\mathsf{Sim}_1$ and $\mathsf{NIZK}'.\mathsf{Sim}_2$. In this way, we remove any information concerning the keys of the honest parties from $(\pi_h)_{h\in H}$. Hybrid 1 is indistinguishable from Hybrid 0 due to the multi-theorem zero-knowledge of NIZK'.

Formally speaking, the simulator generates the URS and the anti-rushing messages $(\mathsf{S}_h, \pi_h)_{h\in H}$ as follows (the red text indicates what changed since the last hybrid).

1. $\forall h \in H : \quad \mathsf{S}_h \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}_{\mathsf{AR}}[\mathsf{SendMsg}, k_h, K_h, h])$
2. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^\lambda)$
3. $\forall h \in H : \quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (\mathsf{S}_h, h)\big)$

Next, for every q from 1 to the number of random oracle queries issued by the adversary, we proceed from Hybrid 2.q to Hybrid 13.q.

Hybrid 2.q. In this hybrid, we schedule the q-th oracle response $\big(\hat{v}, (\hat{u}_i)_{i\in[n]}\big)$ before generating the programs of the honest parties. Furthermore, for every honest party h, we puncture the key K_h in $(\hat{u}_h, \hat{v})$ and we store it in S_h. We also program the latter to output the appropriate result when $(\hat{u}_h, \hat{v})$ is input. Observe that with overwhelming probability, such input does not activate the trapdoor in S_h. By the correctness of puncturing, the input-output behaviour of the honest parties' programs is the same as in the previous hybrid. Hence, indistinguishability holds by the security of iO.

The formal steps performed by the simulator in order to generate $(\mathsf{S}_h, \pi_h)_{h\in H}$ are described below.

1. $\hat{v} \xleftarrow{\$} \{0,1\}^\lambda$
2. $\forall i \in [n] : \quad \hat{u}_i \xleftarrow{\$} \{0,1\}^{m\cdot\lambda}$
3. $\forall h \in H : \quad \hat{K}_h \leftarrow \mathsf{Punct}\big(K_h, (\hat{u}_h, \hat{v})\big)$
4. $\forall h \in H : \quad \hat{r}_h \leftarrow F_{K_h}(\hat{u}_h, \hat{v})$
5. $\forall h \in H : \quad \hat{\mathfrak{g}}_h \leftarrow \mathsf{SendMsg}(1^\lambda, h; \hat{r}_h)$
6. $\forall h \in H : \quad \mathsf{S}_h \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}^1_{\mathsf{AR}}[\mathsf{SendMsg}, k_h, \hat{K}_h, h, \hat{u}_h, \hat{v}, \hat{\mathfrak{g}}_h])$ (see Figure 2.14)
7. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^\lambda)$

$\mathcal{P}^1_{\mathsf{AR}}[\mathsf{SendMsg}, k, K, i, \hat{u}, \hat{v}, \hat{\mathsf{g}}]$

Hard-coded. The algorithm $\mathsf{SendMsg}$, the PRF keys k and K and the index i of the party. Moreover, the scheduled oracle response $(\hat{u}, \hat{v})$ to the q-th query and the corresponding output $\hat{\mathsf{g}}$.
Input. Oracle responses $(u, v) \in \{0,1\}^{\lambda \cdot m(\lambda)} \times \{0,1\}^{\lambda}$.

```
1. If $(u, v) = (\hat{u}, \hat{v})$, output $\hat{\mathsf{g}}$
2. $(y_1^0, y_1^1, y_2^0, y_2^1, \ldots, y_m^0, y_m^1) \leftarrow F'_k(v)$
3. For every $j \in [m]$ set
       $x^j \leftarrow \begin{cases} 0 & \text{if } y_j^0 = u^j, \\ 1 & \text{if } y_j^1 = u^j, \\ \perp & \text{otherwise.} \end{cases}$
4. If $x^j \neq \perp$ for every $j \in [m]$, output $(x^1, x^2, \ldots, x^m)$.
5. Set $r \leftarrow F_K(u, v)$.
6. Output $\mathsf{g} \leftarrow \mathsf{SendMsg}(1^\lambda, i; r)$.
```

Figure 2.14: The Anti-Rushing Program

8. $\forall h \in H: \quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (\mathsf{S}_h, h)\big)$

Hybrid 3.q. In this hybrid, on behalf of every honest party h, the simulator generates the element $\hat{\mathsf{g}}_h$ hard-coded into S_h using true randomness $\hat{r}_h$ instead of the output of F. Moreover, if the anti-rushing messages $(\mathsf{armsg}_i)_{i \in C}$ of the corrupted parties are valid and correspond to the q-th oracle query, for every $h \in H$, the simulator directly outputs $\hat{r}_h$ to P_h, instead of using F_{K_h}. Observe that this hybrid is indistinguishable from the previous one due to the security of the puncturable PRF F.

The precise procedure used by the simulator to generate $(\mathsf{S}_h, \pi_h)_{h \in H}$ is the following.

1. $\hat{v} \xleftarrow{\$} \{0,1\}^\lambda$
2. $\forall i \in [n]: \quad \hat{u}_i \xleftarrow{\$} \{0,1\}^{m \cdot \lambda}$
3. $\forall h \in H: \quad \hat{K}_h \leftarrow \mathsf{Punct}\big(K_h, (\hat{u}_h, \hat{v})\big)$
4. $\forall h \in H: \quad \hat{r}_h \xleftarrow{\$} \{0,1\}^{L(\lambda)}$
5. $\forall h \in H: \quad \hat{\mathsf{g}}_h \leftarrow \mathsf{SendMsg}(1^\lambda, h; \hat{r}_h)$
6. $\forall h \in H: \quad \mathsf{S}_h \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}^1_{\mathsf{AR}}[\mathsf{SendMsg}, k_h, \hat{K}_h, h, \hat{u}_h, \hat{v}, \hat{\mathsf{g}}_h])$ (see Figure 2.14)
7. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^\lambda)$
8. $\forall h \in H: \quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (\mathsf{S}_h, h)\big)$

Hybrid 4.q. For every honest party h, we now puncture the PRF key k_h in $\hat{v}$. Furthermore, we hard-code into S_h the output of $F'_{k_h}(\hat{v})$ and we use it to compute the result when $\hat{v}$ is input. Since the input-output behaviour of the program remains the same as in the previous hybrid, indistinguishability holds due to the security of iO.

The formal steps performed by the simulator for the generation of the the anti-rushing messages of the honest parties change as follows.

```
$\mathcal{P}^2_{\mathsf{AR}}[\mathsf{SendMsg}, k, K, i, \hat{u}, \hat{v}, \hat{\mathsf{g}}, (\hat{y}^b_j)_{j,b}]$

Hard-coded. The algorithm $\mathsf{SendMsg}$, the PRF keys $k$ and $K$ and the index $i$ of the party. Moreover, the scheduled oracle response $(\hat{u}, \hat{v})$ to the $q$-th query and the corresponding output $\hat{\mathsf{g}}$. Finally, the PRF output $(\hat{y}^b_j)_{j,b}$.
Input. Oracle responses $(u, v) \in \{0,1\}^{\lambda \cdot m(\lambda)} \times \{0,1\}^{\lambda}$.

  1. If $(u, v) = (\hat{u}, \hat{v})$, output $\hat{\mathsf{g}}$
  2. If $v = \hat{v}$, for every $j \in [m]$, set
         $x^j \leftarrow \begin{cases} 0 & \text{if } \hat{y}^0_j = u^j, \\ 1 & \text{if } \hat{y}^1_j = u^j, \\ \perp & \text{otherwise.} \end{cases}$
  3. Otherwise, compute $(y^0_1, y^1_1, y^0_2, y^1_2, \dots, y^0_m, y^1_m) \leftarrow F'_k(v)$ and, for every $j \in [m]$, set
         $x^j \leftarrow \begin{cases} 0 & \text{if } y^0_j = u^j, \\ 1 & \text{if } y^1_j = u^j, \\ \perp & \text{otherwise.} \end{cases}$
  4. If $x^j \neq \perp$ for every $j \in [m]$, output $(x^1, x^2, \dots, x^m)$.
  5. Set $r \leftarrow F_K(u, v)$.
  6. Output $\mathsf{g} \leftarrow \mathsf{SendMsg}(1^\lambda, i; r)$.
```

Figure 2.15: The Anti-Rushing Program

1. $\hat{v} \xleftarrow{\$} \{0,1\}^\lambda$
2. $\forall i \in [n]: \quad \hat{u}_i \xleftarrow{\$} \{0,1\}^{m \cdot \lambda}$
3. $\forall h \in H: \quad \hat{K}_h \leftarrow \mathsf{Punct}\big(K_h, (\hat{u}_h, \hat{v})\big)$
4. $\forall h \in H: \quad \hat{k}_h \leftarrow \mathsf{Punct}'(k_h, \hat{v})$
5. $\forall h \in H: \quad (\hat{y}^b_{h,j})_{j,b} \leftarrow F'_{k_h}(\hat{v})$
6. $\forall h \in H: \quad \hat{r}_h \xleftarrow{\$} \{0,1\}^{L(\lambda)}$
7. $\forall h \in H: \quad \hat{\mathsf{g}}_h \leftarrow \mathsf{SendMsg}(1^\lambda, h; \hat{r}_h)$
8. $\forall h \in H: \quad \mathsf{S}_h \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}^2_{\mathsf{AR}}[\mathsf{SendMsg}, \hat{k}_h, \hat{K}_h, h, \hat{u}_h, \hat{v}, \hat{\mathsf{g}}_h, (\hat{y}^b_{h,j})_{j,b}])$ (see Figure 2.15)
9. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^\lambda)$
10. $\forall h \in H: \quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (\mathsf{S}_h, h)\big)$

Hybrid 5.q. In this hybrid, on behalf of every honest party h, the simulator generates the values $(\hat{y}^b_{h,j})_{j,b}$ sampling them uniformly in $\{0,1\}^\lambda$ instead of using the PRF F'. This hybrid is indistinguishable from the previous one due to the security of the puncturable PRF F'.

The specific steps performed by the simulator for the generation of $(\mathsf{S}_h, \pi_h)_{h \in H}$ are the following.

1. $\hat{v} \xleftarrow{\$} \{0,1\}^\lambda$
2. $\forall i \in [n]: \quad \hat{u}_i \xleftarrow{\$} \{0,1\}^{m\cdot\lambda}$
3. $\forall h \in H: \quad \hat{K}_h \leftarrow \mathsf{Punct}\big(K_h, (\hat{u}_h, \hat{v})\big)$
4. $\forall h \in H: \quad \hat{k}_h \leftarrow \mathsf{Punct}'(k_h, \hat{v})$
5. $\forall h \in H, j \in [m]$ and $b \in \{0,1\}: \quad \hat{y}^b_{h,j} \xleftarrow{\$} \{0,1\}^\lambda$
6. $\forall h \in H: \quad \hat{r}_h \xleftarrow{\$} \{0,1\}^{L(\lambda)}$
7. $\forall h \in H: \quad \hat{\mathsf{g}}_h \leftarrow \mathsf{SendMsg}(1^\lambda, h; \hat{r}_h)$
8. $\forall h \in H: \quad \mathsf{S}_h \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}^2_{\mathsf{AR}}[\mathsf{SendMsg}, \hat{k}_h, \hat{K}_h, h, \hat{u}_h, \hat{v}, \hat{\mathsf{g}}_h, (\hat{y}^b_{h,j})_{j,b}])$ (see Figure 2.15)
9. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^\lambda)$
10. $\forall h \in H: \quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (\mathsf{S}_h, h)\big)$

Hybrid 6.q. In this hybrid, we rely on an injective double-lengthening PRG $\mathsf{PRG}: \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$. Instead of hard-coding the values $(\hat{y}^b_{h,j})_{j,b}$ in S_h, the simulator now stores their images $(\hat{e}^b_{h,j})_{j,b}$ under the PRG PRG, i.e. $\hat{e}^b_{h,j} = \mathsf{PRG}(\hat{y}^b_{h,j})$. Furthermore, when $\hat{v}$ is input in S_h, the program decodes now x^j by comparing $\mathsf{PRG}(u^j)$ to $\hat{e}^0_{h,j}$ and $\hat{e}^1_{h,j}$. Observe that since PRG is injective, $u^j = \hat{y}^b_{h,j}$ if and only if $\mathsf{PRG}(u^j) = \hat{e}^b_{h,j}$. In other words, the input-output behaviour of the programs of the honest parties did not change with respect to the previous hybrid. Therefore, indistinguishability holds by the security of iO.

The formal procedure performed by the simulator for the generation of $(\mathsf{S}_h, \pi_h)_{h\in H}$ is the following.

1. $\hat{v} \xleftarrow{\$} \{0,1\}^\lambda$
2. $\forall i \in [n]: \quad \hat{u}_i \xleftarrow{\$} \{0,1\}^{m\cdot\lambda}$
3. $\forall h \in H: \quad \hat{K}_h \leftarrow \mathsf{Punct}\big(K_h, (\hat{u}_h, \hat{v})\big)$
4. $\forall h \in H: \quad \hat{k}_h \leftarrow \mathsf{Punct}'(k_h, \hat{v})$
5. $\forall h \in H, j \in [m]$ and $b \in \{0,1\}: \quad \hat{y}^b_{h,j} \xleftarrow{\$} \{0,1\}^\lambda$
6. $\forall h \in H, j \in [m]$ and $b \in \{0,1\}: \quad \hat{e}^b_{h,j} \leftarrow \mathsf{PRG}(\hat{y}^b_{h,j})$
7. $\forall h \in H: \quad \hat{r}_h \xleftarrow{\$} \{0,1\}^{L(\lambda)}$
8. $\forall h \in H: \quad \hat{\mathsf{g}}_h \leftarrow \mathsf{SendMsg}(1^\lambda, h; \hat{r}_h)$
9. $\forall h \in H: \quad \mathsf{S}_h \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}^3_{\mathsf{AR}}[\mathsf{SendMsg}, \hat{k}_h, \hat{K}_h, h, \hat{u}_h, \hat{v}, \hat{\mathsf{g}}_h, (\hat{e}^b_{h,j})_{j,b}])$ (see Figure 2.16)
10. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^\lambda)$
11. $\forall h \in H: \quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (\mathsf{S}_h, h)\big)$

Hybrid 7.q. In this hybrid, instead of generating the values $(\hat{e}^b_{h,j})_{j,b}$ using the PRG PRG, the simulator samples them uniformly in $\{0,1\}^{2\lambda}$ for every $h \in H$. By the security of PRG, this hybrid is therefore indistinguishable from the previous one. Here, we are actually relying on the fact that, with overwhelming probability, the terms $(\hat{y}^b_{h,j})_{j,b}$ are not used for the generation the first $q-1$ oracle responses. This is a consequence of the fact that, with overwhelming probability, $\hat{v}$ is different from the nonces in the first $q-1$ oracle answers.

The procedure used by the simulator becomes now the following.

$\mathcal{P}^3_{\mathsf{AR}}[\mathsf{SendMsg}, k, K, i, \hat{u}, \hat{v}, \hat{\mathsf{g}}, (\hat{e}^b_j)_{j,b}]$

Hard-coded. The algorithm SendMsg, PRF keys k and K and the index i of the party. Moreover, the scheduled oracle response $(\hat{u}, \hat{v})$ to the q-th query and the corresponding output $\hat{\mathsf{g}}$. Finally, the PRG outputs $(\hat{e}^b_j)_{j,b}$.

Input. Oracle responses $(u, v) \in \{0,1\}^{\lambda \cdot m(\lambda)} \times \{0,1\}^\lambda$.

1. If $(u, v) = (\hat{u}, \hat{v})$, output $\hat{\mathsf{g}}$
2. If $v = \hat{v}$, for every $j \in [m]$, set

$$x^j \leftarrow \begin{cases} 0 & \text{if } \hat{e}^0_j = \mathsf{PRG}(u^j), \\ 1 & \text{if } \hat{e}^1_j = \mathsf{PRG}(u^j), \\ \perp & \text{otherwise.} \end{cases}$$

3. Otherwise, compute $(y^0_1, y^1_1, y^0_2, y^1_2, \dots, y^0_m, y^1_m) \leftarrow F'_k(v)$ and, for every $j \in [m]$, set

$$x^j \leftarrow \begin{cases} 0 & \text{if } y^0_j = u^j, \\ 1 & \text{if } y^1_j = u^j, \\ \perp & \text{otherwise.} \end{cases}$$

4. If $x^j \neq \perp$ for every $j \in [m]$, output $(x^1, x^2, \dots, x^m)$.
5. Set $r \leftarrow F_K(u, v)$.
6. Output $\mathsf{g} \leftarrow \mathsf{SendMsg}(1^\lambda, i; r)$.

Figure 2.16: The Anti-Rushing Program

1. $\hat{v} \xleftarrow{\$} \{0,1\}^\lambda$
2. $\forall i \in [n]: \quad \hat{u}_i \xleftarrow{\$} \{0,1\}^{m \cdot \lambda}$
3. $\forall h \in H: \quad \hat{K}_h \leftarrow \mathsf{Punct}\big(K_h, (\hat{u}_h, \hat{v})\big)$
4. $\forall h \in H: \quad \hat{k}_h \leftarrow \mathsf{Punct}'(k_h, \hat{v})$
5. $\forall h \in H, j \in [m]$ and $b \in \{0,1\}: \quad \hat{e}^b_{h,j} \xleftarrow{\$} \{0,1\}^{2\lambda}$
6. $\forall h \in H: \quad \hat{r}_h \xleftarrow{\$} \{0,1\}^{L(\lambda)}$
7. $\forall h \in H: \quad \hat{\mathsf{g}}_h \leftarrow \mathsf{SendMsg}(1^\lambda, h; \hat{r}_h)$
8. $\forall h \in H: \quad \mathsf{S}_h \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}^3_{\mathsf{AR}}[\mathsf{SendMsg}, \hat{k}_h, \hat{K}_h, h, \hat{u}_h, \hat{v}, \hat{\mathsf{g}}_h, (\hat{e}^b_{h,j})_{j,b}])$ (see Figure 2.16)
9. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^\lambda)$
10. $\forall h \in H: \quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (\mathsf{S}_h, h)\big)$

Hybrid 8.q. Starting from this hybrid, the program S_h generates the element $\hat{\mathsf{g}}_h$ using the trapdoor mechanism. Compared to the previous stage, the only thing that actually changes is how the values $(\hat{e}^b_{h,j})_{j,b}$ are generated. Specifically, the simulator first encodes the element $\hat{\mathsf{g}}_h$ as a bit string $\hat{x}_h$. Then, for every

$j \in [m]$, it sets $\hat{e}_{h,j}^{\hat{x}_h^j}$ to $\mathsf{PRG}(\hat{u}_h^j)$. The remaining value $\hat{e}_{h,j}^{1-\hat{x}_h^j}$ is instead sampled uniformly in $\{0,1\}^{2\lambda}$ as in the previous hybrid. Observe that in this way, even if we remove the first line from $\mathcal{P}_{\mathsf{AR}}^3$, S_h keeps outputting $\hat{\mathsf{g}}_h$ when $(\hat{u}_h, \hat{v})$ is provided as input. We call the program obtained in this way $\mathcal{P}_{\mathsf{AR}}^4$.

It is possible to conclude that $\mathcal{P}_{\mathsf{AR}}^3$ and $\mathcal{P}_{\mathsf{AR}}^4$ have actually the same input-output behaviour. The claim is trivially verifiable when the input (u, v) coincides with the hard-coded pair $(\hat{u}, \hat{v})$ or $v \neq \hat{v}$. If instead $v = \hat{v}$ and $u \neq \hat{u}$, the matter is a little more complex.

Observe that the image of the PRG PRG has 2^λ elements and they are embedded into a space of cardinality $2^{2\lambda}$. Since the latter is much larger, with overwhelming probability, values uniformly sampled in $\{0,1\}^{2\lambda}$ do not belong to the image of PRG. In particular, this holds for those elements among $(\hat{e}_{h,j}^b)_{j,b}$ that are sampled uniformly. For such values, there is no chance that $\hat{e}_{h,j}^b = G(\hat{u}_h^j)$. As a consequence, the output of $\mathcal{P}_{\mathsf{AR}}^3$ is never generated using the trapdoor when $v = \hat{v}$. In the case of $\mathcal{P}_{\mathsf{AR}}^4$ instead, the only input with $v = \hat{v}$ that activates the trapdoor is $(\hat{u}, \hat{v})$ due to the injectivity of PRG.

In this way, we have proven that Hybrid 8.q is indistinguishable from the previous one due to security of iO. The formal description of the steps performed by the simulator is available below.

1. $\hat{v} \xleftarrow{\$} \{0,1\}^\lambda$
2. $\forall i \in [n]: \quad \hat{u}_i \xleftarrow{\$} \{0,1\}^{m\cdot\lambda}$
3. $\forall h \in H: \quad \hat{K}_h \leftarrow \mathsf{Punct}\big(K_h, (\hat{u}_h, \hat{v})\big)$
4. $\forall h \in H: \quad \hat{k}_h \leftarrow \mathsf{Punct}'(k_h, \hat{v})$
5. $\forall h \in H: \quad \hat{r}_h \xleftarrow{\$} \{0,1\}^{L(\lambda)}$
6. $\forall h \in H: \quad \hat{\mathsf{g}}_h \leftarrow \mathsf{SendMsg}(1^\lambda, h; \hat{r}_h)$
7. For every $h \in H$, rewrite $\hat{\mathsf{g}}_h$ as an m-bit string $\hat{x}_h$.
8. $\forall h \in H$ and $j \in [m]: \quad \hat{e}_{h,j}^{\hat{x}_h^j} \leftarrow \mathsf{PRG}(\hat{u}_h^j)$
9. $\forall h \in H$ and $j \in [m]: \quad \hat{e}_{h,j}^{1-\hat{x}_h^j} \xleftarrow{\$} \{0,1\}^{2\lambda}$
10. $\forall h \in H: \quad \mathsf{S}_h \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}_{\mathsf{AR}}^4[\mathsf{SendMsg}, \hat{k}_h, \hat{K}_h, h, \hat{v}, (\hat{e}_{h,j}^b)_{j,b}])$ (see Figure 2.17)
11. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^\lambda)$
12. $\forall h \in H: \quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (\mathsf{S}_h, h)\big)$

Hybrid 9.q. In this hybrid, the simulator changes the way it generates those values among $(\hat{e}_{h,j}^b)_{j,b}$ that were previously sampled uniformly in $\{0,1\}^{2\lambda}$. For each of them, it indeed chooses a random λ-bit seed $\hat{y}_{h,j}^b$ and sets $\hat{e}_{h,j}^b \leftarrow \mathsf{PRG}(\hat{y}_{h,j}^b)$. This hybrid is therefore indistinguishable from the previous one by the PRG security of PRG. The procedure describing the steps of the simulator is now the following.

1. $\hat{v} \xleftarrow{\$} \{0,1\}^\lambda$
2. $\forall i \in [n]: \quad \hat{u}_i \xleftarrow{\$} \{0,1\}^{m\cdot\lambda}$
3. $\forall h \in H: \quad \hat{K}_h \leftarrow \mathsf{Punct}\big(K_h, (\hat{u}_h, \hat{v})\big)$
4. $\forall h \in H: \quad \hat{k}_h \leftarrow \mathsf{Punct}'(k_h, \hat{v})$
5. $\forall h \in H: \quad \hat{r}_h \xleftarrow{\$} \{0,1\}^{L(\lambda)}$
6. $\forall h \in H: \quad \hat{\mathsf{g}}_h \leftarrow \mathsf{SendMsg}(1^\lambda, h; \hat{r}_h)$

$\mathcal{P}^4_{\mathsf{AR}}[\mathsf{SendMsg}, k, K, i, \hat{v}, (\hat{e}^b_j)_{j,b}]$

Hard-coded. The algorithm SendMsg, the PRF keys k and K and the index i of the party. Moreover, the scheduled nonce $\hat{v}$ for the q-th oracle query and the values $(\hat{e}^b_j)_{j,b}$.
Input. Oracle responses $(u, v) \in \{0,1\}^{\lambda \cdot m(\lambda)} \times \{0,1\}^\lambda$.

1. If $v = \hat{v}$, for every $j \in [m]$, set

$$x^j \leftarrow \begin{cases} 0 & \text{if } \hat{e}^0_j = \mathsf{PRG}(u^j), \\ 1 & \text{if } \hat{e}^1_j = \mathsf{PRG}(u^j), \\ \perp & \text{otherwise.} \end{cases}$$

2. Otherwise, compute $(y^0_1, y^1_1, y^0_2, y^1_2, \ldots, y^0_m, y^1_m) \leftarrow F'_k(v)$ and, for every $j \in [m]$, set

$$x^j \leftarrow \begin{cases} 0 & \text{if } y^0_j = u^j, \\ 1 & \text{if } y^1_j = u^j, \\ \perp & \text{otherwise.} \end{cases}$$

3. If $x^j \neq \perp$ for every $j \in [m]$, output $(x^1, x^2, \ldots, x^m)$.
4. Set $r \leftarrow F_K(u, v)$.
5. Output $\mathsf{g} \leftarrow \mathsf{SendMsg}(1^\lambda, i; r)$.

Figure 2.17: The Anti-Rushing Program

7. For every $h \in H$, rewrite $\hat{\mathsf{g}}_h$ as an m-bit string $\hat{x}_h$.
8. $\forall h \in H$ and $j \in [m]$: $\quad \hat{y}^{\hat{x}^j_h}_{h,j} \leftarrow \hat{u}^j_h$
9. $\forall h \in H$ and $j \in [m]$: $\quad \hat{y}^{1-\hat{x}^j_h}_{h,j} \xleftarrow{\$} \{0,1\}^\lambda$
10. $\forall h \in H, j \in [m]$ and $b \in \{0,1\}$: $\quad \hat{e}^b_{h,j} \leftarrow \mathsf{PRG}(\hat{y}^b_{h,j})$
11. $\forall h \in H$: $\quad \mathsf{S}_h \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}^4_{\mathsf{AR}}[\mathsf{SendMsg}, \hat{k}_h, \hat{K}_h, h, \hat{v}, (\hat{e}^b_{h,j})_{j,b}])$ (see Figure 2.17)
12. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^\lambda)$
13. $\forall h \in H$: $\quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (\mathsf{S}_h, h)\big)$

Hybrid 10.q. In this hybrid, instead of hard-coding the values $(\hat{e}^b_{h,j})_{j,b}$ in S_h, the simulator stores their preimages $(\hat{y}^b_{h,j})_{j,b}$ under the PRG PRG. Furthermore, when $\hat{v}$ is input into S_h, the program decodes x^j by comparing u^j to $\hat{y}^0_{h,j}$ and $\hat{y}^1_{h,j}$. Observe that since PRG is injective, $u^j = \hat{y}^b_{h,j}$ if and only if $\mathsf{PRG}(u^j) = \hat{e}^b_{h,j}$. In other words, the input-output behaviour of the programs of the honest parties did not change with respect to the previous hybrid. Therefore, indistinguishability holds by the security of iO.

The formal procedure performed by the simulator for the generation of $(\mathsf{S}_h, \pi_h)_{h \in H}$ is the following.

1. $\hat{v} \xleftarrow{\$} \{0,1\}^\lambda$
2. $\forall i \in [n]$: $\quad \hat{u}_i \xleftarrow{\$} \{0,1\}^{m \cdot \lambda}$

```
$\mathcal{P}^5_{\mathsf{AR}}[\mathsf{SendMsg}, k, K, i, \hat{v}, (\hat{y}^b_j)_{j,b}]$

Hard-coded. The algorithm SendMsg, the PRF keys $k$ and $K$ and the index $i$ of the party. Moreover,
the scheduled nonce $\hat{v}$ for the $q$-th oracle query and the values $(\hat{y}^b_j)_{j,b}$.
Input. Oracle responses $(u, v) \in \{0,1\}^{\lambda \cdot m(\lambda)} \times \{0,1\}^{\lambda}$.

1. If $v = \hat{v}$, for every $j \in [m]$, set
       $x^j \leftarrow \begin{cases} 0 & \text{if } \hat{y}^0_j = u^j, \\ 1 & \text{if } \hat{y}^1_j = u^j, \\ \perp & \text{otherwise.} \end{cases}$
2. Otherwise, compute $(y^0_1, y^1_1, y^0_2, y^1_2, \ldots, y^0_m, y^1_m) \leftarrow F'_k(v)$ and, for every $j \in [m]$, set
       $x^j \leftarrow \begin{cases} 0 & \text{if } y^0_j = u^j, \\ 1 & \text{if } y^1_j = u^j, \\ \perp & \text{otherwise.} \end{cases}$
3. If $x^j \neq \perp$ for every $j \in [m]$, output $(x^1, x^2, \ldots, x^m)$.
4. Set $r \leftarrow F_K(u, v)$.
5. Output $\mathsf{g} \leftarrow \mathsf{SendMsg}(1^\lambda, i; r)$.
```

Figure 2.18: The Anti-Rushing Program

3. $\forall h \in H : \quad \hat{K}_h \leftarrow \mathsf{Punct}\big(K_h, (\hat{u}_h, \hat{v})\big)$
4. $\forall h \in H : \quad \hat{k}_h \leftarrow \mathsf{Punct}'(k_h, \hat{v})$
5. $\forall h \in H : \quad \hat{r}_h \xleftarrow{\$} \{0,1\}^{L(\lambda)}$
6. $\forall h \in H : \quad \hat{\mathsf{g}}_h \leftarrow \mathsf{SendMsg}(1^\lambda, h; \hat{r}_h)$
7. For every $h \in H$, rewrite $\hat{\mathsf{g}}_h$ as an m-bit string $\hat{x}_h$.
8. $\forall h \in H$ and $j \in [m] : \quad \hat{y}^{\hat{x}^j_h}_{h,j} \leftarrow \hat{u}^j_h$
9. $\forall h \in H$ and $j \in [m] : \quad \hat{y}^{1-\hat{x}^j_h}_{h,j} \xleftarrow{\$} \{0,1\}^\lambda$
10. $\forall h \in H : \quad \mathsf{S}_h \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}^5_{\mathsf{AR}}[\mathsf{SendMsg}, \hat{k}_h, \hat{K}_h, h, \hat{v}, (\hat{y}^b_{h,j})_{j,b}])$ (see Figure 2.18)
11. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^\lambda)$
12. $\forall h \in H : \quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (\mathsf{S}_h, h)\big)$

Hybrid 11.q. In this hybrid, we finally change the oracle response to the q-th query of the adversary, substituting the random values $(\hat{u}_h)_{h \in H}$ with the encryption of the elements $(\hat{\mathsf{g}}_h)_{h \in H}$.

Actually, the only thing that changes is the distribution of the terms $(\hat{y}^b_{h,j})_{j,b}$. Indeed, they are not uniform in $\{0,1\}^\lambda$ anymore, but they are generated by the simulator as $F'_{k_h}(\hat{v})$. Since $\hat{y}^{\hat{x}^j_h}_{h,j} = \hat{u}^j_h$ for every $j \in [m]$ and $h \in H$, we also modify the values $(\hat{u}_h)_{h \in H}$ accordingly.

Observe that this hybrid is indistinguishable from the previous one by the security of the puncturable PRF F'. The formal procedure used by the simulator for the generation of $(\mathsf{S}_h, \pi_h)_{h\in H}$ and the q-th oracle response becomes the following.

1. $\hat{v} \xleftarrow{\$} \{0,1\}^{\lambda}$
2. $\forall i \in C: \quad \hat{u}_i \xleftarrow{\$} \{0,1\}^{m\cdot\lambda}$
3. $\forall h \in H: \quad \hat{k}_h \leftarrow \mathsf{Punct}'(k_h, \hat{v})$
4. $\forall h \in H: \quad \hat{r}_h \xleftarrow{\$} \{0,1\}^{L(\lambda)}$
5. $\forall h \in H: \quad \hat{\mathsf{g}}_h \leftarrow \mathsf{SendMsg}(1^{\lambda}, h; \hat{r}_h)$
6. For every $h \in H$, rewrite $\hat{\mathsf{g}}_h$ as an m-bit string $\hat{x}_h$.
7. $\forall h \in H: \quad (\hat{y}^b_{h,j})_{j,b} \leftarrow F'_{k_h}(\hat{v})$
8. $\forall h \in H$ and $j \in [m]: \quad \hat{u}^j_h \leftarrow \hat{y}^{\hat{x}^j_h}_{h,j}$
9. $\forall h \in H: \quad \hat{K}_h \leftarrow \mathsf{Punct}\big(K_h, (\hat{u}_h, \hat{v})\big)$
10. $\forall h \in H: \quad \mathsf{S}_h \xleftarrow{\$} \mathsf{iO}(1^{\lambda}, \mathcal{P}^5_{\mathsf{AR}}[\mathsf{SendMsg}, \hat{k}_h, \hat{K}_h, h, \hat{v}, (\hat{y}^b_{h,j})_{j,b}])$ (see Figure 2.18)
11. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^{\lambda})$
12. $\forall h \in H: \quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (\mathsf{S}_h, h)\big)$

Hybrid 12.q. In this hybrid, the simulator generates the programs of the honest parties by obfuscating $\mathcal{P}_{\mathsf{AR}}$, as in the original protocol. We however keep replying to the q-th oracle query of the adversary as in Hybrid 11.q. Indistinguishability from the previous stage is guaranteed by the security of iO. As a matter of fact, the programs $\mathcal{P}^5_{\mathsf{AR}}$ (as in Hybrid 11.q) and $\mathcal{P}_{\mathsf{AR}}$ have the same input-output behaviour.

Formally, the anti-rushing messages $(\mathsf{S}_h, \pi_h)_{h\in H}$ are generated as follows.

1. $\forall h \in H: \quad \mathsf{S}_h \xleftarrow{\$} \mathsf{iO}(1^{\lambda}, \mathcal{P}_{\mathsf{AR}}[\mathsf{SendMsg}, k_h, K_h, h])$ (see Figure 2.12)
2. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^{\lambda})$
3. $\forall h \in H: \quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (\mathsf{S}_h, h)\big)$

The procedure for the generation of the q-th oracle response is instead the following.

1. $\hat{v} \xleftarrow{\$} \{0,1\}^{\lambda}$
2. $\forall i \in C: \quad \hat{u}_i \xleftarrow{\$} \{0,1\}^{m\cdot\lambda}$
3. $\forall h \in H: \quad \hat{r}_h \xleftarrow{\$} \{0,1\}^{L(\lambda)}$
4. $\forall h \in H: \quad \hat{\mathsf{g}}_h \leftarrow \mathsf{SendMsg}(1^{\lambda}, h; \hat{r}_h)$
5. For every $h \in H$, rewrite $\hat{\mathsf{g}}_h$ as an m-bit string $\hat{x}_h$.
6. $\forall h \in H: \quad (\hat{y}^b_{h,j})_{j,b} \leftarrow F'_{k_h}(\hat{v})$
7. $\forall h \in H$ and $j \in [m]: \quad \hat{u}^j_h \leftarrow \hat{y}^{\hat{x}^j_h}_{h,j}$

Observe that the URS and the anti-rushing messages $(\mathsf{armsg}_h)_{h\in H}$ of the honest parties can be now generated independently of the q-th oracle response. It is therefore possible to produce the q-th oracle answer on the fly, after receiving the corresponding query.

Hybrid 13.q. In this hybrid, the simulator generates the elements $(\hat{\mathsf{g}}_h)_{h\in H}$ hidden in the q-th oracle response using the functionality $\mathcal{F}_{\mathsf{NoRush}}$. The operation is actually performed only if the q-th oracle query consists of n elements $(\mathsf{S}_i, \pi_i)_{i\in[n]}$ where the values $(\mathsf{S}_h, \pi_h)_{h\in H}$ coincide with the anti-rushing messages of the honest parties in the only round of interactions and, for every $i \in C$,

$$\mathsf{NIZK}'.\mathsf{Verify}\big(\mathsf{urs}, \pi_i, (\mathsf{S}_i, i)\big) = 1.$$

In such cases, the simulator extracts the witnesses from the NIZK proofs of the corrupted parties using $\mathsf{NIZK}'.\mathsf{Extract}$, obtaining the corresponding PRF keys k_i and K_i. By the extractability of NIZK', the procedure is successful with overwhelming probability. If the q-th oracle query does not satisfy the properties described above, the response is generated as in the previous stage.

More precisely, the simulator now samples the nonce $\hat{v}$ and the terms $(\hat{u}_i)_{i\in C}$ uniformly and retrieves the randomness $\hat{r}_i$ used for the generation of $\hat{\mathsf{g}}_i = \mathsf{S}_i(\hat{u}_i, \hat{v})$ for every $i \in C$. Now, it is indeed possible to perform the operation as the PRF key K_i is no longer secret. Moreover, with overwhelming probability, the pair $(\hat{u}_i, \hat{v})$ does not activate the trapdoor in S_i. The values $(\hat{\mathsf{g}}_h)_{h\in H}$ are finally generated by sending $(\hat{\mathsf{g}}_i, \hat{r}_i)_{i\in C}$ to $\mathcal{F}_{\mathsf{NoRush}}$.

At the end, if the anti-rushing messages of the corrupted parties correspond to the q-th oracle query, the simulator retrieves the randomness of the honest parties $(\hat{r}_h)_{h\in H}$ by sending the label of the q-query to the functionality.

Observe that this hybrid is indistinguishable from the previous one by the witness extractability of NIZK'. The formal steps used for the generation of the q-th oracle query become now the following.

1. $\hat{v} \xleftarrow{\$} \{0,1\}^\lambda$
2. $\forall i \in C: \quad \hat{u}_i \xleftarrow{\$} \{0,1\}^{m\cdot\lambda}$
3. $\forall i \in C: \quad (k_i, K_i, w_i) \leftarrow \mathsf{NIZK}'.\mathsf{Extract}\big(\mathsf{urs}, \tau, (\mathsf{S}_i, i), \pi_i\big)$
4. $\forall i \in C: \quad \hat{r}_i \leftarrow F_{K_i}(\hat{u}_i, \hat{v})$
5. $\forall i \in C: \quad \hat{\mathsf{g}}_i \leftarrow \mathsf{SendMsg}(1^\lambda, i; \hat{r}_i)$
6. Send $\big(\mathsf{Query}, (\hat{\mathsf{g}}_i, \hat{r}_i)_{i\in C}\big)$ to $\mathcal{F}_{\mathsf{NoRush}}$, obtaining $\big(\mathsf{id}, (\hat{\mathsf{g}}_h)_{h\in H}\big)$ as a reply.
7. For every $h \in H$, rewrite $\hat{\mathsf{g}}_h$ as an m-bit string $\hat{x}_h$.
8. $\forall h \in H: \quad (\hat{y}^b_{h,j})_{j,b} \leftarrow F'_{k_h}(\hat{v})$
9. $\forall h \in H$ and $j \in [m]: \quad \hat{u}^j_h \leftarrow \hat{y}^{\hat{x}^j_h}_{h,j}$

Hybrid 14. This stage corresponds to the ideal world and it just formalises what has been achieved through the series of hybrids we described above.

At this point, the simulator generates the anti-rushing messages of the honest parties and the URS as in Hybrid 1. Moreover, it replies to all the oracle queries as in Hybrid 13.q, receiving ideal $(\mathsf{g}_h)_{h\in H}$ from the functionality $\mathcal{F}_{\mathsf{NoRush}}$. We recall that each of these samples is associated with a label id, so there is a one-to-one correspondence between labels and oracle queries. When the adversary selects the anti-rushing messages $(\mathsf{S}_i, \pi_i)_{i\in C}$ of the corrupted parties, the simulator sends the label of the corresponding oracle query to the functionality. The latter will take care of outputting the associated randomness to the honest parties. $\square$

Theorem 2.5.3. Suppose that $\mathsf{DS} = (\mathsf{Gen}, \mathsf{Sample})$ is a semi-maliciously secure distributed sampler for the distribution $\mathcal{D}$. Assume that there exists an anti-rusher for $\mathsf{DS.Gen}$. Then, there exists an actively secure distributed sampler for $\mathcal{D}$.

On the novelty of this compiler. Observe that the idea of a compiler converting passive protocols into actively secure ones is not new. The most famous example is GMW [GMW87], which achieves this by adding ZK proofs proving the well-formedness of all the messages in the protocol. The novelty of our construction consists of doing this without increasing the number of rounds. GMW deals with rushing by requiring all the parties to commit to their randomness at the beginning of the protocol and then prove that all the messages in the interaction are consistent with the initial commitments. A passively secure one-round protocol would therefore be compiled, in the best case, into a 2-round one.

Although the techniques were inspired by [HJK+16], this work employs the ideas in a new context, generalising them to multiple players and applying them in multiparty protocols. Observe indeed that [HJK+16] devised the techniques to construct adaptively secure universal samplers. To some extent, we still use them to prevent the adversary from making adaptive choices.

2.6 Public-Key PCFs for Reverse-Samplable Correlations

We now consider the concept of a *distributed correlation sampler*, where the distribution $\mathcal{D}$ produces *private, correlated* outputs $R_1, R_2, \dots, R_n$, where R_i is given only to the i-th party. This can also model the case where the distribution $\mathcal{D}$ has only one output $R = R_1 = \cdots = R_n$, which must be accessible only to the parties that took part in the computation (but not to outsiders; unlike with a distributed sampler).

PCGs and PCFs. The concept of distributed correlation samplers has been previously studied in the form of pseudorandom correlation generators (PCGs) [BCGI18, BCG+19a, BCG+19b, BCG+20b] and pseudorandom correlation functions (PCFs)[BCG+20a, OSY21]. These are tailored to distributions with n outputs, each one addressed to a different player. Specifically, they consist of two algorithms $(\mathsf{Gen}, \mathsf{Eval})$: Gen is used to generate n short correlated seeds or keys, one for each party. Eval is then used to locally expand the keys and non-interactively produce a large amount of correlated randomness, analogously to the non-correlated setting of a PRG (for PCG) or PRF (for PCF).

Both PCGs and PCFs implicitly rely on a trusted dealer for the generation and distribution of the output of Gen, which in practice can be realized using a secure multiparty protocol. The communication overhead of this computation should be small, compared with the amount of correlated randomness obtained from Eval.

If we consider a one-round protocol to distribute the output of Gen, the message of the i-th party and the corresponding randomness r_i act now as a kind of public/private key pair (r_i is necessary to retrieve the i-th output.) Such a primitive is called a *public-key PCF* [OSY21]. Orlandi *et al.* [OSY21] built public-key PCFs for the random OT and vector-OLE correlations based on Paillier encryption with a common reference string (a trusted RSA modulus). In this section, we will build public-key PCFs for general correlations, while avoiding trusted setups.

2.6.1 Correlation Functions and their Properties

Instead of considering singe-output distributions $\mathcal{D}$, we now consider n-output correlations $\mathcal{C}$. We also allow different samples from $\mathcal{C}$ to themselves be correlated by some secret parameters, which allows handling correlations such as vector-OLE and authenticated multiplication triples (where each sample depends on some fixed MAC keys). This is modelled by allowing each party i to input a *master secret* mk_i into $\mathcal{C}$. These additional inputs are independently sampled by each party using an algorithm Secret.

Some example correlations.

Previous works have focussed on a simple class of *additive correlations*, where the outputs $R_1, \dots, R_n$ form an additive secret sharing of values sampled from a distribution. This captures, for instance, oblivious transfer, (vector) oblivious linear evaluation and (authenticated) multiplication triples, which are all useful correlations for secure computation tasks. Vector OLE and authenticated triples are also examples requiring a master secret, which is used to fix a secret scalar or secret MAC keys used to produce samples. Assuming

LWE, we can construct public-key PCFs for any additive correlation [BCG$^+$20a], using homomorphic secret-sharing based on multi-key FHE [DHRW16]. However, we do not know how to build PCFs for broader classes of correlations, except for in the two-party setting and relying on subexponentially secure iO [DHRW16].

As motivation, consider the following important types of non-additive correlations:

- *Pseudorandom secret sharing.* This can be seen as a correlation that samples sharings of uniformly random values under some linear secret sharing scheme. Even for simple t-out-of-n threshold schemes such as Shamir, the best previous construction requires $\binom{n}{t}$ complexity [CDI05].
- *Garbled circuits.* In the two-party setting, one can consider a natural garbled circuit correlation, which for some circuit C, gives a garbling of C to one party, and all pairs of input wire labels to the other party. Having such a correlation allows preprocessing for secure 2-PC, where in the online phase, the parties just use oblivious transfer to transmit the appropriate input wire labels.[3] Similarly, this can be extended to the multi-party setting, by for instance, giving n parties the garbled circuit together with a secret-sharing of the input wire labels.

For garbled circuits, it may also be useful to consider a variant that uses a master secret, if e.g. we want each garbled circuit to be sampled with a fixed offset used in the free-XOR technique [KS08].

Reverse-Samplable Correlations.

The natural way to define a public-key PCF would be a one-round protocol implementing the functionality that samples from the correlation function $\mathcal{C}$ and distributes the outputs. However, Boyle *et al.* [BCG$^+$19b] prove that for PCGs, any construction satisfying this definition in the plain model would require that the messages be at least as long as the randomness generated, which negates one of the main advantages of using a PCF. Following the approach of Boyle *et al.*, in this section we adopt a weaker definition. We require that no adversary can distinguish the real samples of the honest parties from simulated ones which are *reverse sampled* based on the outputs of the corrupted players. This choice restricts the set of correlation functions to those whose outputs are efficiently reverse-samplable[4]. We formalise this property below.

Definition 2.6.1 (Reverse Samplable Correlation Function with Master Secrets). An n-party correlation function with master secrets is a pair of PPT algorithms $(\mathsf{Secret}, \mathcal{C})$ with the following syntax:

- Secret takes as input the security parameter 1^λ and the index of a party $i \in [n]$. It outputs the i-th party's master correlation secret mk_i.
- $\mathcal{C}$ takes as input the security parameter 1^λ and the master secrets $\mathsf{mk}_1, \dots, \mathsf{mk}_n$. It outputs n correlated values $R_1, R_2, \dots, R_n$, one for each party.

We say that $(\mathsf{Secret}, \mathcal{C})$ is reverse samplable if there exists a PPT algorithm $\mathsf{RSample}$ such that, for every set of corrupted parties $C \subsetneq [n]$ and master secrets $(\mathsf{mk}_i)_{i\in[n]}$ and $(\mathsf{mk}'_h)_{h\in H}$ in the image of Secret, no PPT adversary is able to distinguish between $\mathcal{C}(1^\lambda, \mathsf{mk}_1, \mathsf{mk}_2, \dots, \mathsf{mk}_n)$ and

$$\left\{ (R_1, R_2, \dots, R_n) \;\middle|\; \begin{array}{l} \forall i \in C: \quad \mathsf{mk}'_i \leftarrow \mathsf{mk}_i \\ (R'_1, R'_2, \dots, R'_n) \xleftarrow{\$} \mathcal{C}(1^\lambda, \mathsf{mk}'_1, \mathsf{mk}'_2, \dots, \mathsf{mk}'_n) \\ \forall i \in C: \quad R_i \leftarrow R'_i \\ (R_h)_{h\in H} \xleftarrow{\$} \mathsf{RSample}\big(1^\lambda, C, (R_i)_{i\in C}, (\mathsf{mk}_i)_{i\in C}, (\mathsf{mk}_h)_{h\in H}\big) \end{array} \right\}$$

Notice that indistinguishability cannot rely on the secrecy of the master secrets $(\mathsf{mk}_i)_{i\in[n]}$ and $(\mathsf{mk}'_h)_{h\in H}$, since the adversary could know their values. Furthermore, $\mathsf{RSample}$ does not take as input the same master secrets that were used for the generation of the outputs of the corrupted parties. The fact that indistinguishability holds in spite of this implies that the elements $(R_i)_{i\in C}$ leak no information about the master secrets of the honest players.

[3]Note that formally, in the presence of malicious adversaries, preprocessing garbled circuits in this way requires the garbling scheme to be adaptively secure [BHR12].

[4]In the examples above, reverse-samplability is possible for pseudorandom secret-sharing, but not for garbled circuits, since we should not be able to find valid input wire labels when given only a garbled circuit.

$\mathcal{G}_{\text{PCF-Corr}}(\lambda)$

Initialisation.

1. $b \xleftarrow{\$} \{0,1\}$
2. $\forall i \in [n]: \quad (\mathsf{sk}_i, \mathsf{pk}_i) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, i)$
3. $\forall i \in [n]: \quad \mathsf{mk}'_i \xleftarrow{\$} \mathsf{Secret}(1^\lambda, i)$
4. Activate the adversary with input $(1^\lambda, (\mathsf{pk}_i)_{i\in[n]})$.

Repeated querying. On input $(\mathsf{Correlation}, x)$ from the adversary where $x \in \{0,1\}^{l(\lambda)}$, compute

1. $\forall i \in [n]: \quad R_i^0 \leftarrow \mathsf{Eval}(i, \mathsf{pk}_1, \dots, \mathsf{pk}_n, \mathsf{sk}_i, x)$
2. $(R_i^1)_{i\in[n]} \xleftarrow{\$} \mathcal{C}(1^\lambda, \mathsf{mk}'_1, \dots, \mathsf{mk}'_n)$
3. Give $(R_1^b, R_2^b, \dots, R_n^b)$ to the adversary.

Output. The adversary wins if its final output is b.

Figure 2.19: Correctness Game for the Public-Key PCF

2.6.2 Defining Public Key PCFs

We now formalise the definition of public key PCF as it was sketched at the beginning of the section. We start by specifying the syntax, we will then focus our attention on security, in particular against semi-malicious and active adversaries.

Definition 2.6.2 (Public-Key PCF with Master Secrets). A public-key PCF for the n-party correlation function with master secrets $(\mathsf{Secret}, \mathcal{C})$ is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Eval})$ with the following syntax:

- Gen takes as input the security parameter 1^λ and the index of a party $i \in [n]$, and outputs the PCF key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ of the i-th party. Gen needs $L(\lambda)$ bits of randomness.

- Eval takes as input an index $i \in [n]$, n PCF public keys, the i-th PCF private key sk_i and a nonce $x \in \{0,1\}^{l(\lambda)}$. It outputs a value R_i corresponding to the i-th output of $\mathcal{C}$.

Every public-key PCF $(\mathsf{Gen}, \mathsf{Eval})$ for $\mathcal{C}$ induces a one-round protocol $\Pi_{\mathcal{C}}$. This is the natural construction in which every party broadcasts pk_i output by Gen, and then runs Eval on all the parties' messages, its own private key and various nonces.

Definition 2.6.3 (Semi-Maliciously Secure Public-Key PCF for Reverse Samplable Correlation). Let $(\mathsf{Secret}, \mathcal{C})$ be an n-party, reverse samplable correlation function with master secrets. A public-key PCF $(\mathsf{Gen}, \mathsf{Eval})$ for $(\mathsf{Secret}, \mathcal{C})$ is semi-maliciously secure if the following properties are satisfied.

- **Correctness.** No PPT adversary can win the game $\mathcal{G}_{\text{PCF-Corr}}(\lambda)$ (see Figure 2.19) with noticeable advantage.

- **Security.** There exists a PPT extractor $\mathsf{Extract}$ such that for every set of corrupted parties $C \subsetneq [n]$ and corresponding randomness $(\rho_i)_{i\in C}$, no PPT adversary can win the game $\mathcal{G}_{\text{PCF-Sec}}^{C,(\rho_i)_{i\in C}}(\lambda)$ (see Figure 2.20) with noticeable advantage.

Correctness requires that the samples output by the PCF are indistinguishable from those produced by $\mathcal{C}$ even if the adversary receives all the public keys. Security instead states that a semi-malicious adversary learns no information about the samples and the master secrets of the honest players except what can be deduced from the outputs of the corrupted parties themselves.

Like for distributed samplers, the above definition can be adapted to passive security by modifying the security game. Specifically, it would be sufficient to sample the randomness of the corrupted parties inside the game, perhaps relying on a simulator when $b = 1$.

$$\mathcal{G}_{\text{PCF-Sec}}^{C,(\rho_i)_{i\in C}}(\lambda)$$

Initialisation.

1. $b \xleftarrow{\$} \{0,1\}$
2. $\forall h \in H: \quad \rho_h \xleftarrow{\$} \{0,1\}^{L(\lambda)}$
3. $\forall i \in [n]: \quad (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{Gen}(1^\lambda, i;\, \rho_i)$
4. $(\mathsf{mk}_i)_{i\in C} \leftarrow \mathsf{Extract}(C, \rho_1, \rho_2, \ldots, \rho_n)$.
5. $\forall h \in H: \quad \mathsf{mk}'_h \xleftarrow{\$} \mathsf{Secret}(1^\lambda, h)$
6. Activate the adversary with 1^λ and provide it with $(\mathsf{pk}_i)_{i\in[n]}$ and $(\rho_i)_{i\in C}$.

Repeated querying. On input $(\mathsf{Correlation}, x)$ from the adversary where $x \in \{0,1\}^{l(\lambda)}$, compute

1. $\forall i \in [n]: \quad R_i^0 \leftarrow \mathsf{Eval}(i, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathsf{sk}_i, x)$
2. $\forall i \in C: \quad R_i^1 \leftarrow R_i^0$
3. $(R_h^1)_{h\in H} \xleftarrow{\$} \mathsf{RSample}\big(1^\lambda, C, (R_i^1)_{i\in C}, (\mathsf{mk}_i)_{i\in C}, (\mathsf{mk}'_h)_{h\in H}\big)$
4. Give $(R_1^b, R_2^b, \ldots, R_n^b)$ to the adversary.

Output. The adversary wins if its final output is b.

Figure 2.20: Security Game for the Public-Key PCF

In our definition, nonces are adaptively chosen by the adversary; however, in a *weak* PCF [BCG+20a], the nonces are sampled randomly or selected by the adversary ahead of time. We can define a weak public-key PCF similarly, and use the same techniques as Boyle *et al.* [BCG+20a] to convert a weak public-key PCF into a public-key PCF by means of a random oracle.

Active security. We define actively secure public-key PCFs using an ideal functionality, similarly to how we defined actively secure distributed samplers.

Definition 2.6.4 (Actively Secure Public-Key PCF for Reverse Samplable Correlation). Let $(\mathsf{Secret}, \mathcal{C})$ be an n-party reverse samplable correlation function with master secrets. A public-key PCF $(\mathsf{Gen}, \mathsf{Eval})$ for $(\mathsf{Secret}, \mathcal{C})$ is *actively secure* if the corresponding one-round protocol $\Pi_\mathcal{C}$ implements the functionality $\mathcal{F}_\mathcal{C}^{\mathsf{RSample}}$ (see Figure 2.21) against a static and active adversary corrupting up to $n-1$ parties.

Any protocol that implements $\mathcal{F}_\mathcal{C}^{\mathsf{RSample}}$ will require either a CRS or a random oracle; this is inherent for

$$\mathcal{F}_\mathcal{C}^{\mathsf{RSample}}$$

Initialisation. On input Init from every honest party and the adversary, the functionality samples $\mathsf{mk}_h \xleftarrow{\$} \mathsf{Secret}(1^\lambda, h)$ for every $h \in H$ and waits for $(\mathsf{mk}_i)_{i\in C}$ from the adversary.
Correlation. On input a fresh nonce $x \in \{0,1\}^{l(\lambda)}$ from a party P_j, the functionality waits for $(R_i)_{i\in C}$ from the adversary. Then, it computes

$$(R_h)_{h\in H} \xleftarrow{\$} \mathsf{RSample}\big(1^\lambda, C, (R_i)_{i\in C}, (\mathsf{mk}_i)_{i\in C}, (\mathsf{mk}_h)_{h\in H}\big),$$

sends R_j to P_j and stores $\big(x, (R_i)_{i\in[n]}\big)$. If x has already been queried, the functionality retrieves the stored tuple $\big(x, (R_i)_{i\in[n]}\big)$ and outputs R_j to P_j.

Figure 2.21: The Actively Secure Public-Key PCF Functionality for Reverse Samplable Correlation

meaningful correlation functions, since the simulator needs to retrieve the values $(R_i)_{i\in C}$ in order to forward them to $\mathcal{F}_{\mathcal{C}}^{\mathsf{RSample}}$. Therefore, some kind of trapdoor is needed.

Notice also that the algorithm $\mathsf{RSample}$ takes as input the master secrets of the corrupted parties. We can therefore assume that whenever the values $(R_i)_{i\in C}$ chosen by the adversary are inconsistent with $(\mathsf{mk}_i)_{i\in C}$ or with $\mathcal{C}$ itself, the output of the reverse sampler is $\perp$. As a consequence, an actively secure public-key PCF must not allow the corrupted parties to select these irregular outputs; otherwise distinguishing between real world and ideal world would be trivial.

2.6.3 Public-Key PCF with Trusted Setup

We will build our semi-maliciously secure public-key PCF by first relying on a trusted setup and then removing it by means of a distributed sampler. A public-key PCF with trusted setup is defined by Definition 2.6.2 to include an algorithm Setup that takes as input the security parameter 1^λ and outputs a CRS. The CRS is then provided as an additional input to the evaluation algorithm Eval, but not to the generation algorithm Gen. (If Gen required the CRS, then substituting Setup with a distributed sampler would give us a two-round protocol, not a one-round protocol.)

We say that a public-key PCF with trusted setup is semi-maliciously secure if it satisfies Definition 2.6.3, after minor tweaks to the games $\mathcal{G}_{\text{PCF-Corr}}(\lambda)$ and $\mathcal{G}_{\text{PCF-Sec}}^{C,(\rho_i)_{i\in C}}(\lambda)$ to account for the modified syntax. Notice that in the latter, the extractor needs to be provided with the CRS but not with the randomness used to produce it. If that was not the case, we would not be able to use a distributed sampler to remove the CRS.

Definition of public key PCF with trusted setup. We formalise the concept of public key PCF with trusted setup describing its syntax and security properties. The definitions closely resemble those of public key PCF (see Definition 2.6.2 and Definition 2.6.3). The only difference consists in the CRS generated by the setup algorithm, which is now provided as an additional input to Eval and $\mathsf{Extract}$ and whose value is always disclosed to the adversary in the security games.

Definition 2.6.5 (Public Key PCF with Trusted Setup). A public key PCF with trusted setup for the n-party correlation function with master secret $(\mathsf{Secret}, \mathcal{C})$ is a triple of PPT algorithms $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Eval})$ with the following syntax:

- Setup takes as input the security parameter 1^λ and outputs a CRS S.
- Gen takes as input the security parameter 1^λ and the index of a party $i \in [n]$, outputting the PCF key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ of the i-th party. The algorithm needs $L(\lambda)$ bits of randomness.
- Eval takes as input an index $i \in [n]$, a CRS S, n PCF public keys, one for each party, the PCF private key sk_i of the i-th party and a nonce $x \in \{0,1\}^{l(\lambda)}$. The output is a value R_i corresponding to the i-th output of $\mathcal{C}$.

Definition 2.6.6 (Semi-Maliciously Secure Public Key PCF with Trusted Setup for Reverse Samplable Correlation). Let $(\mathsf{Secret}, \mathcal{C})$ be an n-party, reverse samplable correlation function with master secret. A public key PCF with trusted setup $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Eval})$ for $(\mathsf{Secret}, \mathcal{C})$ is semi-maliciously secure if the following properties are satisfied.

- **Correctness.** No PPT adversary can win the game $\mathcal{G}_{\text{SetupCorr}}(\lambda)$ (see Figure 2.22) with noticeable advantage.
- **Security.** There exists a PPT extractor $\mathsf{Extract}$ such that for every set of corrupted parties $C \subsetneq [n]$ and corresponding randomness $(\rho_i)_{i\in C}$, no PPT adversary can win the game $\mathcal{G}_{\text{SetupSec}}^{C,(\rho_i)_{i\in C}}(\lambda)$ (see Figure 2.23) with noticeable advantage.

$$\mathcal{G}_{\text{SetupCorr}}(\lambda)$$

Initialisation.

1. $b \overset{\$}{\leftarrow} \{0,1\}$
2. $S \overset{\$}{\leftarrow} \mathsf{Setup}(1^\lambda)$
3. $\forall i \in [n]: \quad (\mathsf{sk}_i, \mathsf{pk}_i) \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda, i)$
4. $\forall i \in [n]: \quad \mathsf{mk}'_i \overset{\$}{\leftarrow} \mathsf{Secret}(1^\lambda, i)$
5. Activate the adversary with 1^λ and provide it with $(\mathsf{pk}_i)_{i\in[n]}$ and S.

Repeated querying. On input $(\mathsf{Correlation}, x)$ from the adversary where $x \in \{0,1\}^{l(\lambda)}$, compute

1. $\forall i \in [n]: \quad R_i^0 \leftarrow \mathsf{Eval}(i, S, \mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n, \mathsf{sk}_i, x)$
2. $(R_i^1)_{i\in[n]} \overset{\$}{\leftarrow} \mathcal{C}(1^\lambda, \mathsf{mk}'_1, \ldots, \mathsf{mk}'_n)$
3. Give $(R_1^b, R_2^b, \ldots, R_n^b)$ to the adversary.

Output. The adversary wins if its final output is b.

Figure 2.22: Correctness Game for Public Key PCFs with Setup

$$\mathcal{G}_{\text{SetupSec}}^{C,(\rho_i)_{i\in C}}(\lambda)$$

Initialisation.

1. $b \overset{\$}{\leftarrow} \{0,1\}$
2. $S \overset{\$}{\leftarrow} \mathsf{Setup}(1^\lambda)$
3. $\forall h \in H: \quad \rho_i \overset{\$}{\leftarrow} \{0,1\}^{L(\lambda)}$
4. $\forall i \in [n]: \quad (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{Gen}(1^\lambda, i;\, \rho_i)$
5. $(\mathsf{mk}_i)_{i\in C} \leftarrow \mathsf{Extract}(C, S, \rho_1, \rho_2, \ldots, \rho_n)$.
6. $\forall h \in H: \quad \mathsf{mk}'_h \overset{\$}{\leftarrow} \mathsf{Secret}(1^\lambda, h)$
7. Activate the adversary with 1^λ and provide it with $(\mathsf{pk}_i)_{i\in[n]}$, S and $(\rho_i)_{i\in C}$.

Repeated querying. On input $(\mathsf{Correlation}, x)$ from the adversary where $x \in \{0,1\}^{l(\lambda)}$, compute

1. $\forall i \in [n]: \quad R_i^0 \leftarrow \mathsf{Eval}(i, S, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathsf{sk}_i, x)$
2. $\forall i \in C: \quad R_i^1 \leftarrow R_i^0$
3. $(R_h^1)_{h\in H} \overset{\$}{\leftarrow} \mathsf{RSample}\big(1^\lambda, C, (R_i^1)_{i\in C}, (\mathsf{mk}_i)_{i\in C}, (\mathsf{mk}'_h)_{h\in H}\big)$
4. Give $(R_1^b, R_2^b, \ldots, R_n^b)$ to the adversary.

Output. The adversary wins if its final output is b.

Figure 2.23: Security Game for Public Key PCFs with Setup

Our public-key PCF with trusted setup. Our construction is based once again on iO. The key of every party i is a simple PKE pair $(\mathsf{sk}_i, \mathsf{pk}_i)$. The generation of the correlated samples and their distribution is handled by the CRS, which is an obfuscated program. Specifically, the latter takes as input the public keys of the parties and a nonce $x \in \{0,1\}^{l(\lambda)}$. After generating the master secrets $\mathsf{mk}_1, \mathsf{mk}_2, \dots, \mathsf{mk}_n$ using Secret and the correlated samples $R_1, R_2, \dots, R_n$ using $\mathcal{C}$, the program protects their privacy by encrypting them under the provided public keys. Specifically, R_i and mk_i are encrypted using pk_i, making the i-th party the only one able to retrieve the underlying plaintext.

The randomness used for the generation of the samples, the master secrets and the encryption is produced by means of two puncturable PRF keys k and K, known to the CRS program. The CRS program is equipped with two keys: k and K. The first one is used to generate the master secrets; the input to the PRF is the sequence of all public keys $(\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n)$). The master secrets remain the same if the nonce x varies. The second PRF key is used to generate the randomness fed into $\mathcal{C}$ and the encryption algorithm; here, the PRF input consists of all the program inputs. As a result, any slight change in the inputs leads to completely unrelated ciphertexts and samples.

On the size of the nonce space. Unfortunately, in order to obtain semi-maliciously security, we need to assume that the nonce space is of polynomial size. In the security proof, we need to change the behaviour of the CRS program for all nonces. This is due to the fact that we cannot rely on the reverse samplability of the correlation function as long as the program contains information about the real samples of the honest players. If the number of nonces is exponential, our security proof would rely on a non-polynomial number of hybrids and therefore we would need to assume the existence of sub-exponentially secure primitives.

The formal description of our solution. Our public-key PCF with trusted setup for $(\mathsf{Secret}, \mathcal{C})$ is described in Figure 2.24 together with the program $\mathcal{P}_{\mathsf{CG}}$ used as a CRS.

Our solution relies on an IND-CPA PKE scheme $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and two puncturable PRFs F and F'. We assume that the output of the first one is naturally split into $n+1$ blocks, the initial one as big as the randomness needed by $\mathcal{C}$, the remaining ones the same size as the random tape of $\mathsf{PKE.Enc}$. We also assume that the output of F' is split into n blocks as big as the randomness used by Secret.

Theorem 2.6.7 (Public Key PCFs with Trusted Setup). Let $(\mathsf{Secret}, \mathcal{C})$ be an n-party, reverse samplable correlation function with master secrets. If $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is an IND-CPA PKE scheme, iO is an indistinguishability obfuscator, (F, Punct) and (F', Punct') are puncturable PRFs with the properties described above and $l(\lambda)$ is $\mathsf{polylog}(\lambda)$, the construction presented in Figure 2.24 is a semi-maliciously secure public-key PCF with trusted setup for $(\mathsf{Secret}, \mathcal{C})$.

Furthermore, if PKE, iO, (F, Punct) and (F', Punct') are sub-exponentially secure, the public-key PCF with trusted setup is semi-maliciously secure even if $l(\lambda)$ is $\mathsf{poly}(\lambda)$.

In both cases, the size of the CRS and the PCF keys is $\mathsf{poly}(l)$.

Proof. We show that the public key PCF with trusted setup described in Figure 2.24 is semi-maliciously secure. We prove both correctness and security in one go. As a matter of fact, correctness can be regarded as the special case of security in which $C = \emptyset$. Observe that in such case, we can always assume that $\mathsf{RSample}$ samples directly from $\mathcal{C}$ using randomly chosen master secrets $(\mathsf{mk}'_i)_{i \in [n]}$.

We proceed by a sequence of 12 indistinguishable hybrids (some of them repeated for every possible nonce value) going from the real world (Hybrid 0) to the ideal world (Hybrid 12). The size of the nonce space will affect the proof only on the number of reductions needed. Specifically, the number of hybrids will be polynomial if and only if the cardinality of the nonce space is polynomial. In the other cases, we will need to assume the existence of sub-exponentially secure primitives.

We always assume that the PRF keys k and K hard-coded into the correlation generation program CGP are randomly sampled in $\{0,1\}^\lambda$. Moreover, in every stage, we assume that the PKE pairs of the parties are all generated according to the protocol, using the randomness parametrising the game in the case of the corrupted players. We denote the i-th pair by $(\hat{\mathsf{sk}}_i, \hat{\mathsf{pk}}_i)$.

Hybrid 0. This is the initial hybrid and corresponds to the execution of the security game with $b = 0$. The challenger creates the correlation generation program by obfuscating $\mathcal{P}_{\mathsf{CG}}$ (see Figure 2.24). Also the

Public-Key PCF with Trusted Setup

```
$\mathsf{Setup}(1^\lambda)$
    1. $k \xleftarrow{\$} \{0,1\}^\lambda$
    2. $K \xleftarrow{\$} \{0,1\}^\lambda$
    3. Output $\mathsf{CGP} \xleftarrow{\$} \mathsf{iO}\big(1^\lambda, \mathcal{P}_{\mathsf{CG}}[k,K]\big)$
$\mathsf{Gen}(1^\lambda, i)$
    1. Output $(\mathsf{sk}_i, \mathsf{pk}_i) \xleftarrow{\$} \mathsf{PKE.Gen}(1^\lambda)$
$\mathsf{Eval}(i, \mathsf{CGP}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathsf{sk}_i, x)$
    1. $(c_1, c_2, \ldots, c_n) \leftarrow \mathsf{CGP}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n, x)$
    2. $(R_i, \mathsf{mk}_i) \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_i, c_i)$
    3. Output $R_i$.
```

$\mathcal{P}_{\mathsf{CG}}[k,K]$

Hard-coded. Two puncturable PRF keys k and K.
Input. n public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ and a nonce $x \in \{0,1\}^{l(\lambda)}$.

```
1. $(r, r_1, r_2, \ldots, r_n) \leftarrow F_K(\mathsf{pk}_1, \ldots, \mathsf{pk}_n, x)$.
2. $(s_1, s_2, \ldots, s_n) \leftarrow F'_k(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$
3. $\forall i \in [n]: \quad \mathsf{mk}_i \leftarrow \mathsf{Secret}(1^\lambda, i; s_i)$
4. $(R_1, R_2, \ldots, R_n) \leftarrow \mathcal{C}(1^\lambda, \mathsf{mk}_1, \ldots, \mathsf{mk}_n; r)$
5. $\forall i \in [n]: \quad c_i \leftarrow \mathsf{PKE.Enc}\big(\mathsf{pk}_i, (R_i, \mathsf{mk}_i); r_i\big)$
6. Output $c_1, c_2, \ldots, c_n$.
```

Figure 2.24: A Public-Key PCF with Trusted Setup

```
$\mathcal{P}^1_{\mathsf{CG}}[k, K, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}]$

**Hard-coded.** The puncturable PRF keys $k$ and $K$, the public keys $(\hat{\mathsf{pk}}_i)_{i\in[n]}$ and the master secrets $(\hat{\mathsf{mk}}_i)_{i\in[n]}$.
**Input.** A nonce $x \in \{0,1\}^{l(\lambda)}$ and $n$ public keys $\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n$.

  1. $(r, r_1, r_2, \dots, r_n) \leftarrow F_K(\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n, x)$.
  2. If $\mathsf{pk}_i = \hat{\mathsf{pk}}_i$ for every $i \in [n]$, set $\mathsf{mk}_i \leftarrow \hat{\mathsf{mk}}_i$ for every $i \in [n]$.
  3. Otherwise, perform the following operations
      (a) $(s_1, s_2, \dots, s_n) \leftarrow F'_k(\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n)$
      (b) $\forall i \in [n]: \quad \mathsf{mk}_i \leftarrow \mathsf{Secret}(1^\lambda, i;\, s_i)$
  4. $(R_1, R_2, \dots, R_n) \leftarrow \mathcal{C}(1^\lambda, \mathsf{mk}_1, \mathsf{mk}_2, \dots, \mathsf{mk}_n;\, r)$
  5. $\forall i \in [n]: \quad c_i \leftarrow \mathsf{PKE.Enc}\big(\mathsf{pk}_i, (R_i, \mathsf{mk}_i);\, r_i\big)$
  6. Output $c_1, c_2, \dots, c_n$.
```

Figure 2.25: The Correlation Generation Program

keys of the honest parties are generated following the protocol. Those of the corrupted players are instead derived using the randomness parametrising the game. Finally, the challenger replies to all the sampling queries using Eval.

Hybrid 1. In this hybrid, we puncture the PRF key k in the list of public keys of the players. We also store, in the correlation generation program CGP, the master secrets corresponding to the punctured position and we use them to compute the output when the public keys of the parties are fed into it. In this way, the input-output behaviour of CGP does not change with respect to the previous hybrid. Therefore, indistinguishability follows from the security of obfuscation.

The formal steps performed by the challenger for the generation of CGP are now the following (the red text highlights what changed since the last stage).

1. $\hat{k} \leftarrow \mathsf{Punct}'\big(k, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n)\big)$
2. $(\hat{s}_1, \hat{s}_2, \dots, \hat{s}_n) \leftarrow F'_k(\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n)$
3. $\forall i \in [n]: \quad \hat{\mathsf{mk}}_i \leftarrow \mathsf{Secret}(1^\lambda, i;\, \hat{s}_i)$
4. $\mathsf{CGP} \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}^1_{\mathsf{CG}}[\hat{k}, K, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}])$ (see Figure 2.25)

Hybrid 2. In this hybrid, we change how we produce the master secrets $(\hat{\mathsf{mk}}_i)_{i\in[n]}$. Specifically, instead of generating the randomness of Secret by means of F'_k, we sample it uniformly. By the security of the puncturable PRF, this hybrid is indistinguishable from the previous one.

Formally, the challenger generates CGP as follows.

1. $\hat{k} \leftarrow \mathsf{Punct}'\big(k, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n)\big)$
2. $\forall i \in [n]: \quad \hat{\mathsf{mk}}_i \xleftarrow{\$} \mathsf{Secret}(1^\lambda, i)$
3. $\mathsf{CGP} \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}^1_{\mathsf{CG}}[\hat{k}, K, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}])$ (see Figure 2.25)

$\mathcal{P}_{\mathsf{CG}}^{\hat{x},2}[k, K, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}, (\mathsf{mk}'_h)_{h\in H}, H, \hat{x}]$

Hard-coded. The puncturable PRF keys k and K, the public keys $(\hat{\mathsf{pk}}_i)_{i\in[n]}$ and the master secrets $(\hat{\mathsf{mk}}_i)_{i\in[n]}$ and $(\mathsf{mk}'_i)_{i\in H}$, the set of honest parties H and the nonce $\hat{x}$.
Input. A nonce $x \in \{0,1\}^{l(\lambda)}$ and n public keys $\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n$.

1. $(r, r_1, r_2, \dots, r_n) \leftarrow F_K(\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n, x)$.
2. If $\mathsf{pk}_i = \hat{\mathsf{pk}}_i$ for every $i \in [n]$ and $x \geq_{\mathsf{lex}} \hat{x}$, set $\mathsf{mk}_i \leftarrow \hat{\mathsf{mk}}_i$ for every $i \in [n]$.
3. If $\mathsf{pk}_i = \hat{\mathsf{pk}}_i$ for every $i \in [n]$ and $x <_{\mathsf{lex}} \hat{x}$, set $\mathsf{mk}_h \leftarrow \mathsf{mk}'_h$ for every $h \in H$ and $\mathsf{mk}_i \leftarrow \hat{\mathsf{mk}}_i$ for every $i \in C$.
4. Otherwise, perform the following operations
   (a) $(s_1, s_2, \dots, s_n) \leftarrow F'_k(\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n)$
   (b) $\forall i \in [n]: \quad \mathsf{mk}_i \leftarrow \mathsf{Secret}(1^\lambda, i;\, s_i)$
5. $(R_1, R_2, \dots, R_n) \leftarrow \mathcal{C}(1^\lambda, \mathsf{mk}_1, \mathsf{mk}_2, \dots, \mathsf{mk}_n;\, r)$
6. $\forall i \in [n]: \quad c_i \leftarrow \mathsf{PKE.Enc}\big(\mathsf{pk}_i, (R_i, \mathsf{mk}_i);\, r_i\big)$
7. Output $c_1, c_2, \dots, c_n$.

Figure 2.26: The Correlation Generation Program

We now consider the nonce space $\{0,1\}^{l(\lambda)}$ and we order it using the lexicographical order $<_{\mathsf{lex}}$. Let ϵ denote the minimum and Ω the maximum. We apply the series of hybrids from 3 to 10 for every nonce $\hat{x}$, starting from ϵ and following the lexicographical order.

Hybrid 3.$\hat{x}$. In this hybrid, the challenger samples additional master secrets $(\mathsf{mk}'_h)_{h\in H}$ for the honest parties and hard-codes them into CGP along with $\hat{x}$ and H. When the nonce x input in CGP is strictly smaller than $\hat{x}$ and the provided public keys coincide with the ones of the parties, the program generates the samples $(R_i)_{i\in[n]}$ substituting $\hat{\mathsf{mk}}_h$ with mk'_h for every $h \in H$. Furthermore, when the nonce x is strictly smaller than $\hat{x}$, the challenger replies to the correlation queries using $\mathsf{RSample}$, providing it with $(\hat{\mathsf{mk}}_i)_{i\in[n]}$.

Observe that for $\hat{x} = \epsilon$, the input-output behaviour of CGP has not changed with respect to Hybrid 2. Moreover, the challenger never uses $\mathsf{RSample}$ to reply to the correlation queries. If instead $\hat{x} \neq \epsilon$, we will see that the input-output behaviour of CGP has not changed with respect to the previous hybrid either and the challenger replies to the correlation queries as it did before. We conclude that indistinguishability holds in both cases due to the security of the obfuscator.

The formal steps used by the challenger for the generation of CGP are now the following.

1. $\hat{k} \leftarrow \mathsf{Punct}'\big(k, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n)\big)$
2. $\forall i \in [n]: \quad \hat{\mathsf{mk}}_i \xleftarrow{\$} \mathsf{Secret}(1^\lambda, i)$
3. $\forall h \in H: \quad \mathsf{mk}'_h \xleftarrow{\$} \mathsf{Secret}(1^\lambda, h)$
4. $\mathsf{CGP} \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}_{\mathsf{CG}}^{\hat{x},2}[\hat{k}, K, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}, (\mathsf{mk}'_h)_{h\in H}, H, \hat{x}])$ (see Figure 2.26)

The reply to $(\mathsf{Correlation}, x)$ when $x <_{\mathsf{lex}} \hat{x}$ is instead computed as follows.

1. $(c_i)_{i\in[n]} \leftarrow \mathsf{CGP}(\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n, x)$
2. $\forall i \in C: \quad (R_i, \mathsf{mk}_i) \leftarrow \mathsf{PKE.Dec}(\hat{\mathsf{sk}}_i, c_i)$

$\mathcal{P}_{\mathsf{CG}}^{\hat{x},3}[k, K, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}, (\mathsf{mk}'_h)_{h\in H}, H, \hat{x}, (\hat{c}_i)_{i\in[n]}]$

Hard-coded. The puncturable PRF keys k and K, the public keys $(\hat{\mathsf{pk}}_i)_{i\in[n]}$ and the master secrets $(\hat{\mathsf{mk}}_i)_{i\in[n]}$ and $(\mathsf{mk}'_i)_{i\in H}$, the set of honest parties H, the nonce $\hat{x}$ and the ciphertexts $(\hat{c}_i)_{i\in[n]}$.
Input. A nonce $x \in \{0,1\}^{l(\lambda)}$ and n public keys $\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n$.

```
1. If $\mathsf{pk}_i = \hat{\mathsf{pk}}_i$ for every $i \in [n]$ and $x = \hat{x}$, output $(\hat{c}_i)_{i\in[n]}$.
2. $(r, r_1, r_2, \dots, r_n) \leftarrow F_K(\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n, x)$.
3. If $\mathsf{pk}_i = \hat{\mathsf{pk}}_i$ for every $i \in [n]$ and $x \geq_{\mathsf{lex}} \hat{x}$, set $\mathsf{mk}_i \leftarrow \hat{\mathsf{mk}}_i$ for every $i \in [n]$.
4. If $\mathsf{pk}_i = \hat{\mathsf{pk}}_i$ for every $i \in [n]$ and $x <_{\mathsf{lex}} \hat{x}$, set $\mathsf{mk}_h \leftarrow \mathsf{mk}'_h$ for every $h \in H$ and $\mathsf{mk}_i \leftarrow \hat{\mathsf{mk}}_i$ for
   every $i \in C$.
5. Otherwise, perform the following operations
   (a) $(s_1, s_2, \dots, s_n) \leftarrow F'_k(\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n)$
   (b) $\forall i \in [n]: \quad \mathsf{mk}_i \leftarrow \mathsf{Secret}(1^\lambda, i;\, s_i)$
6. $(R_1, R_2, \dots, R_n) \leftarrow \mathcal{C}(1^\lambda, \mathsf{mk}_1, \mathsf{mk}_2, \dots, \mathsf{mk}_n;\, r)$
7. $\forall i \in [n]: \quad c_i \leftarrow \mathsf{PKE.Enc}\big(\mathsf{pk}_i, (R_i, \mathsf{mk}_i);\, r_i\big)$
8. Output $c_1, c_2, \dots, c_n$.
```

Figure 2.27: The Correlation Generation Program

3. $(R_h)_{h\in H} \xleftarrow{\$} \mathsf{RSample}\big(1^\lambda, C, (R_i)_{i\in C}, (\hat{\mathsf{mk}}_i)_{i\in C}, (\hat{\mathsf{mk}}_h)_{h\in H}\big)$

Hybrid 4.$\hat{x}$. In this hybrid, we puncture the PRF key K in the tuple consisting of the public keys of the parties and the nonce $\hat{x}$, i.e. $(\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n, \hat{x})$. Moreover, we program CGP to output the appropriate ciphertexts when the punctured position is given as input. Since the input-output behaviour of the program has not changed with respect to the previous hybrid, indistinguishability follows from the security of iO.

The formal procedure used by the challenger for the generation of CGP is now the following.

1. $\hat{k} \leftarrow \mathsf{Punct}'\big(k, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n)\big)$
2. $\forall i \in [n]: \quad \hat{\mathsf{mk}}_i \xleftarrow{\$} \mathsf{Secret}(1^\lambda, i)$
3. $\forall h \in H: \quad \mathsf{mk}'_h \xleftarrow{\$} \mathsf{Secret}(1^\lambda, h)$
4. $\hat{K} \leftarrow \mathsf{Punct}\big(K, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n, \hat{x})\big)$
5. $(\hat{r}, \hat{r}_1, \hat{r}_2, \dots, \hat{r}_n) \leftarrow F_K(\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n, \hat{x})$
6. $(\hat{R}_i)_{i\in[n]} \leftarrow \mathcal{C}(1^\lambda, \hat{\mathsf{mk}}_1, \hat{\mathsf{mk}}_2, \dots, \hat{\mathsf{mk}}_n; \hat{r})$
7. $\forall i \in [n]: \quad \hat{c}_i \leftarrow \mathsf{PKE.Enc}\big(\hat{\mathsf{pk}}_i, (\hat{R}_i, \hat{\mathsf{mk}}_i); \hat{r}_i\big)$
8. $\mathsf{CGP} \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}_{\mathsf{CG}}^{\hat{x},3}[\hat{k}, \hat{K}, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}, (\mathsf{mk}'_h)_{h\in H}, H, \hat{x}, (\hat{c}_i)_{i\in[n]}])$ (see Figure 2.27)

Hybrid 5.$\hat{x}$. In this hybrid, we change how we generate the samples $(\hat{R}_i)_{i\in[n]}$ and encrypt them. Specifically, instead of producing the randomness using the PRF F_K, we sample it uniformly. Observe that this hybrid and the previous one are indistinguishable by the security of the puncturable PRF F.

The formal procedure used by the challenger to generate CGP becomes the following.

1. $\hat{k} \leftarrow \mathsf{Punct}'\big(k, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n)\big)$
2. $\forall i \in [n]: \quad \hat{\mathsf{mk}}_i \xleftarrow{\$} \mathsf{Secret}(1^\lambda, i)$
3. $\forall h \in H: \quad \mathsf{mk}'_h \xleftarrow{\$} \mathsf{Secret}(1^\lambda, h)$
4. $\hat{K} \leftarrow \mathsf{Punct}\big(K, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n, \hat{x})\big)$
5. $(\hat{R}_i)_{i\in[n]} \xleftarrow{\$} \mathcal{C}(1^\lambda, \hat{\mathsf{mk}}_1, \hat{\mathsf{mk}}_2, \dots, \hat{\mathsf{mk}}_n)$
6. $\forall i \in [n]: \quad \hat{c}_i \xleftarrow{\$} \mathsf{PKE.Enc}\big(\hat{\mathsf{pk}}_i, (\hat{R}_i, \hat{\mathsf{mk}}_i)\big)$
7. $\mathsf{CGP} \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}^{\hat{x},3}_{\mathsf{CG}}[\hat{k}, \hat{K}, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}, (\mathsf{mk}'_h)_{h\in H}, H, \hat{x}, (\hat{c}_i)_{i\in[n]}])$ (see Figure 2.27)

Hybrid 6.$\hat{x}$. In this hybrid, the challenger replies to the query $(\mathsf{Correlation}, \hat{x})$ directly sending the values $(\hat{R}_i)_{i\in[n]}$ sampled by $\mathcal{C}$ during the generation of PCG. This hybrid is indistinguishable from the previous one by the correctness of iO and PKE.

Hybrid 7.$\hat{x}$. In this hybrid, we change how we produce the samples $(\hat{R}_i)_{i\in[n]}$. Specifically, we first generate $(R'_i)_{i\in[n]}$ using the correlation function $\mathcal{C}$ and substituting mk'_h to $\hat{\mathsf{mk}}_h$ for every $h \in H$. Then, we obtain $(\hat{R}_h)_{h\in H}$ by feeding the original master secrets $(\hat{\mathsf{mk}}_i)_{i\in[n]}$ and $(R'_i)_{i\in C}$ into $\mathsf{RSample}$. Finally, we set $\hat{R}_i$ to R'_i for every $i \in C$. Observe that this hybrid is indistinguishable from the previous one by the reverse samplability of $(\mathsf{Secret}, \mathcal{C})$.

The formal steps performed by the challenger for the generation of CGP become the following.

1. $\hat{k} \leftarrow \mathsf{Punct}'\big(k, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n)\big)$
2. $\hat{K} \leftarrow \mathsf{Punct}\big(K, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n, \hat{x})\big)$
3. $\forall i \in [n]: \quad \hat{\mathsf{mk}}_i \xleftarrow{\$} \mathsf{Secret}(1^\lambda, i)$
4. $\forall h \in H: \quad \mathsf{mk}'_h \xleftarrow{\$} \mathsf{Secret}(1^\lambda, h)$
5. $\forall i \in C: \quad \mathsf{mk}'_i \leftarrow \hat{\mathsf{mk}}_i$
6. $(R'_i)_{i\in[n]} \xleftarrow{\$} \mathcal{C}(1^\lambda, \mathsf{mk}'_1, \mathsf{mk}'_2, \dots, \mathsf{mk}'_n)$
7. $(\hat{R}_h)_{h\in H} \xleftarrow{\$} \mathsf{RSample}\big(1^\lambda, C, (R'_i)_{i\in C}, (\hat{\mathsf{mk}}_i)_{i\in C}, (\hat{\mathsf{mk}}_h)_{h\in H}\big)$
8. $\forall i \in C: \quad \hat{R}_i \leftarrow R'_i$
9. $\forall i \in [n]: \quad \hat{c}_i \xleftarrow{\$} \mathsf{PKE.Enc}\big(\hat{\mathsf{pk}}_i, (\hat{R}_i, \hat{\mathsf{mk}}_i)\big)$
10. $\mathsf{CGP} \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}^{\hat{x},3}_{\mathsf{CG}}[\hat{k}, \hat{K}, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}, (\mathsf{mk}'_h)_{h\in H}, H, \hat{x}, (\hat{c}_i)_{i\in[n]}])$ (see Figure 2.27)

Hybrid 8.$\hat{x}$. In this stage, instead of hard-coding into CGP the encryption of $(\hat{R}_h, \hat{\mathsf{mk}}_h)$, for every $h \in H$, we store the encryption of (R'_h, mk'_h). The challenger, however, replies to the query $(\mathsf{Correlation}, \hat{x})$ by feeding the actual samples of the corrupted parties and the original master secrets $(\hat{\mathsf{mk}}_i)_{i\in[n]}$ into $\mathsf{RSample}$. Observe that the challenger does not need to know the secret-keys of the honest parties to reply to the queries $(\mathsf{Correlation}, x)$ with $x \neq \hat{x}$. The knowledge of the keys K and k permits indeed to recompute the samples $(R_i)_{i\in[n]}$. This fact allows us to reduce the indistinguishability between Hybrid 7 and 8 to the IND-CPA security of PKE.

The formal steps performed by the challenger for the generation of CGP become the following.

1. $\hat{k} \leftarrow \mathsf{Punct}'\big(k, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n)\big)$

2. $\hat{K} \leftarrow \mathsf{Punct}\big(K, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n, \hat{x})\big)$

3. $\forall i \in [n]: \quad \hat{\mathsf{mk}}_i \xleftarrow{\$} \mathsf{Secret}(1^\lambda, i)$

4. $\forall h \in H: \quad \mathsf{mk}'_h \xleftarrow{\$} \mathsf{Secret}(1^\lambda, h)$

5. $\forall i \in C: \quad \mathsf{mk}'_i \leftarrow \hat{\mathsf{mk}}_i$

6. $(R'_i)_{i\in[n]} \xleftarrow{\$} \mathcal{C}(1^\lambda, \mathsf{mk}'_1, \mathsf{mk}'_2, \dots, \mathsf{mk}'_n)$

7. $\forall i \in [n]: \quad \hat{c}_i \xleftarrow{\$} \mathsf{PKE.Enc}\big(\hat{\mathsf{pk}}_i, (R'_i, \mathsf{mk}'_i)\big)$

8. $\mathsf{CGP} \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}_{\mathsf{CG}}^{\hat{x},3}[\hat{k}, \hat{K}, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}, (\mathsf{mk}'_h)_{h\in H}, H, \hat{x}, (\hat{c}_i)_{i\in[n]}])$ (see Figure 2.27)

Furthermore, the response to $(\mathsf{Correlation}, \hat{x})$ is now computed as follows.

1. $(c_i)_{i\in[n]} \leftarrow \mathsf{CGP}(\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n, \hat{x})$

2. $\forall i \in C: \quad (R_i, \mathsf{mk}_i) \leftarrow \mathsf{PKE.Dec}(\hat{\mathsf{sk}}_i, c_i)$

3. $(R_h)_{h\in H} \xleftarrow{\$} \mathsf{RSample}\big(1^\lambda, C, (R_i)_{i\in C}, (\hat{\mathsf{mk}}_i)_{i\in C}, (\hat{\mathsf{mk}}_h)_{h\in H}\big)$

Hybrid 9.$\hat{x}$. In this hybrid, we change how we generate the samples $(R'_i)_{i\in[n]}$ and we encrypt them. Specifically, instead of sampling the randomness for $\mathcal{C}$ and $\mathsf{PKE.Enc}$ uniformly, we rely on the output of the PRF F_K when its nonce is the position where we had previously punctured. Indistinguishability from the previous stage holds by the security of the puncturable PRF F.

The formal steps performed by the challenger for the generation of CGP become the following.

1. $\hat{k} \leftarrow \mathsf{Punct}'\big(k, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n)\big)$

2. $\hat{K} \leftarrow \mathsf{Punct}\big(K, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n, \hat{x})\big)$

3. $(\hat{r}, \hat{r}_1, \hat{r}_2, \dots, \hat{r}_n) \leftarrow F_K(\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n, \hat{x})$

4. $\forall i \in [n]: \quad \hat{\mathsf{mk}}_i \xleftarrow{\$} \mathsf{Secret}(1^\lambda, i)$

5. $\forall h \in H: \quad \mathsf{mk}'_h \xleftarrow{\$} \mathsf{Secret}(1^\lambda, h)$

6. $\forall i \in C: \quad \mathsf{mk}'_i \leftarrow \hat{\mathsf{mk}}_i$

7. $(R'_i)_{i\in[n]} \leftarrow \mathcal{C}(1^\lambda, \mathsf{mk}'_1, \mathsf{mk}'_2, \dots, \mathsf{mk}'_n;\ \hat{r})$

8. $\forall i \in [n]: \quad \hat{c}_i \leftarrow \mathsf{PKE.Enc}\big(\hat{\mathsf{pk}}_i, (R'_i, \mathsf{mk}'_i);\ \hat{r}_i\big)$

9. $\mathsf{CGP} \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}_{\mathsf{CG}}^{\hat{x},3}[\hat{k}, \hat{K}, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}, (\mathsf{mk}'_h)_{h\in H}, H, \hat{x}, (\hat{c}_i)_{i\in[n]}])$ (see Figure 2.27)

Hybrid 10.$\hat{x}$. In this hybrid, we do not puncture K anymore and we remove $(\hat{c}_i)_{i\in[n]}$ from CGP. When the public keys of the parties and the nonce $\hat{x}$ are input into the program, we compute the output running the same procedure as if the nonce was strictly smaller than $\hat{x}$. Observe that the input-output behaviour of the program is the same as in the previous hybrid, therefore indistinguishability follows from the security of iO.

The formal procedure adopted by the challenger for the generation of CGP becomes the following.

1. $\hat{k} \leftarrow \mathsf{Punct}'\big(k, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n)\big)$

2. $\forall i \in [n]: \quad \hat{\mathsf{mk}}_i \xleftarrow{\$} \mathsf{Secret}(1^\lambda, i)$

3. $\forall h \in H: \quad \mathsf{mk}'_h \xleftarrow{\$} \mathsf{Secret}(1^\lambda, h)$

$\mathcal{P}^{\hat{x},4}_{\mathsf{CG}}[k, K, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}, (\mathsf{mk}'_h)_{h\in H}, H, \hat{x}]$

Hard-coded. The puncturable PRF keys k and K, the public keys $(\hat{\mathsf{pk}}_i)_{i\in[n]}$ and the master secrets $(\hat{\mathsf{mk}}_i)_{i\in[n]}$ and $(\mathsf{mk}'_i)_{i\in H}$, the set of honest parties H and the nonce $\hat{x}$.
Input. A nonce $x \in \{0,1\}^{l(\lambda)}$ and n public keys $\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n$.

1. $(r, r_1, r_2, \dots, r_n) \leftarrow F_K(\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n, x)$.
2. If $\mathsf{pk}_i = \hat{\mathsf{pk}}_i$ for every $i \in [n]$ and $x >_{\text{lex}} \hat{x}$, set $\mathsf{mk}_i \leftarrow \hat{\mathsf{mk}}_i$ for every $i \in [n]$.
3. If $\mathsf{pk}_i = \hat{\mathsf{pk}}_i$ for every $i \in [n]$ and $x \leq_{\text{lex}} \hat{x}$, set $\mathsf{mk}_h \leftarrow \mathsf{mk}'_h$ for every $h \in H$ and $\mathsf{mk}_i \leftarrow \hat{\mathsf{mk}}_i$ for every $i \in C$.
4. Otherwise, perform the following operations
   (a) $(s_1, s_2, \dots, s_n) \leftarrow F'_k(\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n)$
   (b) $\forall i \in [n]: \quad \mathsf{mk}_i \leftarrow \mathsf{Secret}(1^\lambda, i;\, s_i)$
5. $(R_1, R_2, \dots, R_n) \leftarrow \mathcal{C}(1^\lambda, \mathsf{mk}_1, \mathsf{mk}_2, \dots, \mathsf{mk}_n;\, r)$
6. $\forall i \in [n]: \quad c_i \leftarrow \mathsf{PKE.Enc}\big(\mathsf{pk}_i, (R_i, \mathsf{mk}_i);\, r_i\big)$
7. Output $c_1, c_2, \dots, c_n$.

Figure 2.28: The Correlation Generation Program

4. $\mathsf{CGP} \overset{\$}{\leftarrow} \mathsf{iO}(1^\lambda, \mathcal{P}^{\hat{x},4}_{\mathsf{CG}}[\hat{k}, K, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}, (\mathsf{mk}'_h)_{h\in H}, H, \hat{x}])$ (see Figure 2.28)

The next step of the proof is to repeat Hybrid 3-10 for the following value of the nonce $\hat{x}$. When the procedure has been applied to all the elements of the nonce space, we move to Hybrid 11.

Hybrid 11. In this stage, we change how we generate the master secrets $(\mathsf{mk}'_h)_{h\in H}$ and $(\hat{\mathsf{mk}}_i)_{i\in C}$. Specifically, instead of sampling the randomness fed into Secret uniformly, we rely on the output of the PRF F'_k when its nonce is the position where we had previously punctured. Observe that this hybrid is indistinguishable from the previous one by the security of the puncturable PRF F'.

The formal procedure used by the challenger for the generation of CGP becomes the following.

1. $\hat{k} \leftarrow \mathsf{Punct}'\big(k, (\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n)\big)$
2. $(\hat{s}_1, \hat{s}_2, \dots, \hat{s}_n) \leftarrow F'_k(\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n)$
3. $\forall i \in [n]: \quad \mathsf{mk}'_i \overset{\$}{\leftarrow} \mathsf{Secret}(1^\lambda, i; \hat{s}_i)$
4. $\forall i \in C: \quad \hat{\mathsf{mk}}_i \leftarrow \mathsf{mk}'_i$
5. $\forall h \in H: \quad \hat{\mathsf{mk}}_h \overset{\$}{\leftarrow} \mathsf{Secret}(1^\lambda, h)$
6. $\mathsf{CGP} \overset{\$}{\leftarrow} \mathsf{iO}(1^\lambda, \mathcal{P}^{\Omega,4}_{\mathsf{CG}}[\hat{k}, K, (\hat{\mathsf{pk}}_i, \hat{\mathsf{mk}}_i)_{i\in[n]}, (\mathsf{mk}'_h)_{h\in H}, H, \Omega])$ (see Figure 2.28)

Hybrid 12. This is the final stage and corresponds to the ideal world. In this hybrid, we do not puncture k anymore and we generate CGP by obfuscating the original program $\mathcal{P}_{\mathsf{CG}}$ (see Figure 2.24). Observe that the input-output behaviour of CGP is the same as in Hybrid 11. Indeed, there is no $x >_{\text{lex}} \hat{x}$ because $\hat{x}$ has reached the maximum. Indistinguishability holds by the security of iO.

Also notice that the challenger replies to every query $(\mathsf{Correlation}, x)$ using the following procedure.

1. $(c_i)_{i\in[n]} \leftarrow \mathsf{CGP}(\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n, x)$

2. $\forall i \in C: \quad (R_i, \mathsf{mk}_i) \leftarrow \mathsf{PKE.Dec}(\hat{\mathsf{sk}}_i, c_i)$

3. $(R_h)_{h\in H} \xleftarrow{\$} \mathsf{RSample}\big(1^\lambda, C, (R_i)_{i\in C}, (\hat{\mathsf{mk}}_i)_{i\in C}, (\hat{\mathsf{mk}}_h)_{h\in H}\big)$

Furthermore, observe that the master secrets $(\hat{\mathsf{mk}}_h)_{h\in H}$ are sampled at random using Secret and they are independent of the public keys and CGP. Finally, we formalise the operations of the extractor.

$\mathsf{Extract}(C, \mathsf{CGP}, \rho_1, \dots, \rho_n)$

1. $\forall i \in [n]: \quad (\hat{\mathsf{sk}}_i, \hat{\mathsf{pk}}_i) \leftarrow \mathsf{PKE.Gen}(1^\lambda; \rho_i)$

2. $(c_i)_{i\in[n]} \leftarrow \mathsf{CGP}(\hat{\mathsf{pk}}_1, \hat{\mathsf{pk}}_2, \dots, \hat{\mathsf{pk}}_n, \epsilon)$

3. $\forall i \in C: \quad (R_i, \mathsf{mk}_i) \leftarrow \mathsf{PKE.Dec}(\hat{\mathsf{sk}}_i, c_i)$

4. Output $(\mathsf{mk}_i)_{i\in C}$.

Observe that in every hybrid the size of CGP is polynomial in the length $l(\lambda)$ of the nonces. Furthermore, the size of the keys is always independent of the size of the nonce space. □

2.6.4 Our Public-Key PCFs

As mentioned in the previous section, once we obtain a semi-maliciously secure public-key PCF with trusted setup, we can easily remove the CRS using a distributed sampler. We therefore obtain a public-key PCF with security against semi-malicious adversaries. If the size of the CRS and the keys of the initial construction is logarithmic in the size of the nonce space, the key length after removing the setup is still polynomial in $l(\lambda)$.

Theorem 2.6.8 (Semi-Maliciously Secure Public Key PCFs). Let $(\mathsf{Secret}, \mathcal{C})$ be an n-party, reverse samplable correlation function with master secrets. Suppose that $\mathsf{pkPCFS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Eval})$ is a semi-maliciously secure public-key PCF with trusted setup for $(\mathsf{Secret}, \mathcal{C})$. Moreover, assume that there exists a semi-maliciously secure n-party distributed sampler for pkPCFS.Setup. Then, public-key PCFs for $(\mathsf{Secret}, \mathcal{C})$ with semi-malicious security exist.

We will not prove Theorem 2.6.8 formally. Security follows from the fact that distributed samplers implement the functionality that samples directly from the underlying distribution. From this point of view, it is fundamental that the randomness input into Setup is not given as input to the extractor of the public-key PCF pkPCFS.

Active security in the random oracle model.

If we rely on a random oracle, it is easy to upgrade a semi-maliciously secure public-key PCF to active security. We can use an anti-rusher (see Section 2.5.1) to deal with rushing and malformed messages. If the key size of the semi-malicious construction is polynomial in $l(\lambda)$, after compiling with the anti-rusher, the key length is still $\mathsf{poly}(l)$. The technique described above allows us to deduce the security of our solution from the semi-malicious security of the initial public-key PCF. The result is formalised by the following theorem. Again, we will not provide a formal proof.

Theorem 2.6.9 (Actively Secure Public Key PCFs in the Random Oracle Model). Let $(\mathsf{Secret}, \mathcal{C})$ be an n-party, reverse samplable correlation function with master secret. Assume that $\mathsf{pkPCF} = (\mathsf{Gen}, \mathsf{Eval})$ is a semi-maliciously secure public-key PCFs for $(\mathsf{Secret}, \mathcal{C})$ and suppose there exists an anti-rusher for the associated protocol. Then, actively secure public-key PCFs for $(\mathsf{Secret}, \mathcal{C})$ exist.

Active security from sub-exponentially secure primitives.

So far, all our constructions rely on polynomially secure primitives. However, we often work in the random oracle model. We now show that it is possible to build actively secure public-key PCFs in the URS model assuming the existence of sub-exponentially secure primitives. Furthermore, these constructions come with no restrictions on the size of the nonce space.

Our solution is obtained by assembling a sub-exponentially and semimaliciously secure public-key PCF with trusted setup with a sub-exponentially and semi-maliciously secure distributed sampler. We add witness-extractable NIZKs proving the well-formedness of the messages. Like for our semi-malicious construction, if the size of the CRS and the keys of the public-key PCF with trusted setup is polynomial in the nonce length $l(\lambda)$, after composing with the DS, the key size remains $\mathsf{poly}(l)$.

Theorem 2.6.10 (Actively Secure Public Key PCFs from Subexponentially Secure Primitives). Let $(\mathsf{Secret}, \mathcal{C})$ be an n-party, reverse samplable correlation function with master secret. Suppose that $\mathsf{pkPCFS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Eval})$ is a sub-exponentially and semi-maliciously secure public-key PCF with trusted setup for $(\mathsf{Secret}, \mathcal{C})$. Assume that there exists a sub-exponentially and semi-maliciously secure n-party distributed sampler for $\mathsf{pkPCFS.Setup}$. If there exist simulation-extractable NIZKs with URS proving the well-formedness of the sampler shares and the PCF public keys, there exists an actively secure public-key PCF for $(\mathsf{Secret}, \mathcal{C})$ in the URS model.

Proof. Let $\mathsf{pkPCFS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Eval})$ be the sub-exponentially and semi-maliciously secure public key PCF with trusted setup for $(\mathsf{Secret}, \mathcal{C})$. Assume that $\mathsf{DS} = (\mathsf{Gen}, \mathsf{Sample})$ is a sub-exponentially and semi-maliciously secure distributed sampler for $\mathsf{pkPCFS.Setup}$. Moreover, suppose that the algorithms $\mathsf{DS.Gen}$ and $\mathsf{pkPCFS.Gen}$ need $L(\lambda)$ and $L'(\lambda)$ bits of randomness for their execution respectively. We rely on a PRG PRG mapping a λ-bit seed into a pseudorandom string of $L(\lambda) + L'(\lambda)$ bits. We assume that the output of G is naturally split into two blocks of length $L(\lambda)$ and $L'(\lambda)$ bits respectively. Finally, let $\mathsf{NIZK}' = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ be a simulation-extractable NIZK proving the well-formedness of sampler shares and PCF public keys. Specifically, in the relation corresponding to NIZK', the statement consists of a tuple (U, pk, i), whereas the witness is a pair (s, sk) such that

$$U = \mathsf{DS.Gen}(1^\lambda, i; r), \qquad (\mathsf{sk}, \mathsf{pk}) = \mathsf{pkPCFS.Gen}(1^\lambda, i; r'), \qquad (r, r') = \mathsf{PRG}(s).$$

Our actively secure public key PCF $\Pi_{\mathsf{exp}\text{-}\mathcal{C}}$ is described in Figure 2.29. We now prove that no PPT adversary is able to distinguish between $\Pi_{\mathsf{exp}\text{-}\mathcal{C}}$ and the composition of $\mathcal{F}_{\mathcal{C}}^{\mathsf{RSample}}$ with a PPT simulator we are going to present. The proof proceeds by a series of 6 indistinguishable hybrids (some of them repeated multiple times) going from $\Pi_{\mathsf{exp}\text{-}\mathcal{C}}$ (Hybrid 0) to the ideal world (Hybrid 6).

Hybrid 0. This hybrid coincides with the real world. The simulator runs the protocol $\Pi_{\mathsf{exp}\text{-}\mathcal{C}}$ on behalf of the honest parties. Specifically, it starts its execution producing the URS for the NIZK proofs, it generates the sampler shares and the keys of the honest parties, proves their well-formedness and sends everything except the private keys to the adversary. Moreover, the simulator replies to the correlation queries using $\mathsf{pkPCFS.Eval}$ as in $\Pi_{\mathsf{exp}\text{-}\mathcal{C}}$.

Hybrid 1. In this hybrid, we change how we generate the URS and the NIZK proofs of the honest parties. Specifically, we substitute them with the output of the simulators $\mathsf{NIZK}'.\mathsf{Sim}_1$ and $\mathsf{NIZK}'.\mathsf{Sim}_2$. Indistinguishability between Hybrid 0 and 1 follows from the multi-theorem zero-knowledge of NIZK'. Formally, the steps performed by the simulator for the generation of the messages of the honest parties are the following (the red text indicates what changed since the last hybrid).

1. $\forall h \in H : \quad (r_h, r'_h) \leftarrow \mathsf{PRG}(s_h)$

2. $\forall h \in H : \quad U_h \xleftarrow{\$} \mathsf{DS.Gen}(1^\lambda, h)$

3. $\forall h \in H : \quad (\mathsf{sk}_h, \mathsf{pk}_h) \leftarrow \mathsf{pkPCFS.Gen}(1^\lambda, h; r'_h)$

4. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_1(1^\lambda)$

$\Pi_{\mathsf{exp}\text{-}\mathcal{C}}$

URS. The protocol needs a URS $\mathsf{urs} \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Gen}(1^\lambda)$ for the NIZK proofs.

Initialisation. Each party P_i performs the following steps.

1. $s_i \xleftarrow{\$} \{0,1\}^\lambda$
2. $(r_i, r'_i) \leftarrow \mathsf{PRG}(s_i)$
3. $U_i \leftarrow \mathsf{DS.Gen}(1^\lambda, i;\, r_i)$
4. $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{pkPCFS.Gen}(1^\lambda, i;\, r'_i)$
5. $\pi_i \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Prove}\big(1^\lambda, \mathsf{urs}, (U_i, \mathsf{pk}_i, i), (s_i, \mathsf{sk}_i)\big)$
6. Broadcast $(U_i, \mathsf{pk}_i, \pi_i)$ and wait for a similar message from every other party.
7. If there exists $j \in [n]$ such that $\mathsf{NIZK}'.\mathsf{Verify}\big(\mathsf{urs}, \pi_j, (U_j, \mathsf{pk}_j, j)\big) = 0$ abort.
8. $S \leftarrow \mathsf{DS.Sample}(U_1, U_2, \dots, U_n)$

Correlation. On input a nonce $x \in \{0,1\}^{l(\lambda)}$, each party P_i outputs $R_i \leftarrow \mathsf{pkPCFS.Eval}\big(i, S, (\mathsf{pk}_j)_{j\in[n]}, \mathsf{sk}_i, x\big)$.

Figure 2.29: Actively Secure Public Key PCF based on Sub-Exponentially Secure Primitives

5. $\forall h \in H: \quad \pi_h \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Sim}_2\big(\mathsf{urs}, \tau, (U_h, \mathsf{pk}_h, h)\big)$

Hybrid 2. In this hybrid, we change how we generate the randomness $(r_h, r'_h)_{h\in H}$ of the honest parties. Specifically, instead of expanding the seed s_h, we sample r_h and r'_h uniformly in $\{0,1\}^{L(\lambda)}$ and $\{0,1\}^{L'(\lambda)}$ respectively. Observe that this hybrid is indistinguishable from the previous one by the PRG security of PRG.

Next, we repeat the hybrids from 3 to 5 for every possible choice of the seeds of the corrupted parties $\hat{\rho} := (\hat{s}_i)_{i\in C}$. We follow the lexicographical order starting from the minimum.

Hybrid 3.$\hat{\rho}$. In this hybrid, we change how we generate the sampler shares of the honest parties. Let $(\hat{r}_i, \hat{r}'_i)$ be $\mathsf{PRG}(\hat{s}_i)$ for every $i \in C$. We rely on $\mathsf{DS.Sim}$, providing it with the randomness $(\hat{r}_i)_{i\in C}$ and an element $\hat{S}$ produced by $\mathsf{pkPCFS.Setup}$.

We also change the way we reply to the correlation queries. At the beginning of its execution, the simulator samples a random mk'_h for every $h \in H$. When it receives the messages of the corrupted parties, the simulator extracts the witnesses from the NIZK proofs, obtaining the seeds $\rho = (s_i)_{i\in C}$ and the corresponding secret keys. The operation can be performed due to simulation-extractability. Furthermore, the simulator derives the master secrets $(\mathsf{mk}_i)_{i\in C}$ of the corrupt parties by computing $(r_i, r'_i) \leftarrow \mathsf{PRG}(s_i)$ for every $i \in C$ and running $\mathsf{pkPCFS.Extract}$ on $(r'_i)_{i\in[n]}$ and S. The reply to the correlation queries is then computed as follows.

- If $\rho = \hat{\rho}$, the simulator substitutes the output of $\mathsf{DS.Sample}$ with $\hat{S}$ in $\mathsf{pkPCFS.Eval}$. It answers with the results.
- If $\rho >_{\mathsf{lex}} \hat{\rho}$, the simulator replies as in the real protocol.
- If $\rho <_{\mathsf{lex}} \hat{\rho}$, the simulator extracts the outputs of the corrupted parties by relying on their private keys, feeds the obtained values into $\mathsf{RSample}$ along with $(\mathsf{mk}_i)_{i\in C}$ and $(\mathsf{mk}'_h)_{h\in H}$ and, at the end, answers with the results.

Observe that this hybrid is indistinguishable from the previous one by the semi-malicious security of DS. Notice that when $\hat{\rho}$ is minimum, the simulator never relies on $\mathsf{RSample}$. The formal steps used by the simulator to generate $(U_h)_{h\in H}$ become the following.

1. $\forall i \in C: \quad (\hat{r}_i, \hat{r}'_i) \leftarrow \mathsf{PRG}(\hat{s}_i)$
2. $\hat{S} \xleftarrow{\$} \mathsf{pkPCFS.Setup}(1^\lambda)$
3. $(U_h)_{h \in H} \xleftarrow{\$} \mathsf{DS.Sim}\big(1^\lambda, C, \hat{S}, (\hat{r}_i)_{i \in C}\big)$

The reply to $(\mathsf{Correlation}, x)$ is instead computed as follows.

1. $S \leftarrow \mathsf{DS.Sample}(U_1, U_2, \ldots, U_n)$
2. $\forall i \in C: \quad (s_i, \mathsf{sk}_i) \leftarrow \mathsf{NIZK'.Extract}\big(\mathsf{urs}, \tau, (U_i, \mathsf{pk}_i, i), \pi_i\big)$
3. $\forall i \in C: \quad (r_i, r'_i) \leftarrow \mathsf{PRG}(s_i)$
4. $(\mathsf{mk}_i)_{i \in C} \leftarrow \mathsf{pkPCFS.Extract}(C, S, r'_1, r'_2, \ldots, r'_n)$
5. $\rho \leftarrow (s_i)_{i \in C}$
6. If $\rho = \hat{\rho}$, compute $R_h \leftarrow \mathsf{pkPCFS.Eval}\big(h, \hat{S}, (\mathsf{pk}_i)_{i \in [n]}, \mathsf{sk}_h, x\big)$ $\forall h \in H$.
7. If $\rho >_{\mathsf{lex}} \hat{\rho}$, compute $R_h \leftarrow \mathsf{pkPCFS.Eval}\big(h, S, (\mathsf{pk}_i)_{i \in [n]}, \mathsf{sk}_h, x\big)$ $\forall h \in H$.
8. If $\rho <_{\mathsf{lex}} \hat{\rho}$, compute
   (a) $\forall i \in C: \quad R_i \leftarrow \mathsf{pkPCFS.Eval}\big(i, S, (\mathsf{pk}_j)_{j \in [n]}, \mathsf{sk}_i, x\big)$
   (b) $(R_h)_{h \in H} \xleftarrow{\$} \mathsf{RSample}\big(1^\lambda, C, (R_i)_{i \in C}, (\mathsf{mk}_i)_{i \in C}, (\mathsf{mk}'_h)_{h \in H}\big)$

Hybrid 4.$\hat{\rho}$. In this hybrid, we change how we generate the outputs of the honest parties when the seeds of the corrupted players coincide with $\hat{\rho}$. Specifically, we now reverse sample them. We can indeed retrieve the samples addressed to the corrupted parties using their private keys. The latter can be extracted from the corresponding NIZK proofs along with the seeds of the corrupted players. Using $\mathsf{pkPCFS.Extract}$, it is also possible to derive the master secrets $(\mathsf{mk}_i)_{i \in C}$.

Observe that this hybrid is indistinguishable from the previous one by the semi-malicious security of pkPCFS. As a matter of fact, if the randomness ρ chosen by the adversary is different from $\hat{\rho}$, the two stages are perfectly identical. If instead $\rho = \hat{\rho}$, we can reduce distinguishability between Hybrid 2.$\hat{\rho}$ and 3.$\hat{\rho}$ to the security game $\mathcal{G}^{C, (\hat{r}'_i)_{i \in C}}_{\text{SetupSec}}(\lambda)$.

When $\rho = \hat{\rho}$, the simulator replies to $(\mathsf{Correlation}, x)$ using the following procedure.

1. $\forall i \in C: \quad (s_i, \mathsf{sk}_i) \leftarrow \mathsf{NIZK'.Extract}\big(\mathsf{urs}, \tau, (U_i, \mathsf{pk}_i, i), \pi_i\big)$
2. $\forall i \in C: \quad (r_i, r'_i) \leftarrow \mathsf{PRG}(s_i)$
3. $(\mathsf{mk}_i)_{i \in C} \leftarrow \mathsf{pkPCFS.Extract}(C, \hat{S}, r'_1, r'_2, \ldots, r'_n)$
4. $\forall i \in C: \quad R_i \leftarrow \mathsf{pkPCFS.Eval}\big(i, \hat{S}, (\mathsf{pk}_j)_{j \in [n]}, \mathsf{sk}_i, x\big)$
5. $(R_h)_{h \in H} \xleftarrow{\$} \mathsf{RSample}\big(1^\lambda, C, (R_i)_{i \in C}, (\mathsf{mk}_i)_{i \in C}, (\mathsf{mk}'_h)_{h \in H}\big)$

Hybrid 5.$\hat{\rho}$. In this hybrid, we revert to the original procedure for the generation of the sampler shares of the honest parties. Specifically, we do not rely anymore on $\mathsf{DS.Sim}$, but we use $\mathsf{DS.Gen}$. Observe that this hybrid is indistinguishable from the previous one by the semi-malicious security of DS.

If $\rho = \hat{\rho}$, the procedure used by the simulator to reply to $(\mathsf{Correlation}, x)$ becomes the following.

1. $S \leftarrow \mathsf{DS.Sample}(U_1, U_2, \ldots, U_n)$
2. $\forall i \in C: \quad (s_i, \mathsf{sk}_i) \leftarrow \mathsf{NIZK'.Extract}\big(\mathsf{urs}, \tau, (U_i, \mathsf{pk}_i, i), \pi_i\big)$
3. $\forall i \in C: \quad (r_i, r'_i) \leftarrow \mathsf{PRG}(s_i)$

$\mathcal{S}_{\mathsf{exp}\text{-}\mathcal{C}}$

Initialisation.

1. $\forall h \in H: \quad r'_h \xleftarrow{\$} \{0,1\}^{L'(\lambda)}$
2. $\forall h \in H: \quad U_h \xleftarrow{\$} \mathsf{DS.Gen}(1^\lambda, h)$
3. $\forall h \in H: \quad (\mathsf{sk}_h, \mathsf{pk}_h) \leftarrow \mathsf{pkPCFS.Gen}(1^\lambda, h; r'_h)$
4. $(\mathsf{urs}, \tau) \xleftarrow{\$} \mathsf{NIZK'.Sim}_1(1^\lambda)$
5. $\forall h \in H: \quad \pi_h \xleftarrow{\$} \mathsf{NIZK'.Sim}_2\big(\mathsf{urs}, \tau, (U_h, \mathsf{pk}_h, h)\big)$
6. Send $(U_h, \mathsf{pk}_h, \pi_h)_{h\in H}$ to the adversary and wait for $(U_i, \mathsf{pk}_i, \pi_i)_{i\in C}$ as a reply.
7. If there exists $i \in C$ such that $\mathsf{NIZK'.Verify}\big(\mathsf{urs}, \pi_i, (U_i, \mathsf{pk}_i, i)\big) = 0$, send Abort to the functionality.
8. $S \leftarrow \mathsf{DS.Sample}(U_1, U_2, \dots, U_n)$
9. $\forall i \in C: \quad (s_i, \mathsf{sk}_i) \leftarrow \mathsf{NIZK'.Extract}\big(\mathsf{urs}, \tau, (U_i, \mathsf{pk}_i, i), \pi_i\big)$
10. $\forall i \in C: \quad (r_i, r'_i) \leftarrow \mathsf{PRG}(s_i)$
11. $(\mathsf{mk}_i)_{i\in C} \leftarrow \mathsf{pkPCFS.Extract}(C, S, r'_1, r'_2, \dots, r'_n)$
12. Send $(\mathsf{mk}_i)_{i\in C}$ to the functionality.

Correlation. On input $(\mathsf{Correlation}, x)$ where $x \in \{0,1\}^{l(\lambda)}$, the simulator sends to the functionality $R_i \leftarrow \mathsf{pkPCFS.Eval}\big(i, S, (\mathsf{pk}_j)_{j\in[n]}, \mathsf{sk}_i, x\big)$ for every $i \in C$.

Figure 2.30: Simulator for $\Pi_{\mathsf{exp}\text{-}\mathcal{C}}$

4. $(\mathsf{mk}_i)_{i\in C} \leftarrow \mathsf{pkPCFS.Extract}(C, S, r'_1, r'_2, \dots, r'_n)$
5. $\forall i \in C: \quad R_i \leftarrow \mathsf{pkPCFS.Eval}\big(i, S, (\mathsf{pk}_j)_{j\in[n]}, \mathsf{sk}_i, x\big)$
6. $(R_h)_{h\in H} \xleftarrow{\$} \mathsf{RSample}\big(1^\lambda, C, (R_i)_{i\in C}, (\mathsf{mk}_i)_{i\in C}, (\mathsf{mk}'_h)_{h\in H}\big)$

The next step is to repeat Hybrid 3-5 for the next value $\hat{\rho}$ of the seeds of the corrupted parties. If $\hat{\rho}$ has reached the maximum, we move to Hybrid 6.

Hybrid 6. This hybrid corresponds to the ideal world and summarises what we have achieved so far. In this final stage, the simulator selects the sampler shares and the keys of the honest parties as in the original protocol, using however true randomness instead of expanding PRG seeds. The URS and the NIZK proofs are generated by relying on the simulators $\mathsf{NIZK'.Sim}_1$ and $\mathsf{NIZK'.Sim}_2$ as in Hybrid 1.

When the simulator receives the messages of the corrupted parties from the adversary, it extracts their seeds and private keys from the zero-knowledge proofs using $\mathsf{NIZK'.Extract}$. At that point, it has all the necessary information to retrieve the master secrets $(\mathsf{mk}_i)_{i\in C}$ of the corrupted players by means of $\mathsf{pkPCFS.Extract}$. The values are sent to $\mathcal{F}_{\mathcal{C}}^{\mathsf{RSample}}$.

Upon receiving any query $(\mathsf{Correlation}, x)$, the simulator is also able to compute the outputs of the corrupted parties as it knows their private keys. So it is just sufficient to forward the results to the functionality. The latter will take care of the generation and distribution of the samples of the honest players using $\mathsf{RSample}$. The formal description of the simulator is available is Figure 2.30.

□

$\mathcal{F}_{\mathcal{C}}^{\mathsf{Ideal}}$

Initialisation. On input Init from every honest party and the adversary, the functionality activates and enters the querying phase.

Querying phase.

- On input $(\mathsf{Query}, \mathsf{id}, \mathcal{C})$ from the adversary where $\mathcal{C}$ is a (n, ℓ, r, t)-correlation that has not been queried previously, the functionality samples $(R_i)_{i\in[n]} \xleftarrow{\$} \mathcal{C}(1^\lambda)$, sends $(R_i)_{i\in C}$ to the adversary and stores the tuple $\big(\mathsf{id}, \mathcal{C}, (r_i)_{i\in[n]}\big)$.
- On input $(\mathsf{Choice}, \hat{\mathsf{id}})$ from the adversary, the functionality stores $\hat{\mathsf{id}}$, ends the querying phase and begins the correlation phase. If $\hat{\mathsf{id}} = \mathsf{Abort}$, the functionality outputs $\perp$ to every honest party and halts.

Correlation phase. On input a (n, ℓ, r, t)-correlation $\mathcal{C}$ from party P_i, where a tuple $\big(\hat{\mathsf{id}}, \mathcal{C}, (R_i)_{i\in[n]}\big)$ has not been stored previously, the functionality samples $(R_1, R_2, \dots, R_n) \xleftarrow{\$} \mathcal{C}(1^\lambda)$, stores $\big(\hat{\mathsf{id}}, \mathcal{C}, (R_i)_{i\in[n]}\big)$ and outputs R_i to P_i. If $\big(\hat{\mathsf{id}}, \mathcal{C}, (R_i)_{i\in[n]}\big)$ has been previously stored (in either the querying or correlation phase), the functionality outputs R_i to P_i.

Figure 2.31: The Functionality for Ideal Public Key PCFs with Active Security

2.7 Ideal Public Key PCFs and Distributed Universal Samplers

We now inspect the feasibility of ideal public key PCFs. The term is used to denote one-round constructions implementing the functionality that directly samples the outputs from adaptively chosen correlations and distributes them to the parties. In contrast with the other PCFs we described, ideal public key PCFs are not tailored to any specific correlation function; instead, the correlation function can be chosen on the fly. However, they can exist only in the random oracle model.

Defining ideal public key PCFs. We deal with generic correlations $\mathcal{C}$ without master secrets. $\mathcal{C}$ takes no inputs, and generates n correlated outputs. It does not need to satisfy any specific properties (in particular it is not required to be reverse samplable). Since the correlations supported by an ideal public key PCF are restricted to those whose description is polynomially bounded, we define the class of (n, ℓ, r, t)-correlations as the set of functions mapping r bits of randomness into n t-bit outputs and having an ℓ-bit description as a circuit.

The syntax of ideal public key PCFs is derived from that of their non-ideal counterparts (see Definition 2.6.2). The only difference is that the evaluation algorithm Eval takes as input the description of an (n, ℓ, r, t)-correlation instead of a nonce.

Definition 2.7.1 (Ideal Public-Key PCF). Let $\ell(\lambda)$, $r(\lambda)$ and $t(\lambda)$ be polynomials. An ideal public-key PCF for (n, ℓ, r, t)-correlations is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Eval})$ with the following syntax:

- Gen takes as input the security parameter 1^λ and the index of a party $i \in [n]$, and outputs the PCF key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ of the i-th party.
- Eval takes as input an index $i \in [n]$, n PCF public keys, the i-th PCF private key sk_i and the description of an (n, ℓ, r, t)-correlation $\mathcal{C}$. It outputs a value R_i corresponding to the i-th output of $\mathcal{C}$.

Definition 2.7.2 (Ideal Public Key PCF with Active Security). An ideal public key PCF $(\mathsf{Gen}, \mathsf{Eval})$ for (n, ℓ, r, t)-correlations satisfies *active security* if the corresponding one-round protocol $\Pi_{\mathcal{C}}$ implements the functionality $\mathcal{F}_{\mathcal{C}}^{\mathsf{Ideal}}$ (see Figure 2.31) against a static and active adversary corrupting up to $n-1$ parties.

Like in the definition of the actively secure distributed sampler (see Definition 2.3.3), the adversary is allowed to request different samples stored under different labels. Afterwards, the adversary can specify a label of its choice, forcing the functionality to output the associated values to the honest players. We must allow this kind of influence in order to model rushing; an active adversary can always wait for the messages of the honest parties and adaptively choose the reply of the corrupted players. She is allowed to rerun the procedure as many time as it desire, repeatedly re-generating the messages of the corrupted parties and obtaining different collections of samples. The adversary can then use the messages that led to the most favourable results.

We will build our ideal public key PCFs upon a new primitive called a *distributed universal sampler*. We will present and analyse it in the following subsection.

2.7.1 Distributed Universal Samplers

A distributed universal sampler (DUS) generalises the concept of distributed sampler. Recall that a distributed sampler is tailored to output a single sample from some fixed distribution $\mathcal{D}$. In some applications, for instance when we need to sample from multiple distributions, chosen on-the-fly, this may be too restrictive.

What we want instead is analogous to a universal sampler (US) [HJK+16], where a trusted dealer first generates and publishes a sampler U. Later, the parties can use U to sample from arbitrary distributions, learning no additional information about the randomness used to generate the output. With a *distributed* universal sampler, we aim to remove the trusted dealer, sampling from generic distributions in a distributed way and with only one round of interaction.

Formally, (distributed) universal samplers do not support completely generic distributions, but are instead restricted to those whose descriptions are polynomially-bounded. We therefore define the class of (ℓ, r, t)-distributions as the set of all distributions converting r bits of randomness into a t-bit output and having an ℓ-bit description as a circuit.

The syntax of a DUS is obtained by augmenting the Sample algorithm of a DS with an additional input, namely the description of the distribution from which to sample the output.

Definition 2.7.3 (Distributed Universal Sampler). Let $\ell(\lambda)$, $r(\lambda)$ and $t(\lambda)$ be polynomials. An n-party distributed universal sampler for (ℓ, r, t)-distributions consists of a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Sample})$ with the following syntax.

1. Gen is a probabilistic algorithm taking as input the security parameter 1^λ and a party index $i \in [n]$ and outputting a sampler share U_i for party i. Suppose that the procedure needs $L(\lambda)$ bits of randomness.

2. Sample is a deterministic algorithm taking as input n shares of the sampler $U_1, U_2, \dots, U_n$ and the description of an (ℓ, r, t)-distribution $\mathcal{D}$, outputting a sample R.

Similarly to a DS, any distributed universal sampler $\mathsf{DUS} = (\mathsf{Gen}, \mathsf{Sample})$ naturally corresponds to a one-round protocol Π_{DUS}, where each party first broadcasts a message output by Gen, and then, for every required sample, runs Sample on input all the parties' messages and the desired distribution $\mathcal{D}$.

As in the setting of universal samplers [HJK+16], we can classify a DUS in two main ways: security for distributions chosen *selectively* by the adversary ahead of time, and security for adversaries who can *adaptively* choose distributions on-the-fly. We refer to the first class as *one-time security*, and the latter as *reusable security*.

One-Time Distributed Universal Samplers

While reusable distributed universal samplers need a random oracle independently of the power of the adversary, it is possible to build one-time DUSs with semi-malicious security in the plain model. Indeed, we now consider a one-time, selective security definition, where the sampler may only be queried once, and on a distribution $\mathcal{D}$ that is fixed ahead of time. We formalise the idea.

Definition 2.7.4 (Distributed Universal Sampler with One-Time Semi-Malicious Security). A distributed universal sampler $(\mathsf{Gen}, \mathsf{Sample})$ satisfies *one-time semi-malicious security* if there exists a PPT simulator

Sim such that, for every set of corrupted parties $C \subsetneq [n]$, corresponding randomness $(\rho_i)_{i\in C}$ and (ℓ, r, t)-distribution $\mathcal{D}$, the following two distributions are computationally indistinguishable.

$$\left\{\begin{matrix}(U_i)_{i\in[n]}, (\rho_i)_{i\in C}\\ R\end{matrix}\;\middle|\;\begin{matrix}\forall i \in H: & \rho_i \xleftarrow{\$} \{0,1\}^{L(\lambda)}\\ \forall i \in [n]: & U_i \leftarrow \mathsf{Gen}\big(1^\lambda, i; \rho_i\big)\\ \multicolumn{2}{l}{R \leftarrow \mathsf{Sample}(U_1, U_2, \dots, U_n, \mathcal{D})}\end{matrix}\right\}$$

$$\left\{\begin{matrix}(U_i)_{i\in[n]}, (\rho_i)_{i\in C}\\ R\end{matrix}\;\middle|\;\begin{matrix}R \xleftarrow{\$} \mathcal{D}\\ (U_i)_{i\in[n]} \xleftarrow{\$} \mathsf{Sim}\big(1^\lambda, C, \mathcal{D}, R, (\rho_i)_{i\in C}\big)\end{matrix}\right\}$$

Just as with a DS, we can adapt the above definition to passive security by sampling the randomness of the corrupted parties inside the game in the real world and by generating it using the simulator in the ideal world. In a definition for active security, we must also account for a rushing adversary, who may adaptively choose the sampler shares of the corrupted parties after seeing those of the honest parties. In other words, in the ideal world, the adversary would be allowed to select the final output R from a list of samples generated by the functionality. We do not consider this notion here, since our actively secure construction actually satisfies the stronger notion of a *reusable* DUS (see Section 2.7.1).

Universal samplers. We will build our one-time DUSs starting from their non-distributed version [HJK+16]. The corresponding definition is available in Section 2.2.6. A one-time universal sampler is a pair of PPT algorithms ($\mathsf{US.Setup}, \mathsf{US.Sample}$) with the same syntax as the reusable case. The main difference is that the random oracle is no longer needed. Security is defined by stating that no PPT adversary can distinguish the real samplers from fake ones specifically programmed to output an ideal sample R when used in conjunction with a distribution $\mathcal{D}$ selected ahead of time. The property is required to hold for every (ℓ, r, t)-distribution $\mathcal{D}$. In other words, the main novelty is that we now program the sampler for one specific distribution selected ahead of time, whereas in the reusable case, we did that for multiple distributions adaptively chosen by the adversary.

Construction of a one-time DUS. Given distributed samplers and a one-time universal sampler, it is quite straightforward to build a semi-maliciously secure one-time DUS. We can simply substitute the trusted dealer generating the one-time sampler U with a semi-maliciously secure DS for $\mathsf{US.Setup}$. Security follows from the programmability of one-time universal samplers and the fact that, by definition, a DS implements the functionality that directly samples from the underlying distribution. This idea is formalised in the following theorem. We will not, however, provide a formal proof.

Theorem 2.7.5 (One-Time Distributed Universal Samplers). Suppose that ($\mathsf{US.Setup}, \mathsf{US.Sample}$) is a one-time universal sampler for (ℓ, r, t)-distributions. Assume that ($\mathsf{DS.Gen}, \mathsf{DS.Sample}$) is a semi-maliciously secure distributed sampler for $\mathsf{US.Setup}$. Then, there exists a one-time distributed universal samplers for (ℓ, r, t)-distributions with semi-malicious security.

Reusable Distributed Universal Samplers.

In a reusable DUS, the shares output by Gen can be reused to obtain an arbitrary number of samples, from distributions that are chosen adaptively by the adversary. Note that, as is the case for (non-distributed) universal samplers [HJK+16], this notion is impossible to realize in the standard model, and our construction will use a random oracle.

We model security by requiring that the sampler can be used to obtain a one-round protocol that securely realizes an ideal sampling functionality, which can be queried adaptively.

Definition 2.7.6 (Reusable Distributed Universal Sampler). A distributed universal sampler $\mathsf{DUS} = (\mathsf{Gen}, \mathsf{Sample})$ for (ℓ, r, t)-distributions satisfies *reusable active security* if the corresponding one-round protocol Π_{DUS} implements the functionality $\mathcal{F}_{\mathsf{DUS}}$ (see Figure 2.32) against a static and active PPT adversary.

$\mathcal{F}_{\mathsf{DUS}}$

Initialisation. On input Init from every honest party and the adversary, the functionality activates and enters the querying phase.

Querying phase.

- On input $(\mathsf{Query}, \mathsf{id}, \mathcal{D})$ from the adversary, where $\mathcal{D}$ is an (ℓ, r, t)-distribution and the pair $(\mathsf{id}, \mathcal{D})$ has not been queried previously, the functionality samples $R \xleftarrow{\$} \mathcal{D}$, sends R to the adversary and stores $(\mathsf{id}, \mathcal{D}, R)$.

- On input $(\mathsf{Choice}, \widehat{\mathsf{id}})$ from the adversary, the functionality stores $\widehat{\mathsf{id}}$, ends the querying phase and begins the sampling phase. If $\widehat{\mathsf{id}} = \mathsf{Abort}$, the functionality outputs $\perp$ to every honest party and halts.

Sampling phase. On input an (ℓ, r, t)-distribution $\mathcal{D}$ from party P_i, where a tuple $(\widehat{\mathsf{id}}, \mathcal{D}, R)$ has not been stored previously, the functionality samples $R \xleftarrow{\$} \mathcal{D}$, stores $(\widehat{\mathsf{id}}, \mathcal{D}, R)$ and outputs R to P_i. If $(\widehat{\mathsf{id}}, \mathcal{D}, R)$ has been previously stored (in either the querying or sampling phase), the functionality outputs R to P_i.

Figure 2.32: The Reusable Distributed Universal Sampler Functionality for Active Security

Adaptively secure universal samplers. We construct a reusable DUS starting from an adaptively secure universal sampler. We briefly recall the corresponding definition [HJK+16], the formal version of which is available in Section 2.2.6. An adaptively secure universal sampler is a pair of PPT algorithms $(\mathsf{US.Setup}, \mathsf{US.Sample})$, the first one of which is used by a trusted dealer to generate a sampler U. By feeding U and (ℓ, r, t)-distributions $\mathcal{D}$ to the second algorithm (and using the random oracle), anyone can obtain a sample R. Importantly, $\mathsf{US.Sample}$ is deterministic, so a set of parties can use U to generate public samples non-interactively. Security requires that no PPT adversary $\mathcal{A}$ can distinguish the real sampler U and the original oracle responses from fake ones specifically programmed to output ideal samples from the (ℓ, r, t)-distributions chosen by $\mathcal{A}$ on-the-fly. Hofheinz *et al.* [HJK+16] present an adaptively secure universal sampler for (ℓ, r, t)-distributions whose size is $\mathsf{poly}(\ell, r, t)$.

Our reusable distributed universal sampler. It turns out that designing a reusable DUS is rather straightforward. It suffices to generate the adaptively secure sampler U using a DS for $\mathsf{US.Setup}$. Observe that if we use the construction of [HJK+16], the size of the sampler shares is $\mathsf{poly}(\ell, r, t)$. Security follows from the adaptive programmability of US and the fact that the DS implements the functionality that directly samples from the associated distribution. We formalise this in the theorem below, again, given without proof.

Theorem 2.7.7 (Reusable Distributed Universal Samplers in the Random Oracle Model). Suppose that $(\mathsf{US.Setup}, \mathsf{US.Sample})$ is an adaptively secure universal sampler for (ℓ, r, t)-distributions using a random oracle $\mathcal{H}$. Assume that there exists an actively secure distributed sampler for $\mathsf{US.Setup}$. Then, there exists a reusable distributed universal sampler for (ℓ, r, t)-distributions with active security.

2.7.2 Building Ideal Public Key PCFs upon Distributed Universal Samplers

If reusable distributed universal samplers exist, constructing ideal public key PCFs becomes easy. We can use the reusable DUS to produce correlated samples $(R_j)_{j \in [n]}$ and then protect their privacy using a PKE scheme. Specifically, each party P_i can generate a PKE pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ and broadcast the public counterpart along with a reusable DUS share. In order to produce and deal correlated outputs $(R_j)_{j \in [n]}$ from a correlation function $\mathcal{C}$, it is sufficient to query the DUS with the distribution that samples from $\mathcal{C}$ and outputs the encryption of R_j under pk_j for every $j \in [n]$. In this way, only the party j can retrieve the value of the j-th sample R_j. If we rely on the DUS built on top of the US of Hofheinz *et al.* [HJK+16], the key size of our public key PCF is $\mathsf{poly}(\ell, r, n \cdot t)$.

Security. Proving the security of this construction is an easy task. We rely on UC composability, and substitute the DUS with the corresponding functionality $\mathcal{F}_{\mathsf{DUS}}$ (see Figure 2.32). Then, by the IND-CPA security of the PKE scheme, we can substitute the ciphertexts addressed to the honest parties in the $\mathcal{F}_{\mathsf{DUS}}$ responses with the encryption of random values. We substitute the ciphertexts of the corrupt parties with encryptions of the elements provided by the functionality $\mathcal{F}_{\mathcal{C}}^{\mathsf{Ideal}}$ (see Figure 2.31). Our result is formalised in the following theorem; we do not provide a formal proof.

Theorem 2.7.8 (Ideal Public Key PCFs in the Random Oracle Model). If there exists an IND-CPA PKE scheme and an n-party reusable distributed universal sampler with active security, there exists an actively secure ideal public key PCFs for (n, ℓ, r, t)-correlations.

Bibliography

[AJJM20] Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 28–57. Springer, Heidelberg, November 2020.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Heidelberg, August 2019.

[BCG+20a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020.

[BCG+20b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Heidelberg, August 2020.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[Bd94] Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 274–285. Springer, Heidelberg, May 1994.

[BGG19] Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *FC 2018 Workshops*, volume 10958 of *LNCS*, pages 64–77. Springer, Heidelberg, March 2019.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BGI+14a] Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 387–404. Springer, Heidelberg, August 2014.

[BGI14b] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. https://eprint.iacr.org/2017/1050.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, Heidelberg, December 2012.

[BP15] Nir Bitansky and Omer Paneth. ZAPs and non-interactive witness indistinguishability from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 401–427. Springer, Heidelberg, March 2015.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 342–362. Springer, Heidelberg, February 2005.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, March 2015.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Heidelberg, August 2016.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st FOCS*, pages 308–317. IEEE Computer Society Press, October 1990.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 323–341. Springer, Heidelberg, August 2007.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 97–111. Springer, Heidelberg, August 2006.

[GPSZ17] Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 156–181. Springer, Heidelberg, April / May 2017.

[HIJ$^+$17] Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 181–211. Springer, Heidelberg, December 2017.

[HJK$^+$16] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 715–744. Springer, Heidelberg, December 2016.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

[LZ17] Qipeng Liu and Mark Zhandry. Decomposable obfuscation: A framework for building applications of obfuscation from polynomial hardness. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 138–169. Springer, Heidelberg, November 2017.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Heidelberg, October 2021.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Heidelberg, August 2019.

Chapter 3

On the (Im)Possibility of Distributed Samplers:Lower Bounds and Party-Dynamic Constructions

Damiano Abram, Maciej Obremski, Peter Scholl

Abstracts. Distributed samplers, introduced by Abram, Scholl and Yakoubov (Eurocrypt '22), are a one-round, multi-party protocol for securely sampling from any distribution. We give new lower and upper bounds for constructing distributed samplers in challenging scenarios. First, we consider the feasibility of distributed samplers with a malicious adversary in the standard model; the only previous construction in this setting relies on a random oracle. We show that for any UC-secure construction in the standard model, even with a CRS, the output of the sampling protocol must have low entropy. This essentially implies that this type of construction is useless in applications.

Secondly, we study the question of building distributed samplers in the party-dynamic setting, where parties can join in an ad-hoc manner, and the total number of parties is unbounded. Here, we obtain positive results. First, we build a special type of unbounded universal sampler, which after a trusted setup, allows sampling from any distributed with unbounded size. Our construction is in the *shared randomness model*, where the parties have access to a shared random string, and uses indistinguishability obfuscation and somewhere statistically binding hashing. Next, using our unbounded universal sampler, we construct distributed universal samplers in the party-dynamic setting. Our first construction satisfies one-time selective security in the shared randomness model. Our second construction is reusable and secure against a malicious adversary in the random oracle model. Finally, we show how to use party-dynamic, distributed universal samplers to produce ideal, correlated randomness in the party-dynamic setting, in a single round of interaction.

3.1 Introduction

Many cryptographic protocols require public parameters to be generated in a secure manner. This is the case, for instance, with trusted parameters used in many succinct zero-knowledge proofs [BCCT12], or trusted RSA moduli used in cryptographic accumulators [Bd94]. Using incorrectly or insecurely generated parameters in these settings can have devastating results, often completely breaking the desired security properties. As a result, when such parameters are needed, the parties involved may wish to run a secure multi-party computation protocol to generate them, guaranteeing security as long at least one of the parties is honest. However, this type of setup protocol is typically expensive to carry out and coordinate.

Universal samplers, introduced by Hofheinz et al. [HJK+16], offer a partial solution to this problem. A universal sampler produces a single set of public parameters, which can later be used to securely sample from *any* distribution. In their strongest form, note that universal samplers are inherently tied to the random oracle model: in fact, they can be seen as a type of random oracle for sampling from arbitrary, structured distributions, without leaking the underlying random coins in the process.

A downside of universal samplers is that they still require a trusted setup, even if it only needs to be done once. Distributed samplers, recently introduced by Abram, Scholl and Yakoubov [ASY22, AWZ23], work around this issue by allowing parameters to be sampled using a secure multi-party protocol with *minimal interaction*. Each party publishes a single message, after which all parties can obtain a sample from the desired distribution. More formally, a distributed sampler for n parties and a distribution $\mathcal{D}$ is defined by a pair of algorithms $(\mathsf{Gen}, \mathsf{Sample})$, such that given a set of messages $U_i = \mathsf{Gen}(1^\lambda, i)$, for $i \in [n]$, one can compute a sample $R \leftarrow \mathsf{Sample}(U_1, \ldots, U_n)$. The security requirement essentially states that this one-round protocol must securely realize the ideal functionality for sampling from $\mathcal{D}$, even when up to $n-1$ parties are corrupted.

The basic definition considers a 'one-time' or static setting, where there is a single distribution $\mathcal{D}$ that is fixed ahead of time, and the parties can only obtain a single sample from $\mathcal{D}$. This can be considered either with security against a passive adversary, or an active adversary. Active security is particularly challenging, due to the need to handle a rushing adversary, who may choose their messages U_i *after* seeing the messages U_j of the other parties. This allows an attacker to "grind" different choices of their randomness, obtaining different U_i, until finding an output R that she likes. So, the best form of security one can hope for in this setting is a relaxation of the ideal functionality for sampling, where the adversary first obtains several samples from $\mathcal{D}$, before settling on a final output. A stronger variety of distributed samplers is one that is reusable, for an unbounded number of queries. This is known as a distributed universal sampler. Similarly to the case of a (non-distributed) universal sampler, this is only possible to construct in the random oracle model.

Abram et al [ASY22] constructed distributed samplers in the plain model (no CRS) for any distribution based on indistinguishability obfuscation and multi-key fully homomorphic encryption. Their first construction is secure only against a *semi-malicious*[1] and non-rushing adversary. This was then upgraded to malicious security in the programmable random oracle model, with a construction that is also reusable, and secure for adaptive choices of the desired distributions. On top of this, they showed how distributed samplers can be used for sampling arbitrary forms of *correlated randomness*, often used in MPC protocols, with a one-round protocol.

We note that the constructions in [ASY22] are proven secure assuming that the underlying primitives are secure against polynomially bounded adversaries. This is in contrast to similar primitives like non-interactive MPC [HIJ+17] or probabilistic iO [CLTV15], for which the only general constructions are based on subexponentially secure primitives. This highlights that the setting of computing randomized functionalities, where no party has a private input, seems easier than that of general computations.

3.1.1 Our Results

In this work, we further explore the feasibility of distributed samplers, pushing their lower and upper limits with both impossibility results and more powerful constructions. We focus on security in the UC model [Can01], which gives strong composability guarantees. See Table 3.1.1 for an overview of our results and prior relevant work.

Impossibility of Distributed Samplers Without Random Oracles.

We first pose the question: is it possible to build actively secure distributed samplers in the *standard model*, that is, without random oracles? As a starting point, we observe that actively secure distributed samplers cannot be built without a common reference string (CRS) in the UC model. This is an immediate consequence

[1]A semi-malicious adversary is one who follows the protocol, but may choose their random tape arbitrarily.

Primitive	Security model	Setup	Feasibility
Universal sampler	Static	None	poly FE [LZ17]
Universal sampler	Adaptive	RO	poly iO [HJK+16]
Unbounded US	Static	Shared rand.	poly iO + SSB [AOS23, §4.1]
Distributed sampler	Semi-malicious	None	poly iO + mkFHE [ASY22]
Reusable DS	Malicious, UC	RO	poly iO + mkFHE + NIZK [ASY22]
Distributed sampler	Malicious, ind.	CRS	subexp iO + subexp mkFHE + ELFs +... [AWZ23]
Distributed sampler	Malicious, UC	None	impossible [HV16]
Distributed sampler	Malicious, UC	CRS	impossible (**§3.4**)
Party-dynamic DS	Semi-malicious	Shared rand.	poly iO + mkFHE [AOS23, §5]
Reusable, P-D DS	Malicious, UC	RO	poly iO + mkFHE + NIZK [AOS23, §5]

Table 3.1.1: Overview of the feasibility of universal and distributed samplers in different settings. RO = random oracle; poly/subexp = polynomial/subexponential hardness; mkFHE = multi-key FHE; ELF = extremely lossy function

of the UC impossibility for same-output probabilistic functions of [CKL03], since the function $\mathcal{D}(1^\lambda)$ we want to compute has an unpredictable output and no inputs.

We observe also that generic actively secure distributed samplers without a CRS cannot exist even in the standalone model with black-box simulation. If that was not the case, by sequentially composing a distributed sampler with 2-round active OT protocols in the CRS model such as [PVW08] or [DGH+20], we would obtain a 3-round OT protocol with active security and black-box simulation in the plain model. The latter is known to be impossible [HV16].

For this reason, we investigate the CRS model. At first glance, it seems that distributed samplers are then trivial: the CRS can directly encode a sample from the desired distribution. This solution does not even need interaction. However, interactive distributed samplers with a CRS may have some advantages over the trivial construction, if the CRS can be reused multiple times and/or is easier to generate, either by being short or unstructured (i.e. a uniformly random string). We prove that if the construction is secure against rushing adversaries in the UC model, none of the above properties can be satisfied in the standard model.

All of these impossibilities come from our main result, below. Although the impossibility is in the UC model, we show that it even holds for a restricted class of adversaries who always follow the protocol, but behave in a rushing manner, sending their messages after receiving those of the honest parties. This only strengthens our impossibility result.

Theorem 3.1.2 (Informal, c.f. Thm. 3.4.1). For any distributed sampler secure against rushing adversaries in the UC model, for a distribution $\mathcal{D}$ where $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log \lambda)$, we have that $\mathsf{H}(R \,|\, \mathsf{crs}) = O(\log \lambda)$, where crs denotes the CRS and R the output of the distributed sampler.[2]

This essentially rules out this flavour of distributed sampler for all practical applications, as we discuss in the following corollaries.

Corollary 1: the collision probability is large. An immediate consequence of small Shannon entropy is that the output of the distributed sampler has a high probability of a collision if the CRS is not changed. This implies that in applications where more than one sample from $\mathcal{D}$ is needed, the same CRS cannot be reused.

[2]We use H_∞ to denote the min-entropy. We use H to denote the Shannon entropy. We refer to Section 3.3.2 for formal definitions.

Corollary 2: the CRS must be long. Less trivially, we show that this means that the CRS must be at most $O(\lambda)$ bits smaller than the Yao incompressibility entropy of $\mathcal{D}$. Recall that this roughly measures the compressed size of a sample from $\mathcal{D}$, after applying any efficient compression algorithm. As a result, the CRS must be almost as long as an output of $\mathcal{D}$, after applying compression.

Corollary 3: the CRS must be ugly. Finally, we show that in meaningful scenarios, the CRS must inherently be *structured*, or "ugly", meaning that it requires private coins to sample. In practice, this type of CRS must be generated by a trusted party or multi-party computation protocol, whereas obtaining a CRS that can be sampled from uniform randomness is much easier, relying only on a public source of randomness (or a hash function modelled as a random oracle).

Conclusion. Put together, the above corollaries show that UC-secure distributed samplers in the standard model, with rushing adversaries, are essentially useless. Since the CRS can only be used once, is structured, and as long as an output of $\mathcal{D}$, in practice it will most likely be no easier to generate the CRS than to just generate a sample from $\mathcal{D}$.

Open questions. Our main impossibility result is in the UC model, with polynomial-time simulation and dishonest majority. Recall that in this setting, rewinding is not allowed and simulation is inherently black-box[3]. We leave to future work the question of proving impossibilities — or finding constructions — for different settings, such as an honest majority, rewinding and non-black-box simulation.

Positive Results: Party-Dynamic Distributed Samplers.

On the positive side, we give new results in settings where the parties have access to a random oracle, or in some cases, a public source of uniform randomness, called the *shared randomness model*. The main difference between these settings is that random oracles are an idealised model that assumes the existence of an exponential amount of randomness to which all the parties have access. The shared randomness model, instead, is a more realistic setting in which all parties have access to randomness that grows polynomially in the size of their inputs.

We construct *party-dynamic* distributed universal samplers, where the messages are independent of the distribution we want to sample from, the set of participants and their number, which is a priori unbounded. We analyse two notions of security. In one-time, semi-malicious security, the messages are used to generate a single sample, and the underlying distribution and set of parties are chosen ahead of time. With reusable, active security, the same messages are used to generate samples for multiple distributions and multiple subsets of participants, both adaptively chosen by the adversary. Distributed universal samplers, i.e. distributed samplers where the messages are independent of the distribution, were already built in [ASY22]. Prior to this work, however, all constructions were tailored to a specific set of players, which forced a restart of the protocol if participants joined or left.

Applications. Constructions supporting dynamic participants are ideally suited to non-interactive setup ceremonies for SNARKs in a permisionless setting, such as blockchains. More generally, they can be used for trusted setup in MPC protocols: imagine a world where every institution (e.g. governments, NGOs, intergovernmental organisations, private companies,...) publishes a distributed universal sampler message on a public bulletin board. Any set of parties that wants to run an MPC protocol can now non-interactively generate any CRS or correlated randomness[4] they want by just combining the sampler messages of the institutions they trust. The desired randomness is secure as long as just one of the participant's randomness is kept private. Furthermore, since our construction is party-dynamic, new organisations can join the protocol

[3]Although the UC model allows the simulator to depend on the real-world adversary, the notion of security is still black-box. Indeed, it can be proven that a protocol is UC-secure if and only if it is secure against the "dummy adversary", who simply follows the instructions given by the environment [Can01]. By reframing the model in this way, we obtain a form of black-box simulation: security requires the existence of a single simulator that works for every environment.

[4]Using a party-dynamic distributed correlation sampler, discussed below.

at any time without requiring further action from the others. Of course, the use of iO makes our solution currently impractical. However, we highlight that the task of obfuscating circuits is only required by the institutions (which likely have more resources); the parties just need to evaluate the resulting programs. In other words, for our solution to become practical, obfuscating does not need to be extremely efficient, what matters is the efficiency of the evaluation.

Our results. A key tool we introduce for our party-dynamic constructions is an *unbounded universal sampler*. Universal samplers are a way of securely sampling from any distribution, after a trusted setup phase which outputs some public parameters, called the *sampler parameters*. Previous constructions [HJK+16, LZ17] require the sampler parameters to be at least as large as the maximum size of the distribution. In the unbounded setting, we impose no such constraint: the circuit-size of the distribution can be arbitrarily large. Since the sample may be bigger than the sampler parameters, this inherently means that we need some additional source of randomness (such as the shared randomness model, or random oracle). An immediate application of unbounded universal samplers is to compile any protocol with a large CRS into one with a small, reusable CRS in the random oracle model. This technique was recently applied in [ABI+23] to build a private simultaneous messages protocol with succinct public parameters and messages that have logarithmic size in the function input.

Theorem 3.1.3 (Informal). Assuming polynomially secure iO and somewhere statistically binding hashing, there exist unbounded universal samplers in the shared randomness model.

Using the unbounded universal sampler, we obtain the following.

Theorem 3.1.4 (Informal). Assuming polynomially secure iO and multi-key FHE, there exist party-dynamic distributed universal samplers for any distribution, which are:

- One-time secure against a non-rushing, semi-malicious adversary, in the shared randomness model
- Reusable and secure against a malicious and static adversary, in the UC model with local random oracle (and assuming NIZK)

Party-Dynamic, Distributed Correlation Samplers.

As an application of our party-dynamic distributed samplers, we show how they can be used to obtain party-dynamic, distributed, universal *correlation samplers*, where after each party publishing a single, short, message, any subset of parties can obtain large amounts of correlated samples $R_1, \ldots, R_n$, defined by some arbitrary, correlated distributions (adaptively chosen after the messages are sent). Formally, we phrase this construction in the language of (public-key) pseudorandom correlation functions [BCG+20, ASY22].

Theorem 3.1.5 (Informal). Assuming polynomially secure iO and multi-key FHE, there exist party-dynamic, public-key, pseudorandom universal correlation functions, for adaptively-chosen correlations in the UC model with local random oracle.

Such primitive can be used, for instance, to build party-dynamic MPC with an information-theoretical online phase [DPSZ12, IKM+13, IOZ14] and non-interactive offline phase: when a party joins the protocol, it just needs to sent its public-key for the pseudorandom correlation function. After that, it can immediately join the online phase, without the other players' need to regenerate their pseudorandom correlation function keys.

3.1.2 Related Work

Follow-up work on distributed samplers. In [AWZ23], Abram, Waters and Zhandry presented solutions to circumvent the impossibility proven in this paper. Instead of aiming for a simulation-based security definition, they show that, using strong primitives (including subexponential iO) but no random oracle, it is possible to implement game-based definitions for distributed samplers that allow removing trusted setups in one round while preserving the hardness of search problems and the security of most protocols against active adversaries.

The lower bounds in this paper provide an argument supporting that the complex game-based definitions of [AWZ23] are necessary, as the more natural simulation-based definition is unachievable without a random oracle. We point out that a simulation-based definition would be, in principle, desirable as there exist situations for which the definitions of [AWZ23] are not sufficient. For instance, their notion of a hardness-preserving distributed sampler does not allow removing the CRS from a NIZK while preserving soundness. This is because hardness-preserving distributed samplers preserve the hardness of games only when the challenger is efficient. Their second notion of indistinguishability-preserving distributed samplers also does not work in all contexts. For example, consider the functionality $\mathcal{F}$ that provides the adversary with several RSA moduli, lets the adversary choose one of them (denote the chosen modulus by N), and then allows MPC over $\mathbb{Z}_N$ (this is an interesting setting, e.g. for using MPC tools from [OSY21] that require a trusted setup). There exists a protocol Π that, given an RSA modulus as CRS, implements $\mathcal{F}$. However, if we apply an indistinguishability-preserving distributed sampler [AWZ23], the result Π' no longer implements the functionality $\mathcal{F}$ (this can be proven using an entropy based argument, as we did in this paper).

We highlight that the ideas of [AWZ23] cannot be used to construct unbounded universal samplers without random oracles. Indeed, the main obstacle that [AWZ23] managed to overcome is the unpredictability of the output of one-round protocols when the adversary adopts rushing behaviour. The main challenge of unbounded universal samplers is instead the incompressibility of ideal samples: how can we argue that the unbounded universal sampler produces outputs that look ideal if the entropy in the construction is even smaller than the entropy of one ideal sample? This issue is immediately inherited by all party-dynamic primitives we introduced in this work.

iO for Turing machines. Our construction for unbounded universal samplers uses garbled circuits to achieve succinctness, in a similar way to a construction of iO for Turing machines by Garg and Srinivasan [GS18]. The settings where these techniques are used are significantly different. One key difference is also that in our setting we are able to prove security relying only on polynomially secure primitives, while all existing constructions of iO for Turing machines rely on subexponentially secure primitives in their security proofs. We note that another construction of iO for Turing machines [BFK+19] uses the shared randomness model to avoid the size of the obfuscated program growing with a bound on the input. This is related to our use of shared randomness for removing the size dependency in our succinct universal sampler, however, the techniques are different.

Laconic function evaluation for Turing machines. Recently, Döttling, Gajland and Malavolta [DGM23] showed how to construct laconic function evaluation for Turing machines (TM-LFE), also using the techniques of [GS18]. Is it tempting to think that one can build unbounded universal samplers from the above primitive, as follows: the sampler consists of a TM-LFE hash key and an obfuscated program that, on input the digest of a distribution $\mathcal{D}$, outputs the TM-LFE encoding of a pseudorandom string r. In order to obtain a sample from the distribution $\mathcal{D}$, we retrieve the encoding produced by the obfuscated program on input a digest of $\mathcal{D}$. By decrypting this encoding using $\mathcal{D}$, we obtain $\mathcal{D}(1^\lambda; r)$ without learning any additional information.

This is not, however, an unbounded universal sampler: if we rely on a TM-LFE scheme satisfying simulation-based security, the size of the encoding produced by the obfuscated program is at least as big as the sample $\mathcal{D}(1^\lambda; r)$. Therefore, also the size of the sampler is bigger than the samples it produces. This is exactly what we want to avoid in unbounded universal samplers. In [DGM23], the authors also introduced a weaker indistinguishability-based security definition for TM-LFE, which can have encodings that are sublinear in the size of $\mathcal{D}(1^\lambda; r)$. However, this security definition is too weak for unbounded universal samplers: it would only guarantee that if $\mathcal{D}(1^\lambda; r) = \mathcal{D}(1^\lambda; r')$, the adversary cannot tell whether the sampler used r or r'.

Non-interactive key exchange. The setting of party-dynamic distributed samplers is similar to unbounded non-interactive key exchange (NIKE), which can be built using iO [KRS15]. NIKE is in some way similar to a distributed sampler for the uniform distribution, but it satisfies a weaker security definition: the output of the NIKE is guaranteed to look random only if no party is corrupted. This implies, for instance, that the derived output may depend only on the randomness of one party. Distributed samplers instead achieve security even when the adversary takes part in the computation. This difference allows NIKE to avoid many issues related to entropy.

One-round MPC. Distributed samplers can also be viewed as an inputless version of non-interactive MPC [HIJ+17]. We recall that non-interactive MPC unavoidably achieves a weak definition of security in which the adversary is allowed to learn the residual function (i.e. the function obtained by fixing the inputs of the honest parties while leaving the other inputs free). To achieve this, the primitive needs to rely on a PKI.

The fact that distributed samplers have no inputs gives a huge advantage: it allows us to satisfy a standard definition of security, without even needing PKIs. Notice that the naive idea of running an NIMPC protocol that, on input $r_1, \dots, r_n$, outputs $\mathcal{D}(1^\lambda; r_1 \oplus \cdots \oplus r_n)$ does not give a distributed sampler for $\mathcal{D}$, due to the residual function attack.

Two-round reusable MPC. Another related primitive is multi-party, reusable non-interactive secure computation (MrNISC) [BL20], which performs MPC in the party-dynamic setting with minimal interaction. In their construction, based on LWE, parties use the first round to publish encryptions of their input, and later, can publish second round messages for computing any desired function with a subset of parties. While related to distributed samplers, MrNISC does not allow secret randomness to be used in the function, unless it is encoded as part of the inputs in the first round; therefore, it does not seem to help with building a distributed sampler.

Roadmap.

In Section 3.2, we present a technical overview of our results. We describe preliminaries in Appendix 3.3.2. In Section 3.4, we formalise our lower bounds. We discuss succinct and unbounded universal samplers in [AOS23, Section 4]. Finally, we present our party-dynamic constructions in [AOS23, Section 5].

3.2 Technical Overview

We now give a high-level overview of the techniques used to obtain our results.

Notation. We denote the security parameter by λ. Even when not explicitly written, we assume that all random variables depend on λ. We use bold font to denote vectors, e.g. $\boldsymbol{v}$, single coordinates will be indicated using subscripts, e.g. v_i. The symbol $\sim_c$ denotes computational indistinguishability. We represent the set of corrupted players by C, the set of honest players is instead denoted by H. We indicate the bit-length of any string s by $|s|$. If c is a circuit, we use a similar notation $|c|$ to denote the number of gates. We use $\mathsf{struct}(c)$ to denote the structure of c. With an abuse of notation, we identify distributions $\mathcal{D}$ with circuits mapping uniformly random strings of bits into samples. We say that a distribution is efficient if its circuit has $\mathsf{poly}(\lambda)$ size.

3.2.1 (Im)possibility of Distributed Samplers without Random Oracle

As we motivated in the introduction, actively secure distributed samplers in the plain model with black-box simulation are impossible. In the CRS model, instead, they are trivial to build: the CRS can directly encode a sample from the underlying distribution. The result is a distributed sampler in which the parties do not even need to communicate, since they just output the CRS.

We study how interactive constructions can improve upon the trivial solution. In principle, the advantages can be multiple: the same CRS can be reused in many distributed sampler executions producing independent-looking outputs. Moreover, the CRS of distributed samplers can be nicer (i.e. easier to generate) than the

THE FUNCTIONALITY $\mathcal{F}_{\mathcal{D}}$

Initialisation. On input Init from every honest party and the adversary, the functionality activates and sets $Q := \emptyset$. (Q will be used to keep track of queries.) If all the parties are honest, the functionality outputs $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$ to every honest party and sends R to the adversary, then it halts.

Query. On input Query from the adversary, the functionality samples $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$ and creates a fresh label id. It sends (id, R) to the adversary and adds the pair to Q.

Output. On input $(\mathsf{Output}, \widehat{\mathsf{id}})$ from the adversary, the functionality retrieves the only pair $(\mathsf{id}, R) \in Q$ with $\mathsf{id} = \widehat{\mathsf{id}}$. Then, it outputs R to every honest party and terminates.

Figure 3.1: The distributed sampler functionality for rushing adversaries

direct encoding of a sample, for instance because of the smaller size, or because it is unstructured (i.e. a uniformly random string of bits). The result of our analysis is that none of the above properties can be satisfied: without a random oracle, distributed samplers essentially provide no advantage over the trivial solution. In order for this impossibility to hold, we do not even need to aim for active security, it suffices that the adversary is *strongly semi-malicious*: it may adaptively choose the randomness of the corrupted parties after seeing the honest messages, but all corrupted players follow the protocol.

On the Unpredictability of Distributed Samplers in the CRS Model.

All the negative results mentioned above are consequences of the main theorem of this work: in a strongly semi-malicious distributed sampler, where the underlying distribution $\mathcal{D}$ has high min-entropy, namely $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log \lambda)$, the Shannon entropy of the output conditioned on the CRS is $O(\log \lambda)$.

All through the paper we carefully juggle different variants of entropy, each bringing a unique set of properties we require during the proofs. Shannon entropy H has a powerful chain rule. Collision entropy H_2 gives us an elegant tool for building distinguishers, but lacks a chain rule and is not invariant under computational indistinguishability (i.e. for two computationally indistinguishable random variables, H_2 can be vastly different). We also use min-entropy H_∞, this is the smallest of the above mentioned and has the fewest properties. Our assumption on the entropy of the random source $\mathcal{D}$ is $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log \lambda)$ (clearly the task is trivial if $\mathcal{D}$ is constant) – this becomes the weakest assumption one can make using any of the above notions (and thus makes our theorem stronger). Finally, Yao's entropy is the only entropy we use that remains invariant under computational indistinguishability (i.e. two computationally indistinguishable random variables have the same Yao entropy). For formal definitions please refer to Section 3.3.2.

Distributed samplers against a rushing adversary. In order to understand the idea behind the result, we need to recall the definition of distributed samplers with security against an active adversary [ASY22]. The corresponding functionality provides the adversary with as many samples from the underlying distribution as the adversary wants. The adversary can then select one of these values; the functionality outputs it to all the honest parties. This kind of behaviour is needed to model the fact that, in the case of a rushing adversary, the corrupted parties see the honest messages before they publish their own. In other words, before committing to a choice, they can always test their candidate messages and discard them if they are not happy.

Definition 3.2.1 (Distributed sampler - security against rushing adversaries). Let $\mathcal{D}(1^\lambda)$ be an efficiently samplable distribution. An n-party actively secure (resp. strongly semi-maliciously secure) distributed sampler for $\mathcal{D}(1^\lambda)$ is a one-round protocol implementing the functionality $\mathcal{F}_{\mathcal{D}}$ (see Figure 3.1) against a static and active (resp. strongly semi-malicious) adversary corrupting up to $n-1$ parties.

The security model. We consider the UC model against the "dummy adversary", the one that simply follows the instructions given by the environment. We recall that a protocol is UC-secure if and only if it is secure against the dummy adversary [Can01]. In this setting, there exists a unique simulator that works for every environment. Since the role of the adversary is essentially assumed by the environment, we will use the terms adversary and environment interchangeably. We work in the dishonest majority setting.

Our proof will only consider adversaries that behave honestly, i.e. they choose the randomness of the corrupted parties uniformly at random and they follow the protocol. Notice that since we are proving a lower bound, considering very weak adversaries such as the honest one makes our results even stronger.

In the ideal world, the outputs are restricted to a small set. The simulator of the distributed sampler needs to provide the honest parties' messages and the CRS to the adversary before learning the choices of the corrupted players. Since the simulator runs in polynomial time, the number of samples received from the functionality before the delivery is polynomially bounded. Let the corresponding set be Q.

Once the adversary supplies the corrupted messages, the output of the protocol is fixed (indeed, we cannot rewind the adversary, as the UC model does not allow it). If the latter belongs to Q, the simulator can easily instruct the functionality to output the right sample to all honest players. If instead that is not the case, the only choice left for the simulator is to keep querying the functionality for new samples and hope for a collision. Since the distribution has high min-entropy, this occurs with negligible probability. In other words, the output must belong to Q with overwhelming probability. If that does not happen, the adversary can easily distinguish the real protocol from the ideal world as the simulator is not able to make the honest parties output the right result.

In the real world, the output is easily predictable from the CRS and the messages of the honest parties. Let R denote the output of the distributed sampler, let crs be the CRS and let U_H and U_C denote the messages of the honest and the corrupted parties respectively. The fact that the CRS and the messages of the honest parties restrict the output is a set of polynomial size is a strong property. In particular, the latter implies that $\mathsf{H}(R|\mathsf{crs}, U_H) = O(\log \lambda)$. This equality holds in the ideal world, but what about the real world? Unfortunately, Shannon's entropy does not behave well under computational indistinguishability, i.e. computationally indistinguishable random variables may have very different entropy. We prove, however, that if the adversary honestly follows the protocol in the real world, $\mathsf{H}(R|\,\mathsf{crs}, U_H) = O(\log \lambda)$.

Consider the distinguisher that, after receiving the CRS and U_H, keeps regenerating the messages of the corrupted parties following the protocol, and stores the outputs obtained in this way. In the ideal world, the distinguisher will never obtain more than $q(\lambda)$ different samples, where $q(\lambda)$ is a polynomial upper-bound on the cardinality of Q, the set of values queried by the simulator to the functionality. We notice that without loss of generality $q(\lambda)$ is known to the distinguisher as the simulator is fixed.

Using a technical argument based on entropy, we show that if $\mathsf{H}(R|\,\mathsf{crs}, U_H)$ is not $O(\log \lambda)$, in the real world, there exists a non-negligible function $\delta(\lambda)$ such that for every polynomial $j(\lambda)$, the j-th output obtained by the distinguisher differs from all the previous ones with probability at least $\delta(\lambda)$. The crucial point is that $\delta(\lambda)$ is independent of j. Indeed, as j increases, the probability of obtaining new outputs becomes lower (the probability of colliding with one of the previous outcomes gets higher and higher). If this probability decreases too fast, the number of different outputs obtained by the distinguisher may converge to a certain threshold smaller than $q(\lambda)$. Since the probability is always bounded from below by $\delta(\lambda)$, however, in the real world, the distinguisher is able to obtain more than $q(\lambda)$ different outputs in a polynomial number of steps. This is sufficient to break the security of distributed samplers.

The final result: an easy application of the strong chain rule. At this point, proving our theorem becomes simple. Since we are considering an honest adversary, the result described in the previous paragraph immediately implies that $\mathsf{H}(R|\,\mathsf{crs}, U_C)$ is also $O(\log \lambda)$. Furthermore, U_H is independent of U_C, given the CRS. In other words, $\mathsf{H}(U_H|\mathsf{crs}) = \mathsf{H}(U_H|\mathsf{crs}, U_C)$. By a simple application of the strong chain rule for Shannon's entropy, it is easy to show that $\mathsf{H}(R|\,\mathsf{crs}) = O(\log \lambda)$.

Indeed, consider the entropy diagram in Figure 3.2.[5] Observe that $\mathsf{H}(R|\mathsf{crs})$ corresponds to the union of

[5]The diagram is not completely general as some of the intersections between the sets are empty, however, the figure is sufficiently generic to describe our argument.

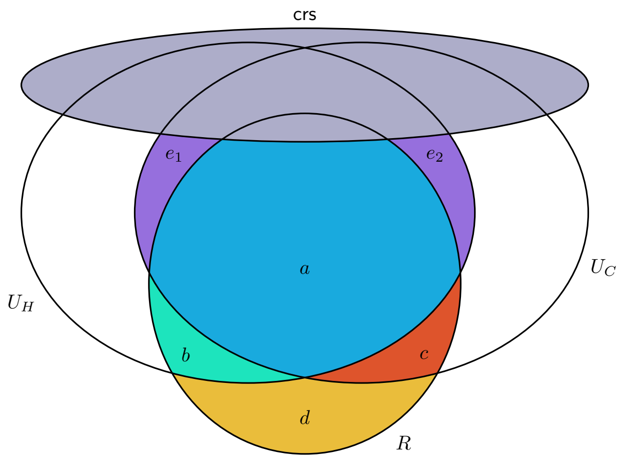


Figure 3.2: Entropy diagram of the distributed sampler.

the blue, red, green and yellow areas, i.e. $\mathsf{H}(R|\mathsf{crs}) = a+b+c+d$. We know that $\mathsf{H}(R|\mathsf{crs}, U_H)$ corresponds to the union of the red and yellow areas, so, $c+d = \mathsf{H}(R|\mathsf{crs}, U_H)$. Similarly, $\mathsf{H}(R|\mathsf{crs}, U_C)$ corresponds to the union of the green and yellow areas, so, $b+d = \mathsf{H}(R|\mathsf{crs}, U_C)$. We also observe that the union of the blue and purple areas correspond to $\mathsf{H}(U_H|\mathsf{crs}) - \mathsf{H}(U_H|\mathsf{crs}, U_C) = 0$, so $a+e_1+e_2 = 0$. Finally, we notice that both e_1+e_2 and d are non-negative. Indeed, the former corresponds to $\mathsf{H}(U_H|\mathsf{crs}, R) - \mathsf{H}(U_H|\mathsf{crs}, R, U_C) \geq 0$, whereas the latter corresponds to $\mathsf{H}(R|\mathsf{crs}, U_H, U_C) \geq 0$. The fact that $e_1+e_2 \geq 0$ also implies that $a \leq 0$, so

$$\mathsf{H}(R|\mathsf{crs}) = a+b+c+d \leq b+c+2d = \mathsf{H}(R|\mathsf{crs}, U_H) + \mathsf{H}(R|\mathsf{crs}, U_C) = O(\log\lambda).$$

Bad News for Distributed Samplers.

All the results we discuss below hold in absence of a random oracle and for distributed samplers that achieve UC-security against a strongly semi-malicious adversary.

Distributed sampler CRSs cannot be used twice. The first corollary of Theorem 3.1.2 is that two distributed sampler executions using the same CRS have colliding outputs with non-negligible probability. We recall that our theorem applies when the min-entropy of the underlying distribution is high, i.e. $\omega(\log\lambda)$. For all such distributions, the collision probability is negligible, i.e. two independent samples from $\mathcal{D}(1^\lambda)$ will almost always be different. As a consequence, by reusing the same CRS twice, we obtain samples that do not look independent.

The reason at the base of our first corollary is that, by a simple application of Jensen's inequality, the average collision entropy $\widetilde{\mathsf{H}}_2(R|\,\mathsf{crs})$ is bounded from above by $\mathsf{H}(R|\,\mathsf{crs}) = O(\log\lambda)$. We recall that the average collision entropy is defined as

$$\widetilde{\mathsf{H}}_2(R|\,\mathsf{crs}) := -\log\big(\mathbb{P}[R = R']\big)$$

where R and R' are two distributed sampler outputs computed using the same CRS crs and the probability is also over the randomness of crs. We conclude that $\mathbb{P}[R = R'] \geq 1/\mathsf{poly}(\lambda)$.

Distributed sampler CRSs are long. We prove that CRSs of strongly semi-malicious distributed samplers cannot be small: they can be at most $O(\log\lambda)$ bits shorter than the Yao entropy of the underlying distribution $\mathsf{H}_{\mathsf{Yao}}(\mathcal{D})$[6].

[6]The Yao entropy of $\mathcal{D}$ roughly measures how much a sample from $\mathcal{D}$ can be compressed in polynomial time without losing information.

We prove this result by first observing that $\mathsf{H}_{\mathsf{Yao}}(R|\,\mathsf{crs}) = O(\log\lambda)$. Indeed, as we motivated in the previous paragraph, two distributed sampler executions using the same CRS have colliding outputs with non-negligible probability. We can therefore consider the Yao's compressor that outputs nothing and the associated decompressor that, provided with the CRS crs, reruns the distributed sampler protocol in its head and outputs the result R'. With $1/\mathsf{poly}(\lambda)$ probability, R' coincides with the input of the compressor.[7] This is enough to conclude that $\mathsf{H}_{\mathsf{Yao}}(R|\,\mathsf{crs}) = O(\log\lambda)$.

We then show that $\mathsf{H}_{\mathsf{Yao}}(R|\,\mathsf{crs}) \geq \mathsf{H}_{\mathsf{Yao}}(R) - |\mathsf{crs}|$. We prove this by noticing that, given a compressor-decompressor pair (c', d') for $\mathsf{H}_{\mathsf{Yao}}(R|\,\mathsf{crs})$, we can build a compressor-decompressor pair (c, d) for $\mathsf{H}_{\mathsf{Yao}}(R)$ as follows: c provides its input R to the distributed sampler simulator, corrupting no party. It obtains a fake CRS crs' that looks like the real one. It then outputs $c'(R, \mathsf{crs}')$ along with crs'. The decompressor d is exactly the same as d'. The success probability of (c, d) is the same as for (c', d') except for a negligible quantity. The size of the compressed string has however grown by $|\mathsf{crs}|$ bits, increasing $\mathsf{H}_{\mathsf{Yao}}(R)$ by the same amount.

We point out that, in order to prove the above inequality, we cannot use the Yao chain rule of [KPW13, Appendix B] as their compressor for $\mathsf{H}_{\mathsf{Yao}}(R)$ has $O(2^{|\mathsf{crs}|})$ size.

Distributed sampler CRSs are ugly. Suppose that there exists a strongly semi-malicious distributed sampler for the distribution $\mathcal{D}$ having CRS crs. We prove that it is possible to *non-interactively* and *securely* generate a sample from $\mathcal{D}$ given only crs and public random coins. In other words, if there exists a distributed sampler with nice CRS, also the underlying distribution can be encoded in a nice CRS. The second solution may be preferable as it often requires less communication. As an additional corollary, if the distributed sampler uses a URS (i.e. the CRS is a random string of bits), we can sample from $\mathcal{D}$ using just public random coins. So, in the random oracle model, we would not even need a CRS.

Our idea is that, given crs and public random coins, each party can just rerun the distributed sampler protocol with crs as CRS and the public coins as randomness for the players. The result R is clearly indistinguishable from a sample from $\mathcal{D}$. However, in order to prove that this protocol is secure, we need to be able to simulate crs and the public coins, given R.

We simulate crs by feeding R to the distributed sampler simulator (we corrupt no party). Unfortunately, the simulator cannot provide us with the randomness used by the parties. We proceed by brute-force: we rerun the protocol in our head using the fake CRS and we hope that the output collides with R. If we fail, we retry sampling a new fake CRS. Once we succeed, we output the fake crs and the randomness of the parties that led to the collision.

By the first corollary of Theorem 3.1.2, we know that, *on average* over R, the collision probability is $1/\mathsf{poly}(\lambda)$. So, for a *polynomial fraction* of all possible values R, the simulation succeeds after a polynomial number of tries. For the remaining fraction of the support of $\mathcal{D}$, our approach fails, meaning that the CRS and the randomness of the parties might leak too much information about the output.

In other words, the sampling protocol we described is secure only for a polynomial fraction of the support of $\mathcal{D}$. The good news is that it is possible to tell if the result of our non-interactive sampling protocol lies in the secure subset or not: the parties can locally run the simulator. If it succeeds with sufficiently high frequency, they can be sure their output is secure, otherwise, they need to discard it, generate a new crs and public coins and rerun the protocol. Since there is a polynomial fraction of the support of $\mathcal{D}$ that will not be discarded, the players need a polynomial number of attempts before succeeding. We also point out that the distribution of the outputs will be biased, but not significantly: if $\mathcal{D}$ describes the distribution of another protocol's CRS, it is still secure to use the outputs of our procedure as CRSs for such protocol.

How General is the Impossibility?

Our arguments seem to apply not only to the UC model but also to the more powerful settings of security with superpolynomial simulation, and standalone security with rewinding. Informally, what Theorem 3.1.2 is saying is that the size of the distributed sampler messages sets an information-theoretic bound B on the

[7]We can make the decompressor deterministic using a PRF.

number of samples that a simulator can encode in the messages it produces. An adversary can rerun the distributed sampler protocol in its head a number of times that is significantly larger than B. In the real world, it is supposed to obtain more than B distinct outputs, on the other hand, in the ideal world, this does not happen. This suggests that the impossibility holds even if we rely on superpolynomial simulation.

Even rewinding does not seems to help: in the ideal world, with high probability, the output R of the distributed sampler has non-negligible probability of being resampled (i.e., if the distinguisher reruns the protocol in its head, regenerating the messages of the corrupted players, it has a high chance of reobtaining R after a few tries). This is because R was the result of the rewinding process. If R had a low probability of being resampled, the probability that rewinding output R would have been low in the first place. On the other hand, in the real world, R has very low probability of being resampled (we want $\mathsf{H}(R|\sigma, U_H) = \omega(\log \lambda)$, otherwise, we rerun into the problems of the UC model). This leads to a successful attack. Whether these ideas can be formalised will be part of future work.

3.2.2 Constructing Unbounded Universal Samplers

Our first positive result is a construction of an unbounded universal sampler in the shared randomness model. Recall that in a universal sampler (US), the trusted setup algorithm outputs some sampler parameters U, which are later used to securely sample from a distribution $\mathcal{D}$. Our goal is to ensure that the size of the circuit that samples from $\mathcal{D}$ may be unbounded, and in particular, independent of U.

Succinct, Bounded Universal Samplers.

We start by building a US that is not totally unbounded, but is succinct, meaning that the size of U is only polylogarithmic in the maximum circuit size L of the supported distribution $\mathcal{D}$. To see the challenge in achieving this, recall that the sampler parameters in the selective, one-time universal sampler by Hofheinz *et al.* [HJK+16] consist of an obfuscated program. To sample from a distribution $\mathcal{D}$, the program is fed with the circuit describing $\mathcal{D}$. It then uses a puncturable PRF to generate random bits used to sample from $\mathcal{D}$ and outputs the result. If we want to obtain succinctness then there is no way the obfuscated program can evaluate the sampling circuit, which may now be significantly larger than the sampler parameters. Therefore, we cannot even provide $\mathcal{D}$ as input to the program, let alone evaluate it.

Taking advantage of the locality of garbled circuits. Our solution is to use garbled circuits. We obfuscate a program $\mathsf{SUSProg}$, which, instead of evaluating $\mathcal{D}$ itself, will output a garbling of $\mathcal{D}$ along with one random label for each input wire and both labels for each output wire. At any point in time, a party can evaluate the garbled circuit produced by $\mathsf{SUSProg}$ obtaining a sample from $\mathcal{D}$.

The big advantage of garbled circuits is its locality: as long as there is way to retrieve the labels associated with the input and output wires of any gate g, we can garble g without knowing the whole circuit to which g belongs. Specifically, each execution of $\mathsf{SUSProg}$ takes as input a single gate of $\mathcal{D}$ and outputs its garbling. The description of the gate will consists of a type (input, output, XOR or AND) and identifiers for the input and output wires of the gate[8]. Since the operations $\mathsf{SUSProg}$ needs to perform are now independent of $\mathcal{D}$, the size of $\mathsf{SUSProg}$ can remain small. A similar idea was adopted by Garg and Srinivasan for the construction of obfuscation for Turing machines [GS18] (see Section 3.1.2 for more discussion).

Making the garbled gates consistent. The first problem is that we need to ensure that different gates are garbled consistently, in that whenever a wire of the circuit is re-used, the same wire labels are used. As a consequence, all the executions of $\mathsf{SUSProg}$ associated with $\mathcal{D}$ cannot be independent, they all need to have access to some common information.

To ensure this, we use a master garbling key mk to derive, using a PRF F, the randomness needed by the garbling and the random bits given as input to $\mathcal{D}$. Formally, the labels associated with a wire w will be

[8]Notice that the terminology distinguishes between input gate and input wire of a gate. The first one is used to denote an input to the *circuit*, the second one is used to denote the input to a *gate*, i.e. the bit to which we apply an XOR or an AND. A similar discussion applies to output gates and output wires.

```
THE PROGRAM $\mathcal{P}_{\mathsf{SUS}}[K, \mathsf{hk}]$

Hardcoded: A PRF key $K$ and an SSB hash key $\mathsf{hk}$.
Input: SSB hash $z$ of $\mathcal{D}$, index $i$, gate $g$ and SSB proof $\pi$.

    1. If $\mathsf{Hash.Verify}(\mathsf{hk}, z, i, g, \pi) = 0$, output $\perp$.
    2. $\mathsf{mk} \leftarrow F_1(K, z)$
    3. Output $\mathsf{Garble}(1^\lambda, g, \mathsf{mk})$
```

Figure 3.3: Warm-up attempt for the unobfuscated SUS program

$(k_w^0, k_w^1) \leftarrow F(\mathsf{mk}, w)$. For each input gate g, we also use F to sample a random input bit, and give out the corresponding wire label. For each XOR or AND gate g, we additionally use F to sample a permutation to reorder the ciphertexts. For each output gate, we provide both wire labels.

We observe that every execution of SUSProg associated with $\mathcal{D}$ needs to retrieve the same key mk. Furthermore, different distributions $\mathcal{D}$ and $\mathcal{D}'$ need to use independent-looking garbling keys. If that is not the case, we risk garbling different circuits using the same labels, which would compromise privacy.

We solve these issues by providing SUSProg also with a hash z of the circuit $\mathcal{D}$. Since the size of z is $O(\log L)$, we can input it to SUSProg without any troubles. The obfuscated program SUSProg will be equipped with a puncturable PRF F_1 and a key K. Using z as input for F_1, SUSProg will retrieve mk and use the latter to garble the provided gate. By the collision resistance of the hash function, different distributions will correspond to different hashes and so, by the security of the puncturable PRF, to independent-looking garbling keys. To make this argument compatible with indistinguishability obfuscation, we use a somewhere statistically binding (SSB) hash function [HW15].

Limiting the leakage using SSB hashing. So far, nothing prevents the adversary from garbling a circuit using SUSProg while providing an inconsistent digest z. This means that the adversary can retrieve the randomness used to produce the sample from $\mathcal{D}$ by simply garbling the identity function along with $z = \mathsf{Hash}(\mathcal{D})$.

Luckily, SSB hash functions help us in countering this attack. Indeed, SSB hashing can be used to prove that a certain gate g is the i-th element in the preimage of z. So, if we provide the proof along with z, g, i and the SSB hash key hk, the obfuscated program is able to check if g really is the i-th gate of $\mathcal{D}$. If the verification succeeds, the program can garble g using mk, otherwise, it can simply output $\perp$.

SSB hash functions set an upper bound on the length of the messages that can be hashed. In our construction, we set this to $L(\lambda)$ blocks[9]. A nice feature of some SSB hashing schemes [HW15] is that both the hash key and the SSB proofs have size $O(\log L)$. Furthermore, the proofs can also be verified in $O(\log L)$ time. In other words, verifying the proofs in the code of SUSProg does not blow up the size of the program.

We present the construction so far in the program shown in Figure 3.3. To summarise, the adversary can make SUSProg output only the garbling of $\mathcal{D}$ or independent-looking information. Indeed, any execution inputting a hash other than z would lead to an independent-looking garbling key and hence, independent-looking information. If instead z is input, all the adversary can receive is the garbled gates of $\mathcal{D}$. If it tries to provide a different gate, the hash check will fail.

Taking control over the outputs with a trapdoor. To prove security, we need to argue that our program reveals no information in addition to the output of the garbled circuit. This is formalised by saying that we can simulate SUSProg given a sample R from $\mathcal{D}$. Clearly, the simulated SUSProg needs to output

[9]Each block will be the description of a different gate.

R when run on $\mathcal{D}$. Unfortunately, our obfuscated program cannot satisfy this property in the current state. Indeed, the sample R may contain significantly more information than the size of $\mathsf{SUSProg}$.

Here, we rely on the shared randomness model, where we require any party obtaining a sample to have a long, uniform string $\boldsymbol{u}$. Using $\boldsymbol{u}$, we equip $\mathsf{SUSProg}$ with a trapdoor that allows us to program its output in the security proof; we do this using the delayed backdoor programming technique from the adaptive universal sampler in [HJK+16], also used in the malicious constructions of [ASY22]. To garble the i-th gate g_i, we provide $\mathsf{SUSProg}$ with a u_i, corresponding to the i-th block of the randomness $\boldsymbol{u}$. We hardcode into our program an additional key k for a special kind of authenticated encryption scheme. In each execution, after verifying the SSB proofs, $\mathsf{SUSProg}$ tries to decrypt u_i using k. If decryption succeeds, the program outputs the underlying plaintext, otherwise it resumes its usual behaviour, i.e. it garbles the provided gate.

The encryption scheme, which is based on puncturable PRFs, is designed so that ciphertexts are indistinguishable from random strings, but the overwhelming majority of strings are not valid ciphertexts. When a random u_i is input into $\mathsf{SUSProg}$, then, the probability of activating the trapdoor is negligible. In the simulation, however, $\boldsymbol{u}$ will be the encryption of a garbled circuit simulated using R and $\mathcal{D}$. By the security of iO, the adversary will not be able to tell if the output is generated using the trapdoor or the standard procedure.

Binding the trapdoor to the distribution. Finally, there is one weakness remaining in the construction: we need to bind the random string $\boldsymbol{u}$ to the distribution $\mathcal{D}$. At the moment, the adversary can easily tell if $\boldsymbol{u}$ hides the encryption of a random circuit or not. It can simply garble $\mathcal{D}$ twice, once using $\boldsymbol{u}$ and once inputting a random string. If the outputs differ, it must be that $\boldsymbol{u}$ activates the trapdoor.

Clearly, we cannot prevent the adversary from choosing the distribution and the random string as it pleases, however, we can make sure that for different choices of $(\mathcal{D}, \boldsymbol{u})$, we obtain independent-looking executions. Specifically, instead of equipping $\mathsf{SUSProg}$ with a hardcoded trapdoor key k, we generate k along with mk using the PRF F_1. Recall that the input given to F_1 is a hash of $\mathcal{D}$. In this way, different distributions would use different trapdoor keys and so $\boldsymbol{u}$ would activate the trapdoor only in conjunction with $\mathcal{D}$.

Finally, we also want to ensure that when given different random strings, the garbled circuit output by $\mathsf{SUSProg}$ changes. That corresponds to having a different garbling key mk. To ensure this, in each execution, we provide $\mathsf{SUSProg}$ also with an SSB hash h of $\boldsymbol{u}$. We then input h into the puncturable PRF F_1 along with z. In conclusion, we obtain a different garbling key and a different trapdoor key for every choice of distribution and random string. To ensure that the string u_i input to the program is consistent with the hash h, we additionally modify $\mathsf{SUSProg}$ to receive an SSB proof that u_i is the i-th block of the preimage of h. The program checks the proof and outputs the garbled gate only if the verification succeeds. Otherwise, it outputs $\perp$.

To summarise, if the adversary does not input (h, z) into $\mathsf{SUSProg}$, the program outputs information that looks independent of the sample R. If it inputs (z, h) instead, the adversary is forced to provide a pair (g_i, u_i) for a certain $i \in [L]$ where g_i denotes the i-th gate of $\mathcal{D}$. If this is the case, the adversary receives the scheduled garbling of g_i, otherwise, it receives $\perp$.

We present an informal description of the final version of the program in Figure 3.4. For the complete, formal construction and its security proof, we refer to [AOS23, Section 4.1].

From Succinct to Unbounded Universal Samplers.

Once we have a succinct, but bounded, US, it is quite straightforward to obtain an unbounded US. Our construction will simply run the setup procedure from the succinct US, and output its sampler parameters U. This already allows us to sample from any distribution $\mathcal{D}$ up to some polynomial bound. To sample from a larger $\mathcal{D}$, we simply use U to run the setup algorithm for a second succinct US, with a bound of twice the size (since the first US was succinct, this will always be possible for a sufficiently large security parameter). This process is then iterated until we have a sampler that can support the distribution $\mathcal{D}$.

For technical reasons, to prove this construction secure we need an additional property of the unbounded US, which we call randomness extractability. Intuitively, this says that given a sampler output R and the

```
THE PROGRAM $\mathcal{P}_{\mathsf{SUS}}[K, \mathsf{hk}]$

Hardcoded: A PRF key $K$ and an SSB hash key $\mathsf{hk}$.
Input: SSB hashes $h$ and $z$ of $\boldsymbol{u}$ and $\mathcal{D}$ respectively, index $i \in [L]$, random string $v$, gate $g$ and SSB proofs $\pi$ and $\pi'$.

  1. $b \leftarrow \mathsf{Hash.Verify}\big(\mathsf{hk}, h, i, v, \pi\big)$
  2. $b' \leftarrow \mathsf{Hash.Verify}\big(\mathsf{hk}, z, i, g, \pi'\big)$
  3. If $b = 0$ or $b' = 0$, output $\bot$.
  4. $(\mathsf{mk}, k) \leftarrow F_1\big(K, (h, z)\big)$
  5. $x \leftarrow \mathsf{Dec}(k, v)$
  6. If $x \neq \bot$, output $x$.
  7. Otherwise, output $\mathsf{Garble}(1^\lambda, g, \mathsf{mk})$
```

Figure 3.4: Informal description of the unobfuscated SUS program

randomness that was used to compute the sampler parameters, it is possible to extract the randomness that "explains" the output R from distribution $\mathcal{D}$. We show that this property holds for our construction, and in fact is easily achievable in a generic way for any universal sampler.

3.2.3 Building Unbounded and Party-Dynamic, Distributed Universal Samplers

Our next goal is to obtain unbounded *distributed universal samplers*, where the sampler is derived from n messages, one from each out of a set of n parties. As well as allowing the choice of distribution $\mathcal{D}$ to be unbounded, and not tied to the sampler parameters, here we also want the sampler to be *party-dynamic*, so the set of parties can be chosen dynamically from an unbounded set of possible parties.

A toy construction of a *bounded*, party-dynamic distributed sampler can be easily obtained from any n-party distributed sampler for a fixed number of parties: each party simply runs the i-party distributed sampler algorithm, for $i = 2, \dots, n$, and publishes all the $n-1$ messages. Of course, this construction requires the size of each message to scale at least linearly with n.

To get an unbounded construction, we modify this blueprint by instead having each party publish a single message consisting of an unbounded universal sampler. Later, to sample from a distribution with some size-n subset of the parties, those parties' unbounded USes will each be used to generate an n-party distributed sampler message on-the-fly. Since we use an unbounded US, this construction is inherently tied to the shared randomness model, where the subset of n parties must all hold a common string of uniform bits to obtain their sample. We prove security in the one-time setting, against a non-rushing and semi-malicious adversary.

Modelling Active Security. In the non-rushing setting, modelling security is quite straightforward and similar to the case of non-party-dynamic definitions. When moving to an active adversary, however, we have to be careful how to define security. Recall that with a static number of parties, active security of a distributed sampler is defined using an ideal functionality, which allows the adversary to obtain several samples from the distribution $\mathcal{D}$, before settling on one it likes. This corresponds to the fact that in a construction, every choice of a corrupt party's randomness may lead to a different result from $\mathcal{D}$.

In the party-dynamic setting, we consider a static adversary in the UC model: whenever a new party joins the system, the adversary must decide whether that party is corrupted or not. At the same time, we need a way to model the fact that the adversary can try candidate messages of a corrupt party P_j, obtaining different samples, before P_j has actually joined the system. To do this, we allow the adversary to input a label id_j, corresponding to a new choice of message for P_j, and can then obtain a sample for the desired subset of parties that includes P_j. When P_j eventually joins, the adversary can either choose one of the previously sent labels, "fixing" the relevant outputs to the corresponding samples, or choose a fresh label which leads to freshly sampled outputs.

Achieving Active Security and Reusability. Next, we upgrade our construction to be actively secure, and also reusable for an unbounded number of queries on arbitrary distributions. We do this in a black-box way, starting from any one-time secure, party-dynamic construction. The main idea is to have each party publish an *adaptive* (or reusable), bounded universal sampler [HJK+16] as its message, together with a NIZK showing that it is well-formed. Then, whenever a subset of parties wants to obtain a sample, the adaptive US is used to generate a message for a one-time, party-dynamic distributed US. By relying on a reusable US, we ensure that each message from the one-time, party-dynamic construction is only used once. Recall that our one-time, party-dynamic construction requires a source of public, shared randomness $\boldsymbol{u}$ to obtain the sample; to generate $\boldsymbol{u}$ in a reusable way, we use a random oracle.

There is still one problem with this approach, though. An adversary may still adaptively choose the messages of the corrupt parties, and the distribution $\mathcal{D}$, *after* seeing the honest parties' messages from the one-time, party-dynamic distributed US (which is not secure against a rushing adversary). To fix this, we again rely on the random oracle model. We force the adversary to commit to its messages before seeing these messages, by making it query the random oracle with input the subset of parties, distribution $\mathcal{D}$, and adaptive universal sampler messages. The output of the random oracle is a λ-bit tag, which is fed into the adaptive US before generating the one-time messages. Since the tag is unpredictable, this ensures that the adversary cannot learn any outputs without first committing to its messages.

Party-Dynamic, Public-Key Pseudorandom Universal Correlation Functions (PCFs).

Our last construction is an application of party-dynamic distributed universal samplers, for generating correlated randomness. A public-key PCF [BCG+20, ASY22] is a one-round protocol for securely sampling from n correlated random variables, where each party obtains one of the outputs, while learning nothing of the other parties' outputs. We show how to build public-key PCFs in the party-dynamic setting where the correlation is adaptively chosen after the messages of the parties are sent. Our construction is quite simple, and follows the blueprint of the previous construction for a fixed number of parties [ASY22]: each party sends a public key for a PKE scheme, plus a message for a distributed universal sampler. The distributed universal sampler messages are then used to sample from the distribution that encrypts the n outputs of the correlation function under each of the parties' public keys, allowing only the correct party to recover its output. By relying on our party-dynamic distributed universal sampler, we immediately obtain a public-key, universal PCF in the party-dynamic setting.

We present the construction directly in the actively secure and reusable setting (in the random oracle model). Because of this, we achieve the stronger notion of an ideal public-key PCF, which securely realizes the ideal sampling functionality (with suitable relaxations to account for rushing adversaries). In contrast, without a random oracle, this type of PCF is impossible to achieve, unless one allows the parties' messages to be as long as the total output length of all queries to the correlation.

3.3 Preliminaries

3.3.1 Distributed Samplers

Distributed samplers (DS) are a strong primitive allowing n parties to securely generate CRSs with a single round of interaction. Specifically, a distributed sampler for the distribution $\mathcal{D}(1^\lambda)$ is a one-round protocol that generates a sample R from $\mathcal{D}(1^\lambda)$ without revealing any information except R itself.

The notion was introduced for the first time by Abram, Scholl and Yakoubov in [ASY22]. In the paper, the authors show how to build the primitive from indistinguishability obfuscation and multi-key FHE. In this section, we recall their definition considering multiple adversarial models.

We start by consider security against a weakly semi-malicious adversary, i.e. a *non-rushing* adversary that, as in the semi-honest model, follows the protocol, but before beginning the execution, it chooses the random tapes of the corrupted parties as it prefers. If the adversary follows the protocol but instead chooses the randomness of the corrupted parties after seeing the honest messages, we say that we are dealing with a *strongly semi-malicious* adversary.

Definition 3.3.1 (Weakly semi-maliciously secure distributed sampler). Let $\mathcal{D}(1^\lambda)$ be an efficiently samplable distribution. An n-party distributed sampler (DS) for $\mathcal{D}(1^\lambda)$ is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{Sample})$ having the following syntax:

1. Gen is a probabilistic algorithm taking as input the security parameter 1^λ and the index i of the party running it. The output is the distributed sampler message U_i of the i-th party. We assume that Gen needs $M(\lambda)$ bits of randomness.

2. Sample is a deterministic algorithm taking as input n distributed sampler messages $(U_j)_{j\in[n]}$, one for each party. The output is a sample R.

We say that the distributed sampler is *weakly semi-maliciously* secure if there exists a PPT simulator Sim such that, for every subset $C \subsetneq [n]$ of corrupted parties and associated randomness $(\rho_i)_{i\in C}$, the following two distributions are computationally indistinguishable.

$$\left\{ \begin{array}{c} (U_i)_{i\in H} \\ (\rho_i)_{i\in C}, R \end{array} \middle| \begin{array}{ll} \rho_i \xleftarrow{\$} \{0,1\}^{M(\lambda)} & \forall i \in H \\ U_i \leftarrow \mathsf{Gen}(1^\lambda, i;\, \rho_i) & \forall i \in [n] \\ R \leftarrow \mathsf{Sample}(U_1, \dots, U_n) & \end{array} \right\}$$

$$\left\{ \begin{array}{c} (U_i)_{i\in H} \\ (\rho_i)_{i\in C}, R \end{array} \middle| \begin{array}{l} R \xleftarrow{\$} \mathcal{D}(1^\lambda) \\ (U_i)_{i\in H} \xleftarrow{\$} \mathsf{Sim}\big(1^\lambda, C, R, (\rho_i)_{i\in C}\big) \end{array} \right\}$$

The security definition essentially states that even for the worst randomness choice of the corrupted parties, the honest messages leak no information except the output itself. Observe that if we run Sample over the simulated messages, the output coincides with R with overwhelming probability. Notice that any adversary corrupting no party but just listening to the conversations is always able to obtain the output. Indeed, the latter is just a deterministic function of the transcript.

It is possible to reformulate the above definition by saying that weakly semi-maliciously secure distributed sampler is a one-round protocol implementing the functionality that provides all the parties with the same sample R from $\mathcal{D}(1^\lambda)$. Unfortunately, it is impossible to implement the above functionality against rushing adversaries. Indeed, after receiving the messages of the honest parties, the adversaries can always rerun the protocol in its head multiple times, changing only the messages of the corrupted parties. In this way, the attacker obtains multiple samples from $\mathcal{D}(1^\lambda)$, it can therefore choose the one it likes the most and send the corresponding corrupted messages in the protocol. In other words, the adversary can always choose the output among a set of polynomially many samples. For this reason, in the rushing setting, distributed samplers are defined as in Definition 3.2.1.

3.3.2 Notions of Entropy

In information theory, entropy is used to measure the unpredictability of random variables. After almost a century of research, several definitions have been formalised. In this appendix, we recall some of the important notions and the related properties. We start with Shannon's entropy [Sha48].

Definition 3.3.2 (Shannon's entropy). Let X be a random variable having finite support. The Shannon's entropy of X is

$$\mathsf{H}(X) := -\sum_x \mathbb{P}[X = x] \cdot \log\big(\mathbb{P}[X = x]\big).$$

We recall also the notion of conditional Shannon's entropy.

Definition 3.3.3 (Conditional Shannon's entropy). Let X and Y be random variables having finite support and let E be an event. The Shannon's entropy of X conditioned on E is

$$\mathsf{H}(X|E) := -\sum_x \mathbb{P}[X = x|E] \cdot \log\big(\mathbb{P}[X = x|E]\big).$$

The Shannon's entropy of X conditioned on Y is instead

$$\mathsf{H}(X|Y) := \sum_y \mathbb{P}[Y = y] \cdot \mathsf{H}(X|Y = y).$$

Shannon's entropy satisfies an important property called *the strong chain rule*. We recall it below.

Theorem 3.3.4 (Strong chain rule). Let X and Y be random variables with finite support. Then,

$$\mathsf{H}(X, Y) = \mathsf{H}(Y) + \mathsf{H}(X|Y).$$

Notice that (X, Y) is a random variable, so $\mathsf{H}(X, Y)$ is defined as in Definition 3.3.2. We also recall the following properties of Shannon's entropy.

Lemma 3.3.5. Let X, Y and Z be random variables with finite support. Then,

- If X is uniform over a set of cardinality m, $\mathsf{H}(X) = \log m$.
- If X is independent of Y, given Z, $\mathsf{H}(X|Y, Z) = \mathsf{H}(X|Z)$.
- $\mathsf{H}(X|Y, Z) \leq \mathsf{H}(X|Z)$.
- If f is a deterministic function, $\mathsf{H}(f(X)) \leq \mathsf{H}(X)$.

We now recall other definitions of entropy that are used to prove our results.

Definition 3.3.6 (Max entropy). Let X be a random variable with finite support, let E be an event. We define the max entropy of X to be

$$\mathsf{H}_0(X) = \log|\mathsf{Supp}(X)|.$$

We define the max entropy of X conditioned on E to be

$$\mathsf{H}_0(X|E) = \log|\mathsf{Supp}(X|E)|.$$

Definition 3.3.7 (Min entropy). Let X be a random variable with finite support, let E be an event. We define the min entropy of X to be

$$\mathsf{H}_\infty(X) = -\log\big(\max_x \mathbb{P}[X = x]\big).$$

We define the min entropy of X conditioned on E to be

$$\mathsf{H}_\infty(X|E) = -\log\big(\max_x \mathbb{P}[X = x|E]\big).$$

Finally, we recall the definition of collision entropy.

Definition 3.3.8 (Collision entropy). Let X and Y be random variables with finite support, let E be an event. We define the collision entropy of X to be

$$\mathsf{H}_2(X) = -\log\Big(\sum_x \mathbb{P}[X = x]^2\Big) = -\log\big(\mathbb{P}[X = X']\big),$$

where X' is independent and identically distributed to X. We define the collision entropy of X conditioned on E to be

$$\mathsf{H}_2(X|E) = -\log\Big(\sum_x \mathbb{P}[X = x|E]^2\Big).$$

The average collision entropy of X given Y is instead

$$\widetilde{\mathsf{H}}_2(X|Y) = -\log\Big(\sum_{x,y} \Pr[Y = y] \cdot \Pr[X = x|Y = y]^2\Big).$$

All the above definitions of entropy are not equivalent. For instance, Shannon's entropy can assume values that are significantly larger than min and collision entropy. The definitions are however related by the following well-known inequalities.

Theorem 3.3.9. Let X be a random variable with finite support, let E be an event. We have that

$$0 \leq \mathsf{H}_\infty(X) \leq \mathsf{H}_2(X) \leq \mathsf{H}(X) \leq \mathsf{H}_0(X),$$
$$0 \leq \mathsf{H}_\infty(X|E) \leq \mathsf{H}_2(X|E) \leq \mathsf{H}(X|E) \leq \mathsf{H}_0(X|E) \leq \mathsf{H}_0(X).$$

Yao's Incompressibility Entropy.

All the entropy notions we recalled above are great for measuring information theoretic properties, however, they all suffer from an important disadvantage, namely, they do not behave well under computational indistinguishability. Specifically, if $X \sim_c X'$, the entropy of X can be significantly different from the entropy of X', it does not matter which of the above definitions we consider.

We solve this issue by relying on a notion of computational entropy [Yao82, HLR07]. We recall the definition.

Definition 3.3.10 (Yao's entropy). Let $(X_\lambda)_{\lambda\in\mathbb{N}}$ be an ensemble of random variables. We say that the Yao entropy of X is smaller or equal to $k(\lambda)$, written $\mathsf{H}_{\mathsf{Yao}}(X) \leq k(\lambda)$, if there exists a pair of polynomial sized deterministic circuits $(c_\lambda, d_\lambda)_{\lambda\in\mathbb{N}}$ such that

$$\mathbb{P}[d_\lambda(c_\lambda(X)) = X] \geq \frac{2^{\ell(\lambda)}}{2^{k(\lambda)}} - \mathsf{negl}(\lambda).$$

In the above formula, $\ell(\lambda)$ denotes the output size of c_λ. The circuit c_λ is called a compressor, whereas d_λ is called a decompressor.

In [HLR07], Hsiao, Lu and Reyzin generalised the definition to the conditional case. We recall it below.

Definition 3.3.11 (Conditional Yao's entropy). Let $(X_\lambda)_{\lambda\in\mathbb{N}}$ and $(Y_\lambda)_{\lambda\in\mathbb{N}}$ be two ensembles of random variables. We say that the Yao entropy of X conditioned on Y is smaller or equal to $k(\lambda)$, written $\mathsf{H}_{\mathsf{Yao}}(X|Y) \leq k(\lambda)$, if there exists a pair of polynomial sized deterministic circuits $(c_\lambda, d_\lambda)_{\lambda\in\mathbb{N}}$ such that

$$\mathbb{P}[d(c(X,Y),Y) = X] \geq \frac{2^{\ell(\lambda)}}{2^{k(\lambda)}} - \mathsf{negl}(\lambda).$$

In the above formula, $\ell(\lambda)$ denotes the output size of c_λ. The circuit c_λ is called a compressor, whereas d_λ is called a decompressor. If $\mathsf{H}_{\mathsf{Yao}}(X|Y) \leq k(\lambda)$ where $k(\lambda)$ is $O(\log\lambda)$, we will simply write that $\mathsf{H}_{\mathsf{Yao}}(X|Y) = O(\log\lambda)$.

Essentially, Yao's incompressibility entropy measures how much it is possible to compress, in polynomial time, samples from a distribution X given that the outcome of the possibly correlated random variable Y is known.

We observe that Yao's entropy can assume values that are significantly larger than Shannon's entropy. Examples of this kind are the outputs of PRGs. In some particular cases, however, also the opposite is true. For instance, there exist distributions X such that $\mathsf{H}_\infty(X) = O(\log\lambda)$ but $\mathsf{H}(X) = \omega(\log\lambda)$. For all such X, we have $\mathsf{H}_{\mathsf{Yao}}(X) = O(\log\lambda)$ (consider the compressor that outputs the empty string and the decompressor that outputs the most likely element).

The following well-known lemma formalises the fact that Yao's entropy preserves under computational indistinguishability.

Lemma 3.3.12. Let $(X_\lambda, Y_\lambda)_{\lambda\in\mathbb{N}}$ and $(X'_\lambda, Y'_\lambda)_{\lambda\in\mathbb{N}}$ be two ensembles of random variables such that $(X_\lambda, Y_\lambda) \sim_c (X'_\lambda, Y'_\lambda)$. Then, $\mathsf{H}_{\mathsf{Yao}}(X|Y) \leq k(\lambda)$ if and only if $\mathsf{H}_{\mathsf{Yao}}(X'|Y') \leq k(\lambda)$.

We highlight that Yao's entropy is not the only notion of computational entropy [Rey11, HILL99, HLR07]. Among all the studied notions, it is however the one assuming highest values [HLR07]. We decided to use Yao's entropy exactly for this reason, making the results presented in this paper as strong as possible.

3.4 Impossibility of Distributed Samplers without Random Oracle

We now present and prove our main theorem, namely that in a strong semi-malicious distributed sampler $\mathsf{H}(R|\sigma) = O(\log\lambda)$. The idea was sketched in the technical overview (see Section 3.2.1).

Theorem 3.4.1. Let $\mathcal{D}(1^\lambda)$ be an efficient distribution such that $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log\lambda)$. In a strongly semi-maliciously secure distributed sampler for $\mathcal{D}(1^\lambda)$ in the UC model, we have that $\mathsf{H}(R \,|\, \mathsf{crs}) = O(\log\lambda)$.

Proof. Consider the distributed sampler execution in which the adversary controls a subset C of parties, but behaves exactly as in the protocol. In particular, the adversary waits to receive the messages of the honest parties and then it generates random (and independent) messages for the corrupted parties.

Let Sample be the algorithm used by the parties to reconstruct their output. In order to be as general as possible, compared to Definition 3.3.1, we change the syntax of the procedure by providing it also with the CRS crs, the index i of the party running it and the randomness used by P_i to generate its DS message U_i. Let $\mathsf{Gen}(1^\lambda, \mathsf{crs}, j)$ be the algorithm used by party P_j to generate its message. We assume that such algorithm requires $L_j(\lambda)$ bits of randomness. Suppose that the generation of the CRS requires $L(\lambda)$ bits of randomness.

Consider the security game of the protocol, we start by focusing on the ideal world. Since the simulator runs in polynomial time, there exists a polynomial upper bound $q(\lambda)$ on the number of samples the simulator queries to the functionality before providing (crs, U_H) to the adversary. Let Q be the set containing the responses to these queries.

Claim 3.4.1. *In the ideal world, with overwhelming probability, $R \in Q$.*

Proof of the claim. After the adversary provides U_C, the output of the protocol R is determined and known to the adversary. Notice that, if everybody is honest in the real world, all the parties obtain the same R with overwhelming probability, so R is well defined. If that was not the case, the adversary can easily distinguish the protocol from the simulation (in the ideal world, the honest parties always output the same value, in the real one they would not). After receiving U_C, the simulator needs to communicate to the functionality that it must output R to the honest parties.

Now, observe that

$$\mathbb{P}[\mathcal{D}(1^\lambda) = R] = \sum_x \mathbb{P}[R = x] \cdot \mathbb{P}[\mathcal{D}(1^\lambda) = x] \leq$$
$$\leq \max_x \; \mathbb{P}[\mathcal{D}(1^\lambda) = x] = 2^{-\mathsf{H}_\infty(\mathcal{D})} = 2^{-\omega(\log\lambda)}.$$

So, $\mathbb{P}[\mathcal{D}(1^\lambda) = R]$ is negligible.

As a consequence, the simulator can make the honest parties output R only if $R \in Q$. Indeed, once R is fixed, the probability that any subsequent query to the functionality collides with R is negligible. We conclude that $R \in Q$ with overwhelming probability, otherwise it would be possible to distinguish between real world and ideal world. ■

The next claim is used to prove that, in the real world, $\mathsf{H}(R \,|\, U_H, \mathsf{crs}) = O(\log\lambda)$. We introduce some notation.

Let $p(\lambda)$ be a polynomial and let ι be the index of a fixed corrupted party. Consider the algorithm $\mathcal{D}'_{U_H,\mathsf{crs}}(1^\lambda)$ defined as follows:

1. $\forall j \in C: \quad r_j \xleftarrow{\$} \{0,1\}^{L_j(\lambda)}$
2. $\forall j \in C: \quad U'_j \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \mathsf{crs}, j;\, r_j)$
3. Output $R' \leftarrow \mathsf{Sample}(\mathsf{crs}, U_H, U'_C, \iota, r_\iota)$

Let S be the random variable denoting the set of samples produced by running $\mathcal{D}'_{U_H,\mathsf{crs}}(1^\lambda)$ $p(\lambda)$ times.

Let E_0 be the random variable having value 1 if R is well defined (i.e. every party outputs the same value), 0 otherwise. Similarly, let E_1 be random variable having value 1 if $R \in S$, 0 otherwise. Finally, we define $M(\lambda) := L(\lambda) + \sum_{i\in[n]} L_i(\lambda)$.

Claim 3.4.2. *Suppose that, in the real world, $\mathsf{H}(R\,|\,U_H, \mathsf{crs})$ is not $O(\log\lambda)$. Then, for every $\lambda_0 \in \mathbb{N}$, there exists $\lambda \geq \lambda_0$ such that $\mathbb{P}[R \notin S | E_0 = 1] \geq 1/M(\lambda)$.*

Proof of the claim. Observe that, for every x, y, z, w, we have

$$\mathbb{P}[R = x\,|U_H = y, \mathsf{crs} = z, S = w] = \mathbb{P}[R = x\,|U_H = y, \mathsf{crs} = z].$$

Indeed, given $U_H = y$ and $\mathsf{crs} = z$, the value of R is determined only by the randomness used to generate U_C. Such randomness is independent of S. So, $\mathsf{H}(R\,|\,U_H, \mathsf{crs}, S) = \mathsf{H}(R\,|\,U_H, \mathsf{crs})$. We have that

$$\begin{aligned}\mathsf{H}(R\,|\,U_H, \mathsf{crs}) &= \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S) \leq \mathsf{H}(R, E_0\,|\,U_H, \mathsf{crs}, S) = \\ &= \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_0) + \mathsf{H}(E_0\,|\,U_H, \mathsf{crs}, S) \leq \\ &\leq \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_0) + \mathsf{H}_0(E_0) = \\ &= \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_0) + 1. \end{aligned} \tag{3.1}$$

Now, we have that $\mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_0)$ is equal to

$$\mathbb{P}[E_0 = 0] \cdot \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_0 = 0) + \mathbb{P}[E_0 = 1] \cdot \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_0 = 1). \tag{3.2}$$

We know that $\mathbb{P}[E_0 = 0]$ is negligible, moreover,

$$\mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_0 = 0) \leq \mathsf{H}_0(R) \leq \log\Big(2^{L(\lambda)} \cdot \prod_{i\in[n]} 2^{L_i(\lambda)}\Big) = M(\lambda).$$

We have proven that $\mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_0 = 0) = \mathsf{poly}(\lambda)$, so, putting it together with (3.1) and (3.2), we obtain

$$\mathsf{H}(R\,|\,U_H, \mathsf{crs}) \leq \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_0 = 1) + 1 + \mathsf{negl}(\lambda). \tag{3.3}$$

Now, we observe that

$$\begin{aligned}\mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_0 = 1) &\leq \mathsf{H}(R, E_1\,|\,U_H, \mathsf{crs}, S, E_0 = 1) = \\ &= \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_1, E_0 = 1) + \mathsf{H}(E_1\,|\,U_H, \mathsf{crs}, S, E_0 = 1) \leq \\ &\leq \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_1, E_0 = 1) + \mathsf{H}_0(E_1) = \\ &= \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_1, E_0 = 1) + 1. \end{aligned} \tag{3.4}$$

Furthermore, $\mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_1, E_0 = 1)$ is equal to

$$\mathbb{P}[E_1 = 0|E_0 = 1] \cdot \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_1 = 0, E_0 = 1) + \\ \mathbb{P}[E_1 = 1|E_0 = 1] \cdot \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_1 = 1, E_0 = 1). \tag{3.5}$$

We observe that

$$\begin{aligned}\mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, &E_1 = 1, E_0 = 1) = \\ &= \sum_w \mathbb{P}[S = w] \cdot \mathsf{H}(R\,|\,U_H, \mathsf{crs}, S = w, E_1 = 1, E_0 = 1) \leq \\ &\leq \sum_w \mathbb{P}[S = w] \cdot \mathsf{H}_0(R\,|\,S = w, E_1 = 1, E_0 = 1) \leq \\ &\leq \sum_w \mathbb{P}[S = w] \cdot \log\big(p(\lambda)\big) = \log\big(p(\lambda)\big). \end{aligned} \tag{3.6}$$

We also notice that

$$\mathsf{H}(R\,|\,U_H, \mathsf{crs}, S, E_1 = 0, E_0 = 1) \leq \mathsf{H}_0(R) \leq L(\lambda) + \sum_{i\in[n]} L_i(\lambda) = M(\lambda). \tag{3.7}$$

Now, suppose that there exists $\lambda_0 \in \mathbb{N}$ such that, for all $\lambda \geq \lambda_0$, $\mathbb{P}[E_1 = 0|E_0 = 1] \leq 1/M(\lambda)$. We would have that

$$\begin{aligned}
\mathsf{H}(R \,|\, U_H, \mathsf{crs}) &\leq \mathsf{H}(R \,|\, U_H, \mathsf{crs}, S, E_0 = 1) + 1 + \mathsf{negl}(\lambda) \leq && \text{by (3.3)}\\
&\leq \mathsf{H}(R \,|\, U_H, \mathsf{crs}, S, E_1, E_0 = 1) + 2 + \mathsf{negl}(\lambda) \leq && \text{by (3.4)}\\
&\leq \frac{1}{M(\lambda)} \cdot M(\lambda) + \log\big(p(\lambda)\big) + 2 + \mathsf{negl}(\lambda) = && \text{by (3.5),(3.6),(3.7)}\\
&= \log\big(p(\lambda)\big) + 3 + \mathsf{negl}(\lambda).
\end{aligned}$$

So, $\mathsf{H}(R \,|\, U_H, \mathsf{crs})$ would be $O(\log \lambda)$ contradicting our initial assumption. We conclude that for every $\lambda_0 \in \mathbb{N}$, there exists $\lambda \geq \lambda_0$ such that $\mathbb{P}[E_1 = 0|E_0 = 1] \geq 1/M(\lambda)$. ■

Claim 3.4.3. *In the real world,* $\mathsf{H}(R \,|\, U_H, \mathsf{crs}) = O(\log \lambda)$.

Proof of the claim. By contradiction suppose that, in the real world, $\mathsf{H}(R \,|\, U_H, \mathsf{crs})$ is not $O(\log \lambda)$.

Now, consider the adversary $\mathcal{A}$ that, given σ, U_H, samples $(q(\lambda)+1) \cdot \lambda \cdot M(\lambda)$ independent elements from $\mathcal{D}'_{U_H,\mathsf{crs}}(1^\lambda)$ and outputs 1 if and only if it obtains strictly more than $q(\lambda)$ distinct values in this way. We show that such adversary can distinguish between real world and ideal world with non-negligible advantage.

First of all, we notice that $\mathcal{A}$ runs in polynomial time. Let $R'_{j,\iota}$ be the output of the j-th execution of $\mathcal{D}'_{U_H,\mathsf{crs}}(1^\lambda)$. Observe that the distribution of $R'_{j,\iota}$ conditioned on U_H, crs is the same as the distribution of R_ι, the output of the ι-th party, conditioned on U_H, crs. By Claim 3.4.1,

$$\mathbb{P}[R'_{j,\iota} \notin Q] = \mathbb{P}[R_\iota \notin Q] \leq \mathbb{P}[E_0 = 0] + \mathbb{P}[R \notin Q] = \mathsf{negl}(\lambda).$$

Hence, by the union bound and due to the fact that $|Q| \leq q(\lambda)$, in the ideal world, $\mathcal{A}$ outputs 0 with overwhelming probability.

Now, let S_j be the random variable containing the values of the first $j-1$ samples from $\mathcal{D}'_{U_H,\mathsf{crs}}(1^\lambda)$. We know that, in the real world,

$$\begin{aligned}
\mathbb{P}[R'_{j,\iota} \in S_j] = \mathbb{P}[R_\iota \in S_j] &= \\
&= \mathbb{P}[E_0 = 0] \cdot \mathbb{P}[R_\iota \in S_j | E_0 = 0] + \mathbb{P}[E_0 = 1] \cdot \mathbb{P}[R_\iota \in S_j | E_0 = 1] \leq \\
&\leq \mathsf{negl}(\lambda) + \mathbb{P}[R \in S_j | E_0 = 1].
\end{aligned}$$

We conclude that, for every $\lambda_0 \in \mathbb{N}$, there exists a $\lambda \geq \lambda_0$ such that

$$\mathbb{P}[R'_{j,\iota} \in S_j] \leq 1 - \frac{1}{2M(\lambda)}$$

Now, we observe that, for every $\lambda_0 \in \mathbb{N}$, there exists a $\lambda \geq \lambda_0$ such that

$$\mathbb{P}\left[|S_{j\cdot\lambda\cdot M}| = |S_{(j+1)\cdot\lambda\cdot M}|\right] \leq \left(1 - \frac{1}{2M(\lambda)}\right)^{\lambda\cdot M(\lambda)}$$

and so, by the union bound, for the same values of λ,

$$\begin{aligned}
\mathbb{P}[|S_{(q(\lambda)+1)\cdot\lambda\cdot M+1}| \leq q(\lambda)] \leq& \mathbb{P}\big[\exists j \text{ s.t. } |S_{j\cdot\lambda\cdot M}| = |S_{(j+1)\cdot\lambda\cdot M}|\big] \leq \\
\leq& (q(\lambda)+1) \cdot \left(1 - \frac{1}{2M(\lambda)}\right)^{\lambda\cdot M(\lambda)}
\end{aligned}$$

Observe that

$$\lim_{\lambda\to\infty} (q(\lambda)+1) \cdot \left(1 - \frac{1}{2M(\lambda)}\right)^{\lambda\cdot M(\lambda)} = 0,$$

so, $\mathbb{P}[|S_{(q(\lambda)+1)\cdot\lambda\cdot M+1}| \leq q(\lambda)]$ is definitively smaller than 1/3. Therefore, for every $\lambda_0 \in \mathbb{N}$, there exists a $\lambda \geq \lambda_0$ such that

$$\begin{aligned}\mathrm{Adv}_{\mathcal{A}}(\lambda) = \Big|\mathbb{P}[\mathcal{A}(1^\lambda) = 0|\text{ideal}] - \mathbb{P}[\mathcal{A}(1^\lambda) = 0|\text{real}]\Big| \geq \\ \geq 1 - \mathsf{negl}(\lambda) - 1/3 \geq 1/2 - 1/3.\end{aligned}$$

We conclude that $\mathcal{A}$ distinguishes between the real world and the ideal world with non-negligible advantage. This contradicts the security of the distributed sampler, therefore, in the real world, $\mathsf{H}(R\,|\,U_H, \mathsf{crs}) = O(\log\lambda)$. ■

Claim 3.4.4. *In the real world, we have that* $\mathsf{H}(R\,|\,U_C, \mathsf{crs}) = O(\log\lambda)$.

Proof of the claim. The messages of the corrupted parties are distributed as in the fully honest case. So, switching the role of honest and corrupted parties does not affect the distribution of $(\sigma, (U_i)_{i\in[n]}, R)$. By applying the result of Claim 3.4.3 on the new set of corrupted parties, we obtain that $\mathsf{H}(R\,|\,U_C, \mathsf{crs}) = O(\log\lambda)$. ■

Claim 3.4.5. *In the real world,* $\mathsf{H}(R\,|\,\mathsf{crs}) = O(\log\lambda)$.

Proof of the claim. By the strong chain rule of Shannon's entropy, we have that

$$\begin{aligned}&\mathsf{H}(R\,|\,U_H,\mathsf{crs}) + \mathsf{H}(R\,|\,U_C,\mathsf{crs}) - \mathsf{H}(R\,|\,U_H,U_C,\mathsf{crs}) + \\ &\;+ \mathsf{H}(U_H\,|\,\mathsf{crs}) - \mathsf{H}(U_H\,|\,U_C,\mathsf{crs}) + \\ &\;+ \mathsf{H}(U_C\,|\,R,U_H,\mathsf{crs}) - \mathsf{H}(U_C\,|\,R,\mathsf{crs}) = \\ =&\underline{\mathsf{H}(R,U_H\,|\,\mathsf{crs})} - \underline{\mathsf{H}(U_H\,|\,\mathsf{crs})} + \underline{\mathsf{H}(R,U_C\,|\,\mathsf{crs})} - \underline{\mathsf{H}(U_C\,|\,\mathsf{crs})} + \\ &\;- \underline{\mathsf{H}(R,U_H,U_C\,|\,\mathsf{crs})} + \underline{\mathsf{H}(U_H,U_C\,|\,\mathsf{crs})} + \\ &\;+ \underline{\mathsf{H}(U_H\,|\,\mathsf{crs})} - \underline{\mathsf{H}(U_H,U_C\,|\,\mathsf{crs})} + \underline{\mathsf{H}(U_C\,|\,\mathsf{crs})} + \\ &\;+ \underline{\mathsf{H}(U_C,R,U_H\,|\,\mathsf{crs})} - \underline{\mathsf{H}(R,U_H\,|\,\mathsf{crs})} - \underline{\mathsf{H}(U_C,R\,|\,\mathsf{crs})} + \mathsf{H}(R\,|\,\mathsf{crs}) = \\ =&\mathsf{H}(R\,|\,\mathsf{crs}).\end{aligned}$$

We observe that $\mathsf{H}(U_C\,|\,R,U_H,\mathsf{crs}) \leq \mathsf{H}(U_C\,|\,R,\mathsf{crs})$ and $\mathsf{H}(R\,|\,U_H,U_C,\mathsf{crs}) \geq 0$. Moreover, since U_H and U_C are independent given crs, $\mathsf{H}(U_H\,|\,\mathsf{crs}) = \mathsf{H}(U_H\,|\,U_C,\mathsf{crs})$. We conclude that, by Claim 3.4.3 and 3.4.4, $\mathsf{H}(R\,|\,\mathsf{crs}) \leq \mathsf{H}(R\,|\,U_H,\mathsf{crs}) + \mathsf{H}(R\,|\,U_C,\mathsf{crs}) = O(\log\lambda)$. ■

□

3.4.1 Distributed Sampler CRSs Cannot be Used Twice

We now discuss the first consequence of Theorem 3.4.1. Suppose that our distribution $\mathcal{D}(1^\lambda)$ has high min-entropy, i.e. $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log\lambda)$. We observe that the probability that two independent samples from $\mathcal{D}(1^\lambda)$ collide is negligible. Indeed, denoting the two independent outputs by R and R', we have

$$\mathbb{P}[R = R'] = 2^{-\mathsf{H}_2(\mathcal{D})} \leq 2^{-\mathsf{H}_\infty(\mathcal{D})} = 2^{-\omega(\log\lambda)}.$$

We show, however, that if we run a strongly semi-maliciously secure distributed sampler for $\mathcal{D}(1^\lambda)$ twice using the same CRS, the outputs collide with non-negligible probability. As a consequence, we cannot hope to reuse the same CRS to generate independent looking samples.

Corollary 3.4.2. Let $\mathcal{D}(1^\lambda)$ be an efficiently samplable distribution such that $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log\lambda)$. Consider a strongly semi-maliciously secure distributed sampler protocol for $\mathcal{D}(1^\lambda)$ in the UC model and let R and R' denote the outputs of two protocol executions having the same CRS. Then, $\mathbb{P}[R = R'] \geq 1/\mathsf{poly}(\lambda)$.

Proof. By Theorem 3.4.1, we know that $\mathsf{H}(R|\mathsf{crs}) = O(\log\lambda)$. Now, we have that

$$\mathbb{P}[R = R'] = \sum_z \mathbb{P}[\mathsf{crs} = z] \cdot \mathbb{P}[R = R'|\mathsf{crs} = z] = \sum_z \mathbb{P}[\mathsf{crs} = z] \cdot 2^{-\mathsf{H}_2(R|\mathsf{crs}=z)} \geq$$
$$\geq \sum_z \mathbb{P}[\mathsf{crs} = z] \cdot 2^{-\mathsf{H}(R|\mathsf{crs}=z)}$$

We observe that $f(x) := 2^{-x}$ is a convex function, so, by Jensen's inequality

$$\mathbb{P}[R = R'] \geq \sum_z \mathbb{P}[\mathsf{crs} = z] \cdot 2^{-\mathsf{H}(R|\mathsf{crs}=z)} \geq 2^{-\sum_z \mathbb{P}[\mathsf{crs}=z]\cdot\mathsf{H}(R|\mathsf{crs}=z)} = 2^{-\mathsf{H}(R|\mathsf{crs})}.$$

Observe that $2^{-\mathsf{H}(R|\mathsf{crs})} = 1/\mathsf{poly}(\lambda)$. □

3.4.2 Distributed Sampler CRSs Cannot be Short

A second consequence of Theorem 3.4.1 is that the CRSs of strongly semi-maliciously secure distributed samplers cannot be short. Specifically, the bit-length of the CRS $|\mathsf{crs}|$ must be at most $O(\log\lambda)$ bits shorter than $\mathsf{H}_{\mathsf{Yao}}(\mathcal{D})$.

The Yao entropy $\mathsf{H}_{\mathsf{Yao}}(R|\mathsf{crs})$ must be small. Although Yao's entropy and Shannon's entropy can assume very different values, we prove that if $\mathsf{H}(R|\mathsf{crs}) = O(\log\lambda)$, also $\mathsf{H}_{\mathsf{Yao}}(R|\mathsf{crs}) = O(\log\lambda)$. Indeed, by Corollary 3.4.2, we know that two distributed sampler executions using the same CRS have colliding outputs with non-negligible probability. We can therefore consider the compressor that on input (R, crs) outputs the empty string and the decompressor that on input crs, runs the distributed sampler using crs as CRS, outputting the result R'. Since there is a $1/\mathsf{poly}(\lambda)$ probability that $R = R'$, we conclude that $\mathsf{H}_{\mathsf{Yao}}(R|\mathsf{crs}) = O(\log\lambda)$. Observe that we can make the decompressor deterministic using a PRF.

A chain rule for Yao's entropy. To conclude our argument, we show that

$$\mathsf{H}_{\mathsf{Yao}}(R|\mathsf{crs}) \geq \mathsf{H}_{\mathsf{Yao}}(R) - |\mathsf{crs}|. \tag{3.8}$$

By the security of distributed samplers in the fully honest case, R and $\mathcal{D}(1^\lambda)$ are computationally indistinguishable, so, $\mathsf{H}_{\mathsf{Yao}}(R) = \mathsf{H}_{\mathsf{Yao}}(\mathcal{D})$. From this, we easily deduce that $\mathsf{H}_{\mathsf{Yao}}(\mathcal{D}) - |\mathsf{crs}| \leq O(\log\lambda)$.

We highlight that in [KPW13, Appendix B], Krenn *et al.* proved the chain rule for Yao's entropy, which seems to immediately imply (3.8). Unfortunately, this is not the case. The idea at the base of their proof is that given a compressor-decompressor pair (c, d) for $\mathsf{H}_{\mathsf{Yao}}(R|\mathsf{crs})$, we can build a new compressor-decompressor pair for $\mathsf{H}_{\mathsf{Yao}}(R)$ with the same success probability as (c, d). On input R, the new compressor performs a brute-force search for a crs' such that $d(c(R, \mathsf{crs}'), \mathsf{crs}') = R$, then it outputs $c(R, \mathsf{crs}'), \mathsf{crs}'$. The decompressor instead is identical to d. Since, the output size of the new compressor is $|\mathsf{crs}|$ bits larger than c's, we obtain that $\mathsf{H}_{\mathsf{Yao}}(R|\mathsf{crs}) \geq \mathsf{H}_{\mathsf{Yao}}(R) - |\mathsf{crs}|$. Observe however that if $|\mathsf{crs}|$ is more than $O(\log\lambda)$, the new compressor does not run in polynomial time. That prevents us from using their result.

We notice that in our setting, the new compressor does not need to perform a brute-force search. Indeed, in order to obtain a crs', it can just feed R to the distributed sampler simulator for the fully honest case and pick the CRS contained in the simulated view. The latter is indistinguishable from the the real CRS used for the generation of R. This allows us to prove the chain rule even if $|\mathsf{crs}|$ is more than $O(\log\lambda)$.

Corollary 3.4.3. Suppose that $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log\lambda)$. If OWFs exist, the CRS crs of a strongly semi-maliciously secure distributed sampler for $\mathcal{D}(1^\lambda)$ in the UC model must satisfy $\mathsf{H}_{\mathsf{Yao}}(\mathcal{D}) - |\mathsf{crs}| \leq O(\log\lambda)$.

Proof. We start by proving the following claim.

Claim 3.4.6. *In the distributed sampler protocol,* $\mathsf{H}_{\mathsf{Yao}}(R|\mathsf{crs}) = O(\log\lambda)$.

Proof of the claim. Consider the following randomised pair of compressor and decompressor:

- On input a pair (R, crs), the compressor c outputs the empty string.
- On input crs, the decompressor d runs the distributed sampler protocol in-its-head using crs as CRS. Then, it outputs the result R'.

Observe that $\mathbb{P}[d(c(R, \mathsf{crs}), \mathsf{crs}) = R] = \mathbb{P}[R = R']$. Since R and R' are the outputs of two distributed sampler executions using the same CRS, by Corollary 3.4.2, we know that

$$\mathbb{P}[d(c(R, \mathsf{crs}), \mathsf{crs}) = R] = \frac{1}{\mathsf{poly}(\lambda)}$$

Now, consider the deterministic decompressor d' that performs exactly the same operations as d, but uses a PRF F to generate its randomness, i.e. d' has a random PRF key K hard-coded in its circuit and it generates its randomness by computing $F(K, \mathsf{crs})$. By the security of the PRF

$$\mathbb{P}[d'(c(R, \mathsf{crs}), \mathsf{crs}) = R] \geq \mathbb{P}[d(c(R, \mathsf{crs}), \mathsf{crs}) = R] - \mathsf{negl}(\lambda) = \frac{1}{\mathsf{poly}(\lambda)}.$$

Notice that the probability in the first term is also over K. So,

$$\begin{aligned}\frac{1}{\mathsf{poly}(\lambda)} \leq & \mathbb{P}[d'(c(R, \mathsf{crs}), \mathsf{crs}) = R] = \sum_x \frac{1}{|K|} \cdot \mathbb{P}[d'(c(R, \mathsf{crs}), \mathsf{crs}) = R | K = x] \leq \\ & \leq \max_x \mathbb{P}[d'(c(R, \mathsf{crs}), \mathsf{crs}) = R | K = x].\end{aligned}$$

Let $\hat{x}$ be the value of x associated with the maximum, let $d'_{\hat{x}}$ be the decompressor having $K = \hat{x}$. We have proven that the pair $(c, d'_{\hat{x}})$ is successful in compressing and decompressing with probability greater than $1/\mathsf{poly}(\lambda)$. We conclude that $\mathsf{H}_{\mathsf{Yao}}(R|\mathsf{crs}) = O(\log \lambda)$. ■

Claim 3.4.7. *In the distributed sampler protocol,* $\mathsf{H}_{\mathsf{Yao}}(R|\mathsf{crs}) \geq \mathsf{H}_{\mathsf{Yao}}(R) - |\mathsf{crs}|$.

Proof of the claim. Suppose that $\mathsf{H}_{\mathsf{Yao}}(R|\mathsf{crs}) \leq k(\lambda)$ for some function $k(\lambda)$. Then, there exists a compressor-decompressor pair (c, d) such that

$$\mathbb{P}[d(c(R, \mathsf{crs}'), \mathsf{crs}') = R] \geq \frac{2^\ell}{2^k} - \mathsf{negl}(\lambda).$$

We now design a new compressor $\hat{c}$ for R: $\hat{c}$ feeds its input R to the distributed sampler simulator Sim for the fully-honest case. The latter provides a CRS crs' and the messages of all the parties. The compressor outputs $c(R, \mathsf{crs}')$ as well as crs'.

Let crs be the CRS used to generate R. By the security of the distributed sampler in the fully honest case, we know that the triple $(\mathsf{crs}, (U_i)_{i \in [n]}, R)$ in the protocol is computationally indistinguishable from $(\mathsf{Sim}(1^\lambda, R'), R')$ where $R' \xleftarrow{\$} \mathcal{D}(1^\lambda)$. We conclude that the pairs (R, crs) and (R, crs') are also computationally indistinguishable. As a consequence,

$$\Big|\mathbb{P}[d(c(R, \mathsf{crs}), \mathsf{crs}) = R] - \mathbb{P}[d(c(R, \mathsf{crs}'), \mathsf{crs}') = R]\Big| = \mathsf{negl}(\lambda).$$

We conclude that

$$\begin{aligned}\mathbb{P}[d(\hat{c}(R)) = R] = \mathbb{P}[d(c(R, \mathsf{crs}'), \mathsf{crs}') = R] \geq \mathbb{P}[d(c(R, \mathsf{crs}'), \mathsf{crs}') = R] - \mathsf{negl}(\lambda) \geq \\ \geq \frac{2^\ell}{2^k} - \mathsf{negl}(\lambda) = \frac{2^{\ell + |\mathsf{crs}|}}{2^{k + |\mathsf{crs}|}} - \mathsf{negl}(\lambda).\end{aligned}$$

Observe that $\ell + |\mathsf{crs}|$ is the output size of $\hat{c}$. Notice also that $\hat{c}$ is not deterministic, however, we can make it so adopting the same technique used in Claim 3.4.6, i.e. by generating its randomness using a PRF. Specifically, consider the deterministic compressor $\hat{c}'$ which has a PRF key K hard-coded in its circuit and generates its randomness by computing $F(K, R)$, performing then the same operations as $\hat{c}$. By the security of the PRF, we have that

$$\Big|\mathbb{P}[d(\hat{c}'(R)) = R] - \mathbb{P}[d(\hat{c}(R)) = R]\Big| = \mathsf{negl}(\lambda).$$

So,

$$\mathbb{P}[d(\hat{c}'(R)) = R] \geq \frac{2^{\ell+|\mathsf{crs}|}}{2^{k+|\mathsf{crs}|}} - \mathsf{negl}(\lambda).$$

As in the proof of the previous claim, the probability in the first term is also over K. So,

$$\begin{aligned}\frac{2^{\ell+|\mathsf{crs}|}}{2^{k+|\mathsf{crs}|}} - \mathsf{negl}(\lambda) \leq &\mathbb{P}[d(\hat{c}'(R)) = R] = \sum_x \frac{1}{|K|} \cdot \mathbb{P}[d(\hat{c}'(R)) = R | K = x] \leq \\ &\leq \max_x \mathbb{P}[d(\hat{c}'(R)) = R | K = x].\end{aligned}$$

Let $\hat{x}$ be the value of x associated with the maximum, let $\hat{c}'_{\hat{x}}$ be the decompressor having $K = \hat{x}$. We have proven that the pair $(\hat{c}'_{\hat{x}}, d)$ succeeds in compressing and decompressing R with probability greater than $2^{\ell+|\mathsf{crs}|}/2^{k+|\mathsf{crs}|} - \mathsf{negl}(\lambda)$, so $\mathsf{H}_{\mathsf{Yao}}(R) \leq k(\lambda) + |\mathsf{crs}|$. We conclude that $\mathsf{H}_{\mathsf{Yao}}(R|\mathsf{crs}) \geq \mathsf{H}_{\mathsf{Yao}}(R) - |\mathsf{crs}|$. ■

By Claims 3.4.6 and 3.4.7, we have $O(\log \lambda) = \mathsf{H}_{\mathsf{Yao}}(R|\mathsf{crs}) \geq \mathsf{H}_{\mathsf{Yao}}(R) - |\mathsf{crs}|$. We notice that, by the security of the distributed sampler in the fully honest case, R is computationally indistinguishable from $\mathcal{D}(1^\lambda)$, so, $\mathsf{H}_{\mathsf{Yao}}(R) = \mathsf{H}_{\mathsf{Yao}}(\mathcal{D})$. We conclude that $\mathsf{H}_{\mathsf{Yao}}(\mathcal{D}) - |\mathsf{crs}| \leq O(\log \lambda)$. □

3.4.3 Distributed Sampler CRSs Cannot be (too) Nice

The *niceness* of a CRS cannot be defined in a mathematical way. However, informally speaking, we can say that a CRS is nicer than another if it is easier to produce in an MPC setting, e.g. a uniformly random string of bits (i.e. a URS) is simpler to generate than a random RSA modulus of unknown factorisation. Indeed, if we aim for security with abort, we can generate a URS using a simple commit-then-reveal approach. On the other hand, all the state-of-the-art constructions for the generation of RSA moduli rely on rejection sampling: first the parties generate secret-shared (or encrypted) candidate primes p and q, they multiply them and apply expensive (bi)primality tests on the secret-shared data [FLOP18, HMR+19, CCD+20]. After sufficiently many trials (the number depends on the size of the modulus), the players obtain a valid RSA modulus with high probability. This results in rather complex protocols.

In the previous sections, we have seen that the CRS of distributed samplers cannot be used more then once and cannot be short. These facts suggest that directly encoding a sample from $\mathcal{D}(1^\lambda)$ in a CRS is probably better than relying on a distributed sampler protocol for $\mathcal{D}(1^\lambda)$. At least in the first case, the parties spare one round of interaction. But what if the CRS used by the distributed sampler is nicer than any encoding of $\mathcal{D}(1^\lambda)$? We prove that this cannot happen as it is always possible to non-interactively generate samples from $\mathcal{D}(1^\lambda)$ using only distributed sampler CRSs and public random coins.

Non-interactive generation of samples from $\mathcal{D}(1^\lambda)$ using a distributed sampler CRS and random bits. In this section, we observe that a distributed sampler for $\mathcal{D}(1^\lambda)$ allows us to construct an efficient deterministic function De that maps pairs consisting of a distributed sampler CRS crs and uniformly random bits r into values R that are computationally indistinguishable from $\mathcal{D}(1^\lambda)$. This algorithm trivially outputs the result of the distributed sampler using crs as CRS and r as randomness for the parties. The interesting fact is that if the distributed sampler is strongly semi-maliciously secure in the UC model, the algorithm is efficiently invertible with non-negligible probability. Specifically, we prove that there exists an efficient PPT algorithm test, that outputs 1 with $1/\mathsf{poly}(\lambda)$ probability when run over a sample $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$. Moreover, if this event occurs, it is possible to efficiently find a pair (crs, r) that looks random over $\mathsf{De}^{-1}(R)$.

In other words, if we provide the parties with a distributed sampler CRS crs and public uniformly random bits r, the parties can obtain a sample R from $\mathcal{D}(1^\lambda)$ without any interaction. Furthermore, with $1/\mathsf{poly}(\lambda)$ probability, (crs, r) reveals no information in addition to what can be already inferred from R. Finally, the honest parties can tell when (crs, r) reveals too much information, having therefore the opportunity to reject R and restart.

THE GAME $\mathcal{G}^{\mathsf{Inv}}_{\mathcal{A}}(1^\lambda)$

The challenger performs the following operations

1. $b \xleftarrow{\$} \{0,1\}$
2. $\mathsf{crs} \xleftarrow{\$} \mathsf{CRS}(1^\lambda)$, $r \xleftarrow{\$} \mathcal{U}(1^\lambda)$
3. $R \leftarrow \mathsf{De}(\mathsf{crs}, r)$
4. If $\mathsf{test}(R) = 0$, go back to step 2.
5. If $b = 0$, provide the adversary with (crs, r), otherwise provide it with $\mathsf{En}(R)$.

The adversary wins if it terminates its execution outputting b.

Figure 3.5: Invertibility game

Biases in the distribution. Notice that the selective rejection biases the distribution of the output. However, any cryptographic protocol basing its security on R will remain secure even if R is sampled according to the biased distribution. Indeed, at least a polynomial fraction of the samples is not rejected. If the protocol was insecure in the new setting, there would be a non-negligible probability of sampling a bad R even in the original protocol, which would therefore be insecure.

Why is De invertible with non-negligible probability? Our idea is based on the fact that in a strongly semi-malicious distributed sampler, $\mathsf{H}(R|\mathsf{crs})$ is small. In other words, the CRS of the protocol describes the output with high precision. Obtaining the exact CRS used for the generation of R is usually hard, however, the simulator for the fully-honest case can provide us with a functionally equivalent object.

In order to completely describe R, we need to extract also randomness r for the parties, so that, using r in conjunction with the CRS, we obtain R. Unfortunately, the simulator cannot provide much help here. Indeed, given $(\mathsf{crs}', (U_i)_{i\in[n]}) \xleftarrow{\$} \mathsf{Sim}(1^\lambda, R)$, it is usually hard to extract the randomness r used to generate $(U_i)_{i\in[n]}$. Actually, such r might not even exist. Luckily, in Corollary 3.4.2, we have proven that two executions of a strongly semi-maliciously secure distributed sampler using the same CRS have colliding outputs with $1/\mathsf{poly}(\lambda)$ probability. So, if we run the distributed sampler protocol again using crs' as CRS, we obtain R again with non-negligible probability. Clearly, in the new execution, the value of $(U_i)_{i\in[n]}$ has probably changed, however, this time we know the randomness r' used to generate the messages.

On average invertibility. We observe that the probability of succeeding in inverting De is also over the outcome of R. In other words, we just know that the *average* probability of inverting R is $1/\mathsf{poly}(\lambda)$. That does not mean that the overwhelming majority of values R is efficiently invertible: if R assumes an unlucky value, we can try to invert as many times as we want without any hope of succeeding. We could prove, however, that there always exists a polynomial fraction of the space of events for which R is easy to invert.

We also noticed that it is possible to efficiently test if R is easy to invert or not. Indeed, we can just try to invert it many times, if the success frequency is lower than a certain threshold, we can reject R, otherwise, we accept it. We used the Chernoff bound to find the threshold and the maximum number of inversion attempts. In particular, we needed test to succeed with at least $1/\mathsf{poly}(\lambda)$ probability.

The special case of URSs. As a corollary of the result described in this section, if the distributed sampler uses a URS, it is possible to securely generate samples from $\mathcal{D}(1^\lambda)$ using public random coins only. In particular, in the random oracle model, the parties can securely sample from $\mathcal{D}(1^\lambda)$ without interacting and without needing any CRS.

Corollary 3.4.4. Suppose that $\mathsf{H}_\infty(\mathcal{D}) = \omega(\log \lambda)$ and there exists a strongly semi-maliciously secure distributed sampler for $\mathcal{D}(1^\lambda)$ in the UC model. Let $\mathsf{CRS}(1^\lambda)$ be the algorithm used to generate the CRS of the protocol. Then, there exist a deterministic polynomial algorithm De and PPT algorithms En and test such that

- $\mathsf{De}\big(\mathsf{CRS}(1^\lambda), \mathcal{U}(1^\lambda)\big) \sim_c \mathcal{D}(1^\lambda)$
- $\mathbb{P}[\mathsf{test}\big(\mathcal{D}(1^\lambda)\big) = 1] \geq 1/\mathsf{poly}(\lambda)$
- No PPT adversary can win the game $\mathcal{G}^{\mathsf{Inv}}_{\mathcal{A}}(1^\lambda)$ (see Figure 3.5) with non-negligible advantage.

Proof. We start by defining the algorithm De, which on input crs and random string r, runs the distributed sampler protocol using crs as CRS and r as randomness for the parties, outputting the result.

Claim 3.4.8. $\mathsf{De}\big(\mathsf{CRS}(1^\lambda), \mathcal{U}(1^\lambda)\big) \sim_c \mathcal{D}(1^\lambda)$

Proof of the claim. We observe that $R := \mathsf{De}\big(\mathsf{CRS}(1^\lambda), \mathcal{U}(1^\lambda)\big)$ is distributed exactly as the distributed sampler output. By the security of the distributed sampler, we conclude that $R \sim_c \mathcal{D}(1^\lambda)$. ■

We now define the PPT algorithm Inv as follows: Inv feeds the input R to the distributed sampler simulator for the fully honest case. In this way, it obtains a fake CRS crs' and messages $(U_i)_{i\in[n]}$. Finally, Inv picks a uniformly random string r and outputs (crs', r).

Claim 3.4.9. *Let R denote a sample from $\mathcal{D}(1^\lambda)$. Then, there exists a polynomial $q(\lambda)$ such that*

$$\mathbb{P}[\mathsf{De}(\mathsf{Inv}(R)) = R] \geq \frac{1}{q(\lambda)}$$

Proof of the claim. Let $(\mathsf{crs}', r) := \mathsf{Inv}(R)$. By the security of distributed samplers, we know that (crs', R) is computationally indistinguishable from $(\mathsf{crs}, \mathsf{De}(\mathsf{crs}, r))$ where $\mathsf{crs} \xleftarrow{\$} \mathsf{CRS}(1^\lambda)$ and r is uniformly random. We conclude that for r and r' independent and uniformly random

$$\big(\mathsf{crs}, \mathsf{De}(\mathsf{crs}, r'), \mathsf{De}(\mathsf{crs}, r)\big) \sim_c \big(\mathsf{crs}', R, \mathsf{De}(\mathsf{crs}', r)\big) \tag{3.9}$$

By Corollary 3.4.2, we know that

$$\mathbb{P}[\mathsf{De}(\mathsf{crs}, r') = \mathsf{De}(\mathsf{crs}, r)\big] \geq \frac{1}{\mathsf{poly}(\lambda)}$$

We conclude that, by (3.9),

$$\mathbb{P}[\mathsf{De}(\mathsf{Inv}(R)) = R] = \mathbb{P}[\mathsf{De}(\mathsf{crs}', r') = R] \geq \frac{1}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda) \geq \frac{1}{\mathsf{poly}(\lambda)}$$

■

We now define the algorithm test as follows: on input R, test checks if $\mathsf{De}(\mathsf{Inv}(R)) = R$ for $4\lambda \cdot q(\lambda)$ times. If the equation is satisfied less than λ times, test outputs 0, otherwise it output 1.

In a similar way, we define En: on input R, En computes $(\mathsf{crs}', r) \xleftarrow{\$} \mathsf{Inv}(R)$ and checks if $\mathsf{De}(\mathsf{crs}', r) = R$. If that is the case, it outputs (crs', r). Otherwise, it repeats the operation. If the procedure fails for more than $8\lambda \cdot q(\lambda)$, En outputs $\perp$.

Claim 3.4.10.

$$\mathbb{P}[\mathsf{test}\big(\mathcal{D}(1^\lambda)\big) = 1] \geq \frac{1}{8q(\lambda)}$$

Proof of the claim. We define $p_x := \mathbb{P}[\mathsf{De}(\mathsf{Inv}(x)) = x]$. Let R be a sample from $\mathcal{D}(1^\lambda)$. By Claim 3.4.9, we know that

$$\mathbb{E}[p_R] = \sum_x \mathbb{P}[R = x] \cdot p_x = \mathbb{P}[\mathsf{De}(\mathsf{Inv}(R)) = R] \geq \frac{1}{q(\lambda)}$$

Since $0 \leq p_x \leq 1$, we have that $\mathbb{E}[p_R^2] \leq \mathbb{E}[p_R]$, so, by the Paley-Zygmund inequality,

$$\mathbb{P}\Big[p_R \geq \frac{1}{2q(\lambda)}\Big] \geq \mathbb{P}\Big[p_R > \frac{1}{2}\mathbb{E}[p_R]\Big] \geq \frac{1}{4} \cdot \frac{\mathbb{E}[p_R]^2}{\mathbb{E}[p_R^2]} \geq \frac{1}{4} \cdot \mathbb{E}[p_R] \geq \frac{1}{4q(\lambda)}$$

Define now $\Omega_{\mathsf{good}} = \{x | p_x \geq \frac{1}{2q(\lambda)}\}$. Suppose now that $x \in \Omega_{\mathsf{good}}$. If we run the check $\mathsf{De}(\mathsf{Inv}(x)) = x$ for $8\lambda \cdot q(\lambda)$ times, by the Chernoff bound, we know that it succeeds more than

$$\frac{1}{2} \cdot 4\lambda q(\lambda) \cdot p_x \geq \lambda$$

times with overwhelming probability. So if $x \in \Omega_{\mathsf{good}}$, $\mathsf{test}(x) = 1$ with overwhelming probability. We conclude that

$$\begin{aligned} \mathbb{P}[\mathsf{test}(R) = 1] &\geq \mathbb{P}[\mathsf{test}(R) = 1 | R \in \Omega_{\mathsf{good}}] \cdot \mathbb{P}[R \in \Omega_{\mathsf{good}}] \geq \\ &\geq \min_{x \in \Omega_{\mathsf{good}}} \mathbb{P}[\mathsf{test}(x) = 1] \cdot \mathbb{P}[R \in \Omega_{\mathsf{good}}] \geq \frac{1}{8q(\lambda)} \end{aligned}$$

■

Claim 3.4.11. *No PPT adversary can win the game $\mathcal{G}^{\mathsf{Inv}}_{\mathcal{A}}(1^\lambda)$ (see Figure 3.5) with non-negligible advantage.*

Proof of the claim. As in the previous claim, let $p_x := \mathbb{P}[\mathsf{De}(\mathsf{Inv}(x)) = x]$. Define now $\Omega_{\mathsf{bad}} = \{x | p_x \leq \frac{1}{8q(\lambda)}\}$. Suppose that $x \in \Omega_{\mathsf{bad}}$. If we run the check $\mathsf{De}(\mathsf{Inv}(x)) = x$ for $4\lambda \cdot q(\lambda)$ times, by the Chernoff bound, we know that it succeeds less than

$$2 \cdot 4\lambda q(\lambda) \cdot p_x \leq \lambda$$

times with overwhelming probability. So, if $x \in \Omega_{\mathsf{bad}}$, $\mathsf{test}(x) = 0$ with overwhelming probability.

Let $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$. We observe that

$$\mathbb{P}[\mathsf{En}(R) = \bot | \mathsf{test}(R) = 1] \leq \mathbb{P}[R \in \Omega_{\mathsf{bad}} | \mathsf{test}(R) = 1] + \mathbb{P}[\mathsf{En}(R) = \bot | R \notin \Omega_{\mathsf{bad}}].$$

We know that

$$\begin{aligned} \mathbb{P}[R \in \Omega_{\mathsf{bad}} | \mathsf{test}(R) = 1] &\leq \frac{\mathbb{P}[\mathsf{test}(R) = 1 | R \in \Omega_{\mathsf{bad}}]}{\mathbb{P}[\mathsf{test}(R) = 1]} \leq \\ &\leq 8q(\lambda) \cdot \max_{x \in \Omega_{\mathsf{bad}}} \mathbb{P}[\mathsf{test}(x) = 1] = \mathsf{negl}(\lambda). \end{aligned}$$

Furthermore,

$$\mathbb{P}[\mathsf{En}(R) = \bot | R \notin \Omega_{\mathsf{bad}}] \geq \min_{x \notin \Omega_{\mathsf{bad}}} \mathbb{P}[\mathsf{En}(x) = \bot].$$

Notice that for every $x \notin \Omega_{\mathsf{bad}}$, $p_x > \frac{1}{8q(\lambda)}$. Observe also that En tries to invert the input up to $8\lambda \cdot q(\lambda) > \lambda / p_x$ times, so, with overwhelming probability $\mathsf{En}(x) \neq \bot$. We conclude that $\mathbb{P}[\mathsf{En}(R) = \bot | \mathsf{test}(R) = 1] = \mathsf{negl}(\lambda)$.

Now, suppose that $\mathsf{En}(R) = (\mathsf{crs}', r') \neq \bot$. We recall that crs' is obtained by running $\mathsf{Sim}(1^\lambda, R)$, so by the security of distributed samplers $(R, \mathsf{crs}') \sim_c (\mathsf{De}(\mathsf{crs}, r), \mathsf{crs})$ where $\mathsf{crs} \xleftarrow{\$} \mathsf{CRS}(1^\lambda)$ and $r \xleftarrow{\$} \mathcal{U}(1^\lambda)$. We also know that r' is random conditioned on satisfying $\mathsf{De}(\mathsf{crs}', r') = R$. In other words, (R, crs', r') is computationally indistinguishable from $(\mathsf{De}(\mathsf{crs}, r), \mathsf{crs}, r)$. We conclude our proof observing that $(R, \mathsf{En}(R)) \sim_c (\mathsf{De}(\mathsf{crs}, r), \mathsf{En}(\mathsf{De}(\mathsf{crs}, r)))$. ■

□

Bibliography

[ABI+23] Damiano Abram, Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Varun Narayanan. Cryptography from planted graphs: Security with logarithmic-size messages. In Guy Rothblum and Hoeteck Wee, editors, *Theory of Cryptography*, pages 286–315, Cham, 2023. Springer Nature Switzerland.

[AOS23] Damiano Abram, Maciej Obremski, and Peter Scholl. On the (Im)possibility of Distributed Samplers: Lower Bounds and Party-Dynamic Constructions. Cryptology ePrint Archive, Report 2023/863, 2023. https://eprint.iacr.org/2023/863.

[ASY22] Damiano Abram, Peter Scholl, and Sophia Yakoubov. Distributed (correlation) samplers: How to remove a trusted dealer in one round. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 790–820. Springer, Heidelberg, May / June 2022.

[AWZ23] Damiano Abram, Brent Waters, and Mark Zhandry. Security-Preserving Distributed Samplers: How to Generate Any CRS in One Round Without Random Oracles. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 489–514, Cham, 2023. Springer Nature Switzerland.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020.

[Bd94] Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 274–285. Springer, Heidelberg, May 1994.

[BFK+19] Saikrishna Badrinarayanan, Rex Fernando, Venkata Koppula, Amit Sahai, and Brent Waters. Output compression, MPC, and iO for turing machines. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 342–370. Springer, Heidelberg, December 2019.

[BL20] Fabrice Benhamouda and Huijia Lin. Mr NISC: Multiparty reusable non-interactive secure computation. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 349–378. Springer, Heidelberg, November 2020.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CCD+20] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and abhi shelat. Multiparty generation of an RSA modulus. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 64–93. Springer, Heidelberg, August 2020.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 68–86. Springer, Heidelberg, May 2003.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, March 2015.

[DGH+20] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Daniel Masny, and Daniel Wichs. Two-round oblivious transfer from CDH or LPN. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 768–797. Springer, Heidelberg, May 2020.

[DGM23] Nico Döttling, Phillip Gajland, and Giulio Malavolta. Laconic function evaluation for turing machines. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *Public-Key Cryptography – PKC 2023*, pages 606–634, Cham, 2023. Springer Nature Switzerland.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[FLOP18] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 331–361. Springer, Heidelberg, August 2018.

[GS18] Sanjam Garg and Akshayaram Srinivasan. A simple construction of iO for turing machines. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 425–454. Springer, Heidelberg, November 2018.

[HIJ+17] Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 181–211. Springer, Heidelberg, December 2017.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HJK+16] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 715–744. Springer, Heidelberg, December 2016.

[HLR07] Chun-Yuan Hsiao, Chi-Jen Lu, and Leonid Reyzin. Conditional computational entropy, or toward separating pseudoentropy from compressibility. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 169–186. Springer, Heidelberg, May 2007.

[HMR+19] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient RSA key generation and threshold paillier in the two-party setting. *Journal of Cryptology*, 32(2):265–323, April 2019.

[HV16] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. What security can we achieve within 4 rounds? In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 486–505. Springer, Heidelberg, August / September 2016.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015.

[IKM+13] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 600–620. Springer, Heidelberg, March 2013.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, Heidelberg, August 2014.

[KPW13] Stephan Krenn, Krzysztof Pietrzak, and Akshay Wadia. A counterexample to the chain rule for conditional HILL entropy - and what deniable encryption has to do with it. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 23–39. Springer, Heidelberg, March 2013.

[KRS15] Dakshita Khurana, Vanishree Rao, and Amit Sahai. Multi-party key exchange for unbounded parties from indistinguishability obfuscation. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 52–75. Springer, Heidelberg, November / December 2015.

[LZ17] Qipeng Liu and Mark Zhandry. Decomposable obfuscation: A framework for building applications of obfuscation from polynomial hardness. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 138–169. Springer, Heidelberg, November 2017.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Heidelberg, October 2021.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[Rey11] Leonid Reyzin. Some notions of entropy for cryptography - (invited talk). In Serge Fehr, editor, *ICITS 11*, volume 6673 of *LNCS*, pages 138–142. Springer, Heidelberg, May 2011.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27:623–656, 1948.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd FOCS*, pages 80–91. IEEE Computer Society Press, November 1982.

Chapter 4

Security-Preserving Distributed Samplers: How to Generate any CRS in One Round without Random Oracles

Damiano Abram, Brent Waters, Mark Zhandry

Abstract. A distributed sampler is a way for several mutually distrusting parties to non-interactively generate a common reference string (CRS) that all parties trust. Previous work constructs distributed samplers in the random oracle model, or in the standard model with very limited security guarantees. This is no accident, as standard model distributed samplers with full security were shown impossible. In this work, we provide new definitions for distributed samplers which we show achieve meaningful security guarantees in the standard model. In particular, our notion implies that the hardness of a wide range of security games is preserved when the CRS is replaced with a distributed sampler. We also show how to realize our notion of distributed samplers. A core technical tool enabling our construction is a new notion of single-message zero knowledge.

4.1 Introduction

Many protocols require a common reference string to be generated by a third party in order to securely run the protocol. Importantly, the security of the protocol requires that the any secrets revealed during setup are hidden from the parties of the protocol. For example, if the protocol relies on a public RSA modulus for a reference string, the parties of the protocol must not know the prime factors. Such a structured common reference string requires placing enormous trust in the third party, and naturally leads to the question:

What happens if the trusted third party is actually not trustworthy?

Digging deeper, there may be many potential third parties who are willing to run the setup: maybe certain state organizations (e.g. NIST) as well and independent organizations (e.g. EFF). Some participants in the protocol may trust some third parties, while some participants only trust other third parties, and there may be no overlap between the trusted parties. How can we ensure that all protocol participants trust the reference string?

An obvious solution is for all potential third parties to run an MPC protocol to generate the reference string. Then, as long as each participant trusts a single third party, they will trust the reference string (CRS).

However, engaging in an MPC protocol can be a logistical burden for these third parties. For comparison, in a situation where the CRS is generated by a single trusted third party, that party can simply post the reference string they produce to some public domain. In contrast, if many third parties are engaging in an MPC protocol to compute the reference string, this requires the many third parties to send several messages back-and-forth between each other.

Another issue is the difficulty of updating the CRS if we want to expand the number of involved trusted parties. For example, suppose third parties A, B, C engaged in an MPC protocol to generate a CRS such as an RSA modulus N. At some later date, users u, v wish to engage in a protocol using an RSA modulus, but user u only trusts a new third party D and not A, B, C. Meanwhile v does not trust D since it is new. Unfortunately, this would require A, B, C to come back online and interact with D to create a new modulus N'. A, B, C may be unable or unwilling to do so, as it would be an unreasonable burden to re-run the MPC any time a trusted setup was requested with a new third party.

Solution: Distributed Samplers. Abram, Scholl, and Yakoubov [ASY22] proposed the notion of a *distributed sampler*. Here, parties A, B, C each individually run their own setup algorithm locally, arriving at messages U_A, U_B, U_C, which they post to some public domain. Now when a set of users want a CRS generated by A, B, C, they look up U_A, U_B, U_C, and run a procedure which deterministically extracts a CRS from U_A, U_B, U_C. Because the process of computing the CRS from U_A, U_B, U_C is deterministic, all parties can compute it from U_A, U_B, U_C for themselves, and therefore do not require any additional interaction. Thus, the tuple U_A, U_B, U_C now acts as the common reference string, which is simply the concatenation of the individual messages of the various third parties. Informally, as long as a user trusts at least one of the third parties, then they trust the CRS derived from the list of strings that includes that party.

When a set of users wishes to incorporate a new third party D, all they need is for D to generate and post its own U_D. Now the parties can derive a new CRS from U_A, U_B, U_C, U_D. Importantly the original parties A, B, C do not need to do anything to add a new third party. In the follow-up work of [AOS23], a construction is given that maintains security in such a scenario.

Limitations of Existing Work. The work of [ASY22] constructs two kinds of distributed samplers both utilizing indistinguishability obfuscation. The first achieves semi-honest security, where the third parties *honestly* generate their messages but wish to then break a protocol using the generated CRS. Unfortunately, this notion of security is rather limited, since a truly malicious adversary could try to generate their messages dishonestly in order to influence the generated CRS. Such influence over the CRS offers much greater flexibility in breaking the protocol. For example, if the CRS is for a statistically sound proof system, a malicious adversary may try to influence the CRS into a "bad" one where false proofs exist.

The second distributed sampler achieves full malicious security in the UC model. However, the construction requires the random oracle model, and worse requires the full power of programming the random oracle.

Thus, the existing work either requires the full power of the random oracle model, or achieves only a very limited notion of security. This is no accident: as shown by [AOS23], full standard model malicious security is in fact *impossible*. So the question becomes: what kind of malicious security can be meaningfully achieved in the standard model?

4.1.1 Our Work

In this work, we address the above limitations of prior work, by giving new definitions for distributed samplers that avoid the above impossibility while still guaranteeing meaningful security against malicious adversaries, and providing a new instantiation of distributed samplers satisfying this definition. As a crucial step toward this goal, we also investigate single message zero knowledge proofs in the standard model, and provide new constructions with novel features. A summary of our main results follows.

Defining Distributed Samplers. Our first contribution is to define new security notions for distributed samplers. We describe a notion of *security preserving* distributed samplers, which implies that, for any game-based protocol using a reference string, security is preserved by the distributed sampler. That is, if the protocol is secure under a reference string generated by a single trusted third party, then it is also secure when the reference string is generated via a distributed sampler, as long as *at least one* of the parties involved is trusted. We also give some technical definitions of security for distributed samplers that are easier to reason about, and we show that these notions imply adequate notions of security preservation. See Sections 4.5 and 4.6 for details.

Constructing Distributed Samplers. Next, we show how to construct distributed samplers meeting our new definition. We obtain two flavours of the primitive: a CRS-less distributed sampler with security against uniform adversaries and a construction achieving security against non-uniform adversaries by relying on a short, reusable and unstructured CRS.

Our construction uses [ASY22] as a starting basis. However, we need to make several key changes. Critically, we face the following challenge: in order to justify that the reference string is "as good as" an honestly generated one, the reduction needs to be able to embed an actual honestly generated reference string N into the honest third party's message, and somehow force the adversary to generate their own messages in a way that makes the derived reference string equal to N. But in the case of malicious adversaries, whatever strategy the reduction uses, the adversary can seemingly use as well to force the derived reference string to be their own, maliciously generated, N'.

Extractable 1-message zero knowledge. Resolving the above problem requires many tools. One of the main ones is a new 1-message zero knowledge proof, which crucially does not need a CRS. Now, such an object is normally considered impossible, but it can be possible if the simulator is allowed to be non-uniform while the adversary is required to be uniform. Such 1-message zero knowledge leveraging non-uniformity was considered before [BP04]. However, our use of zero knowledge requires several features, such as the ability for the reduction to extract the original proof from the sender's message, that were not present in existing 1-message zero knowledge. We therefore develop a new 1-message zero knowledge proof system with several useful features that we crucially leverage to achieve our notion of distributed samplers.

Updatability. The distributed samplers presented in this work assume that the set of participants is a-priori given. As a consequence, our constructions tolerate inactive parties (their distributed sampler messages can be generated using default randomness), but when new participants join, the protocol needs to restart.

Applications. A direct implication of our results is the existence of a 3-round OT protocol in the plain model (no CRS) with security against active, *uniform* adversaries and non-uniform simulation. This is achieved by directly applying our CRS-less distributed sampler to [PVW08]. More in general, our distributed samplers imply 3-round active MPC in the plain model (no CRS) with security against uniform adversaries and non-uniform simulation [BL18a].

Our distributed samplers can also be used to compile extractable NIZKs into 2-round zero-knowledge proofs of knowledge[1]. The resulting constructions either rely on a short, unstructured CRS or no CRS at all, depending on whether we aim for security against non-uniform adversaries or not. Furthermore, the 2-round protocols satisfy automatically concurrent security, independently of the properties of the original NIZKs.

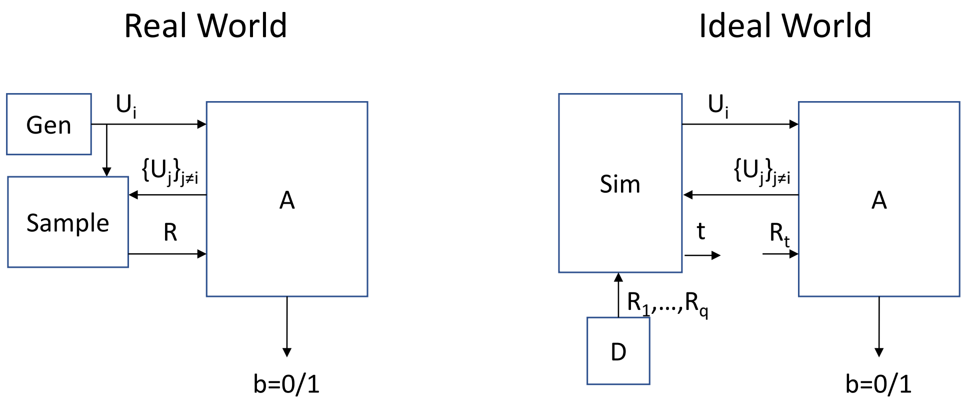
4.2 Technical Overview

4.2.1 New notions of distributed sampler

Full malicious security, and its impossibility. We first recall an informal description of the notion of malicious security obtained by [ASY22], which follows the real/ideal paradigm as shown in Figure 4.1 (We use $\mathcal{D}$ to denote the distribution of honestly generated CRSs. Such distribution can be private-coin). In the

[1]Our techniques do not apply to non-extractable NIZKs. This is due to the challenger of the soundness game being not efficient.

real world, the adversary is given the messages of the honest third parties, and then subsequently generates the messages of the malicious third parties. The challenger then derives the CRS from the combined messages of third parties, and gives it to the adversary. In the ideal world, the honest third party message is instead generated by a simulator (which depends on the adversary), and the simulator is given as input a CRS generated honestly from $\mathcal{D}$. The adversary is then given the simulated message and the honestly generated CRS. Security dictates that the two worlds are indistinguishable, which in particular implies that the derived CRS is equal to the provided honest CRS in the ideal world.



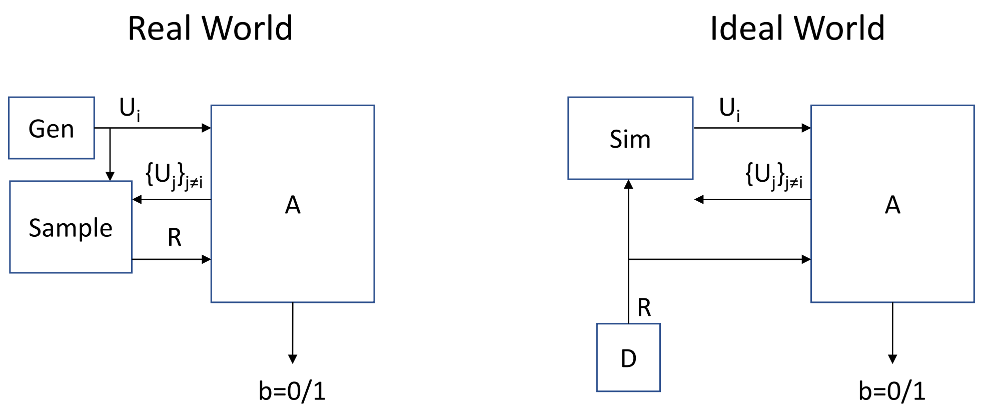
$$|\ \Pr[b = 1|\text{Real World}] - \Pr[b = 1|\text{Ideal World}]\ | < \mathsf{negl}$$

Figure 4.1: An informal explanation of malicious security for distributed samplers. Here, Gen is the algorithm for honestly generating the third party messages U_j and Sample is the algorithm that combines the messages into the derived CRS R. i is the honest user, t is the simulator's choice of which of the honest CRS samples $R_1, \ldots, R_q$ to use.

This brief description is obviously impossible, however. Indeed, a malicious adversary could be *rushing*: *after* seeing the honest party's message, it could generate several sets of malicious third party messages (but even generate them honestly), compute the derived CRSs, and then select the set of third party messages that give a CRS most advantageous to the adversary. This means it is impossible for the simulator to guarantee that any single provided honest CRS is used by the adversary. To capture this ability of rushing adversaries, the definition actually gives the simulator a polynomial number of honestly generated potential CRSs, and the simulator can then choose which one gets sent to the adversary.

The above described notion of security is *still* impossible, as shown by [AOS23]. One basic reason is the following: the simulator has to produce a message U_i, whose length is fixed by the protocol. However, the sequence of honest CRSs provided to the simulator can be arbitrary long, since an arbitrary polynomial-time adversary can generate arbitrarily many sets of third party messages, thereby allowing them to select from an arbitrary polynomial number of CRSs. This means there is no way for a single U_i to embed all of the CRSs. [AOS23] formalize an impossibility, and it seems rather robust, since although their results apply only to the UC model with dishonest majority, different security settings such as standalone security, superpolynomial simulation, honest majority, or having the protocol depend itself on a CRS do not seem to solve the problem. The positive results of [ASY22, AOS23] therefore employ a random oracle. This avoids the impossibility, since the simulator can now program the random oracle with the various CRSs, instead of programming them into U_i. However, it requires the full power of programming the random oracle, and it is unclear what kind of security this gives in the standard model.

Our first notion: hardness-preserving distributed samplers. We now describe our new notions of security for distributed samplers. The first we describe is that of *hardness preserving*, which is given informally in Figure 4.2. There are two main differences from the security notion described. First, only a single honest CRS is given to the simulator in the ideal world. This is necessary in the standard model, as there is no way to program an unbounded number of CRSs into a fixed length simulated message. Note that with this change we can no longer hope to force the derived CRS to be equal to the provided honest CRS, except possibly with inverse polynomial probability. This means an adversary can distinguish real from ideal in the majority of cases. So the second change is to relax indistinguishability to the following. We only require that if the adversary outputs 1 in the real world with non-negligible probability ϵ_1, then it also outputs 1 in the ideal world with non-negligible probability ϵ_2. But ϵ_1 and ϵ_2 do not need to be close, and ϵ_2 can be far lower than ϵ_1.



$$\Pr[b = 1|\text{Ideal World}] < \mathsf{negl} \Rightarrow \Pr[b = 1|\text{Real World}] < \mathsf{negl}$$

Figure 4.2: An informal explanation of hardness-preserving security for distributed samplers. It is the same as Figure 4.1, except that there is only a single honest CRS in the ideal world, and the relation between success probabilities in the two worlds is relaxed.

The obvious question is then: what kind of guarantees does such a relaxed definition provide? We show that hardness preserving distributed samplers are good for guaranteeing security for various *search* tasks. These are tasks where the adversary's goal is to output some value with non-negligible probability (as opposed to distinguishing tasks, where the goal is to output a value with probability non-negligibly larger than 1/2).

More precisely, we consider a general search game between a challenger and adversary, where at some step the challenger is provided with an honestly generated CRS, which it uses in its own internal logic but also sends to the adversary. We can compile such a game into one where the CRS is generated via distributed samplers, and the adversary controls all but one of the trusted third parties. A diagram of such a game and its compilation is given in Figure 4.3. We show the following:

Theorem 4.2.1 (informal)*.* If a distributed sampler is hardness-preserving and the search game is hard, then the compiled search game is also hard.

Notice that there exists a non-negligible security loss between the original search game and the compiled version. Furthermore, the loss depends on the running time of the adversary. This is unavoidable: a rushing adversary can regenerate the corrupted party distributed sampler messages in its head many times, looking for an output that gives a higher chance of solving the search problem. The advantage will therefore degrade proportionally to the number of such trials, which is proportional to the running time.

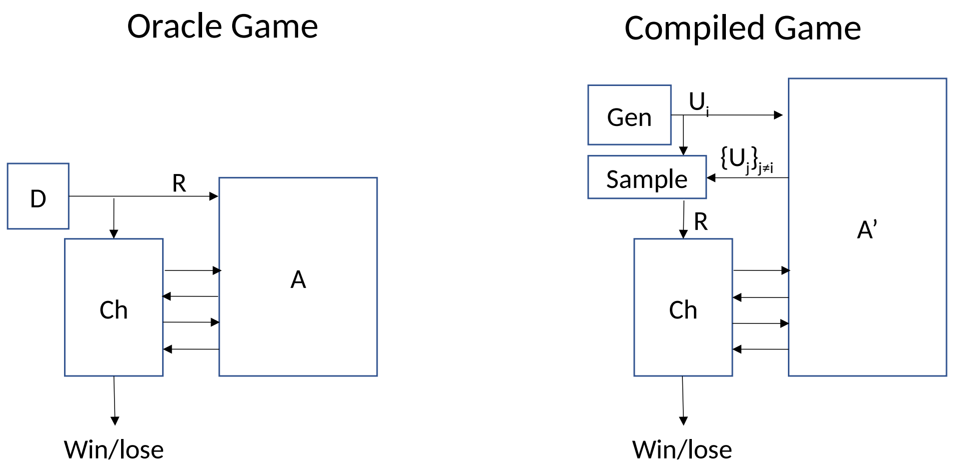


Figure 4.3: Search games and their compilations. The figure on the left is a search game utilizing an honest CRS, while the figure on the right is the compiled game using a distributed sampler to generate the CRS.

Our second notion: indistinguishability-preserving distributed samplers. Hardness-preserving distributed samplers achieve a somewhat limited form of security against active adversaries. For starters, if the game is an indistinguishability game, the notion gives no guarantees. But a more subtle issue is the following. Consider a protocol like a NIZK with CRS. The definition of zero knowledge says that there exists a simulator which simulates both the CRS *and* the proof. Perhaps it generates the CRS such that it knows a certain trapdoor, which allows it to generate a proof without knowing a witness. When using a distributed sampler, we would like the ideal world to reflect this simulated CRS and proof. But this is not a simple matter of plugging in the existing simulated CRS into the simulator for the distributed sampler, as there is no way for the distributed sampler simulator to then use the CRS trapdoor to help generate the proof. In the language of protocols and functionalities, this means that for a protocol Π with CRS which implements a functionality $\mathcal{F}$, the compiled protocol Π' using the distributed sampler to generate the CRS might no longer implement $\mathcal{F}$.

The second distributed sampler notion we introduce, called *indistinguishability-preserving*, tries to tackle this problem. The concept is informally described in Figure 4.4: an indistinguishability-preserving distributed sampler compiles any protocol Π with CRS satisfying the condition at the top of Figure 4.4 for some functionality $\mathcal{F}$ and simulator Sim_Π, into a protocol without CRS satisfying the property at the bottom.

We focus for a moment on the property at the top of Figure 4.4. The condition states that the protocol Π implements the functionality $\mathcal{F}$. However, it actually gives a strictly stronger requirement: in the ideal world, the CRS is simulated using a distribution $\mathcal{D}'$ that produces both a sample R and a trapdoor T. While the adversary receives only R, the simulator Sim_Π receives also T. In the NIZK example, $\mathcal{D}'$ would be the trapdoored CRS, and T is the trapdoor. An important point is that the simulated CRS is independent of any information known to the functionality. Not all protocols have this kind of simulation. For example, the HSS construction of [OSY21] satisfies the property: the CRS is a large RSA modulus distributed identically to the protocol and simulated before interacting with the functionality. On the other hand, imagine a protocol where the CRS consists of an RSA modulus N. Suppose that the protocol allows, e.g., generic MPC modulo N and N is chosen by the functionality (notice that the CRS is given by the functionality). If we use an indistinguishability-preserving distributed sampler to generate N, the compiled protocol will not implement the functionality anymore. This is because, in the simulation, we cannot ensure that the output of the distributed sampler is the modulus N chosen by the functionality.

Moving on to the bottom of Figure 4.4, we observe that, in the ideal world, the sampling algorithm of the distributed sampler has been substituted with a new algorithm called $\mathsf{Trapdoor}$. The latter has the purpose

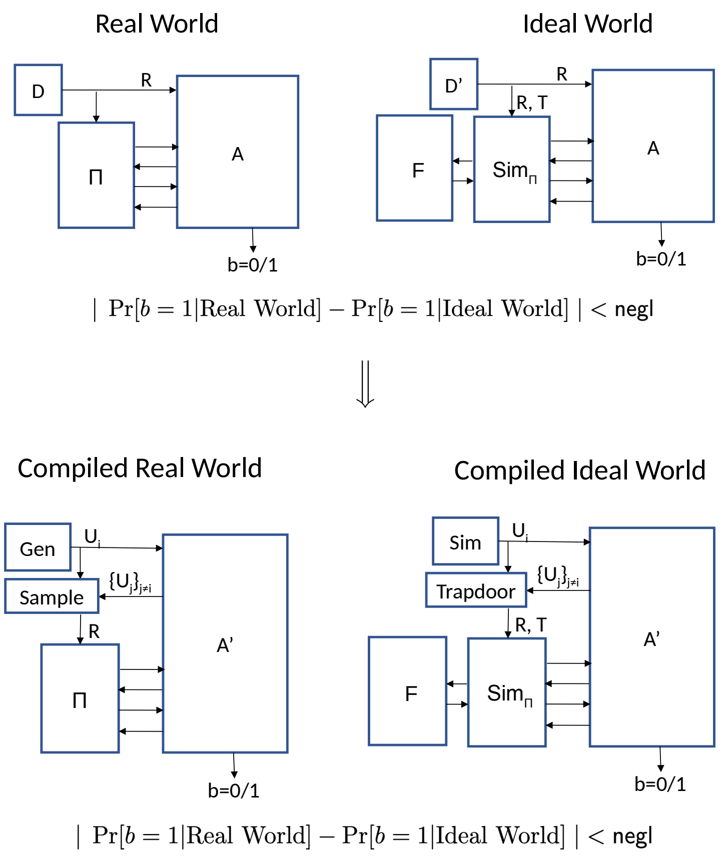


Figure 4.4: An informal explanation of indistinguishability-preserving security for distributed samplers.

of extracting the trapdoors from the outputs of the simulated distributed sampler. The resulting values are then given to Sim_Π. Observe that the property at the bottom implies that the compiled protocol implements $\mathcal{F}$.

Theorem 4.2.2 (informal). If a distributed sampler is indistinguishability-preserving and the protocol Π implements the functionality $\mathcal{F}$ as in the top of Figure 4.4, then the compiled protocol also implements $\mathcal{F}$.

The definition of indistinguishability-preserving distributed sampler is actually more general than what we outlined here: it provides security guarantees even when the sample from $\mathcal{D}$ is not given as a CRS but as an "oracle sample" revealed halfway through the execution of the protocol. It is still possible to compile this kind of protocol using a distributed sampler: instead of executing it at the beginning, the parties will run it at a later stage. Sometimes, when the first round of the protocol Π is independent of the CRS, this fact allows us to compile Π without adding rounds of interaction. For more details, check Section 4.5.2.

Lossy distributed samplers. In the paper, we introduce one last notion: *lossy distributed samplers*. This will be a convenient technical notion that will help us realize our notions of distributed samplers from above. Such a lossy sampler consists of two modes of operation: in addition to a standard mode, in which the output remains unpredictable as long as one party is honest, there exists a lossy mode. When the latter is activated, the output becomes predictable: with overwhelming probability, it will lie in a set of polynomial size determined by the messages of the honest parties. Distinguishing between standard and lossy mode will always be possible, however, for any given PPT adversary. But by choosing sufficiently large parameters for the lossy mode, we ask that the distinguishability advantage for any given adversary can be made an arbitrarily small non-negligible function, i.e. for every PPT $\mathcal{A}$ and $\delta = 1/\mathsf{poly}$, there exists a sufficiently large q[2] such that

$$\left|\Pr\left[\mathcal{A} \to 1 \middle| \mathsf{StandardMode}\right] - \Pr\left[\mathcal{A} \to 1 \middle| \mathsf{LossyMode}(q)\right]\right| \leq \delta.$$

From lossy to hardness-preserving distributed samplers. We use lossy distributed samplers to build hardness-preserving distributed samplers. Consider an adversary $\mathcal{A}$ that, in the real-world game of the hardness-preserving distributed samplers (see Figure 4.2), interacts with the standard mode of the construction. The idea is that if the adversary outputs 1 with non-negligible probability, we can activate the lossy mode with sufficiently large parameters so that $\mathcal{A}$ keeps outputting 1 with non-negligible probability. The main difference is that, now, the output of the construction is all of a sudden predictable.

At this point, we make use of a property that is satisfied by some lossy distributed samplers: *programmability*. The latter guarantees that we can hide an ideal sample $\hat{R} \xleftarrow{\$} \mathcal{D}$ among the outputs of a lossy-mode distributed samplers without the adversary's realizing. Since the output space is polynomial in size, the adversary ends up obliviously selecting $\hat{R}$ as output of the protocol with $p = 1/\mathsf{poly}$ probability. Conditioned on this event, $\mathcal{A}$ still outputs 1 with non-negligible probability ϵ. In conclusion, in the ideal world, the challenger just needs to send lossy-mode messages. The adversary will output 1 with probability at least $p \cdot \epsilon$.

Theorem 4.2.3 (Informal). Any programmable, lossy distributed sampler is hardness-preserving.

4.2.2 Building lossy distributed samplers

We explain how to build programmable, lossy distributed samplers using, among other tools, indistinguishability obfuscation [GGH+13], multi-key FHE [AJJM20], extremely lossy functions (ELFs) [Zha16] and a new primitive called *almost everywhere extractable NIZKs*. We make extensive use of subexponentially secure primitives. The resulting lossy distributed sampler makes use of a short (polynomial in λ, but independent of $\mathcal{D}$), unstructured and reusable CRS (the construction is secure even if the CRS is reused in multiple concurrent instantiations of the protocol, potentially involving different subsets of parties). Our construction originates from the semi-honest distributed sampler of [ASY22]. We briefly recall it.

The encryption program. In [ASY22], a distributed sampler message consists of two obfuscated programs. Adapting the terminology to this paper, we call them the *encryption program* and the *decryption program*.

The encryption program of party P_i takes care of generating a multi-key FHE encryption of a random string s_i under a fresh key pk_i. The output of the construction will be obtained by adding the n random strings $s_1, \dots, s_n$ and feeding the result as randomness for $\mathcal{D}$, i.e. the output sample is $R := \mathcal{D}(1^n; s_1 \oplus \cdots \oplus s_n)$. Observe that thanks to the homomorphic properties of multi-key FHE, given the encryptions of the random strings, everybody is able to derive an encryption of R[3]. The issue is that nobody is able to decrypt it: the output of the multi-key FHE evaluation is encrypted under a "joint key". In order to decrypt, the parties usually need to collaborate: each of them performs a partial decryption of the joint ciphertext and publishes the result. By pooling together the partial plaintexts, everybody can reconstruct the hidden message.

The decryption program. Usually, a multi-key FHE decryption requires interaction. In the distributed samplers of [ASY22], however, the decryption program takes care of everything without needing additional rounds of interaction.

Formally, the decryption program of party P_i takes as input the encryption programs of all the parties and evaluates them. After receiving the encryption of s_j for every $j \in [n]$, the program retrieves an encryption of the output R by applying homomorphic operations on the ciphertexts. Observe that all the decryption programs derive the same joint ciphertext C. The execution terminates by performing a partial decryption of C using the private counterpart of pk_i. The program outputs the resulting partial plaintext.

[2] q is a polynomial that upper bounds the size of the output space.
[3] The fact that the ciphertexts are encrypted under different keys does not constitute a problem.

```
$\mathsf{EProg}[K_i]$

**Hard-coded.** The PPRF key $K_i$.
**Input.** A digest $y$.

  1. $(s_i, r_i, r'_i) \leftarrow F(K_i, y)$
  2. $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Gen}(1^\lambda; r_i)$
  3. $c_i \leftarrow \mathsf{mkFHE.Enc}(\mathsf{pk}_i, s_i; r'_i)$
  4. Output $(\mathsf{pk}_i, c_i)$.
```

Figure 4.5: A sketch of the unobfuscated encryption program of party P_i

Observe that by evaluating all the decryption programs, the parties are able to retrieve all the partial decryptions of C. At that point, reconstructing R is immediate.

Counteracting the residual function attack. A common issue of all 1-round MPC protocols is that an adversary can rerun the protocol in its head many times changing a subset of the messages. The outputs of all these executions are correlated with the inputs of the honest parties. For particular functionalities, this could leak enough information to reconstruct the input of the honest parties.

In distributed samplers, there are no private inputs but we still need to be careful: we need to make sure that, in every distributed sampler execution, the encryption programs use independent looking random strings $s_1, \dots, s_n$. If that was not the case, the adversary might use the residual function attack to learn information about the randomness used in the main execution of the protocol.

In [ASY22], the authors ensure this by feeding the encryption program of each party with the hash of the encryption programs of the other players (notice that inputting the program themselves would not be possible for a matter of sizes). The encryption program generates the randomness for the multi-key FHE key pk_i and the string s_i by inputting the hash into a puncturable PRF. Observe that if any adversary reruns the distributed sampler in its head modifying any of the other messages, the hash fed in the encryption program changes. As a consequence, the program will use an independent looking s_i (and an independent looking multi-key FHE key pair).

In our lossy distributed sampler, the encryption program will remain the same as in [ASY22]. We sketch its code in Figure 4.5.

Adding extractable NIZKs. The main change we bring to the construction is to add non-interactive zero knowledge (NIZK) proofs of the well-formedness of the encryption programs. These proofs will be inputted into the decryption programs. When any of the proofs do not verify, the decryption program will output $\perp$. We sketch their code in Figure 4.6.

In order to describe the lossy mode of the distributed sampler, we assume that the NIZK is *extractable*, which means there is a special trapdoor that allows for extracting from any proof the witness used to generate the proof. We defer the discussion of the exact properties needed until later in this overview.

The lossy mode of the distributed sampler tweaks the programs of one of the honest parties as follows. The encryption program will generate simulated public keys and ciphertexts. The decryption program, instead of verifying the NIZKs, will extract the witnesses from them using the extraction property of the NIZK. From the latter, it will derive the randomness used to generate the multi-keys FHE keys and ciphertexts of the other players. At that point, similarly to [HIJ$^+$17], it simulates the partial decryption instead of directly performing it. We recall that the simulator for the partial decryption takes as input a targeted plaintext R' [AJJM20]. Such value might differ for the actual message hidden in the joint ciphertext C, however, the output of the decryption is still guaranteed to be R'.

$\mathsf{DProg}[K_i, \mathsf{EP}_i, \sigma, (\mathsf{id}_j)_{j \neq i}]$

Hard-coded. The PPRF key K_i, the encryption program EP_i, the CRS for a NIZK σ, the identities of the other parties $(\mathsf{id}_j)_{j \neq i}$.
Input. Set of $n-1$ tuples $(\mathsf{EP}_j, \pi_j)_{j \neq i}$ where EP_j is the encryption program of party P_j and π_j is a NIZK proving its well-formedness.

```
1. $\forall j \neq i: \quad b_j \leftarrow \mathsf{NIZK.Verify}(\sigma, \mathsf{id}_j, \pi_j, \mathsf{EP}_j)$
2. If $\exists j \neq i$ such that $b_j = 0$, output $\perp$
3. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big((\mathsf{EP}_l)_{l \neq j}\big)$
4. $\forall j \in [n]: \quad (\mathsf{pk}_j, c_j) \leftarrow \mathsf{EP}_j(y_j)$
5. $C \leftarrow \mathsf{mkFHE.Eval}\big(\mathcal{D}, c_1, \dots, c_n\big)$
6. $(s_i, r_i, r_i') \leftarrow F(K_i, y_i)$
7. $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Gen}(1^\lambda;\ r_i)$
8. $d_i \leftarrow \mathsf{mkFHE.PartDec}(C, \mathsf{sk}_i)$
9. Output $d_i$
```

Figure 4.6: A sketch of the unobfuscated decryption program of party P_i

Decreasing the size of the output space using an ELF. In the lossy mode, the output of the protocol is decided by the party that sends the lossy-mode programs (those that simulate the multi-key FHE operations). How can we restrict the output space to a set of polynomial size without the adversary's immediate detecting the small output space? After all, the adversary could keep generating outputs, hoping to find a collision. After only a polynomial number of outputs, the adversary would expect to find such a collision in the lossy mode.

To rectify this issue, we have the size of the lossy output space be a polynomial that grows with the adversary's run time and success probability, making sure it is a sufficiently large polynomial that the adversary cannot detect it in the time give.

At a lower level, we use extremely lossy functions (ELFs) [Zha16]. These are randomized algorithms generating deterministic functions with large domain. The primitive has two modes of operations: injective mode and lossy mode. When the first mode is activated, the function is injective. In the other case, the image of the function has size smaller than q, where q is a polynomial parameterizing the lossy mode. The two modes will be always distinguishable with non-negligible advantage. ELFs guarantee that, for any adversary $\mathcal{A}$ and inverse-polynomial δ, by choosing a sufficiently large polynomial q, it is possible to make the distinguishability advantage between the injective mode and the lossy mode smaller than δ.

In our construction, we generate the value R' input in the partial decryption simulator by applying an ELF on the concatenation of the encryption programs of the n parties. The result is then fed in a puncturable PRF. Its output is used as randomness for $\mathcal{D}(1^\lambda)$. In this way, when the ELF has a small image, the distributed sampler will have a small output space. We skecth the code of the lossy-mode programs in Figure 4.7 and Figure 4.8.

Programmability. It is easy to see that our candidate distributed sampler is programmable: in order to hide an ideal sample $\hat{R}$ in the output space, we can just pick a random value $\hat{z}$ in the image of the ELF f and input $\hat{R}$ into the partial decryption simulator whenever $f(\mathsf{EP}_1, \dots, \mathsf{EP}_n) = \hat{z}$. By the security of puncturable PRFs, the changes cannot be detected by the adversary. Furthermore, if the ELF satisfies an

```
$\mathsf{EProg}_{\mathsf{Ls}}[K_i]$

Hard-coded. The PPRF key $K_i$.
Input. A digest $y$.

  1. $(\eta_i, \eta'_i) \leftarrow F'(K_i, y)$
  2. $(\phi, \mathsf{pk}_i, c_i) \leftarrow \mathsf{mkFHE.Sim}_1(1^\lambda;\ \eta_i)$
  3. Output $(\mathsf{pk}_i, c_i)$.
```

Figure 4.7: A sketch of the unobfuscated encryption program for the lossy mode

additional property called *regularity* [Zha16], it is guaranteed that the event $f(\mathsf{EP}_1, \dots, \mathsf{EP}_n) = \hat{z}$ occurs with inverse-polynomial probability.

4.2.3 Security Proof Challenge 1: Simultaneous Extraction and Statistical Soundness

At this point, we can try to prove the security of the candidate lossy distributed sampler. However, there are some challenges that need to be overcome.

The first challenge is the following. In the lossy mode, we need to be able to extract witnesses from valid proofs. However, zero knowledge implies that there are false proofs that contain no witnesses. The existence of these false proofs presents a problem for proving security using indistinguishability obfuscation.

More generally, consider the following general setup. There is a program C_0 receiving n values $x_1, \dots, x_n$ as inputs from n parties along with n NIZKs proving their validity. The program C_0 outputs $\perp$ whenever any of the NIZKs does not verify. In the other cases, it outputs $C(x_1, \dots, x_n)$ where C is some circuit. There also a second program C_1 that, instead of verifying the NIZKs, it tries to extract the witnesses hidden in them (C_1 outputs $\perp$ if the extraction of any witness fails). Then it uses the extracted witnesses to attempt to simulate the same behavior as C_0. The goal is to have obfuscations of C_0 and C_1 be indistinguishable.

The problem of differing inputs. The main issue is that C_0 and C_1 have differing inputs: the zero-knowledge property of the NIZKs guarantees the existence of proofs for which the witness cannot be extracted despite verification succeeds. On these inputs, the behavior of C_0 and C_1 can be easily told apart. In order to apply indistinguishability obfuscation, however, we need C_0 and C_1 to be equivalent programs.

Fortunately, finding these differing inputs is hard. Therefore the natural tool to achieve indistinguishability between obfuscations of C_0 and C_1 would be differing-input obfuscation [BGI+01]. The existence of such primitive is, however, controversial due to some results suggesting its impossibility [GGHW14, BSW16]. In [HIJ+17], Halevi *et al.* faced a problem similar to ours. They solved it by designing NIZKs that can be simulated only for statements hidden in the CRS. Since there is a small number of problematic statements, it is easy to take care of the corresponding executions of C_0 and C_1 using just indistinguishability obfuscation. The solution of Halevi *et al.*, however, compromises the reusability of the CRS and makes it grow with the size of the statements. Since we want to keep the CRS as simple as possible, we follow a different approach.

Indistinguishability obfuscation is enough. We rely solely on indistinguishability obfuscation. In [BCP14], Boyle, Chung and Pass showed that, if two programs have a polynomial number of differing inputs and finding any of them is hard, then iO is enough to hide which program was obfuscated. In our application, the number of differing inputs is of course superpolynomial, however, we notice that the result of [BCP14] can be generalized: assume that all differing inputs have a prefix in a set S. If finding an element in S is hard even for adversaries running in time $\mathsf{poly}(\lambda, |S|)$, subexponentially secure iO is sufficient to hide which program was obfuscated.

$\mathsf{DProg}_{\mathsf{Ls}}[K_i, \mathsf{EP}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f]$

Hard-coded. The PPRF key K_i, the encryption program EP_i, the CRS for the almost everywhere extractable NIZK σ, the extraction trapdoors $(\tau_e^j)_{j \neq i}$, a PPRF key K, an ELF f.
Input. Set of $n-1$ tuples $(\mathsf{EP}_j, \pi_j)_{j \neq i}$ where EP_j is the encryption program of party P_j and π_j is an almost everywhere extractable NIZK proving its well-formedness.

```
1. $\forall j \neq i: \quad K_j \leftarrow \mathsf{NIZK.Extract}(\tau_e^j, \pi_j, \mathsf{EP}_j)$
2. If $\exists j \neq i$ such that $K_j = \perp$, output $\perp$
3. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big((\mathsf{EP}_l)_{l \neq j}\big)$
4. $\forall j \neq i: \quad (s_j, r_j, r'_j) \leftarrow F(K_j, y_j)$
5. $z \leftarrow f(\mathsf{EP}_1, \ldots, \mathsf{EP}_n)$
6. $s \leftarrow F(K, z)$
7. $R' \leftarrow \mathcal{D}(1^\lambda; s)$
8. $(\eta_i, \eta'_i) \leftarrow F'(K_i, y_i)$
9. $(\phi, \mathsf{pk}_i, c_i) \leftarrow \mathsf{mkFHE.Sim}_1(1^\lambda; \eta_i)$
10. $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \mathcal{D}, R', (s_j, r_j, r'_j)_{j \neq i}; \eta'_i\Big)$
11. Output $d_i$
```

Figure 4.8: A sketch of the unobfuscated decryption program for the lossy mode

To leverage this observation, we introduce the notion of *almost everywhere extractable NIZKs*. Such NIZKs are designed so that the prefix of all the valid proofs for which the witness cannot be extracted lies in a set S. Finding an element in S is hard even for adversaries running in time $\mathsf{poly}\big(\lambda, |S|\big)$ that are provided with the extraction trapdoor. By using almost everywhere extractable NIZKs together with the generalization of [BCP14], we can show that P_0 and P_1 are hard to distinguish despite the existence of differing-inputs. We discuss building such NIZKs later in this overview.

4.2.4 Security Proof Challenge 2: More Differing Inputs

Decreasing the entropy of the encryption programs. At this point, we can try to prove the security of the candidate lossy distributed sampler. The strategy is the following: using the properties of the almost everywhere extractable NIZKs followed by an input-by-input iO argument, we show that, if the ELF is in injective mode, the lossy-mode programs are indistinguishable from the usual ones. By switching to a lossy ELF, we can then argue that the distinguishability advantage between the modes of the distributed sampler can be made an arbitrarily small inverse-polynomial function.

There is only one problem that hinders this plan: beyond the differing-inputs caused by the NIZK extraction (which are taken care by the almost everywhere extractable NIZKs), there exist other inputs for which the lossy-mode programs have a clearly distinguishable behaviour. Consider indeed two tuples of encryption programs $(\mathsf{EP}_j)_{j \neq i}$ and $(\mathsf{EP}'_j)_{j \neq i}$ having colliding hashes. When these tuples are used along with normal programs for party P_i, the outputs of the protocol will be correlated: in both executions, the programs of P_i use the same random string s_i (see how s_i is generated in Figure 4.5). If instead P_i sent lossy-mode programs, the outputs will look independent of each other (see how $\hat{R}$ is generated in Figure 4.8).

Even if these problematic inputs are hard to find, this time we do not use the trick by Boyle, Chung and Pass [BCP14]. To work around the issue, we decrease the entropy of the encryption programs: we require that they are generated using the randomness produced by a PRG with a small λ-bit seed. The almost everywhere extractable NIZKs will guarantee that the adversary does not break this rule. On the other hand, the lossy-mode programs will use full-entropy randomness. In this way, the total number of valid encryption programs for the corrupted parties becomes smaller than $(2^\lambda)^{n-1}$. By adopting a subexponentially collision-resistant hash function, we can make sure that, with overwhelming probability, there exist no collisions among these $(2^\lambda)^{n-1}$ elements. Moreover, the digests will still be small enough to fit into the encryption programs.

This technique solves also circular dependencies between subexponentially secure primitives: the input-by-input iO argument requires us to work with a number of hybrids that is proportional to the number of valid encryption programs. In each of these hybrids, we need to rely on the security of multi-key FHE. In order for the proof to go through, the size of the multi-key FHE keys therefore needs to increase logarithmically with the number of hybrids. If we used full-entropy encryption programs, the size of the keys would be so large that they would not even fit in the encryption programs anymore. By forcing valid encryption programs to have low entropy, we can hybrid over only the valid programs instead of all possible encryption programs, thereby eliminating the circular dependency. The properties of the NIZK guarantee not only that the adversary cannot find non-valid encryption programs, but that they do not even exist.
With these challenges overcome, we prove the following:

Theorem 4.2.4 (Informal). Assuming almost everywhere extractable NIZKs, subexponential iO, subexponential multi-key FHE, subexponentially collision-resistant hash functions and regular extremely lossy functions, the distributed sampler sketched above is lossy and programmable.

4.2.5 Building indistinguishability-preserving distributed samplers.

A lossy distributed sampler is not necessarily indistinguishability-preserving. We show, however, that the construction described above actually is:

Theorem 4.2.5 (Informal). Assuming almost everywhere extractable NIZKs, subexponential iO, subexponential multi-key FHE, subexponentially collision-resistant hash functions and regular extremely lossy functions, the distributed sampler sketched above is indistinguishability-preserving.

We start by considering a protocol Π that relies on a CRS sampled from the distribution $\mathcal{D}$. We suppose that Π implements a functionality $\mathcal{F}$ as described at the top of Figure 4.4. In particular, in the ideal world, the CRS is simulated using a distribution $\mathcal{D}'$ that outputs a trapdoor T along with the sample R.

A sketch of the proof. We use a hybrid argument beginning from the compilation of the real world using the standard mode of our lossy distributed sampler and ending with the compilation of the ideal world using a simulated mode (see the bottom of Figure 4.4). We prove that the compiled worlds are computationally indistinguishable.

As a first step, we switch the distributed sampler to lossy mode. This already introduces some non-negligible distinguishability advantage in the proof, we will explain later why this does not constitute a problem. On the other hand, the lossy mode allows us to move to a sample space of polynomial size.

Next, we gradually change the distribution of the outputs of the distributed sampler, switching from $\mathcal{D}$ to $\mathcal{D}'$. The technique here is rather simple: we just rely on the security of puncturable PRFs similarly to what we did to argue programmability. Along the way, we gradually switch from the execution of Π, to the execution of the simulator Sim_Π. In particular, there will some subhybrids in which some of the distributed sampler outputs come from $\mathcal{D}$ and some from $\mathcal{D}'$. We run Sim_Π only when the adversary chooses an execution where the sample comes from $\mathcal{D}'$. In these cases, we can retrieve the trapdoor by using the puncturable PRF key K and the ELF hidden in the lossy-mode programs (see Figure 4.8). Observe that, since the sample space is small, switching from $\mathcal{D}$ to $\mathcal{D}'$ needs only a polynomial number of subhybrids. As a consequence, we do not need that $\mathcal{D}$ and $\mathcal{D}'$ are subexponentially indistinguishable, nor that Π implements $\mathcal{F}$ with subexponential security.

In the last hybrid, we switch the ELF in the lossy-mode programs back to injective mode. Once again, the operation introduces a non-negligible distinguishability advantage. However, it allows us to move to a large sample space where all the elements are trapdoored.

The compiled games are indistinguishable. We finally argue why the non-negligible advantage introduced in the first and the last hybrid does not constitute a problem: by contradiction, suppose that there exists an adversary $\mathcal{A}$ that distinguishes between the initial and the final stage with non-negligible advantage ϵ. By choosing sufficiently large parameters for the lossy mode of the ELF (which is used only in the intermediate hybrids, but not in the real and the ideal world), we can ensure that the advantage of $\mathcal{A}$ in the first and the last steps of the proof are both bounded by $\epsilon/4$. The total advantage of $\mathcal{A}$ against the compiled games would therefore be strictly smaller than ϵ, reaching a contradiction.

On the reusability of the CRS of our distributed samplers. It is easy to see that the CRS of a hardness-preserving distributed sampler is always reusable across multiple concurrent executions of the protocol. Indeed, the hardness of the search problem is not affected by the concurrent executions as the latter are always simulatable. On the other hand, the security of an indistinguishability-preserving distributed sampler can be affected by the concurrent executions. The construction presented in this paper, however, does not suffer from this issue.

4.2.6 Building almost everywhere extractable NIZKs

We obtain almost everywhere extractable NIZKs in the CRS model using perfectly sound NIWIs, subexponentially secure injective one-way functions, perfectly binding commitments and perfectly correct identity-based encryption (IBE).

Why consider distributed samplers that need a CRS? It may seem strange to have a distributed sampler — whose purpose is to generate a CRS — in turn rely on a CRS. What is the advantage of generating a CRS using a distributed sampler if the latter still needs a CRS? There are several reasons why a distributed sampler using a CRS can be useful: the CRS of the distributed sampler might be reused multiple times, allowing the production of many samples. The CRS of the distributed sampler protocol can also be simple to generate, perhaps because it is short or because it is unstructured (i.e. a uniform string of bits).

Our Construction. The CRS consists of an IBE master public key and a one-way function challenge v. The proofs are associated to the identity of the party that issues them. Each of them consists of a commitment c_0, an IBE encryption of the witness c_1 under the party's identity and a NIWI guaranteeing that either c_1 contains the witness or c_0 contains the preimage of v. In order to extract the witness, it is sufficient to decrypt c_1.

Observe that, in all valid proofs for which extraction fails, the prefix is a commitment to the preimage of v. Since the one-way function is injective, the number of such prefixes depends only on the size of the randomness of the commitment scheme. As the one-way function is subexponentially secure, we can make v hard to invert even for $\mathsf{poly}(\lambda, |S|)$-time adversaries that have enough power to brute-force the commitment to retrieve the hidden value. This ensures the property we need.

Why to use identity-based encryption? In many applications of almost everywhere extractable NIZKs, we would like to argue that the programs C_0 and C_1 are indistinguishable even if we simulate the NIZKs of the honest parties (clearly, in these situations, C_1 will try to extract the witnesses only from the NIZKs of the corrupted players). The issue is that the NIZK described in the previous paragraph is not simulation-almost everywhere extractable, i.e. leaking simulated proofs may allow distinguishing between C_0 and C_1. On the other hand, disclosing C_1 might compromise the zero-knowledge property of the NIZKs due to the extraction trapdoor hidden into it.

Identity-based encryption allows us to work around the problem: to extract the witness from a NIZK proven under the identity id, we do not need the IBE master secret key, but just the private key associated to id. In other words, if we equip C_1 only with the decryption keys associated to the identities of the corrupted players, we are still able to simulate the proofs of the honest parties. The identities associated with the NIZKs guarantee that no corrupted party can publish one of the simulated proofs as it was its own.

Note that some IBE schemes such as [BF01] have uniformly random public keys. If we also use a one-way *permutation* to generate v, then the CRS is actually uniformly random. As such, our resulting distributed samplers will take a uniformly random CRS, and can be used to generate any arbitrarily structured CRS.

Theorem 4.2.6 (Informal). Assuming perfectly correct IBE, perfectly binding non-interactive commitments, perfectly sound NIWIs and subexponential OWFs, the NIZK sketched above is almost everywhere extractable.

4.2.7 CRS-less NIZKs in the Uniform Setting

All the distributed samplers we described so far make use of a CRS. The latter, needed by the NIZKs in the construction, is short, reusable and unstructured, however, is it possible to completely remove it? For indistinguishability-preserving distributed samplers, this is too much to hope for: if that was not the case, we would obtain a 3-round OT protocol with active security by compiling any 2-round OT protocol with CRS such as [PVW08]. It is known that active OT requires at least 4 rounds [HV16]. We show, however, that, if we restrict to security against uniform adversaries, we can remove the CRS from all our primitives. We obtain this by constructing CRS-less NIZKs that can be plugged in our distributed samplers.

NIZKs against uniform adversaries. The fact that NIZKs do not need CRSs if we restrict to security against uniform adversaries has been known for almost two decades: the fact was proven by Barak and Pass in [BP04] by building a CRS-less NIZK in the stand-alone model. In [BL18b], Bitansky and Lin studied a related question. They designed CRS-less NIZKs with a weak security guarantee against non-uniform adversaries: the number of false statements that can be proven is proportional to the non-uniformity of the adversary. Although this notion does not imply full soundness against uniform adversaries, it is easy to see that their constructions achieve the result. In this way, they indirectly obtain a CRS-less NIZK satisfying a weak form of simulation-soundness [Sah99]: a uniform adversary cannot generate proofs for false statements even if it has oracle access to the NIZK simulator that can be queried only with true statements (in the standard definition of simulation soundness, the simulator can be queried even with false statements).

Beyond these works, the topic remains rather unexplored. In this paper, we show how to construct CRS-less NIZKs achieving full simulation-soundness [Sah99], simulation extractability and almost-everywhere extractability against uniform adversaries. All our constructions rely on the same trick: in order to simulate a proof, we need to use a trapdoor. Such trapdoor will be infeasible to compute for every uniform adversary but not for the simulator as it will be non-uniform.

Uniform-DDH and uniform-LWE. We start by introducing natural variations of the DDH and LWE assumptions that we believe to hold against uniform adversaries.

Consider a uniform deterministic algorithm DDHGen that outputs the description of a cyclic group $\mathbb{G}$ along with two elements $g, h \in \mathbb{G}$ such that no uniform adversary can find the value α such that $h = g^\alpha$. A heuristic instantiation of this algorithm is to use a SHA hash function, or the digits of π, to generate g and h. The uniform-DDH assumption states that no uniform adversary can distinguish between pairs (g^r, h^r) and pairs (g^r, h^s) where r and s are uniformly random elements. Clearly, the assumption cannot hold against non-uniform adversaries: a non-uniform adversary can receive α as part of its non-uniform advice, at that point, distinguishing is trivial. Even uniform quantum adversaries can trivially distinguish by recovering α using Shor's algorithm. We however believe that it is possible to instantiate the assumption so that all uniform, classical PPT adversaries have subexponentially small advantage.

The uniform-LWE assumption follows a similar blueprint: we use a uniform deterministic algorithm LWEGen to generate the matrix $A \in \mathbb{Z}_q^{m\times n}$ describing a lattice. We then assume that no uniform PPT adversary can distinguish $A^\intercal \cdot s + x$ (where s is uniform in $\mathbb{Z}_q^n$ and x is a short vector in $\mathbb{Z}_q^m$) from a random element in $\mathbb{Z}_q^m$. Once again, we cannot hope to achieve security against non-uniform adversaries: if they receive a small vector u such that $A \cdot u = 0$ as part of their non-uniform advice, they can easily break the assumption. We however believe that every uniform, classical or quantum PPT adversary has a subexponentially small advantage.

The first simulation-sound NIZKs.

We obtain simulation-sound NIZKs without CRS using two different approaches. We now describe the first one.

Challengeless one-way functions. The first NIZK makes use of challengeless one-way functions (COWFs): a one-way function in which the challenge is deterministically generated by a uniform algorithm. The guarantee is that no uniform PPT adversary can find a preimage of the challenge.

We actually need two COWFs that are *independently hard*: finding preimages for any of them remains hard even when we are given a preimage for the other one. Uniform-DDH and uniform-LWE easily give a pair of independently hard one-way functions: thanks to the subexponential security of the primitive, we can make sure that, for classical adversaries, breaking uniform-DDH is strictly harder than uniform-LWE (this is achieved by making an appropriate choice of the parameters of the assumptions). On the other hand, in a post-quantum world, uniform-DDH is broken, while uniform-LWE retains its security. If breaking any of the challengeless one-way functions allows an adversary to break the other one, one of these two facts would be contradicted. This kind of trick was used before in [KK19, KNYY21, LPS17].

The first approach. The construction follows the blueprint of [BP04]. The proof consists of two commitments c_0 and c_1 along with a signature and a CRS-less NIWI [BOV03, GOS06a, GOS06b]. The NIZKs prove that either the statement lies in the language or one of the commitments hides a preimage for one of the independently hard challengeless one-way functions COWF_0 and COWF_1. These preimages will be used as trapdoors.

In order to achieve simulation-soundness, we need to ensure that the proof is non-malleable. We therefore generate c_0 and c_1 using a non-interactive CCA commitment without CRS [KS17, LPS17, BL18b, KK19, GKLW21]: each commitment is associated with a tag. The primitive guarantees that, given a commitment, no adversary can derive a commitment to a correlated value under a different tag. In our NIZK, similarly to [GO07], the tag will be a one-time signature verification key. Such key will be used to sign the proof. This ensures that, in order to produce a NIZK for a false statement, the adversary cannot reuse the commitments in the simulated proofs: it needs to at least change the tag (otherwise, it would need to forge a signature). The CCA security of the commitments guarantees the hardness of this task. Therefore, if the adversary manages to prove a false statement is because it discovered one of the trapdoors.

Why do we need two challengeless one-way functions? The reason is that we need to argue that the NIWIs in the simulated proofs leak no information about the trapdoors. When the statement for a simulated proof lies in the language, it is guaranteed that the NIWI does not leak the trapdoor. If that was not the case, by witness indistinguishability, the trapdoor would have been leaked even if the NIWI was generated using the witness for our statement. This contradicts the fact that the trapdoor is hard to compute. What instead if the statement does not lie in the language? In this case, the NIWI does not allow us to tell which trapdoor was used for its generation, however, it might leak some generic information about them, e.g. the minimal trapdoor according to the lexicographical order.

Using two independently hard, challengeless one-way function, we avoid this problem: by the independent hardness, if we use the COWF_0 trapdoors for the simulated proofs, the NIWIs cannot leak any COWF_1 trapdoor and vice-versa. By witness indistinguishability, we conclude that the NIWIs do not leak any of the trapdoors.

Theorem 4.2.7 (Informal). Assuming subexponential independently secure COWFs, non-interactive CCA-commitments without CRS, subexponential CRS-less NIWIs and strong one-time signatures, the CRS-less NIZK sketched above is simulation-sound against uniform adversaries.

The second simulation-sound NIZK.

We describe the second approach to build simulation-sound NIZKs without CRS.

Labelled, challengeless one-way functions (LOWF). Our second simulation-sound NIZK makes instead use of *labelled, challengeless one-way functions* CLOWF: on input any label id, a uniform algorithm deterministically generates a one-way function challenge. The primitive guarantees that no uniform PPT adversary can invert any challenge even given the preimages associated with some of the other labels. A heuristic instantiation of this primitive can use a SHA hash function to generate the verification key for a deterministic signature scheme. In this case, the preimage associated with a label id consists of a signature on id.

The second approach. Building simulation-sound NIZKs with the second approach is perhaps even easier: each proof consists of a commitment c, a CRS-less NIWI, a signature and the relative verification key vk. The NIWI is used to prove that either the statement belongs to the language or c hides a preimage for CLOWF where the label is vk. Such preimage acts as a trapdoor.

We use a signature over the whole proof to ensure that, if the adversary manages to prove a false statement, it uses a fresh verification key (otherwise, it would have succeeded in forging a signature). That means that the adversary needs to find a preimage relative to a fresh label of CLOWF. The trapdoors used in the simulated proof do not help in this task. We can therefore achieve simulation-soundness even with malleable commitments.

Theorem 4.2.8 (Informal). Assuming subexponential LOWF, perfectly binding non-interactive commitments, CRS-less NIWIs and strong one-time signatures, the CRS-less NIZK sketched above is simulation-sound against uniform adversaries.

CRS-less simulation-extractable NIZK.

In order to build simulation-extractable NIZKs, we introduce CRS-less non-interactive extractable commitment schemes. Observe that the primitive can exist only if we restrict to security against uniform adversaries. We build two schemes. The first one is based on uniform-DDH, the second one on uniform-LWE. A commitment consists of an encryption of the value using the public keys deterministically produced by either DDHGen or LWEGen. In the first case, we use ElGamal, in the second case, we use dual-LWE. To extract the value, it is sufficient to perform a decryption (the extractor will be a non-uniform algorithm). The operation is however infeasible for the adversary as the secret key is hard to compute in uniform polynomial-time.

In order to obtain a simulation-extractable NIZK, we simply generate an extractable commitment c to the witness for the statement we want to prove. We then use a simulation-sound NIZK to prove that c is indeed what we claim it to be.

Theorem 4.2.9 (Informal). Assuming CRS-less simulation-sound NIZKs and subexponential CRS-less non-interactive extractable commitments, the CRS-less NIZK sketched above is simulation-extractable against uniform adversaries.

CRS-less almost everywhere extractable NIZK.

We finally present a CRS-less almost everywhere extractable NIZK with security against uniform adversaries. Differently from the construction described in Section 4.2.2, this NIZK will use a single extraction trapdoor for every prover's identity. On the other hand, the scheme will remain almost everywhere extractable even if we provide oracle access to the zero-knowledge simulator (we call the property *simulation-almost everywhere extractability*). This ensures that the obfuscated programs P_0 and P_1 remain indistinguishable even if the proofs of the honest parties are simulated (we recall that P_0 is a program that verifies the NIZKs proving the well-formedness of its inputs, while P_1 instead tries to extract the witnesses from them).

Independently secure labelled one-way functions and extractable commitments. The construction makes use of a labelled challengeless one-way function CLOWF and a non-interactive extractable commitment. The two primitives need to be independently secure: they need to retain their security properties even when we leak the other primitive's trapdoor. We can for instance ensure this using the same trick we adopted for simulation-sound NIZKs: we use a post-quantum extractable commitment (which can be obtained from uniform-LWE) and a quantumly-broken labelled, challengeless one-way function (heuristically, we can obtain it from any DLOG-based deterministic signature).

The reason why we need independently secure primitives is that almost everywhere extractability always requires that the simulation trapdoor (i.e. the trapdoor for CLOWF) is hard to compute in uniform polynomial time even if we leak the extraction trapdoor (i.e. the trapdoor for the extractable commitment). On the other hand, in our construction, the proof of zero-knowledge would require the symmetric relation. Independent security allows us to satisfy both conditions simultaneously.

The simulation-almost everywhere extractable NIZK without CRS. A proof consists of two commitments c_0 and c_1, where c_1 is extractable, along with a CRS-less NIWI. The latter proves that either c_1 hides a witness for the statement we want to prove or c_0 hides a preimage for CLOWF where the label is the identity of the prover. In all the proofs where extraction fails, c_0 will therefore satisfy this second condition.

We select CLOWF so that the preimage for any given label is unique. In this way, the number of prefixes of problematic NIZKs for a given prover identity depends only on the size of the randomness of the commitment scheme. Since CLOWF is subexponentially secure, we can ensure that finding the right CLOWF preimage is infeasible even for $\mathsf{poly}(\lambda, |S|)$-time adversaries ($S$ denotes the set of problematic prefixes) that have enough computational power to recover the value hidden in c_0. Finding elements in S is therefore hard even for $\mathsf{poly}(\lambda, |S|)$-time algorithms. Learning simulated proofs under other provers' indentities does not help the adversary in the task.

Theorem 4.2.10 (Informal). Assume the existence of a subexponential injective LOWF and a CRS-less non-interactive extractable commitment that are independently secure. Assume perfectly binding non-interactive commitments and CRS-less NIWIs. Then, the CRS-less NIZK sketched above is simulation-almost everywhere extractable against uniform adversaries.

4.3 Notation and Preliminaries

In this section, we formalise the notation and recall the known results about distributed samplers.

Basic notation. We denote the security parameter by λ. For any $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, 2, \dots, n\}$. For any binary string x and integer ℓ, $\mathsf{Trunc}_\ell(x)$ denotes the prefix of x consisting of its first ℓ bits. Moreover, for any integers $\ell_0 < \ell_1$, we use $\mathsf{Trunc}_{\ell_0}^{\ell_1}(x)$ to denote the substring of x consisting of the bits from the ℓ_0-th position to the ℓ_1-th one. Given any NP relation $\mathcal{R}$, we denote the corresponding language by $L_\mathcal{R}$.

Algorithm execution. For any deterministic algorithm $\mathcal{A}$ and input x, we use the expression $a \leftarrow \mathcal{A}(x)$ to assign the output of the algorithm $\mathcal{A}$ on input x to the variable a. When $\mathcal{A}$ is probabilistic, we instead use $a \overset{\$}{\leftarrow} \mathcal{A}(x)$. Finally, if $\mathcal{A}$ is randomised, we use $a \leftarrow \mathcal{A}(x; r)$ to mean that a is assigned the output of $\mathcal{A}$ on input x and randomness r. If x is variable, we use $a \leftarrow x$ to assign the value of x to a. If X is a set, instead, we use $a \overset{\$}{\leftarrow} X$ to mean that a is assigned a value sampled from X uniformly at random. If $\mathcal{A}$ and $\mathcal{O}$ are algorithms, for any x and y, we use $\mathcal{A}^{\mathcal{O}(y,\cdot)}(x)$ to denote the value output by $\mathcal{A}$ on input x while having unbounded oracle access to $\mathcal{O}(y, \cdot)$. In other words, at any point in time, $\mathcal{A}$ can send values z to an oracle, which replies with $\mathcal{O}(y, z)$. We use the term *efficient distribution* to denote a uniform PPT algorithm taking only the security parameter as input.

Asymptotic behaviour. We use $\mathsf{negl}(\lambda)$ (resp. $\mathsf{nonegl}(\lambda)$) to denote a generic negligible (resp. non-negligible) function in the security parameter. Similarly, we use $\mathsf{poly}(X_1, \ldots, X_n)$ to denote a generic function that is upper-bounded by a polynomial in the given variables $X_1, \ldots, X_N$. Given two functions $S_0(\lambda)$ and $S_1(\lambda)$, we say that $S_0(\lambda) \ll S_1(\lambda)$ if $S_0(\lambda)$ is a $\mathsf{poly}\big(\lambda, S_1(\lambda)\big)$ function but $S_1(\lambda)$ is not $\mathsf{poly}\big(\lambda, S_0(\lambda)\big)$.

Uniform vs non-uniform adversaries. We recall that a non-uniform algorithm consists of a randomised Turing machine that, at the beginning of its execution, receives a polynomial-size advice string, whose value depends only on the security parameter. A uniform algorithm is instead a randomised Turing machine that receives no such advice string. Throughout the paper, we use AClass to denote either the class of uniform algorithms or the class of non-uniform algorithms. Observe that the latter is strictly larger than the former.

Multiparty computation. In the paper, we deal with multiparty protocols. We always assume the existence of authenticated point-to-point channels along with an authenticated broadcast medium. We often denote the i-th party by P_i. We also assume that each party is associated with a unique identity id known to all the other players. We work with static corruption and we denote the set of honest parties by H. We say that a CRS is unstructured if it is computationally indistinguishable from a uniformly random string of a given length.

Subexponential security. We say that a primitive is subexponentially secure if there exists $e > 0$ such that the advantage of every adversary running in $\mathsf{poly}\big(2^{\lambda^e}\big)$ time in the relative security game is asymptotically smaller than $2^{-\lambda^e}$. In this appendix, we recall security definitions and basic results used in this work.

4.3.1 One-Way Functions

We recall the definition of one-way function (OWF): a function that can be efficiently computed but hard to invert on random instances.

Definition 4.3.1 (One-way function). A one-way function is a pair of uniform PPT algorithms $(\mathsf{Gen}, \mathsf{OWF})$ with the following syntax:

- Gen is randomised, takes as input the security parameter 1^λ and outputs a pair (v, u).
- OWF is deterministic and takes as input the security parameter 1^λ and a value u. The output is a value v.

We require the following properties

- **(Correctness).** For every $\lambda \in \mathbb{N}$, we have

$$\Pr\left[\mathsf{OWF}(1^\lambda, u) = v \middle| (v, u) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)\right] = 1.$$

- **(Security).** For every PPT adversary $\mathcal{A}$, we have

$$\Pr\left[\mathsf{OWF}(1^\lambda, u') = v \middle| (v, u) \xleftarrow{\$} \mathsf{Gen}(1^\lambda), u' \xleftarrow{\$} \mathcal{A}(1^\lambda, v)\right] = \mathsf{negl}(\lambda).$$

We say that the the one-way function is injective if

$$\Pr\left[\exists u' \neq u \quad \mathsf{OWF}(1^\lambda, u') = v \middle| (v, u) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)\right] = 0.$$

One-way functions, including subexponentially secure ones, can be built using well studied assumptions.

4.3.2 Puncturable PRFs

We recall now the definition of puncturable PRF [KPTZ13, BW13, BGI14]. As for a standard PRF, it consists of a keyed functions whose outputs are indistinguishable from random as long as the key remains secret. The primitive, however, satisfies an additional property: it is possible to generate punctured keys. The latter permit evaluating the PRF in any point of its domain except for the punctured position. Furthermore, even if the punctured key is disclosed, the value of the PRF at the punctured position remains indistinguishable from random.

Definition 4.3.2 (Puncturable PRF)*.* Let $p(\lambda)$ and $q(\lambda)$ be polynomial functions. A puncturable PRF with input size $p(\lambda)$ and output size $q(\lambda)$ is a pair of uniform PPT algorithms $(\mathsf{Gen}, F, \mathsf{Punct})$ with the following syntax:

- Gen is randomised, takes as input the security parameter 1^λ and outputs a key K.
- F is deterministic and takes as input a key K and a value $x \in \{0,1\}^{p(\lambda)}$. The output is a pseudorandom string $y \in \{0,1\}^{q(\lambda)}$.
- Punct is deterministic and takes as input a key K and a value $x \in \{0,1\}^{p(\lambda)}$. The output is a punctured key K^*.

We require the following properties.

- **(Correctness).** For every pair of distinct values x and x' in $\{0,1\}^{p(\lambda)}$, we have

$$\Pr\left[F(K,x') = F(K^*,x') \middle| K \xleftarrow{\$} \mathsf{Gen}(1^\lambda),\ K^* \leftarrow \mathsf{Punct}(K,x)\right] = 1.$$

- **(Security).** For every pair of PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, we have

$$\left| \Pr\left[\mathcal{A}_2(\psi, K^*, y_b) = b \middle| \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ K \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \\ (x,\psi) \xleftarrow{\$} \mathcal{A}_1(1^\lambda) \\ K^* \leftarrow \mathsf{Punct}(K,x) \\ y_0 \leftarrow F(K,x) \\ y_1 \xleftarrow{\$} \{0,1\}^{q(\lambda)} \end{array} \right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda).$$

Puncturable PRFs, even with subexponential security, can be easily constructed using one-way functions.

4.3.3 Hash Functions

We recall now the definition of collision resistant hash function. Essentially, the latter consists of a keyed function for which it is hard to find pairs of different elements that are mapped to the same value. Security relies on the unpredictability of the key. It is possible to build subexponentially secure collision resistant hash functions from well studied assumptions.

Definition 4.3.3 (Collision resistant hash function)*.* Let $p(\lambda)$ and $t(\lambda)$ be polynomial functions. A hash function with input size $p(\lambda)$ and digest size $t(\lambda)$ is a pair of uniform PPT algorithms $(\mathsf{Gen}, \mathsf{Hash})$ with the following syntax:

- Gen is randomised, takes as input the security parameter 1^λ and outputs an hash key hk.
- Hash is deterministic and takes as input a hash key hk and a value $x \in \{0,1\}^{p(\lambda)}$. The output is a digest $y \in \{0,1\}^{t(\lambda)}$.

We say that the hash function is collision resistant if, for PPT adversary $\mathcal{A}$, we have

$$\left|\Pr\left[\begin{matrix} x_0 \neq x_1 \\ \mathsf{Hash}(\mathsf{hk}, x_0) = \mathsf{Hash}(\mathsf{hk}, x_1) \end{matrix} \middle| \begin{matrix} \mathsf{hk} \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \\ (x_0, x_1) \xleftarrow{\$} \mathcal{A}(1^\lambda, \mathsf{hk}) \end{matrix}\right]\right| = \mathsf{negl}(\lambda).$$

Applied cryptography makes often use of a keyless version of the above primitive for which finding collisions is still believed to be hard. We formalise the definition below. We highlight that this primitive can hope to achieve security only against uniform adversaries. Indeed, since there is no randomness involved in the construction, a non-uniform adversary can be given a collision as part of its advice string.

Definition 4.3.4 (Keyless collision resistant hash function). Let $p(\lambda)$ and $t(\lambda)$ be polynomial functions. A keyless hash function with input size $p(\lambda)$ and digest size $t(\lambda)$ is a uniform deterministic polynomial time algorithm KHash that takes as input the security parameter 1^λ and a value $x \in \{0,1\}^{p(\lambda)}$. The output is a digest $y \in \{0,1\}^{t(\lambda)}$.

We say that the keyless hash function is collision resistant if, for every *uniform* PPT adversary $\mathcal{A}$, we have

$$\left|\Pr\left[x_0 \neq x_1, \mathsf{KHash}(1^\lambda, x_0) = \mathsf{KHash}(1^\lambda, x_1) \middle| (x_0, x_1) \xleftarrow{\$} \mathcal{A}(1^\lambda)\right]\right| = \mathsf{negl}(\lambda).$$

4.3.4 Commitments

In this subsection, we recall definitions of non-interactive commitments. A non-interactive commitment scheme is a primitive that allows encoding a message m in a string c, called the commitment. By itself, c hides the value of m, so it can be distributed to other parties without fear of revealing its secret. At a later point in time, the commitment can however be opened, disclosing the value hidden into it. The scheme guarantees the hardness of opening c to any value other than m. In other words, after the commitment is opened, the parties can be sure that who generated c had been already committed to revealing m since the time c was sent.

In this paper, we will make use of perfectly binding, computationally hiding non-interactive schemes. In particular, that means that the value hidden in the commitment remains secret only to computationally bounded adversaries. Furthermore, the commitment c uniquely determines the value hidden into it. Such schemes can be built, even with subexponential security, based on well-studied assumptions.

Definition 4.3.5 (Non-interactive commitment scheme). Let $p(\lambda)$ be a polynomial function. A non-interactive commitment scheme with message size $p(\lambda)$ is a uniform PPT algorithm Com that takes as input the security parameter 1^λ and a message $m \in \{0,1\}^{p(\lambda)}$. The output is a commitment c.

We say that the scheme is perfectly binding if, for every $\lambda \in \mathbb{N}$, there exist no pairs (m_0, r_0) and (m_1, r_1) such that $m_0 \neq m_1$ and $\mathsf{Com}(1^\lambda, m_0; r_0) = \mathsf{Com}(1^\lambda, m_1; r_1)$.

We say that the scheme is computationally hiding if, for every pair of PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, we have

$$\left|\Pr\left[\mathcal{A}_2(\psi, c) = b \middle| \begin{matrix} b \xleftarrow{\$} \{0,1\} \\ (m_0, m_1, \psi) \xleftarrow{\$} \mathcal{A}_1(1^\lambda) \\ c \xleftarrow{\$} \mathsf{Com}(1^\lambda, m_b) \end{matrix}\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda).$$

We also recall the definition of computation-enabled CCA commitment [KS17] [LPS17, BL18b, KK19, GKLW21]. This is a particular type of commitment that satisfies non-malleability. That means that given a commitment c hiding a value m, we are not able to derive another commitment c' that hides some value m' correlated to m. This property is formulated by augmenting the commitment algorithm with tags. Formally, we require that, if a value m is committed along with a tag id, m remains hidden even if the adversary has access to an inefficient oracle that extracts the values from the queried commitments. Clearly, the oracle accepts only commitments that use tags different from id.

Obtaining non-interactive non-malleable commitments with large tag space without relying on setups is not an easy task. For this reason, in this paper, we rely on constructions of this kind that achieve security only against uniform adversaries. In particular, the primitive we are interested in satisfies computation-enabled

CCA-Hiding Game

Initialisation: This procedure is run only once, at the beginning of the game.

1. $b \xleftarrow{\$} \{0,1\}$
2. Activate $\mathcal{A}$ with 1^λ.
3. Receive a Turing machine P from the adversary.
4. Run P on no input for at most $t(2^{\lambda^e})$ steps. If P does not terminate before that, provide $\mathcal{A}$ with $\perp$, otherwise, provide it with the first $s(\lambda)$ bits of the output.
5. Receive a tag $\widehat{\mathsf{id}}$ from the adversary.

Value: This procedure can be queried multiple times, both before and after choosing the challenge. Upon receiving pairs (id, c) where $\mathsf{id} \neq \widehat{\mathsf{id}}$, the challenger replies with $\mathsf{Val}(\mathsf{id}, c)$.

Challenge: This procedure can be queried only once. The adversary provides $m_0, m_1 \in \{0,1\}^{p(\lambda)}$. The challenger replies with $c \xleftarrow{\$} \mathsf{CCACom}(1^\lambda, \widehat{\mathsf{id}}, m_b)$.

Win: The adversary wins if it guesses b.

Figure 4.9: CCA-hiding game

CCA security, meaning that, at the beginning of the game we described above, the uniform adversary is allowed to query a possibly inefficient, randomised Turing machine with no input. The challenger provides the adversary with the result of the machine execution.

Definition 4.3.6 (Computation-enabled CCA commitment). Let $p(\lambda)$ and $q(\lambda)$ be polynomial functions, let $e > 0$. A e-computation enabled CCA commitment scheme with message size $p(\lambda)$ and tag size $q(\lambda)$ is a pair of uniform algorithms $(\mathsf{CCACom}, \mathsf{Val})$ with the following syntax:

- CCACom is PPT and takes as input the security parameter 1^λ, a tag $\mathsf{id} \in \{0,1\}^{q(\lambda)}$ and a message $m \in \{0,1\}^{p(\lambda)}$. The output is a commitment c.
- Val is deterministic and inefficient. It takes as input a label id and a commitment c and outputs either a message $m \in \{0,1\}^{p(\lambda)}$ or $\perp$.

We require the following properties.

- **(Correctness).** For every $\lambda \in \mathbb{N}$, $\mathsf{id} \in \{0,1\}^{q(\lambda)}$ and $m \in \{0,1\}^{p(\lambda)}$, we have

$$\Pr\left[\mathsf{Val}(\mathsf{id}, c) = m \middle| c \xleftarrow{\$} \mathsf{CCACom}(1^\lambda, \mathsf{id}, m)\right] = 1.$$

- **(CCA-Hiding).** For every polynomials $t(\lambda)$ and $s(\lambda)$, no uniform PPT adversary $\mathcal{A}$ can win the game in Figure 4.9 with non-negligible advantage.

4.3.5 Strong One-Time Signatures

We recall here the definition of strong one-time signature. Informally, this consists in a signing scheme for which it is hard to craft forgeries if we are given access to just one signature. The scheme is strong in the sense that, given a signature s for a message m, it is even hard to find another signature s' for m. Strong one-time signatures can be built from one-way functions.

Definition 4.3.7 (Strong one-time signature). Let $p(\lambda)$ be a polynomial function. A strong one-time signature is a triple of uniform PPT algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ with the following syntax:

- Gen is randomised and takes as input the security parameter 1^λ. The output is a key pair $(\mathsf{vk}, \mathsf{sk})$.
- Sign is randomised and takes as input a secret key sk and a message $m \in \{0,1\}^{p(\lambda)}$. The output is a signature s.
- Verify is deterministic and takes as input a verification key vk, a message $m \in \{0,1\}^{p(\lambda)}$ and a signature s. The output is a bit $b \in \{0,1\}$.

We require the following properties.

- **(Correctness).** For every $m \in \{0,1\}^{p(\lambda)}$, we have

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m, s) = 1 \middle| (\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda), s \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, m)\right] = 1.$$

- **(Security).** For every pair of PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, we have

$$\Pr\left[\begin{array}{c} (s,m) \neq (\widehat{s}, \widehat{m}) \\ \mathsf{Verify}(\mathsf{vk}, m, s) = 1 \end{array} \middle| \begin{array}{l} (\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \\ (\widehat{m}, \psi) \xleftarrow{\$} \mathcal{A}_1(1^\lambda, \mathsf{vk}) \\ \widehat{s} \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, \widehat{m}) \\ (s, m) \xleftarrow{\$} \mathcal{A}_2(\psi, \widehat{s}) \end{array}\right] = \mathsf{negl}(\lambda).$$

4.3.6 Non-Interactive Witness Indistinguishability

We recall the definition of non-interactive witness-indistinguishable proof (NIWI). Essentially, this consists of a construction specifying how to prove that a given statement x belong to a language using a single message. In order to be efficient, the algorithm that generates the proof needs to receive a witness for x as input. The primitive does not guarantee that the proof keeps the witness secret. It achieves, however, a weaker form of security stating that if there are multiple witnesses for the same statement x, the proof does not disclosed which witness was used for its generation.

It is possible to build subexponentially secure NIWI proofs without setups from various assumptions, specifically, DLIN [GOS06b, GOS06a], derandomisation [BOV03] and indistinguishability obfuscation [BP15].

Definition 4.3.8 (NIWI proof). Let $\mathcal{R}$ be an NP relation. A NIWI proof is a pair of uniform PPT algorithms $(\mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- Prove is randomised and takes as input the security parameter 1^λ, a statement x and a witness w. The output is a proof π.
- Verify is deterministic and takes as input a proof π and a statement x. The output is a bit $b \in \{0,1\}$.

We require the following properties.

- **(Completeness).** There exists a negligible function $\mathsf{negl}(\lambda)$ such that, for every $(x, w) \in \mathcal{R}$, we have

$$\Pr\left[\mathsf{Verify}(\pi, x) = 1 \middle| \pi \xleftarrow{\$} \mathsf{Prove}(1^\lambda, x, w)\right] = 1 - \mathsf{negl}(\lambda).$$

- **(Perfect Soundness).** If $x \notin L_{\mathcal{R}}$, there exists no π such that $\mathsf{Verify}(\pi, x) = 1$.
- **(Witness-Indistinguishability).** For every pair of PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, we have

$$\left|\Pr\left[\mathcal{A}_2(\psi, \pi) = b \middle| \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ (x, w_0, w_1, \psi) \xleftarrow{\$} \mathcal{A}_1(1^\lambda) \\ \pi \xleftarrow{\$} \mathsf{Prove}(1^\lambda, x, w_b) \\ \text{If } (x, w_0) \notin \mathcal{R} \text{ or } (x, w_1) \notin \mathcal{R} : \pi \leftarrow \bot \end{array}\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda).$$

THE IND-ID-CPA GAME

Initialisation: This procedure is run only once, at the beginning of the game.

1. $b \xleftarrow{\$} \{0,1\}$
2. $Q \leftarrow \emptyset$
3. $\widehat{\mathsf{id}} \leftarrow \perp$
4. $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$
5. Activate the adversary with 1^λ and mpk.

Key: This procedure can be queried multiple time, both before and after choosing the challenge. On input identities $\mathsf{id} \in \{0,1\}^{q(\lambda)}$ such that $\mathsf{id} \neq \widehat{\mathsf{id}}$, the challenger adds id to Q and replies with $\mathsf{Extract}(\mathsf{msk}, \mathsf{id})$.

Challenge: This procedure can be queried only once. The adversary provides $m_0, m_1 \in \{0,1\}^{p(\lambda)}$ and $\widehat{\mathsf{id}} \in \{0,1\}^{q(\lambda)} \setminus Q$. The challenger answers with $\mathsf{Enc}(\mathsf{mpk}, \widehat{\mathsf{id}}, m_b)$.

Win: The adversary wins if it guesses b.

Figure 4.10: The IND-ID-CPA game

4.3.7 Identity-Based Encryption

We recall the definition of identity-based encryption (IBE) [Sha84, BF01]. An IBE scheme is a public-key encryption scheme that is augmented with an access policy: each ciphertext and each secret key is associated with an identity. It is possible to decrypt only if two identities match. Holding keys associated with other identities gives no help in retrieving information about the plaintext.

Definition 4.3.9 (Identity-based encryption). Let $p(\lambda)$ and $q(\lambda)$ be polynomial functions. An identity-based encryption scheme (IBE) with message size $p(\lambda)$ and identity size $q(\lambda)$ is a tuple of uniform PPT algorithms $(\mathsf{Setup}, \mathsf{Extract}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- Setup is randomised and takes as input the security parameter 1^λ. The output is a key pair $(\mathsf{mpk}, \mathsf{msk})$.
- $\mathsf{Extract}$ is randomised and takes as input a master secret key msk and an identity $\mathsf{id} \in \{0,1\}^{q(\lambda)}$. The output is a secret-key sk.
- Enc is randomised and takes as input a master public key mpk, an identity $\mathsf{id} \in \{0,1\}^{q(\lambda)}$ and a message $m \in \{0,1\}^{p(\lambda)}$. The output is a ciphertext c.
- Dec is deterministic and takes as input a secret-key sk and a ciphertext c. The output is a message m or $\perp$.

We require the following properties.

- **(Perfect Correctness).** For every $\mathsf{id} \in \{0,1\}^{q(\lambda)}$ and $m \in \{0,1\}^{p(\lambda)}$,

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, c) = m \;\middle|\; \begin{array}{l} (\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ c \xleftarrow{\$} \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m) \\ \mathsf{sk} \xleftarrow{\$} \mathsf{Extract}(\mathsf{msk}, \mathsf{id}) \end{array}\right] = 1.$$

- **(IND-ID-CPA security)**. No PPT adversary can win the game in Figure 4.10 with non-negligible advantage.

Subexponentially secure IBE schemes can be built in the plain model using a large variety of assumptions [CHK03, BB04, Wat05, Gen06, ABB10].

4.3.8 Indistinguishability Obfuscation

We recall the definition of indistinguishability obfuscation [BGI+01, GGH+13]. An indistinguishability obfuscation is an algorithm that modifies a circuit without altering its input-output behaviour. The result is however so "scrambled" that it is hard to tell what the original circuit looked like. In this paper, we use the terms "circuit" and "program" interchangeably.

Definition 4.3.10 (Indistinguishability obfuscator). An indistinguishability obfuscator is a uniform PPT algorithm iO that takes as input the security parameter 1^λ and a circuit C. The output is an obfuscate program $\widetilde{C}$. We require the following properties.

- **(Perfect Correctness).** For every circuit C and input x, we have

$$\Pr\left[C(x) = \widetilde{C}(x) \middle| \widetilde{C} \xleftarrow{\$} \mathsf{iO}(1^\lambda, C)\right] = 1.$$

- **(Security).** For every PPT adversary $\mathcal{A}$ and sampler Samp outputting same-size circuits C_0 and C_1 such that $\forall x : C_0(x) = C_1(x)$ along with auxiliary information aux, we have

$$\left| \Pr\left[\mathcal{A}(1^\lambda, \widetilde{C}, C_0, C_1, \mathsf{aux}) = b \middle| \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ (C_0, C_1, \mathsf{aux}) \xleftarrow{\$} \mathsf{Samp}(1^\lambda) \\ \widetilde{C} \xleftarrow{\$} \mathsf{iO}(1^\lambda, C_b) \end{array}\right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda).$$

Although the initial obfuscation constructions were based on non-standard assumptions [GGH+13], the field has recently shown significant progress. State-of-the-art obfuscators can indeed be based on the subexponential hardness of well-founded problems [JLS21, JLS22]. Notice that the subexponential security of obfuscation is a common assumption in cryptography [CLTV15, DHRW16, HIJ+17].

In this paper, we will use indistinguishability obfuscators satisfying a particular property called injectivity. In other words, it is guaranteed that the obfuscation of distinct circuits will never collide. It is easy to obtain this property by appending a perfectly binding commitment of the unobfuscated circuit to the obfuscated program [CCK+22].

Definition 4.3.11 (Injective indistinguishability obfuscator). We say that an indistinguishability obfuscator iO is injective if, for every $\lambda \in \mathbb{N}$, there exist no pairs (C_0, r_0) and (C_1, r_1) such that $C_0 \neq C_1$ but $\mathsf{iO}(1^\lambda, C_0; r_0) = \mathsf{iO}(1^\lambda, C_1; r_1)$.

4.3.9 Multi-Key FHE

We recall the definition of multi-key fully homomorphic encryption [LTV12, CM15] [MW16]. As standard FHE, multi-key FHE scheme is a public key encryption scheme that allows homomorphically applying functions on encrypted values deriving encryptions of the outputs. The evaluation of the function is performed locally and no information about the plaintexts is revealed in the process. The big advantage of multi-key FHE is that, while standard FHE allows performing operations only between ciphertexts encrypted under the same public key, multi-key FHE suffers from no such restriction: we can evaluate functions on inputs encrypted under different keys, obtaining an encryption of the output under a "joint key". In order to decrypt the latter, the parties need to collaborate: each player will locally compute a partial decryption using its own private key. By pooling together the partial plaintexts, everybody can retrieve the result.

Subexponentially secure multi-key FHE without CRS can be built based on LWE and DSPR [AJJM20], or obfuscation and DDH [DHRW16]. In this paper, we rely on the definition of [AJJM20].

Definition 4.3.12 (Multi-key FHE). An multi-key fully homomorphic encryption scheme is a tuple of uniform PPT algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{PartDec}, \mathsf{FinDec})$ with the following syntax:

- Gen is randomised and takes as input the security parameter 1^λ. The output is a key pair $(\mathsf{pk}, \mathsf{sk})$.

- Enc is randomised and takes as input a public key pk and a message m. The output is a ciphertext c.

- Eval is deterministic and takes as input a function f and n pairs (pk_i, c_i) where n is the number of inputs of f. The output is a ciphertext C encrypted under the joint public key $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$.
- $\mathsf{PartDec}$ is randomised and takes as input a ciphertext C, n public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ for some $n \in \mathbb{N}$, an index $i \in [n]$ and a secret key sk. The output is a partial decryption d.
- FinDec is deterministic and takes as input n partial decryptions $d_1, \ldots, d_n$ for some $n \in \mathbb{N}$. The output is a message m or $\perp$.

We require the following properties.

- **(Correctness).** For every function f with n inputs and values $x_1, \ldots, x_n$, we have

$$\Pr\left[m = f(x_1, \ldots, x_n) \middle| \begin{array}{l} \forall i \in [n] : (\mathsf{pk}_i, \mathsf{sk}_i) \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \\ \forall i \in [n] : c_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}_i, x_i) \\ C \leftarrow \mathsf{Eval}(f, \mathsf{pk}_1, c_1, \ldots, \mathsf{pk}_n, c_n) \\ \forall i \in [n] : d_i \xleftarrow{\$} \mathsf{PartDec}(C, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, i, \mathsf{sk}_i) \\ m \leftarrow \mathsf{FinDec}(d_1, \ldots, d_n) \end{array} \right] = 1.$$

- **(Reusable Semi-Malicious Security).** There exists uniform PPT simulators Sim_1 and Sim_2 such that no PPT adversary $\mathcal{A}$ can win the game in Figure 4.11 with non-negligible advantage.

4.3.10 Extremely Lossy Functions

We recall the definition of extremely lossy function (ELF) [Zha16]. An ELF is a function f parametrised by two values M and r. The former denotes the cardinality of its domain, whereas r denotes an upper bound on the size of the image. When $M = r$, the function is guaranteed to be injective. When $r \neq M$, we say that the ELF is in lossy mode. The primitive ensures that, by choosing a sufficiently large $\mathsf{poly}(\log M)$ value r, the advantage in distinguishing between an injective ELF and a lossy ELF can be made an arbitrarily small inverse polynomial function in $\log M$. Extremely lossy functions can be built based on the *exponential* hardness of DDH over elliptic curves [Zha16].

Definition 4.3.13 (Extremely lossy function). An extremely lossy function (ELF) consists of a uniform PPT algorithm Gen that takes as input two integers M and r. The output is the description of a function f with domain $[M]$. The primitive uses $\log M$ as security parameter. We require the following properties.

- f is computable in $\mathsf{poly}\log(M)$ time and the running time is independent of r.
- If $r = M$, $\Pr\left[\exists x \neq y \text{ s.t. } f(x) = f(y) \middle| f \xleftarrow{\$} \mathsf{Gen}(M, M)\right] = \mathsf{negl}(\log M)$.
- There every $r \in [M]$, $\Pr\left[\left|f([M])\right| \geq r \middle| f \xleftarrow{\$} \mathsf{Gen}(M, r)\right] = \mathsf{negl}(\log M)$.
- For every polynomial p and inverse polynomial function δ, there exists a polynomial q such that, for every adversary $\mathcal{A}$ running in time at most $p(\log M)$, and $r \geq q(\log M)$, we have

$$\left| \Pr\left[\mathcal{A}(M, r, f_b) = 1 \middle| \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ f_1 \xleftarrow{\$} \mathsf{Gen}(M, r) \\ f_0 \xleftarrow{\$} \mathsf{Gen}(M, M) \end{array} \right] - \frac{1}{2} \right| \leq \delta(\log M).$$

In the constructions in this paper, $\log M$ will be both $\mathsf{poly}(\lambda)$ and $\Omega(\lambda)$. Therefore, every negligible function in $\log M$ will also be negligible in λ (and viceversa). Similarly, every polynomial function in $\log M$ will also be polynomial in λ (and viceversa).

We now recall the definition of regular ELF [Zha16]. Informally, an ELF is regular if, by applying the function on a random domain element, we hit all the elements in the image with at least inverse-polynomial probability in r and $\log M$. Regular ELFs can be built based on exponential DDH [Zha16].

The Multi-Key FHE Security Game

Initialisation: This procedure is run only once, at the beginning.

1. $b \xleftarrow{\$} \{0,1\}$
2. Activate the adversary $\mathcal{A}$ with 1^λ
3. Receive $H \subseteq [n]$ from the adversary along with $(x_i)_{i \in H}$.
4. $\forall i \in H : (\mathsf{pk}_i^0, \mathsf{sk}_i^0) \xleftarrow{\$} \mathsf{Gen}(1^\lambda, i)$
5. $\forall i \in H : c_i^0 \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}_i^0, x_i)$
6. $\big(\phi, (\mathsf{pk}_i^1, c_i^1)_{i \in H}\big) \xleftarrow{\$} \mathsf{Sim}_1(1^\lambda, H)$
7. $\forall i \in H : (\mathsf{pk}_i, c_i) \leftarrow (\mathsf{pk}_i^b, c_i^b)$
8. Provide $\mathcal{A}$ with $(\mathsf{pk}_i, c_i)_{i \in H}$.
9. Receive $(x_j, r_j, r'_j)_{j \notin H}$ from $\mathcal{A}$
10. $\forall j \notin H : (\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{Gen}(1^\lambda, j;\, r_j)$
11. $\forall j \notin H : c_j \leftarrow \mathsf{Enc}(\mathsf{pk}_j, x_j;\, r'_j)$

Decryption: This procedure can be queried multiple times. On input a function f with n inputs, compute the following.

1. $C \leftarrow \mathsf{Eval}(f, \mathsf{pk}_1, c_1, \dots, \mathsf{pk}_n, c_n)$
2. $y \leftarrow f(x_1, \dots, x_n)$
3. $\forall i \in H : d_i^0 \xleftarrow{\$} \mathsf{PartDec}(C, \mathsf{pk}_1, \dots, \mathsf{pk}_n, i, \mathsf{sk}_i^0)$
4. $\big(\phi', (d_i^1)_{i \in H}\big) \xleftarrow{\$} \mathsf{Sim}_2\big(\phi, f, y, (x_j, r_j, r'_j)_{j \notin H}\big)$
5. $\phi \leftarrow \phi'$
6. Provide $(d_i^b)_{i \in H}$ to $\mathcal{A}$

Win: The adversary wins if it guesses b.

Figure 4.11: The multi-key FHE security game

Definition 4.3.14 (Regular ELF). An ELF is regular if there exists $s = \mathsf{poly}(\log M, r)$ such that, except with negligible probability over $f \xleftarrow{\$} \mathsf{Gen}(M, r)$, for every $y \in f([M])$, we have

$$\Pr_x\big[f(x) = y \big| x \xleftarrow{\$} [M]\big] \geq \frac{1}{s(\log M, r)}$$

where $\Pr_x$ is a probability taken over the randomness of x.

We also recall the definition of strongly efficiently enumerable ELF [Zha16]. This consists an ELF in which it is possible to reconstruct the image in $\mathsf{poly}(\log M, r)$ time.

Definition 4.3.15 (Strongly efficiently enumerable ELF). An ELF is strongly efficiently enumerable if there exists a randomise algorithm Enum running in $\mathsf{poly}(\log M, r)$ time such that, for every $r \in [M]$,

$$\Pr\left[S \neq f\big([M]\big) \middle| f \xleftarrow{\$} \mathsf{Gen}(M, r), S \xleftarrow{\$} \mathsf{Enum}(M, 1^r, f)\right] \leq \mathsf{negl}(\log M).$$

It easy to show that every regular ELF is strongly efficiently enumerable [Zha16].

Theorem 4.3.16 ([Zha16]). A regular ELF is strongly efficiently enumerable.

4.3.11 Distributed Samplers

Distributed samplers [ASY22] are a powerful primitive allowing n parties to securely generate a sample from a fixed distribution $\mathcal{D}(1^\lambda)$ using a single round of interaction. A natural application of these constructions is the distributed generation of (structured or unstructured) common reference strings in one round. In [ASY22], the authors showed that distributed samplers can also be used to build public-key PCFs [OSY21, ASY22], a primitive producing large amounts of correlated randomness with minimal communication and a single round of interaction.

Known constructions. The notion of distributed sampler was introduced for the first time in [ASY22]. In their work, Abram, Scholl and Yakoubov showed how to build distributed samplers for any efficiently samplable distribution $\mathcal{D}(1^\lambda)$ using strong cryptographic primitives such as polynomially secure indistinguishability obfuscation [BGI$^+$01, GGH$^+$13] and a weaker form of multi-key FHE called multiparty homomorphic encryption (MHE) [AJJM20, MW16]. The authors achieved constructions in the plain model with security against non-rushing semi-malicious adversaries[4], statically corrupting up to $n-1$ parties. In such setting, distributed samplers were defined as one-round protocols that implement the functionality that generates a sample from the distribution $\mathcal{D}(1^\lambda)$ and outputs it to all the parties.

The authors focussed on active security too. They managed to upgrade their constructions to this setting, unfortunately, at the price of relying on random oracles. Active distributed samplers were defined as one-round protocols that implement the functionality $\mathcal{F}_\mathcal{D}$ (see Figure 4.12) in the UC model. Such functionality provides the adversary with a polynomial number of samples from $\mathcal{D}(1^\lambda)$ and lets it choose the one it likes the most as the final output of the protocol. Although $\mathcal{F}_\mathcal{D}$ allows influence to the adversary, the functionality is strong enough to generate CRSs for MPC protocols without compromising security.

Known impossibilities. A recent work by Abram, Obremski and Scholl [AOS23] proved that, without random oracle, it is impossible to build non-trivial active distributed samplers satisfying the definition of [ASY22]. Actually, the impossibility holds even if we try to achieve security against *rushing*, semi-malicious adversaries.

Abram, Obremski and Scholl started by showing that active distributed samplers always need common reference strings. Then, they proved that such CRSs cannot be reused more than once, they cannot be significantly shorter than the Yao entropy of the distribution $\mathsf{H}_{\mathsf{Yao}}(\mathcal{D})$ (they can be at most $O(\log \lambda)$ bits

[4]Similarly to the semi-honest case, a semi-malicious adversary is forced to follow the protocol, but it can choose the randomness tapes of the corrupted players as it prefers. Since the adversary is non-rushing, the choice of the randomness must be taken at the beginning of the protocol, before the honest messages are delivered.

THE ACTIVE DISTRIBUTED SAMPLER FUNCTIONALITY $\mathcal{F}_{\mathcal{D}}$

Sample. On input Sample from the adversary, compute $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$ and reply with R and a unique identifier id. The adversary can query this procedure multiple times.

Output. On input $\widehat{\mathsf{id}}$ from the adversary, the functionality retrieves the sample with identifier $\widehat{\mathsf{id}}$ and outputs it to all honest parties. If such sample is not defined, the functionality aborts.

Figure 4.12: The functionality for active distributed samplers in [ASY22]

shorter) and they cannot be unstructured (unless $\mathcal{D}$ is obliviously samplable[5]). These results, which just assume the existence of OWFs, suggest that, without random oracles, active distributed samplers cannot improve upon the trivial construction in which we directly encode a sample from $\mathcal{D}(1^\lambda)$ in the CRS. In this work, we present how to get around these impossibilities by weakening the security definition of distributed sampler.

4.4 Almost Everywhere Extractable NIZKs

The main purpose of distributed sampler is to generate secure CRSs for multiparty computation protocols using a single round of interaction. As we discussed in the introduction, distributed samplers can be interesting if they rely on CRSs as long as the latter have nice properties such as reusability, short length and unstructuredness.

The distributed sampler we present in this paper will make use of particular NIZKs that, if instantiated with constructions from previous works, would compromise the reusability of the CRS. In this section, we formalise the security properties we require from these primitives. Furthermore, we explain how to realise our definitions obtaining short and unstructured CRSs that do not compromise reusability.

Performing extractions inside obfuscated programs.

We describe the context in which we would like to use our NIZKs. We start from a NIZK satisfying black-box straight-line extraction. We consider an obfuscated program C_0 that receives a NIZK proof π among its input, verifies it and, based on the result, either outputs $\perp$ (when the verification fails) or performs other operations. We would like to argue that this obfuscated circuit is indistinguishable from another obfuscated circuit C_1 that has an extraction trapdoor hardcoded. When C_1 receives π as input, it not only verifies the proof, but it also tries to extract the corresponding witness. If any of the procedures fails, C_1 outputs $\perp$, otherwise, it performs the same operations as C_0.

Since it is hard for the adversary to come up with a proof that verifies but cannot be extracted, one could hope to prove indistinguishability between C_0 and C_1 using obfuscation. Unfortunately, we cannot rely on iO as, due to zero-knowledge, C_0 and C_1 will always have differing inputs. Specifically, we know that simulated proofs exist, verify, but cannot be extracted, so they immediately lead to differing inputs.

The only way to avoid this problem is to rely on constructions in which the CRS only allows simulating proofs for a fixed set of statements S having polynomial size $p(\lambda)$. This idea was for instance used in [HIJ+17]. With this trick, we could augment the extraction trapdoor with a list of witnesses for the statements in S, so the extraction from simulated proof will never fail. This solution, however, has the disadvantage of letting the CRS grow with $p(\lambda)$. That would make the CRS of our distributed sampler long and would hinder reusability.

Differing-input obfuscation would solve our problems. We consider diO. This primitive guarantees the hardness in distinguishing between the obfuscation of two circuits as long as differing inputs are hard to

[5]A distribution is obliviously samplable if given a sample R from $\mathcal{D}(1^\lambda)$, we can simulate the randomness that produced R. In other words, we can securely generate samples but directly feeding public random coins into $\mathcal{D}(1^\lambda)$.

find. Although the existence of general-purpose diO for circuits is often doubted [GGHW14, BSW16], we know that, for some classes of circuits, indistinguishability obfuscators are also differing-input obfuscators. In particular, in [BCP14], Boyle, Chung and Pass proved that this is the case when the number of differing inputs is polynomial. By relying on subexponential secure obfuscation, it is easy to generalise the result of [BCP14] as follows.

Lemma 4.4.1 ([BCP14]). Let Samp be a probabilistic algorithm outputting two circuits C_0, C_1 with input space $\{0,1\}^{m(\lambda)}$, auxiliary information aux and a secret ρ. Let $\mathcal{O}$ be another probabilistic algorithm that on input a pair (ρ, x) outputs a value y. Suppose the following

- there exist efficiently computable values $\ell_0(C_0, C_1, \mathsf{aux}), \ell_1(C_0, C_1, \mathsf{aux})$ and $d(\lambda)$ (the latter potentially superpolynomial) such that

$$\Pr\left[\left|\mathsf{DI}^{\ell_0,\ell_1}_{C_0,C_1}\right| \leq d(\lambda) \middle| (C_0, C_1, \mathsf{aux}, \rho) \xleftarrow{\$} \mathsf{Samp}(1^\lambda)\right] = 1 - \mathsf{negl}(\lambda),$$

  where $\mathsf{DI}^{\ell_0,\ell_1}_{C_0,C_1} := \left\{\mathsf{Trunc}^{\ell_1}_{\ell_0}(x) \middle| C_0(x) \neq C_1(x)\right\}$.

- for every probabilistic adversary $\mathcal{A} \in \mathsf{AClass}$ running in $\mathsf{poly}(\lambda, d(\lambda))$ time,

$$\Pr\left[y \in \mathsf{DI}^{\ell_0,\ell_1}_{C_0,C_1} \middle| \begin{array}{l} (C_0, C_1, \mathsf{aux}, \rho) \xleftarrow{\$} \mathsf{Samp}(1^\lambda) \\ y \xleftarrow{\$} \mathcal{A}^{\mathcal{O}(\rho,\cdot)}(1^\lambda, 1^{d(\lambda)}, C_0, C_1, \mathsf{aux}) \end{array}\right] = \mathsf{negl}(\lambda).$$

Let iO be an indistinguishability obfuscator against which every PPT adversary has advantage at most $\mathsf{negl}(\lambda)/d(\lambda)$. Then, for every PPT adversary $\mathcal{A} \in \mathsf{AClass}$ we have

$$\left|\Pr\left[\mathcal{A}^{\mathcal{O}(\rho,\cdot)}(1^\lambda, C_0, C_1, \widetilde{C}, \mathsf{aux}) = b \middle| \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ (C_0, C_1, \mathsf{aux}, \rho) \xleftarrow{\$} \mathsf{Samp}(1^\lambda) \\ \widetilde{C} \xleftarrow{\$} \mathsf{iO}(1^\lambda, C_b) \end{array}\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda).$$

Sketch of the proof. The proof follows the blueprint of [BCP14][Theorem 6.2], in which they convert a successful PPT distinguisher $\mathcal{A}$ into a successful extractor for (prefixes of) differing inputs. The only differences is that now, $\mathcal{A}$ has unbounded access to $\mathcal{O}(\rho, \cdot)$ and we are looking for substrings of differing-inputs so the binary search will involve only the bits of the inputs in between position $\ell_0(C_0, C_1, \mathsf{aux})$ and $\ell_1(C_0, C_1, \mathsf{aux})$. Furthermore, the parameter $d(\lambda)$ is potentially superpolynomial.

Observe that the extractor of [BCP14][Lemma 6.3] is uniform if $\mathcal{A}$ is uniform. Notice indeed that the extractor does not need to know the advantage $\epsilon(\lambda)$ of $\mathcal{A}$ [6], but only the polynomial $p(\lambda)$ such that, for every $\lambda' \in \mathbb{N}$, there exists $\lambda'' \geq \lambda'$ such that $\epsilon(\lambda'') \geq 1/p(\lambda'')$. Moreover, it runs in time is at most $d^3(\lambda) \cdot \mathsf{poly}(\lambda)$. Finally, it still outputs a prefix of differing-input with non-negligible probability. □

Our goal will be to build particular NIZKs that will allow us to apply the above lemma. More in detail, we want that the prefix of all proofs that verify but cannot be extracted lies in a set VPFE whose elements are hard to compute even for adversaries running in time $d(\lambda) := |\mathsf{VPFE}|$. If we succeed in doing this, assuming the subexponential hardness of iO, we can argue that the obfuscation of C_0 and C_1 are indistinguishable despite the existence of differing inputs. We call the NIZK satisfying this particular property *almost everywhere extractable NIZKs.*

Almost everywhere extractable NIZKs.

In this section, we formalise the properties of the NIZK needed by our distributed samplers. We recall here the definition of identity-based NIZK [KOR05]. Informally, this is a primitive in which both the proving and the verification algorithms are augmented with an input id denoting an identity. Completeness is guaranteed only if the algorithms use the same id.

[6]This could be impossible when the extractor is uniform

Definition 4.4.2 (Identity-based NIZK). Let $\mathcal{R}$ be an NP relation. An identity-based NIZK for $\mathcal{R}$ is a triple of uniform PPT algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax

- Setup is randomised and takes as input the security parameter and outputs a CRS σ.
- Prove is randomised and takes as input the security parameter, the CRS σ, an identity id, a statement x and the corresponding witness w. The output is a proof π.
- Verify is deterministic and takes as input the CRS σ, an identity id, a proof π and a statement x. The output is a bit b representing whether the statement was accepted or not.

We require that the construction satisfies completeness, namely that there exists a negligible function $\mathsf{negl}(\lambda)$ such that, for every $(x, w) \in \mathcal{R}$ and identity id,

$$\Pr\left[\mathsf{Verify}(\sigma, \mathsf{id}, \pi, x) = 1 \middle| \begin{array}{l} \sigma \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ \pi \xleftarrow{\$} \mathsf{Prove}(1^\lambda, \sigma, \mathsf{id}, x, w) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Why do we need identity-based NIZKs? Almost everywhere extractable NIZKs will be identity-based. We recall that our goal is to design NIZKs for which it is difficult to distinguish an obfuscated program that simply verifies the provided proofs and one that instead tries to extract the witnesses. Now, in the security proofs of many applications, e.g. our distributed samplers, the adversary will be given many simulated proofs. In general, these are proofs where extraction fails although the verification succeeds! If we feed any of these proofs to the obfuscated programs, we can trivially discover if the circuit tries to extract the witness or not (in the first case, the output will always be $\perp$).

Identity-based NIZKs allow us to find a way around the problem: we modify the programs so that they will only accept proofs that verify with respect to specific hardcoded identities. If the identities of the simulated proofs differ from the hardcoded ones, the behaviour of the program on input these simulated proofs will be independent of whether extraction if performed or not. In order words, we are using identities to restrict the scope of the proofs.

Alternative approaches. There are two ways we can proceed towards our goal. The first one is to achieve a stronger form of simulation-extractability: forging a valid proof where extraction fails must be hard even if we provide simulated proofs for different identities. Although we use this approach in [AWZ23, Section 10] to build almost everywhere extractable NIZKs with security against uniform adversaries, in this section, we adopt a different solution: we strengthen the notion of zero-knowledge. In particular, the extraction will take place in two steps: first, from the general extraction trapdoor and the identity associated with the proof, we derive an identity-specific trapdoor. Then, we use the latter to extract the witness. We require that zero-knowledge holds even if we leak identity-specific extraction trapdoors where the underlying identities differ from those of the simulated proofs. The obfuscated programs will contain only extraction trapdoors associated with their hardcoded identities.

The two approaches lead to different proving strategies. If we rely on simulation-extractability, the security proof of our application will first consider the hybrid in which the NIZKs of the honest parties are simulated and then we switch to obfuscated programs that try to extract witnesses. If we strengthen zero-knowledge, we will do the opposite: first, we switch to programs that extract the witnesses and then we simulate the proofs of the honest players. The results are equivalent.

In this section, we decided to follow the second approach as it allows us to achieve our goal under weaker assumptions. Following the blueprint in [AWZ23, Section 10], it would also have been possible to adopt the first approach, however, that would require assuming the existence of $B(\lambda)$-bounded labelled one-way functions (that do not need to be challengeless) that are secure against adversaries running in $\mathsf{poly}\big(2^{\lambda^e}, B(\lambda)\big)$ for some $e > 0$.

Defining almost-everywhere extractability. We proceed by formalising the definition of almost everywhere extractable NIZK. The construction relies on two trapdoors τ_s and τ_e, the first one will be used to simulate the proofs, the second one will be used to extract the witnesses. The extraction is divided into two procedures: given a proof π with underlying identity id, we first derive the extraction trapdoor associated with id using $\tau_e^{\mathsf{id}} \xleftarrow{\$} \mathsf{Trap}(\tau_e, \mathsf{id})$. Next, we extract the witness from π using τ_e^{id}. It is straightforward to see that an almost everywhere extractable NIZK is also a non-interactive argument of knowledge.

Definition 4.4.3 (Almost everywhere extractable NIZK). An identity-based NIZK for the NP relation $\mathcal{R}$ is *almost everywhere extractable* if there exists a uniform PPT algorithms $\mathsf{SimSetup}$, Trap and $\mathsf{Extract}$ with the following properties

- No PPT adversary can distinguish between

$$\left\{\sigma \middle| \sigma \xleftarrow{\$} \mathsf{Setup}(1^\lambda)\right\} \qquad \left\{\sigma \middle| (\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimSetup}(1^\lambda)\right\}$$

- The algorithm $\mathsf{Extract}$ is deterministic and, for every $w = \mathsf{Extract}(\tau_e^{\mathsf{id}}, \pi, x)$,

$$\Pr\left[(x, w) \in \mathcal{R} \middle| w \neq \bot\right] = 1.$$

- There exist values $\ell(\lambda) \in [m]$ and $d(\lambda)$ (the latter potentially superpolynomial) and a negligible function $\mathsf{negl}(\lambda)$ such that, for every identity id,

$$\Pr\left[\left|\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}\right| \leq d(\lambda) \middle| (\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{NIZK.SimSetup}(1^\lambda)\right] \geq 1 - \mathsf{negl}(\lambda),$$

  where

$$\mathsf{VPFE}_{\sigma,\tau_e\,\mathsf{id}} := \left\{ \mathsf{Trunc}_\ell(\pi) \middle| \exists (x, r) \text{ s.t. } \begin{array}{r} \mathsf{NIZK.Verify}(\sigma, \mathsf{id}, \pi, x) = 1 \\ \mathsf{NIZK.Trap}(\tau_e, \mathsf{id}; r) = \tau_e^{\mathsf{id}} \\ \mathsf{NIZK.Extract}(\tau_e^{\mathsf{id}}, \pi, x) = \bot \end{array} \right\}$$

- For every probabilistic adversary $\mathcal{A}$ running in $\mathsf{poly}(\lambda, d(\lambda))$ time, there exits a negligible function $\mathsf{negl}(\lambda)$ such that, for every identity id

$$\Pr\left[y \in \mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}} \middle| \begin{array}{l} (\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{NIZK.SimSetup}(1^\lambda) \\ y \xleftarrow{\$} \mathcal{A}(1^\lambda, 1^{d(\lambda)}, \sigma, \tau_e) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

We prove below that, if we use almost everywhere extractable NIZKs and we rely on a subexponentially secure iO scheme, the obfuscation of the programs C_0 and C_1 are indistinguishable.

Lemma 4.4.4. Let NIZK be an almost everywhere extractable NIZK for the relation $\mathcal{R}$. Let $d(\lambda)$ be the upper-bound on $\left|\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}\right|$. Suppose that iO is an indistinguishability obfuscator against which every PPT adversary has advantage at most $\mathsf{negl}(\lambda)/d(\lambda)$. Then, no PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ can win the game in Figure 4.13 with non-negligible advantage.

Proof. Let $\mathcal{A}$ be a PPT adversary. We proceed by means of $m+1$ subhybrids indexed by $i = 0, 1, \ldots, m$. In the i-th of these hybrids, we provide $\mathcal{A}$ with an obfuscation of the program C'_i (see Figure 4.16).

Observe that by the security of iO, when $i = 0$, Hybrid i is indistinguishable from the game in Figure 4.13 when $b = 0$. Similarly, by the security of iO, when $i = m$, Hybrid i is indistinguishable from from the game in Figure 4.13 when $b = 1$. It remains to prove that $\mathcal{A}$ cannot distinguish between Hybrid $i-1$ and Hybrid i for any $i \in [m]$. We rely on Lemma 4.4.1.

We consider the circuit sampler Samp_i that runs $\mathsf{SimSetup}$, provides σ and τ_e to $\mathcal{A}_1$, obtains $C, (\mathsf{id}_j)_{j\in[m]}, \psi$, compute τ_e^j for every $j \in [m]$ and outputs C'_{i-1}, C'_i, $\mathsf{aux} := \psi$ and $\rho := \bot$. Let $\mathcal{O}$ be an algorithm that always returns the empty string. We want to argue that even when aux is revealed, no PPT adversary can distinguish between the obfuscation of C'_{i-1} and C'_i.

diO Game for Almost Everywhere Extractable NIZKs

1. $b \xleftarrow{\$} \{0,1\}$
2. $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimSetup}(1^\lambda)$
3. $\big(C, (\mathsf{id}_j)_{j\in[m]}, \psi\big) \xleftarrow{\$} \mathcal{A}_1(1^\lambda, \sigma, \tau_e)$
4. $\forall j \in [m] : \quad \tau_e^j \xleftarrow{\$} \mathsf{Trap}(\tau_e, \mathsf{id}_j)$
5. $\widetilde{C}_0 \xleftarrow{\$} \mathsf{iO}(1^\lambda, C_0[\sigma, (\mathsf{id}_j)_{j\in[m]}])$ (see Figure 4.14)
6. $\widetilde{C}_1 \xleftarrow{\$} \mathsf{iO}(1^\lambda, C_1[\sigma, (\mathsf{id}_j)_{j\in[m]}, (\tau_e^j)_{j\in[m]}])$ (see Figure 4.15)
7. The adversary wins if $\mathcal{A}_2(\psi, \widetilde{C}_b) = b$.

Figure 4.13: diO game for almost everywhere extractable NIZKs

$C_0\big[\sigma, (\mathsf{id}_j)_{j\in[m]}\big]$

Hard-coded. The NIZK CRS σ, the m identities $(\mathsf{id}_j)_{j\in[m]}$.
Input. A set of inputs $(x_j)_{j\in[m]}$ and a set of proofs $(\pi_j)_{j\in[m]}$.

1. $\forall j \in [m] : \quad b_j \leftarrow \mathsf{NIZK.Verify}(\sigma, \mathsf{id}_j, \pi_j, x_j)$
2. If $\exists j \in [m]$ such that $b_j = 0$, output $\perp$
3. Output $C(x_1, \ldots, x_m)$

Figure 4.14: The circuit C_0

Let $\ell(\lambda)$ and $d(\lambda)$ be the values used in the third and fourth property of our almost everywhere extractable NIZK. Let $\ell_0(\lambda)$ denote the position of the first bit of π_i. Define $\ell_1(\lambda) := \ell_0(\lambda) + \ell(\lambda)$.

The circuits C'_{i-1} and C'_i potentially have differing inputs. Observe that these must be values $(x_j, \pi_j)_{j\in[m]}$ for which $\mathsf{Verify}(\sigma, \mathsf{id}_i, \pi_i, x_i) = 1$ but $\mathsf{Extract}(\tau_e^i, \pi_i, x_i) = \perp$. In other words, we know that for every differing input,

$$\mathsf{DI}^{\ell_0,\ell_1}_{C'_{i-1},C'_i} \subseteq \mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}_i}.$$

With overwhelming probability over the randomness of $\mathsf{SimSetup}$, the latter has at most $d(\lambda)$ elements.

Now, suppose that there exists an adversary $\mathcal{B}$ running in time $\mathsf{poly}\big(\lambda, d(\lambda)\big)$ that can find an element in $\mathsf{DI}^{\ell_0,\ell_1}_{C'_{i-1},C'_i}$ with non-negligible probability given C'_{i-1}, C'_i and $\mathsf{aux} = \psi$. We build an adversary $\mathcal{B}'$ that breaks the fourth property of the almost everywhere extractable NIZK.

The adversary $\mathcal{B}'$ runs an internal copy of $\mathcal{A}_1$ and one of $\mathcal{B}$. It starts by providing the NIZK CRS σ and the trapdoor τ_e it received from its challenger to $\mathcal{A}_1$ obtaining C and $(\mathsf{id}_j)_{j\in[m]}$. Then, for every $j \in [m]$, $\mathcal{B}'$ computes $\tau_e^j \xleftarrow{\$} \mathsf{Trap}(\tau_e, \mathsf{id}_j)$. Finally, it provides $\mathcal{B}$ with C'_{i-1}, C'_i and $\mathsf{aux} := \psi$ and outputs whatever $\mathcal{B}$ outputs. We observe that $\mathcal{B}'$ outputs an element in $\mathsf{DI}^{\ell_0,\ell_1}_{C'_{i-1},C'_i}$ with non-negligible probability. Furthermore, it runs in $\mathsf{poly}\big(\lambda, d(\lambda)\big)$ time.

We conclude that $\mathcal{A}_2$ cannot distinguish between the obfuscation of C'_{i-1} and C'_i even if it is given ψ. This ends the proof. $\square$ $\square$

$C_1\left[\sigma, (\mathsf{id}_j)_{j\in[m]}, (\tau_e^j)_{j\in[m]}\right]$

Hard-coded. The NIZK CRS σ, the m identities $(\mathsf{id}_j)_{j\in[m]}$, the m extraction trapdoors $(\tau_e^j)_{j\in[m]}$.
Input. A set of inputs $(x_j)_{j\in[m]}$ and a set of proofs $(\pi_j)_{j\in[m]}$.

```
1. $\forall j \in [m]: \quad b_j \leftarrow \mathsf{NIZK.Verify}(\sigma, \mathsf{id}_j, \pi_j, x_j)$
2. $\forall j \in [m]: \quad w_j \leftarrow \mathsf{NIZK.Extract}(\tau_e^j, \pi_j, x_j)$
3. If $\exists j \in [m]$ such that $b_j = 0$ or $w_j = \perp$, output $\perp$
4. Output $C(x_1, \ldots, x_m)$
```

Figure 4.15: The circuit C_1

$C_i'\left[i, \sigma, (\mathsf{id}_j)_{j\in[m]}, (\tau_e^j)_{j\le i}\right]$

Hard-coded. The hybrid index i, the NIZK CRS σ where $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimSetup}(1^\lambda)$, the m identities $(\mathsf{id}_j)_{j\in[m]}$, the i extraction trapdoors $(\tau_e^j)_{j\le i}$ where $\tau_e^j \xleftarrow{\$} \mathsf{Trap}(\tau_e, \mathsf{id}_j)$ for every $j \le i$.
Input. A set of inputs $(x_j)_{j\in[m]}$ and a set of proofs $(\pi_j)_{j\in[m]}$.

```
1. $\forall j \le i: \quad w_j \leftarrow \mathsf{NIZK.Extract}(\tau_e^j, \pi_j, x_j)$
2. If $\exists j \le i$ such that $w_j = \perp$, output $\perp$
3. $\forall j \in [m]: \quad b_j \leftarrow \mathsf{NIZK.Verify}(\sigma, \mathsf{id}_j, \pi_j, x_j)$
4. If $\exists j \in [m]$ such that $b_j = 0$, output $\perp$
5. Output $C(x_1, \ldots, x_m)$
```

Figure 4.16: The circuit C_i'

Chosen-ID multi-theorem zero-knowledge.

We now focus on formalising a particular zero-knowledge notion for almost everywhere extractable NIZKs. We call the property *chosen-ID* zero-knowledge. Informally, it says that, as long as τ_e^{id} remains secret, it is impossible to distinguish between a real proof under the identity id and a simulated one produced using the trapdoor τ_s.

This is formalised by giving the adversary access to an oracle that either generates real proofs using witnesses or simulates them using τ_s. We also give access to a second oracle that, on input any identity id, reveals the extraction trapdoor τ_e^{id}. The adversary is allowed to perform multiple adaptive queries to both the oracles with the only restriction that, if an identity is queried to the first oracle, it cannot be queried to the second one and vice-versa. Even with this kind of help, the adversary should not be able to tell if it is given real proofs or fake ones.

Definition 4.4.5 (Chosen-ID Zero-knowledge NIZK). An almost everywhere extractable NIZK $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for $\mathcal{R}$ is chosen-ID zero-knowledge if there exists a uniform PPT algorithm $\mathsf{SimProve}$ such that no PPT adversary $\mathcal{A}$ can win the game in Figure 4.17 with non-negligible advantage.

Chosen-ID Zero-Knowledge Game

Initialisation: This procedure is run only once, at the beginning of the game.

1. $b \xleftarrow{\$} \{0,1\}$
2. $Q_0, Q_1 \leftarrow \emptyset$
3. $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimSetup}(1^\lambda)$
4. Activate the adversary with 1^λ and σ.

Trapdoor: This procedure can be queried multiple times and at any point of the game. Upon receiving any query $(\mathsf{Trap}, \mathsf{id})$ where $\mathsf{id} \notin Q_1$, compute the following.

1. Add id to Q_0
2. $\tau_e^{\mathsf{id}} \xleftarrow{\$} \mathsf{Trap}(\tau_e, \mathsf{id})$
3. Give τ_e^{id} to the adversary.

Prove: This procedure can be queried multiple times and at any point of the game. Upon receiving any query $(\mathsf{Prove}, \mathsf{id}, x, w)$ where $\mathsf{id} \notin Q_0$ and $(x, w) \in \mathcal{R}$, compute the following.

1. Add id to Q_1
2. $\pi^0 \xleftarrow{\$} \mathsf{Prove}(1^\lambda, \sigma, \mathsf{id}, x, w)$
3. $\pi^1 \xleftarrow{\$} \mathsf{SimProve}(\tau_s, \mathsf{id}, x)$
4. Give π^b to the adversary.

Win: The adversary wins if it guesses b.

Figure 4.17: Chosen-ID zero-knowledge game

4.4.1 Building almost everywhere extractable NIZKs

We explain how to build a chosen-ID zero-knowledge, almost everywhere extractable NIZK with security against non-uniform adversaries.

Our construction relies on an identity based encryption scheme, a non-interactive commitment scheme, a subexponentially secure injective one-way function and a NIWI proof. The CRS will consist of the IBE master public key and a challenge v for a one-way function. It is possible to instantiate the primitives so that the CRS is short (i.e. the length depends only on the security parameter) and unstructured.

Let $\mathcal{R}$ be the NP relation we are targetting, suppose that we want to prove that $x \in L_{\mathcal{R}}$ using the identity id. The proof is obtained by encrypting the witness w under the identity id using the IBE scheme. We also commit to 0. Then, we generate a NIWI proving that either the ciphertext is an encryption of w under id, or we committed to the preimage of v. The NIZK proof consists of the concatenation of the commitment, the ciphertext and the NIWI. The verification is a simple check of the latter.

Observe that it is easy to extract the witness by decrypting the ciphertext. Of course, the operation requires knowing the private key associated with the identity id. The latter can be derived from the master secret key of the IBE scheme. Even simulating proofs is rather easy: it is sufficient to encrypt 0, commit to a preimage of v and use the latter as witness for the NIWI. To summarise, the extraction trapdoor will be the master secret key, the soundness trapdoor will be the preimage of v.

Ensuring almost-everywhere extractability. Our idea is that, in all proofs where the witness cannot be extracted, the commitment will hide a preimage of v. In order to ensure this, we will rely on a perfectly correct IBE scheme (if the ciphertext hides the witness, we always succeed in extracting it) and a perfectly sound NIWI (if the ciphertext does not hide a witness, the commitment must hide a preimage of v). Since the one-way function is injective, there will be at most $2^{q(\lambda)}$ ways of committing to a preimage of v. Here, $q(\lambda)$ denotes the length of the randomness used by the commitment[7].

Now, suppose that the commitment is perfectly binding and it is possible to break hiding in $\mathsf{poly}\big(\lambda, 2^{q(\lambda)}\big)$ time. By choosing a sufficiently large security parameter for the one-way function, we can make sure that finding the preimage of v is hard even for adversaries running in $\mathsf{poly}\big(\lambda, 2^{q(\lambda)}\big)$ time. That ensures the last property of almost everywhere extractable NIZKs.

Proving chosen-ID zero-knowledge is instead rather easy. We just rely on witness-indistinguishability, the hiding properties of the commitment and the IND-ID-CPA security of IBE. Notice that a message encrypted under the identity id remains secret as long as the secret-key for id is kept private. Leaking private keys for other identities does not help in retrieving the plaintext.

Formalising the construction. Let $\mathcal{R}$ the NP relation for our almost everywhere extractable NIZK. Consider an IND-ID-CPA identity-based encryption scheme IBE where the master public key mpk is computationally indistinguishable from a uniformly random string. We also require that the scheme satisfies perfect correctness. For instance, we can use the constructions of [BB04, ABB10].

Let Com be a computationally hiding, perfectly binding non-interactive commitment scheme without CRS. Suppose that there exists an algorithm running in superpolynomial time that breaks hiding with probability 1. Finally, let OWF be a subexponentially secure injective one-way function. Furthermore, assume that the one-way function outputs values that are computationally indistinguishable from a uniformly random string. This kind of one-way function can be instantiated e.g. using DLOG.

Finally, we rely on a NIWI scheme without CRS. The underlying relation is the following.

$$\mathcal{R}_{\mathsf{NIWI}} := \left\{ \begin{array}{c|c} \big((\mathsf{mpk}, v, \mathsf{id}, c_0, c_1, x), & c_1 = \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, w;\, r), \quad (x, w) \in \mathcal{R} \\ & \text{OR} \\ (w, r)\big) & w = u, \quad c_0 = \mathsf{Com}(u;\, r), \quad \mathsf{OWF}(1^\lambda, u) = v \end{array} \right\}$$

Our construction is formalised in Figure 4.18 and Figure 4.19.

Theorem 4.4.6. Suppose that Com is a computationally hiding, perfectly binding non-interactive commitment. Assume that the algorithm needs $q_2(\lambda)$ bits of randomness. Suppose that there exists an algorithm running in $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time that breaks the hiding property of Com with probability 1.

Let IBE be an IND-ID-CPA identity-based encryption scheme that satisfies perfect correctness. Let OWF be an injective one-way function that is hard to invert even for adversaries running in $\mathsf{poly}\big(\lambda, 2^{q_2(\lambda)}, S(\lambda)\big)$ time. Suppose that NIWI is a perfectly sound witness-indistinguishable proof system for the relation $\mathcal{R}_{\mathsf{NIWI}}$.

Then, the construction in Figure 4.18 and Figure 4.19 is a chosen-ID zero-knowledge almost everywhere extractable NIZK for $\mathcal{R}$ against non-uniform PPT adversaries.

Proof. Completeness is an immediate consequence of the completeness of NIWI.

Claim 4.4.7. The construction in Figure 4.18 and Figure 4.19 is an almost everywhere extractable NIZK.

Proof of the claim. We start by observing that the CRS σ generated by $\mathsf{Setup}(1^\lambda)$ has exactly the same distribution as the one generated by $\mathsf{SimSetup}(1^\lambda)$. The second property is also straightforward.

Therefore, we focus on the third property of the almost everywhere extractable NIZKs. Let $\ell(\lambda)$ denote the position of the last bit of c_0. We observe that by the perfect soundness of NIWI and the perfect correctness of IBE, in all proofs that verify but cannot be extracted, c_0 is a commitment to a preimage of v. Since the OWF is injective, there exists a unique preimage. Furthermore, the commitment algorithm takes as input $q_2(\lambda)$ bits of randomness. We conclude that, with probability 1, $\mathsf{VPFE}_{\sigma, \tau_e, \mathsf{id}}$ contains at most $2^{q_2(\lambda)}$ elements.

[7]We can assume that $q(\lambda)$ is independent of the length of the committed value. Consider for instance a scheme in which we commit the message bit by bit and all the randomness comes from a PRF.

```
CHOSEN-ID ZERO-KNOWLEDGE, ALMOST EVERYWHERE EXTRACTABLE NIZK - PART 1
Let $q_1(\lambda)$ denote the length of the randomness needed by $\mathsf{IBE.Enc}$.
$\mathsf{Setup}(1^\lambda)$
    1. $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{IBE.Setup}(1^\lambda)$
    2. $(v, u) \xleftarrow{\$} \mathsf{OWF.Gen}(1^\lambda)$
    3. Output $\sigma := (\mathsf{mpk}, v)$
$\mathsf{Prove}\big(1^\lambda, \sigma = (\mathsf{mpk}, v), \mathsf{id}, x, w\big)$
    1. $c_0 \xleftarrow{\$} \mathsf{Com}(1^\lambda, 0)$
    2. $r \xleftarrow{\$} \{0,1\}^{q_1(\lambda)}$
    3. $c_1 \leftarrow \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{id}, w;\, r)$
    4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(1^\lambda, (\mathsf{mpk}, v, \mathsf{id}, c_0, c_1, x), (w, r)\big)$
    5. Output $\pi := (c_0, c_1, \pi')$
$\mathsf{Verify}\big(\sigma = (\mathsf{mpk}, v), \mathsf{id}, \pi = (c_0, c_1, \pi'), x\big)$
    1. Output $\mathsf{NIWI.Verify}\big(\pi', (\mathsf{mpk}, v, \mathsf{id}, c_0, c_1, x)\big)$
```

Figure 4.18: A chosen-ID zero-knowledge, almost everywhere extractable NIZK - Part 1

Now, suppose that an adversary $\mathcal{B}$ running in time $\mathsf{poly}\big(\lambda, 2^{q_2(\lambda)}\big)$ can find an element in $\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}$ with non-negligible probability after being provided with σ and τ_e. We can use this adversary to break the subexponential one-wayness of OWF. Indeed, consider the adversary that after receiving v from its challenger, generates $(\mathsf{mpk}, \mathsf{msk})$ using $\mathsf{IBE.Setup}$ and provides the pair $\sigma = (\mathsf{mpk}, v)$, $\tau_e = \mathsf{msk}$ to $\mathcal{B}$. When the latter replies with c_0, the new adversary retrieves the value hidden in c_0, breaking the hiding property of the commitment. The total running time of $\mathsf{poly}\big(\lambda, 2^{q_2(\lambda)}, S(\lambda)\big)$ and, with non-negligible probability, due to the perfectly binding property of the commitment scheme, the output is the value u such that $\mathsf{OWF}(1^\lambda, u) = v$. We reached a contradiction. This ends the proof of the claim. ■

Claim 4.4.8. The construction in Figure 4.18 and Figure 4.19 satisfies chosen-ID zero-knowledge.

Proof of the claim. We prove the result by means of a sequence of indistinguishable hybrids. We repeat it for $i = 0, 1, \ldots, M$ where M is a polynomial upper bound on the number of Prove queries issued by the adversary (since $\mathcal{A}$ is PPT, M exists). Throughout the proof, for any i, let π_i denote the proof provided to the adversary in the i-th Prove query. Let us denote the identity, the statement and the witness of the latter by id_i, x_i and w_i respectively.

Hybrid i.0. In this hybrid, for every $j \leq i$, we generate the proof π_j using $\mathsf{SimProve}(\tau_s, \mathsf{id}_j, x_j)$. For every $j > i$ instead, we generate the proof π_j using $\mathsf{Prove}(1^\lambda, \sigma, \mathsf{id}_j, x_j, w_j)$. When $i = 0$, this hybrid corresponds to the execution of the chosen-ID zero-knowledge game (see Figure 4.17) with $b = 0$. In particular, the proofs are all generated using the witnesses for $\mathcal{R}$. In all other cases, this hybrid is identical to Hybrid $(i-1)$.3.

Hybrid i.1. In this hybrid, in the i-th proof, instead of committing to 0, we commit to $\tau_s = u$. This hybrid is indistinguishable from the previous one by the hiding property of the commitment scheme. Formally, the operations to generate π_i are the following. All other proofs are generated as in the previous hybrid.

1. $c_0 \xleftarrow{\$} \mathsf{Com}(1^\lambda, u)$

Chosen-ID Zero-Knowledge, Almost Everywhere Extractable NIZK - Part 2

Let $q_2(\lambda)$ denote the length of the randomness needed by Com.

$\mathsf{SimSetup}(1^\lambda)$

1. $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{IBE.Setup}(1^\lambda)$
2. $(v, u) \xleftarrow{\$} \mathsf{OWF.Gen}(1^\lambda)$
3. Output $\sigma := (\mathsf{mpk}, v), \tau_s := u, \tau_e := \mathsf{msk}$

$\mathsf{SimProve}(\tau_s = u, \mathsf{id}, x)$

1. $r \xleftarrow{\$} \{0,1\}^{q_2(\lambda)}$
2. $c_0 \leftarrow \mathsf{Com}(1^\lambda, u; r)$
3. $c_1 \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{id}, 0)$
4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(1^\lambda, (\mathsf{mpk}, v, \mathsf{id}, c_0, c_1, x), (u, r)\big)$
5. Output $\pi := (c_0, c_1, \pi')$

$\mathsf{Trap}(\tau_e = \mathsf{msk}, \mathsf{id})$

1. Output $\mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{id})$

$\mathsf{Extract}(\tau_e^{\mathsf{id}}, \pi = (c_0, c_1, \pi'), x)$

1. $w \leftarrow \mathsf{IBE.Dec}(\tau_e^{\mathsf{id}}, c_1)$
2. If $(x, w) \in \mathcal{R}$, output w, otherwise, output $\perp$.

Figure 4.19: A chosen-ID zero-knowledge, almost everywhere extractable NIZK - Part 2

2. $r \xleftarrow{\$} \{0,1\}^{q_1(\lambda)}$
3. $c_1 \leftarrow \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{id}_i, w_i; r)$
4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(1^\lambda, (\mathsf{mpk}, v, \mathsf{id}_i, c_0, c_1, x_i), (w_i, r)\big)$
5. Output $\pi_i := (c_0, c_1, \pi')$

Hybrid i.2. In this hybrid, instead of using w_i and the randomness used to generate c_1 as witness for NIWI, we use u and the randomness used for c_0. By the witness indistinguishability of NIWI, this hybrid is indistinguishable from the previous one. Formally, the operations to generate π_i are the following. All other proofs are generated as in the previous hybrid.

1. $r \xleftarrow{\$} \{0,1\}^{q_2(\lambda)}$
2. $c_0 \leftarrow \mathsf{Com}(1^\lambda, u; r)$
3. $c_1 \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{id}_i, w_i)$
4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(1^\lambda, (\mathsf{mpk}, v, \mathsf{id}_i, c_0, c_1, x_i), (u, r)\big)$
5. Output $\pi_i := (c_0, c_1, \pi')$

Hybrid i.3. In this hybrid, instead of encrypting w_i using IBE, we encrypt 0. This hybrid is indistinguishable from the previous one by the IND-ID-CPA security of IBE. Notice indeed, that we never provide the adversary with τ_e nor with $\tau_e^{\mathsf{id}_i}$. Formally, the operations to generate π_i are the following. All other proofs are generated as in the previous hybrid.

1. $r \xleftarrow{\$} \{0,1\}^{q_2(\lambda)}$
2. $c_0 \leftarrow \mathsf{Com}(1^\lambda, u; r)$
3. $c_1 \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{id}_i, 0)$
4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(1^\lambda, (\mathsf{mpk}, v, \mathsf{id}_i, c_0, c_1, x_i), (u, r)\big)$
5. Output $\pi_i := (c_0, c_1, \pi')$

Notice that when $i = M$, the last hybrid is identical to the to the execution of the chosen-ID zero-knowledge game (see Figure 4.17) with $b = 1$. This ends the proof of the claim. ■

□

4.5 Weakening Distributed Samplers to Avoid Random Oracles

In this section, we reformulate the concept of distributed sampler under a new light. Although we weaken the simulation-based definition of [ASY22], we obtain a meaningful notion of security against active adversaries. This allows us to build constructions that overcome the impossibilities of [AOS23] without using random oracles.

Syntax of Distributed Samplers.

We start by recalling the syntax of distributed samplers [ASY22].

Definition 4.5.1 (Distributed Sampler). An n-party distributed sampler is a triple of uniform, PPT algorithms $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample})$ with the following syntax:

- Setup is a probabilistic algorithm taking as input the security parameter. The output is a string crs.
- Gen is a probabilistic algorithm taking as input the security parameter, a session identity sid, the index $i \in [n]$ of the party running the algorithm and the string crs. The output is the distributed sampler message U_i of the i-th party.
- Sample is a deterministic algorithm taking as input n distributed sampler messages $U_1, U_2, \dots, U_n$, a session identity sid and the string crs. The output is a sample R.

Observe that distributed samplers are implicitly associated with a one-round protocol with CRS (the latter is generated using $\mathsf{Setup}(1^\lambda)$) producing a sample from a target distribution $\mathcal{D}$. In such protocol, all the parties P_i simultaneously broadcast a distributed sampler message $U_i \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \mathsf{sid}, i, \mathsf{crs})$. After that, everybody retrieves the output $R \leftarrow \mathsf{Sample}(U_1, U_2, \dots, U_n, \mathsf{sid}, \mathsf{crs})$.

Notice that, compared to [ASY22], we augmented the generation and sampling algorithms with a session identity. The latter can be used to restrict the context in which the distributed sampler messages can be used. For instance, it can identify the identities of the parties taking part to the protocol. If the session identity of any of the exchanged messages does not match the expected set of parties, the sampling algorithm will produce $\perp$.

4.5.1 Hardness-Preserving Distributed Samplers.

We now present the first weakening of the original definition. The notion is called *hardness-preserving distributed sampler*. The name refers to the fact that this kind of distributed sampler allows compiling protocols with CRS Π into protocols without CRS Π' while preserving the hardness properties: if the probability of realising an attack against Π is negligible, the probability of realising the same attack against Π' still remains negligible.

An unusual definition of security. Our definition is based on a real-world/ideal-world paradigm where simulation is non-black-box. In the real world, the adversary is provided with a distributed sampler CRS and the message of a honest party. After selecting the distributed sampler messages of the other parties, the adversary is provided with the output of the protocol (notice that the adversary was already able to compute this on its own). In the ideal world, instead, the CRS and the message of the honest party are produced by a simulator. The latter is given an ideal sample $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$. When the adversary answers with the distributed sampler messages of the other players, we do not compute the output of the protocol, we just provide the adversary with R.

The important point is that we do not ask for indistinguishability between the real-world and the ideal world. That would indeed be impossible to achieve. We ask instead that if an adversary $\mathcal{A}$ outputs 1 with non-negligible probability while interacting with the real world, then, $\mathcal{A}$ outputs 1 with non-negligible probability even while interacting with the ideal world.

Definition 4.5.2 (Hardness-Preserving Distributed Sampler). Let $\mathcal{D}(1^\lambda)$ be an efficient distribution. We say that an n-party distributed sampler is hardness-preserving for $\mathcal{D}(1^\lambda)$ against AClass if, for every PPT $\mathcal{A} \in \mathsf{AClass}$, there exists a pair of PPT non-uniform simulators $(\mathsf{SimSetup}_\mathcal{A}, \mathsf{SimGen}_\mathcal{A})$ such that, in the game $\mathcal{G}_{\mathsf{HP}}$ in Figure 4.20,

$$\Pr\left[\mathcal{G}_{\mathsf{HP}}^{\mathcal{A}}(1^\lambda) = 1 \middle| b = 0\right] = \mathsf{nonegl}(\lambda) \implies \Pr\left[\mathcal{G}_{\mathsf{HP}}^{\mathcal{A}}(1^\lambda) = 1 \middle| b = 1\right] = \mathsf{nonegl}(\lambda).$$

Preservation of hardness.

We now explain in what sense distributed samplers satisfying Definition 4.5.2 preserve hardness.

We start by formalising the concept of *game with oracle distribution*. This basically corresponds to a game describing the interaction between n parties connected by authenticated point-to-point channels and a broadcast medium. The adversary has full control over the corrupted players, whereas the operations of the honest parties is managed by the challenger of the game. The novelty compared to the a standard game is that, at some point in time, the parties are all provided with the same ideal sample R from a distribution $\mathcal{D}(1^\lambda)$. The moment in which the sample is delivered is chosen by the parties themselves: by sending a special message (Sample, i), the i-th party declares its approval on delivering R. When all the honest parties expressed their agreement, the sample R is provided to the adversary. When all the corrupted parties agree too, the sample R is given to the challenger too. The adversary wins the game if the challenger terminates its execution outputting 1. For instance, this can mean that the adversary succeeded in performing an attack. We define the advantage as the probability of this event.

Definition 4.5.3 (Game with oracle distribution). An n-party game with oracle distribution is a triple $\mathcal{G} := (\mathcal{D}, \mathsf{Ch})$ where

1. $\mathcal{D}(1^\lambda)$ is an efficient distribution: a uniform, PPT algorithm taking only the security parameter as input.

2. Ch is an efficient challenger: a uniform, PPT, round-based, interactive Turing machine that, for every $i \in [n]$, sends the message (Sample, i) at most once in its execution.

THE HARDNESS-PRESERVING GAME $\mathcal{G}_{\mathsf{HP}}$

Each phase is run only once.

Initialisation Phase:

1. $b \xleftarrow{\$} \{0,1\}$
2. $R^{(1)} \xleftarrow{\$} \mathcal{D}(1^\lambda)$
3. $\mathsf{crs}^{(0)} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$
4. $(\mathsf{crs}^{(1)}, \zeta) \xleftarrow{\$} \mathsf{SimSetup}_{\mathcal{A}}(1^\lambda)$
5. Activate $\mathcal{A}$ with 1^λ and $\mathsf{crs}^{(b)}$.

Generation Phase:

1. Receive $i \in [n]$ and a session identity $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \ldots, \mathsf{id}_{j_n})$ from $\mathcal{A}$
2. $U^{(0)} \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \mathsf{sid}, i, \mathsf{crs}^{(0)})$
3. $U^{(1)} \xleftarrow{\$} \mathsf{SimGen}_{\mathcal{A}}(1^\lambda, \mathsf{sid}, i, \zeta, R^{(1)})$
4. $U_i \leftarrow U^{(b)}$
5. Provide U_i to $\mathcal{A}$

Sampling Phase

1. Receive $(U_j)_{j \neq i}$ from $\mathcal{A}$
2. $R^{(0)} \leftarrow \mathsf{Sample}(U_1, \ldots, U_n, \mathsf{crs}^{(0)})$
3. If $R^{(0)} = \perp$, output 0.
4. Otherwise, provide $R := R^{(b)}$ to $\mathcal{A}$
5. The output of the game is the bit output by $\mathcal{A}$.

Figure 4.20: The hardness-preserving game $\mathcal{G}_{\mathsf{HP}}$

Let $\mathcal{A}$ be a round-based interactive Turing machine. We define $\mathcal{G}_{\mathcal{A}}(1^\lambda)$ to be the output of the game in Figure 4.21.

For every adversary $\mathcal{A}$, we define the advantage of $\mathcal{A}$ in the game $\mathcal{G}$ as

$$\mathsf{Adv}^{\mathcal{G}}_{\mathcal{A}}(\lambda) := \Pr\Big[\mathcal{G}_{\mathcal{A}}(1^\lambda) = 1\Big].$$

We say that $\mathcal{A}$ wins with non-negligible advantage if $\mathsf{Adv}^{\mathcal{G}}_{\mathcal{A}}(\lambda)$ is non-negligible in the security parameter.

Notice that at the beginning of the game in Definition 4.5.3, the adversary is allowed to choose the set of honest parties H and an auxiliary input aux for Ch. In other words, our definition considers only static corruption in the dishonest majority setting.

On the expressiveness of the model. Protocols relying on CRSs can be formulated as games with oracle distribution. In such settings, $\mathcal{D}(1^\lambda)$ represents the distribution from which the CRS is generated. Since the CRS should be given before the beginning of the protocol, in the corresponding game with oracle distribution, the challenger immediately starts by sending (Sample, i) for every $i \in H$. It then waits for

Game with Oracle Distribution

1. Activate the adversary $\mathcal{A}$ with 1^λ.
2. Receive aux, $H \subseteq [n]$ from $\mathcal{A}$.
3. $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$
4. Activate a copy of Ch with 1^λ, H and aux.
5. Relay all the messages from Ch to $\mathcal{A}$.
6. Relay all the messages from $\mathcal{A}$ to Ch.
7. After the challenger has sent (Sample, j) for every $j \in H$, provide $\mathcal{A}$ with R.
8. Only when the above occurred, after $\mathcal{A}$ has sent (Sample, j) for every $j \notin H$, provide R to Ch.
9. Keep relaying the messages between $\mathcal{A}$ and Ch as before.
10. The output of the game is the value output by Ch before halting.

Figure 4.21: Game with oracle distribution

analogous messages from the corrupted players, ignoring all other communication. After that, the challenger runs the protocol with the adversary on behalf of the honest parties.

More in general, games with oracle distributions can be used to analyse the security of protocols that rely on a sampling resource: a functionality that, upon receiving the approval of all players, delivers an ideal sample from a fixed distribution $\mathcal{D}(1^\lambda)$. The sample is leaked to the adversary in advance, at the moment in which all the honest players send their approval.

Compiling games with oracle distributions using distributed samplers. Using a distributed sampler for $\mathcal{D}$, there is a natural way to compile a game with oracle distribution $\mathcal{G} = (\mathcal{D}, \mathsf{Ch})$ into a standard interactive game. The delivery of a special message (Sample, i) in $\mathcal{G}$ will correspond to the delivery of a distributed sampler message U_i from party P_i. The sample R used by the challenger Ch will be the output of the distributed sampler. If the output is $R = \perp$, the challenger always halts outputting 0.

Observe that, as in the game with oracle distribution, the adversary can learn the sample R as soon as all the honest parties deliver their distributed sampler messages. Indeed, the adversary may have already chosen the distributed sampler messages of the corrupted players without revealing them. The honest players (i.e. the challenger) will discover R only when the adversary decides to deliver these messages.

Notice that in the case of a protocol with CRS Π, the compiled game consists of the sequential composition of a distributed sampler with Π, where the former is used to generate the CRS for the latter.

Multi-session security. Distributed samplers sometimes make use of a CRS. We would like the latter to be reusable among multiple sessions involving different subsets of parties. Since all these executions are correlated by the use of the same CRS, the security analysis of the compiled game cannot restrict to single sessions. For this reason, upon activation, we provide the adversary with the distributed sampler CRS and we let it choose the identities of a large number $m > n$ of players that will constitute our universe. At the same time, it also selects the set of honest players H. At that point, the adversary is free to engage in many, possibly simultaneous sessions of the compiled game, all using the same distributed sampler CRS. Each session takes place between n parties chosen by the adversary. The session is uniquely identified by a session label sid consisting of the identities of the n parties and an additional label tag that acts like a counter. Thanks to the latter, it will be possible to have multiple sessions among the same subset of parties. For each session, the adversary is also allowed to choose a different auxiliary input aux. We define the advantage of the adversary as the probability that, in one of the sessions, the challenger outputs 1.

Definition 4.5.4 (Compiled game). Let $\mathcal{G} = (\mathcal{D}, \mathsf{Ch})$ be an n-party game with oracle distribution and let $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample})$ be an n-party distributed sampler. We define the compiled game $\mathcal{G}'$ in Figure 4.22. For any PPT adversary $\mathcal{A}$, we denote the output of the game by $\mathcal{G}'_{\mathcal{A}}(1^\lambda)$. We denote the value output by $\mathcal{A}$ before halting by $\mathcal{A}_{\mathcal{G}'}(1^\lambda)$.

We define the advantage of $\mathcal{A}$ in the game $\mathcal{G}'$ as

$$\mathsf{Adv}^{\mathcal{G}'}_{\mathcal{A}}(\lambda) := \Pr\Big[\mathcal{G}'_{\mathcal{A}}(1^\lambda) = 1\Big].$$

We say that $\mathcal{A}$ wins with non-negligible advantage if $\mathsf{Adv}^{\mathcal{G}'}_{\mathcal{A}}(\lambda)$ is non-negligible in the security parameter.

In the next theorem, we show that if the distributed sampler is hardness-preserving, hard-to-win games with oracle distribution are compiled into standard games that are still hard to win. In other words, if we have a protocol with CRS Π for which all PPT adversaries fail in performing an attack, the attack remains hard to perform even against the compiled protocol Π'.

Theorem 4.5.5. Let $\mathcal{G} := (\mathcal{D}, \mathsf{Ch})$ be an n-party game with oracle distribution such that every PPT adversary $\mathcal{A}$ has negligible advantage against $\mathcal{G}$. Let $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample})$ be an n-party distributed sampler. If DS is hardness-preserving for $\mathcal{D}$ against AClass, there exists no PPT $\mathcal{A}' \in \mathsf{AClass}$ such that $\mathsf{Adv}^{\mathcal{G}'}_{\mathcal{A}'}(\lambda)$ is non-negligible.

The idea at the base of the proof is rather simple. Suppose that an adversary $\mathcal{A}'$ can win against the compiled game with non-negligible advantage. That means that, if we pick a session at random, the session output is 1 with non-negligible probability. Now, we build an the adversary $\mathcal{B}$ against the hardness-preserving property of the distributed sampler. The latter picks a random session ι of the compiled game and simulates it to $\mathcal{A}'$ using the values provided by its challenger. In particular, $\mathcal{B}$ is given the CRS crs, the honest distributed sampler message that is sent for last and the distributed sampler output. The adversary $\mathcal{B}$ halts outputting the outcome of the ι-th session.

In the real-world execution of the distributed sampler, $\mathcal{B}$ outputs 1 with non-negligible probability, so, by the hardness-preserving properties, the same must happen in the ideal-world execution. In the latter, however, in $\mathcal{B}$'s simulation of the ι-th session, the challenger is given an ideal sample from $\mathcal{D}(1^\lambda)$ instead of the actual distributed sampler output. From this, we can easily build a PPT adversary $\mathcal{A}$ that wins against $\mathcal{G}$ with non-negligible advantage.

Proof. Suppose that our game is false and there exists a PPT adversary $\mathcal{A}' \in \mathsf{AClass}$ such that $\mathsf{Adv}^{\mathcal{G}'}_{\mathcal{A}'}(\lambda)$ is non-negligible. Let $M(\lambda)$ be a polynomial upper-bounding the number of $\mathsf{NewSession}$ queries issued by $\mathcal{A}$.

We construct a PPT adversary $\mathcal{B} \in \mathsf{AClass}$ for the hardness-preserving game such that

$$\Pr\left[\mathcal{G}^{\mathcal{B}}_{\mathsf{HP}}(1^\lambda) = 1 \middle| b = 0\right] = \mathsf{nonegl}(\lambda). \tag{4.1}$$

The adversary $\mathcal{B}$ starts its execution by selecting a random value $\iota \xleftarrow{\$} [M]$. Then, it uses the value crs given by its challenger to simulate $\mathcal{G}'$ to an internal copy of $\mathcal{A}'$. It behaves slightly differently in the ι-th $\mathsf{NewSession}$ query. Specifically, let (Sample, i) be the last special message sent by Ch in that session. Instead of providing a distributed sampler message generated using $\mathsf{DS.Gen}$, the adversary $\mathcal{B}$ queries its challenger with i and $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$. It provides the adversary with the answer U_i. Moreover, after all the distributed sampler messages $(U_j)_{j \in [n]}$ have been exchanged, $\mathcal{B}$ does not compute the sample R using $\mathsf{DS.Sample}$, but queries its challenger with $(U_j)_{j \neq i}$. It gives the answer to Ch. All the rest remains as in Figure 4.22. The final output of $\mathcal{B}$ corresponds to the output of the ι-th session.

We observe that if the bit b in the hardness-preserving game is set to 0, the view of $\mathcal{A}'$ in $\mathcal{G}'$ coincides with the one in $\mathcal{B}$'s simulation. So,

$$\Pr\left[\mathcal{G}^{\mathcal{B}}_{\mathsf{HP}}(1^\lambda) = 1 \middle| b = 0\right] \geq \frac{1}{M(\lambda)} \cdot \Pr\left[\mathcal{G}'_{\mathcal{A}'}(1^\lambda) = 1\right].$$

Compiled Game with Oracle Distribution

Initialisation: This procedure is run only once, at the beginning of the game.

1. $\mathsf{crs} \overset{\$}{\leftarrow} \mathsf{DS.Setup}(1^\lambda)$
2. Activate the adversary $\mathcal{A}$ with 1^λ and crs.
3. Receive a list of identities of the parties $\mathsf{ID} := \{\mathsf{id}_1, \ldots, \mathsf{id}_m\}$ from $\mathcal{A}$ along with the subset of honest players $H \subseteq [m]$.

Session: This procedure can be queried multiple times and at any point of the game. Upon receiving any query $(\mathsf{NewSession}, \mathsf{tag}, \mathsf{id}_{j_1}, \ldots, \mathsf{id}_{j_n}, \mathsf{aux})$ where the session identity $\mathsf{sid} := (\mathsf{tag}, \mathsf{id}_{j_1}, \ldots, \mathsf{id}_{j_n})$ has not been queried before, $\mathsf{id}_{j_i} \in \mathsf{ID}$ for every $i \in [n]$, $\mathsf{id}_{j_l} \neq \mathsf{id}_{j_k}$ for every $l \neq k$ and aux, perform the following.

1. Store sid
2. $\forall i \in [n]$ such that $j_i \in H$: $\quad U_i \overset{\$}{\leftarrow} \mathsf{DS.Gen}(1^\lambda, \mathsf{sid}, i, \mathsf{crs})$
3. Activate a new copy of Ch with 1^λ, $H' := \{i \in [n] | j_i \in H\}$ and aux.
4. Relay all the messages from Ch to $\mathcal{A}$ appending sid to them.
5. Relay all the messages from $\mathcal{A}$ with prefix sid to Ch (the prefix is removed).
6. When Ch sends (Sample, i) for any $i \in H'$, provide $\mathcal{A}$ with $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)$.
7. When $\mathcal{A}$ sends $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)$ for any $i \notin H'$, give (Sample, i) to Ch.
8. When all the messages $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)_{i \in [n]}$ have been exchanged, provide Ch with $R \leftarrow \mathsf{DS.Sample}(U_1, \ldots, U_n, \mathsf{sid}, \mathsf{crs})$.
9. Keep relaying the messages between Ch and $\mathcal{A}$ as before.
10. The output of the session is the value output by Ch before halting. If $R = \perp$, the output of the session is 0.

Output: In the game, multiple sessions are run in parallel. The output of the game is 1 if there exists a session that terminates with 1.

Figure 4.22: Compiled game with oracle distribution

The latter is non-negligible. Notice also that since the challenger of $\mathcal{G}'$ is uniform and PPT, $\mathcal{B}$ still belongs to AClass. We have just proven equation (4.1).

By the hardness-preserving property of DS, we know that there exists a pair of PPT algorithms $(\mathsf{SimSetup}_{\mathcal{B}}, \mathsf{SimGen}_{\mathcal{B}})$ such that

$$\Pr\left[\mathcal{G}^{\mathcal{B}}_{\mathsf{HP}}(1^\lambda) = 1 \middle| b = 1\right] = \mathsf{nonegl}(\lambda). \tag{4.2}$$

We can finally build a PPT adversary $\mathcal{A}$ that wins the game $\mathcal{G}$ with non-negligible advantage. The adversary $\mathcal{A}$ runs an internal copy of $\mathcal{A}'$. It starts its execution by sampling $\iota \xleftarrow{\$} [M]$ and running $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{SimSetup}_{\mathcal{B}}(1^\lambda)$. Then, it simulates the game $\mathcal{G}'$ to $\mathcal{A}'$ using crs as CRS for the distributed sampler. The simulation of the game takes place as in Figure 4.22 with the exception of the ι-th session. Let $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$ be the corresponding session identity and aux the corresponding auxiliary input. The adversary $\mathcal{A}$ provides its challenger with aux and the set of honest players $\{i \in [n] | j_i \in H\}$. Then, it relays the messages between $\mathcal{A}'$ and Ch. When $\mathcal{A}$ receives (Sample, i) where $i \in H$ from its challenger, it generates a distributed sampler message U_i and sends it to $\mathcal{A}'$. The operations is always performed using $\mathsf{DS.Gen}$ except for the last honest player. In that case, $\mathcal{A}$ receives an ideal sample R from its challenger, so, it generates U_i using

$$U_i \xleftarrow{\$} \mathsf{DS.SimGen}_{\mathcal{B}}(1^\lambda, \mathsf{sid}, i, \zeta, R).$$

When $\mathcal{A}'$ sends a distributed sample message in the ι-th session on behalf of a corrupted party id_{j_i}, $\mathcal{A}$ sends (Sample, i) to Ch. The adversary $\mathcal{A}$ terminates its execution when $\mathcal{A}'$ does.

We observe that

$$\Pr\left[\mathcal{G}_{\mathcal{A}}(1^\lambda) = 1\right] = \Pr\left[\mathcal{G}^{\mathcal{B}}_{\mathsf{HP}}(1^\lambda) = 1 \middle| b = 1\right] = \mathsf{nonegl}(\lambda).$$

$\square$ $\square$

4.5.2 Indistinguishability Preserving Distributed Samplers

Hardness-preserving distributed samplers guarantee a somewhat limited form a security: they are just meant to preserve the hardness of computations. In other words, if we have two indistinguishable games relying on a CRS, a hardness-preserving distributed sampler does not guarantee that the compiled games are still indistinguishable.

More concretely, suppose that we deal with the security proof of a protocol Π relying on a CRS R. That means that there exists a simulator $\mathcal{S}$ such that Π is indistinguishable from the interaction between $\mathcal{S}$ and a functionality $\mathcal{F}$. A hardness-preserving distributed sampler does not guarantee that the compiled protocol Π' still implements the functionality $\mathcal{F}$. Indeed, how can we simulate the distributed sampler messages sent in Π'? Notice that in its simulation, $\mathcal{S}$ might rely on a trapdoored version of the CRS R. It can be that the outputs of the hardness-preserving distributed sampler never have a trapdoor. Furthermore, even if the trapdoor existed, how would $\mathcal{S}$ retrieve it?

We need our distributed sampler to satisfy additional properties. For this reason, we introduce the notion of *indistinguishability-preserving distributed sampler*. They will guarantee that, under some conditions, if a protocol Π relying on a CRS implements a functionality $\mathcal{F}$ against an active adversary in the UC model, the compiled protocol still implements $\mathcal{F}$. As for the hardness-preserving case, indistinguishability-preserving distributed samplers overcome the impossibilities of [AOS23]. They can therefore be built without using random oracles.

Roadmap for the definition. In order to formalise the definition of indistinguishability-preserving distributed sampler, we need to introduce preliminary concepts. We will define a trapdoored version of games with oracle distribution. This notion is meant to model the behaviour of a simulator that hides trapdoors in the CRSs it produces. In a game with trapdoor oracle distribution, the ideal sample given to the parties hides a trapdoor T. The latter is revealed only to the challenger simultaneously with R. We then define indistinguishability between a game with oracle distribution and a game with trapdoored oracle distribution. Finally, we define indistinguishability-preserving distributed samplers as distributed samplers that compile games with oracle distribution and games with trapdoored oracle distribution preserving indistinguishability.

Games with trapdoored oracle distribution. We introduce the concept of *trapdoored distribution*. Essentially, the latter consists of a distribution $\mathcal{D}'$ that outputs samples R along with trapdoors T. The trapdoor distribution $\mathcal{D}'$ can also be given an auxiliary input aux' of fixed length. The notion is formalised with respect to another (standard) distribution $\mathcal{D}$. We require that for every value aux', the sample R generated by $\mathcal{D}'$ is indistinguishable from the one generated by $\mathcal{D}$.

Definition 4.5.6 (Trapdoored distribution). Let $\mathcal{D}(1^\lambda)$ be an efficient distribution. A trapdoored distribution for $\mathcal{D}$ is a uniform, PPT algorithm $\mathcal{D}'$ which takes as input the security parameter 1^λ and auxiliary information $\mathsf{aux}' \in \{0,1\}^{\ell(\lambda)}$ where $\ell(\lambda)$ is a fixed polynomial. The outputs are a sample R and a trapdoor T. We also require that, for every auxiliary input $\mathsf{aux}' \in \{0,1\}^{\ell(\lambda)}$, the following distributions are indistinguishable

$$\left\{R \middle| R \xleftarrow{\$} \mathcal{D}(1^\lambda)\right\} \qquad \text{and} \qquad \left\{R \middle| (R,T) \xleftarrow{\$} \mathcal{D}'(1^\lambda, \mathsf{aux}')\right\}.$$

Trapdoored distributions are meant to represent the distributions used by simulators of MPC protocols. The auxiliary input aux' can be used to represent any information that the simulator receives from the functionality such as public inputs. It may happen indeed that the simulated CRS depends on these. Examples of this kind are statistically-sound simulation extractable NIZKs [HIJ+17], in which the CRS for a simulated proof is a commitment to the statement.

We formalise the notion of *game with trapdoored oracle distribution*. The concept is similar to the one in Definition 4.5.3. The difference is that now we deal with a trapdoored distribution $\mathcal{D}'$.

Definition 4.5.7 (Game with trapdoored oracle distribution). An n-party game with trapdoored oracle distribution is a triple $\mathcal{G} := (\mathcal{D}', \mathsf{Ch})$ where

1. $\mathcal{D}'$ is a trapdoored distribution.

2. Ch is an efficient challenger: a uniform, PPT, round-based, interactive Turing machine that, for every $i \in [n]$, sends the message (Sample, i) at most once in its execution.

Indistinguishability-preserving distributed samplers will be compatible only with a particular class of games with trapdoored oracle distribution. The interaction between the adversary and the challenger will be analogous to the one in Figure 4.21 with the difference that when the challenger receives the sample R, it may also obtain the corresponding trapdoor T. The adversary instead never receives T. The choice of the auxiliary input aux' given to $\mathcal{D}'$ is made by the challenger when R is given to the adversary. We say that the game satisfies trapdoor security if it is impossible for the adversary to tell if the trapdoor was given to the challenger or not. If the first case, we say that the game is in trapdoor mode, otherwise, we say that the game is in no-trapdoor mode.

Definition 4.5.8 (Trapdoor security). Consider an n-party game with trapdoored oracle distribution $\mathcal{G} = (\mathcal{D}', \mathsf{Ch})$. We say that $\mathcal{G}$ satisfies *trapdoor security* if every PPT adversary $\mathcal{A}$ wins the game in Figure 4.23 with negligible advantage.

Why do we need the above property? Trapdoor security ensures that, independently on whether the trapdoor will be provided, the challenger will be able to conclude its execution obtaining indistinguishable outcomes. Indistinguishability-preserving distributed samplers will guarantee that, if an game with oracle distribution $\mathcal{G}_0 = (\mathcal{D}, \mathsf{Ch}_0)$ is indistinguishable from a game with trapdoored oracle distribution $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$, then, also the compiled games are indistinguishable. In the security proof of our construction, we will switch the challenger of the compiled games from Ch_0 to Ch_1, using the mode of operation in which no trapdoor is given. Then, we gradually modify the output of the distributed sampler, switching from $\mathcal{D}$ to the trapdoored version $\mathcal{D}'$. In other words, there will be some hybrids in which part of the outputs of the distributed sampler are trapdoored, whereas the rest is not. Since there will be no way to predict whether the adversary chooses a trapdoored sample or not, we need to make sure that before R is delivered to it, Ch_1 will not rely on the fact that a trapdoor will be given at some point. Trapdoor security guarantees this.

Trapdoor Security Game

1. $b \xleftarrow{\$} \{0,1\}$
2. Activate the adversary $\mathcal{A}$ with 1^λ.
3. Receive aux and $H \subseteq [n]$ from $\mathcal{A}$.
4. Activate a new copy of Ch with 1^λ, H and aux.
5. Relay all the messages from Ch to $\mathcal{A}$.
6. Relay all the messages from $\mathcal{A}$ to Ch.
7. After the Ch has sent (Sample, i) for every $i \in H$, receive $\mathsf{aux}' \in \{0,1\}^{\ell(\lambda)}$ from Ch, compute $(R, T) \xleftarrow{\$} \mathcal{D}'(1^\lambda, \mathsf{aux}')$ and provide $\mathcal{A}$ with R.
8. After the above occurred and after $\mathcal{A}$ has sent (Sample, i) for every $i \notin H$, provide Ch with R and, if $b = 1$, with T too.
9. Keep relaying the messages between $\mathcal{A}$ and Ch as before.

Win: The adversary wins if it guesses b.

Figure 4.23: Trapdoor security game

Trapdoorable distributed samplers and compiled games. We need to explain how to compile a game with trapdoored oracle distribution. We start by introducing the concept of *trapdoorable distributed sampler*.

Definition 4.5.9 (Trapdoorable distributed sampler). An n-party trapdoorable distributed sampler is a tuple of PPT algorithms $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{SimSetup}, \mathsf{SimGen}, \mathsf{Trap})$ where

1. $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample})$ is an n-party distributed sampler.
2. $\mathsf{SimSetup}(1^\lambda)$ is a PPT algorithm taking as input the security parameter. The output is a simulated CRS crs and the information ζ.
3. $\mathsf{SimGen}(1^\lambda, \mathsf{sid}, i, \zeta, \mathsf{aux}')$ is a PPT algorithm taking as input the security parameter, a session-identity, an index $i \in [n]$, the information ζ and aux'. The output is distributed sampler messages U_i and the trapdoor information ξ.
4. $\mathsf{Trap}\big(\xi, (U_i)_{i\in[n]}\big)$ is a deterministic algorithm taking as input the trapdoor information ξ and the distributed sampler messages $(U_i)_{i\in[n]}$. The output is a pair (R, T).

Essentially, a trapdoorable distributed sampler is a distributed sampler in which the CRS and the messages can be simulated in a way that the outputs will be sampled from a trapdoored distribution $\mathcal{D}'$ instead of $\mathcal{D}$. In other words, the samples will be equipped with trapdoors. The latter can be retrieved from the exchanged messages using the algorithm Trap. The auxiliary information aux' needed by $\mathcal{D}'$ will be hidden in the simulated messages. All the samples produced by the construction will use the same aux'.

We can finally explain how to compile a game with trapdoored oracle distribution using a trapdoorable distributed sampler. The idea is similar to the one explained in Definition 4.5.4. The main differences is that now, the distributed sampler CRS and the last message sent by a honest party in each session are simulated using $\mathsf{SimSetup}$ and SimGen. The auxiliary information input in SimGen will be the one provided by the challenger. When all the distributed sampler messages have been exchanged, we provide the challenger with a pair (R, T) generated using Trap.

Compiled Game with Trapdoored Oracle Distribution

Initialisation: This procedure is run only once, at the beginning of the game.

1. $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{DS.SimSetup}(1^\lambda)$
2. Activate the adversary $\mathcal{A}$ with 1^λ and crs.
3. Receive a list of parties $\mathsf{ID} := \{\mathsf{id}_1, \dots, \mathsf{id}_m\}$ from $\mathcal{A}$ along with the subset of honest players $H \subseteq [m]$.

Session: This procedure can be queried multiple times and at any point of the game. Upon receiving any query $(\mathsf{NewSession}, \mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n}, \mathsf{aux})$ where the session identity $\mathsf{sid} := (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$ has not been queried before, $\mathsf{id}_{j_i} \in \mathsf{ID}$ for every $i \in [n]$, $\mathsf{id}_{j_l} \neq \mathsf{id}_{j_k}$ for every $l \neq k$ and aux, perform the following.

1. Store sid
2. $\forall i$ s.t. $j_i \in H: \quad U_i \xleftarrow{\$} \mathsf{DS.Gen}(1^\lambda, \mathsf{sid}, i, \mathsf{crs})$
3. Activate a new copy of Ch with 1^λ, $H' := \{i \in [n] | j_i \in H\}$ and aux.
4. Relay all the messages from Ch to $\mathcal{A}$ appending sid to them.
5. Relay all the messages from $\mathcal{A}$ with prefix sid to Ch (the prefix is removed).
6. When Ch sends (Sample, i) for any $i \in H'$ except the last one left, provide $\mathcal{A}$ with $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)$.
7. When Ch sends (Sample, i) for the last $i \in H'$, obtain $\mathsf{aux}' \in \{0,1\}^{\ell(\lambda)}$ from Ch_1, compute $(U_i, \xi) \xleftarrow{\$} \mathsf{DS.SimGen}(1^\lambda, \mathsf{sid}, i, \zeta, \mathsf{aux}')$. Then, provide $\mathcal{A}$ with $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)$.
8. When $\mathcal{A}$ sends $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)$ for any $i \notin H'$, give (Sample, i) to Ch.
9. When all the messages $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)_{i \in [n]}$ have been exchanged, compute $(R, T) \leftarrow \mathsf{DS.Trap}\big(\xi, (U_j)_{j \in [n]}\big)$. Provide (R, T) to Ch.
10. Keep relaying the messages between Ch and $\mathcal{A}$ as before.

Figure 4.24: Compiled game with trapdoored oracle distribution

Definition 4.5.10 (Compiled game with trapdoored oracle distribution). Consider an n-party game with trapdoored oracle distribution $\mathcal{G} = (\mathcal{D}, \mathsf{Ch})$ and let $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{SimSetup}, \mathsf{SimGen}, \mathsf{Trap})$ be an n-party trapdoorable distributed sampler.

For any PPT adversary $\mathcal{A}$, we denote by $\mathcal{A}_{\mathcal{G}'}(1^\lambda)$ the value output by $\mathcal{A}$ at the end of the game in Figure 4.24.

Defining indistinguishability-preserving distributed samplers. Indistinguishability-preserving distributed samplers compile indistinguishable games with oracle distributions into standard indistinguishable games. We are interested in the case in which one of the games with oracle distribution is trapdoored.

We define *chosen-sample indistinguishability*. Essentially, the latter says that a game with oracle distribution $\mathcal{G}_0 = (\mathcal{D}, \mathsf{Ch}_0)$ is indistinguishable from a game with trapdoored oracle distribution $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$ if no PPT adversary $\mathcal{A}$ can tell the two apart even if the $\mathcal{A}$ is allowed to choose the sample R. The challenger Ch_1 is never provided with trapdoors.

Definition 4.5.11 (Chosen-sample Indistinguishable games with oracle distribution). Consider any pair

Chosen-Sample Indistinguishability for Games with Oracle Distribution

1. $b \xleftarrow{\$} \{0,1\}$
2. Activate the adversary $\mathcal{A}$ with 1^λ.
3. Receive aux and $H \subseteq [n]$ from $\mathcal{A}$.
4. Activate a copy of Ch_b with 1^λ, H and aux.
5. Relay all the messages from Ch_b to $\mathcal{A}$.
6. Relay all the messages from $\mathcal{A}$ to Ch_b.
7. After the Ch_b has sent (Sample, i) for every $i \in H$ and $\mathcal{A}$ has sent (Sample, i) for every $i \notin H$, let $\mathcal{A}$ choose R and provide it to Ch_b. Do not provide aux' to $\mathcal{A}$.
8. Keep relaying the messages between $\mathcal{A}$ and Ch_b as before.

Win: The adversary wins if it guesses b.

Figure 4.25: Chosen-sample indistinguishability for games with oracle distribution

$(\mathcal{G}_0, \mathcal{G}_1)$ where $\mathcal{G}_0 = (\mathcal{D}, \mathsf{Ch}_0)$ is a game with oracle distribution and $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$ is a game with trapdoored oracle distribution. We say that $\mathcal{G}_0$ and $\mathcal{G}_1$ are chosen-sample indistinguishable if every PPT adversary $\mathcal{A}$ wins the game in Figure 4.25 with negligible advantage.

The reason why we let the adversary choose R is the influence allowed in the compiled games. While in a game with oracle distribution the choice of the sample R is not affected by the adversary, in the compiled games, the adversary has always some influence. If we want the compiled games to be indistinguishable, it is important that the challengers Ch_0 and Ch_1 cannot be told apart, no matter how the adversary influences the choice of R.

We can finally define indistinguishability-preserving distributed samplers.

Definition 4.5.12 (Indistinguishability-preserving distributed sampler). Let $\mathcal{D}(1^\lambda)$ be an efficient distribution and let $\mathcal{D}'$ be a trapdoored distribution for $\mathcal{D}$. We say that an n-party trapdoorable distributed sampler is indistinguishability-preserving for $(\mathcal{D}, \mathcal{D}')$ against AClass if, for every PPT adversary $\mathcal{A} \in \mathsf{AClass}$ and for every pair $(\mathcal{G}_0, \mathcal{G}_1)$ of chosen-sample indistinguishable games where $\mathcal{G}_0 = (\mathcal{D}, \mathsf{Ch}_0)$ is a game with oracle distribution and $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$ is a game with trapdoored oracle distribution satisfying trapdoor security, we have

$$\left| \Pr[\mathcal{A}_{\mathcal{G}_0'}(1^\lambda) = 1] - \Pr[\mathcal{A}_{\mathcal{G}_1'}(1^\lambda) = 1] \right| = \mathsf{negl}(\lambda),$$

where $\mathcal{G}_0'$ and $\mathcal{G}_1'$ are the compiled games.

Applications of indistinguishability-preserving distributed samplers for protocol security.

We now show that, in most cases, indistinguishability-preserving distributed samplers can be used to remove CRSs in MPC protocols at the cost of one additional round of interaction while preserving simulation security. This holds in a context of active adversaries statically corrupting any number of the parties. Our theorem is formalised below.

Theorem 4.5.13. Assume the existence of authenticated point-to-point channels and a broadcast medium. Let Π be an n-party protocol implementing a PPT functionality $\mathcal{F}$ against active PPT adversaries in the UC model with static corruption. Suppose that Π relies on a CRS R generated according to the distribution $\mathcal{D}(1^\lambda)$. Let $\mathcal{S}$ be the corresponding PPT simulator.

Suppose that $\mathcal{S}$ can be regarded as the sequential composition of $\mathcal{S}_1$ and $\mathcal{S}_2$ where $\mathcal{S}_1$ never interacts with the functionality, generates a pair $(R, T) \xleftarrow{\$} \mathcal{D}'(1^\lambda)$ and provides the adversary with the simulated CRS R and $\mathcal{S}_2$ with (R, T).

Assume that $\mathcal{D}'$ is a trapdoored distribution for $\mathcal{D}$. Let DS be an n-party indistinguishability-preserving distributed sampler for $(\mathcal{D}, \mathcal{D}')$. Let Π' be the sequential composition of DS with Π. Then, Π' implements $\mathcal{F}$ against active PPT adversaries in the UC model with static corruption.

Observe that the round complexity of the protocol Π' has only increased by one. The idea at the base of the proof is rather immediate: the protocol Π can be reformulated as a game with oracle distribution $\mathcal{G}_0$. In the latter, the special messages are all exchanged at the beginning of the session. In a similar way, the simulation can be reformulated as a game with trapdoored oracle distribution $\mathcal{G}_1$ in which the auxiliary information given to $\mathcal{D}'$ is the empty string. To be precise, the simulation of Π corresponds to the trapdoor mode of $\mathcal{G}_1$, the no-trapdoor mode of $\mathcal{G}_1$ is instead identical to $\mathcal{G}_0$. Trapdoor security is an immediate consequence of the UC-security of Π. Chosen-sample indistinguishability is instead for free as $\mathcal{G}_0$ and the no-trapdoor mode of $\mathcal{G}_1$ are identical. That is enough to argue that the compiled games $\mathcal{G}_0'$ and $\mathcal{G}_1'$ are indistinguishable too. It is straightforward to notice that if we reformulate the compiled protocol Π' as a game, we obtain $\mathcal{G}_0'$. To terminate the proof, we notice that $\mathcal{G}_1'$ easily leads to a simulator $\mathcal{S}'$ for Π' and $\mathcal{F}$.

Proof. Let H be the set of honest parties. For every $i \in [n]$, let id_i denote the identity of the i-th party.

A single real-world execution of Π can be formulated as a n-party game with oracle distribution $\mathcal{G}_0$. In such game, the challenger Ch_0 immediately sends (Sample, i) for every $i \in H$. Then, it waits for the adversary to send (Sample, i) for every $i \notin H$. It ignores all other communications received before that. Then, Ch_0 runs the protocol Π with $\mathcal{A}$ on behalf of the honest parties.

In a similar way, a single ideal-world execution, can be rephrased as a n-party game with trapdoor oracle distribution $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$ where the challenger Ch_1 behaves as follows:

1. It immediately sends (Sample, i) for every $i \in H$, it sets aux' to be the empty string.
2. It waits for the adversary to send (Sample, i) for every $i \notin H$. It ignores all other communications received before that.
3. If it receives only a sample R, it executes Ch_0 providing it with R
4. It it receives a pair (R, T), it runs $\mathcal{S}_2$ along with $\mathcal{F}$.

By the UC security of Π, $\mathcal{G}_1$ satisfies trapdoor security. Moreover, it is immediate to see that the games $\mathcal{G}_0$ and $\mathcal{G}_1$ are perfectly chosen-sample indistinguishable.

Since DS is indistinguishability-preserving, the compiled games $\mathcal{G}_0'$ and $\mathcal{G}_1'$ are still indistinguishable. Observe that if we reformulate the real-world execution of Π', we obtain $\mathcal{G}_0'$.

We now consider the simulator $\mathcal{S}'$ that generates the distributed sampler CRS crs using $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{SimSetup}(1^\lambda)$. In every session $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$ of the protocol Π' where $\mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n}$ denote the identities of the parties involved, $\mathcal{S}'$ performs the following operations

1. pick i such that $j_i \in H$
2. $\forall l \neq i$ s.t. $j_l \in H: \quad U_l \xleftarrow{\$} \mathsf{DS.Gen}(1^\lambda, \mathsf{sid}, l, \mathsf{crs})$
3. $(U_i, \xi) \xleftarrow{\$} \mathsf{SimGen}(1^\lambda, \mathsf{sid}, i, \zeta)$
4. send $(U_l)_{j_l \in H}$ to the adversary on behalf of the honest parties
5. wait for $(U_l)_{j_l \notin H}$ from the adversary
6. $(R, T) \leftarrow \mathsf{Trap}\big(\xi, (U_l)_{l \in [n]}\big)$
7. run $\mathcal{S}_2(1^\lambda, R, T)$ interacting with the functionality $\mathcal{F}$ and the adversary.

Observe that if we reformulate the interaction between $\mathcal{F}$, $\mathcal{S}'$ and the adversary as a game, we obtain $\mathcal{G}_1'$. We conclude that no active PPT adversary can distinguish between Π' and the composition of $\mathcal{F}$ and $\mathcal{S}'$. This terminates the proof. □ □

In some cases, when the first round of interaction in Π is independent of the CRS, indistinguishability-preserving distributed samplers allow removing the CRS without affecting the round complexity. The result is formalised below.

Theorem 4.5.14. Assume the existence of authenticated point-to-point channels and a broadcast medium. Let Π be an n-party protocol implementing a PPT functionality $\mathcal{F}$ against active PPT adversaries in the UC model with static corruption. Let $\mathcal{S}$ be the corresponding PPT simulator. Suppose that Π can be rewritten as the sequential composition of a one-round protocol Π_1 with no CRS and a protocol Π_2 that relies on a CRS R generated according to the distribution $\mathcal{D}(1^\lambda)$.

Suppose that $\mathcal{S}$ can be regarded as the sequential composition of $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$ where:

- $\mathcal{S}_1$ never interacts with the functionality, generates values $(R, T) \xleftarrow{\$} \mathcal{D}'(1^\lambda)$ and provides the adversary with the simulated CRS R and $\mathcal{S}_3$ with (R, T).

- $\mathcal{S}_2$, never interacts with the functionality, generates the first-round messages of the honest parties using Π_1 and delivers them to the adversary. It passes its internal state to $\mathcal{S}_3$.

Assume that $\mathcal{D}'$ is a trapdoored distribution for $\mathcal{D}$. Let DS be an n-party indistinguishability-preserving distributed sampler for $(\mathcal{D}, \mathcal{D}')$. Let Π' be the composition of DS with Π where DS and Π_1 are run in parallel. Then, Π' implements $\mathcal{F}$ against active PPT adversaries in the UC model with static corruption.

The proof of Theorem 4.5.14 follows the blueprint of the proof of Theorem 4.5.13. Once again, we reformulate Π as a game with oracle distribution $\mathcal{G}_0$. This time the special messages are all sent simultaneously with the first round of communications. Since the simulator $\mathcal{S}$ generates the first round messages exactly as in Π, we can design a game with trapdoored oracle distribution $\mathcal{G}_1$ in which the trapdoor mode is a reformulation of the ideal world whereas the no-trapdoor mode is identical to $\mathcal{G}_0$. Trapdoor security is a consequence of the UC-security of Π, chosen-sample indistinguishability instead comes for free as before. The rest remains as in the proof of Theorem 4.5.13.

Proof. Let H be the set of honest parties. For every $i \in [n]$, let id_i denote the identity of the i-th party.

As before, a single real-world execution of Π can be formulated as a n-party game with oracle distribution $\mathcal{G}_0$. In such game, the challenger Ch_0 immediately sends (Sample, i) for every $i \in H$. Simultaneously, it sends the messages of the honest parties in protocol Π_1. Then, it waits for the adversary to send (Sample, i) for every $i \notin H$ along with the messages of the corrupted players in Π_1. Finally, Ch_0 runs the protocol Π_2 with $\mathcal{A}$ on behalf of the honest parties.

In a similar way, a single ideal-world execution, can be rephrased as a n-party game with trapdoor oracle distribution $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$ where the challenger Ch_1 behaves as follows:

1. It runs $\mathcal{S}_2$. The messages generated by $\mathcal{S}_2$ are delivered to the adversary in conjunction with (Sample, i) for every $i \in H$. The challenger Ch_1 also outputs the empty string aux'.

2. It waits for the adversary to send (Sample, i) for every $i \notin H$, along with the first-round messages of the corrupted parties.

3. If it receives only a sample R, it executes Π_2 on behalf of the honest parties using R as CRS.

4. It it receives a pair (R, T), it runs $\mathcal{S}_3$ along with $\mathcal{F}$. The simulator $\mathcal{S}_3$ is given (R, T) and the messages of the corrupted players in Π_1.

Notice that if Ch_1 receives R but not the trapdoor T, the view of the adversary is the same as in Π. So, by the UC security of Π, $\mathcal{G}_1$ satisfies trapdoor security. Moreover, it is immediate to see that the games $\mathcal{G}_0$ and $\mathcal{G}_1$ are perfectly chosen-sample indistinguishable.

Since DS is indistinguishability preserving, the compiled games $\mathcal{G}_0'$ and $\mathcal{G}_1'$ are still indistinguishable. Observe that if we reformulate the real-world execution of Π' as a game, we obtain $\mathcal{G}_0'$.

We now consider the simulator $\mathcal{S}'$ that generates the distributed sampler CRS crs using $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{SimSetup}(1^\lambda)$. In every session $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$ of the protocol Π' where $\mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n}$ denote the identities of the parties involved, $\mathcal{S}'$ performs the following operations

1. pick i such that $j_i \in H$
2. $\forall l \neq i$ s.t. $j_l \in H: \quad U_l \xleftarrow{\$} \mathsf{DS.Gen}(1^\lambda, \mathsf{sid}, l, \mathsf{crs})$
3. $(U_i, \xi) \xleftarrow{\$} \mathsf{SimGen}(1^\lambda, \mathsf{sid}, i, \zeta)$
4. generate the first-round messages of the honest parties in Π_1 following the protocol. Provide $\mathcal{S}_3$ with the view of the honest players.
5. send $(U_l)_{j_l \in H}$ to the adversary along with the messages generated in the previous step.
6. wait for $(U_l)_{j_l \notin H}$ and the corrupted player messages in Π_1 from the adversary
7. $(R, T) \leftarrow \mathsf{Trap}\big(\xi, (U_l)_{l \in [n]}\big)$
8. run $\mathcal{S}_3(1^\lambda, R, T)$ interacting with the functionality $\mathcal{F}$ and the adversary. $\mathcal{S}_3$ is also given the messages of the corrupted players in Π_1.

Observe that if we reformulate the interaction between $\mathcal{F}$, $\mathcal{S}'$ and the adversary as a game, we obtain $\mathcal{G}_1'$. We conclude that no active PPT adversary can distinguish between Π' and the composition of $\mathcal{F}$ and $\mathcal{S}'$. This terminates the proof. □ □

Generalisations. Sometimes, indistinguishability-preserving distributed samplers can be used to remove CRSs even from UC-secure protocols that satisfy neither of the hypothesis of Theorem 4.5.13 and Theorem 4.5.14. For instance, in some cases, we can let the simulated CRS depend on auxiliary information aux' provided by the functionality. In order for the proofs to go through, however, we need to ask that indistinguishability between real world and ideal world holds even when aux' is leaked to the adversary.

Theorem 4.5.14 can also be generalised in the sense that the simulator $\mathcal{S}$ does not strictly need to follow the protocol in the first round. The important thing, indeed, is to be able to successfully terminate the simulation even if $\mathcal{S}_1$ abruptly refuses to provide the trapdoor T and instead provides a sample R chosen by the adversary ($\mathcal{S}$ can even ask the functionality $\mathcal{F}$ to reveal its internal state when that happens). That would ensure chosen-sample indistinguishability.

The limits of indistinguishability-preserving. Although indistinguishability-preserving distributed samplers allow removing CRSs from a broad range of UC secure protocols, we know that there exist constructions for which this fails. One example in the protocol $\Pi_\mathcal{D}$ in which, after being provided with a CRS R sampled according to $\mathcal{D}(1^\lambda)$, all the parties output R. This protocol trivially implements the functionality $\mathcal{F}_\mathcal{D}$ that generates a sample from $\mathcal{D}(1^\lambda)$ and provides it to all the parties. If indistinguishability-preserving distributed samplers worked for this case we would obtain a distributed sampler for $\mathcal{D}$ satisfying the simulation-based definition of [ASY22]. We know that this is impossible [ASY22, AOS23].

4.6 Lossy Distributed Samplers

In this section, we introduce a new variant of distributed sampler called *lossy distributed samplers*. On their own, lossy distributed samplers are not sufficient to achieve hardness or indistinguishability preservation. However, they are a useful stepping stone towards our goal.

The construction of [ASY22] and its problems with rushing adversaries. In [ASY22], Abram, Scholl and Yakoubov presented a distributed sampler achieving security against semi-malicious non-rushing adversaries in the UC model. In other words, the protocol implements the ideal functionality that provides all the parties with a random sample from $\mathcal{D}(1^\lambda)$. The construction does not rely on random oracles nor CRSs.

There is a property that allows all this: output programming. Specifically, given any distributed sampler messages $(\hat{U}_j)_{j\notin H}$ for the corrupted parties and a random sample $\hat{R}$ from $\mathcal{D}(1^\lambda)$, it is possible to generate fake messages for the honest parties such that, when used in conjunction with $(\hat{U}_j)_{j\notin H}$, the output of the protocol is $\hat{R}$. These fake messages are indistinguishable from the real ones, so no adversary is able to tell if the output was programmed or not.

This property is sufficient to achieve security against non-rushing adversaries in the UC model. Indeed, in this setting, the simulator gets to know the messages of the corrupted parties before generating those of the honest players. So, it can just send fake messages that are programmed to output $\hat{R}$, the sample received from the functionality. In some sense, the simulator is leveraging rushing against the adversary.

The strategy, however, fails against rushing adversaries. Now, indeed, the adversary receives the honest messages first and then it chooses what to send on behalf of the corrupted players. The simulator can still try to program some of the outputs of the distributed sampler, but it can apply the technique only a limited number of times: the samples provided by the functionality have large entropy and so, it is impossible to "hide" many of them in the messages of the honest parties. In conclusion, if the messages of the corrupted players are chosen at random, the simulator cannot predict the choice of the adversary, so, with overwhelming probability, the output of the protocol will not have been programmed.

Unfortunately, the issue we highlighted is not only restricted to the construction of [ASY22], it is part of a more general problem formalised by Abram, Obremski and Scholl in [AOS23]: without random oracle, distributed samplers with UC security against rushing adversaries are essentially impossible. The reason is that, in the protocol, we would like the entropy of the output conditioned on the messages of any subset S of the parties to be high, i.e. $\mathsf{H}\big(R|(U_i)_{i\in S}\big) = \omega(\log\lambda)$. If that was not the case, an adversary corrupting all the parties in S would have too much influence over the output of the protocol, compromising security. On the other hand, in the ideal world, we would like the simulator to generate fake honest messages so that $\mathsf{H}\big(R|(U_i)_{i\in H}\big)$ is small, namely $O(\log\lambda)$. In this way, we can hope to hide ideal samples in the output space so that, even if the adversary decides the messages of the corrupted parties after seeing $(U_i)_{i\in H}$, the output of the protocol will be an ideal sample with high probability. The results presented by Abram, Obremski and Scholl in [AOS23] suggest that, for any such simulator, it is possible to distinguish between the real $(U_i)_{i\in H}$ and the simulated ones.

Introducing lossy distributed samplers. We move back to our goal: building hardness-preserving and indistinguishability-preserving distributed samplers. Although we are not aiming for UC security anymore, having a way to control the output of the distributed sampler is still a desirable property that would simplify our task. In this context, the discussion about entropy in the previous paragraph raised a point we need to face. We do this by introducing the notion of *lossy distributed sampler*.

A lossy distributed sampler is a distributed sampler having two modes of operation. In the standard mode, for every non-empty $H \subseteq [n]$, the entropy $\mathsf{H}\big(R|(U_i)_{i\in H}\big)$ will remain high, namely $\omega(\log\lambda)$. In this way, we can make sure that the influence of the adversary on the protocol is limited. By switching to lossy mode, however, the messages of the honest parties restrict the output in a set of polynomial size, with high probability. In other words, in the lossy mode, the outputs of the protocol becomes predictable. This allows us to deal with rushing.

Unavoidably, an adversary can always distinguish between a distributed sampler in standard mode and one in lossy mode [8]. However, lossy distributed samplers permit making the distinguishability advantage arbitrarily small: for every polynomial $p(\lambda)$ and inverse polynomial function $\delta(\lambda)$, we can set the parameters of the lossy mode so that no adversary running in time at most $p(\lambda)$ can distinguish between the standard

[8]In the standard mode, running the protocol twice produces different outputs with overwhelming probability, in lossy mode, instead, there is a non-negligible probability of obtaining a collision.

mode and the lossy mode with advantage greater than $\delta(\lambda)$. Observe that this property strongly resembles the one of ELFs [Zha16]. This is not a coincidence, as ELFs will be one of the building blocks for lossy distributed samplers.

We now present the precise definition.

Definition 4.6.1 (Lossy distributed sampler). An *lossy distributed sampler* for AClass is an n-party distributed sampler $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample})$ for which there exists a tuple of PPT algorithms $(\mathsf{LossySetup}, \mathsf{LossyGen}, \mathsf{Project}, \mathsf{Extract})$ with the following syntax:

- $\mathsf{LossySetup}$ is randomised and takes as input the security parameter and an integer $q \in \mathbb{N}$. The output is a lossy distributed sampler crs, the CRS trapdoor ζ.

- $\mathsf{LossyGen}$ is uniform, randomised and takes as input the security parameter, a session identity sid, an index $i \in [n]$ and the CRS information ζ. The output is a lossy distributed sampler message U_i and the extraction trapdoor ξ.

- $\mathsf{Project}$ is uniform, deterministic and takes as input the CRS trapdoor ζ, n distributed sampler messages $(U_i)_{i\in[n]}$ and a session identity sid. The output is an element z.

- $\mathsf{Extract}$ is uniform, deterministic and takes as input an extraction trapdoor ξ and a value z. The output is a sample R.

A lossy distributed sampler satisfies the following properties.

- **(Arbitrarily small advantage).** For every polynomial $p(\lambda)$ and inverse polynomial function $\delta(\lambda)$, there exists a polynomial $q(\lambda)$ such that every adversary $\mathcal{A} \in \mathsf{AClass}$ running in time at most p can win the game in Figure 4.26 with advantage asymptotically smaller than δ.

- **(Small support).** For every polynomial $q(\lambda)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that, for every session identity sid and index $i \in [n]$,

$$\Pr\left[\left|\mathsf{Supp}_\zeta\right| > q(\lambda) \middle| \begin{array}{l} (\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{LossySetup}\big(1^\lambda, q(\lambda)\big) \\ (U_i, \xi) \xleftarrow{\$} \mathsf{LossyGen}(1^\lambda, \mathsf{sid}, i, \zeta) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Supp}_\zeta := \big\{\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big) \big| \big(\mathsf{sid}, (U_j)_{j\in[n]}\big) \in \{0,1\}^*\big\}$.

Notice that the lossy mode is split into two parts: a lossy setup $\mathsf{LossySetup}$ and a lossy generation algorithm $\mathsf{LossyGen}$. The lossy setup takes as input the parameter $q(\lambda)$ and outputs a fake CRS along with a trapdoor ζ. The lossy generation algorithm takes as input the trapdoor ζ and the index of party $i \in [n]$. The output is the lossy message for P_i. In order to switch the distributed sampler to lossy mode, it is sufficient that a single party sends a message in lossy mode. When that happens, with high probability, the output of Sample is obtained by first projecting the exchanged messages in a set of polynomial size and then deterministically mapping the result into a sample from $\mathcal{D}(1^\lambda)$. Observe, however, that our definition does not guarantee that this occurs with overwhelming probability, but just with probability $1 - \delta(\lambda)$, where $\delta(\lambda)$ is an arbitrarily small inverse-polynomial quantity. Informally, this means that the lossy mode restricts most of the outputs of the construction in a set of polynomial size.

In order to make the distinguishability advantage between standard and lossy mode arbitrarily small, it is important that $\mathsf{Project}$ and $\mathsf{Extract}$ are hard to compute when the trapdoors ζ and ξ are kept secret.

Regularity of lossy distributed samplers. We now formulate the definition of regular lossy distributed sampler. Essentially, this consists of a lossy distributed sampler for which the output of the lossy mode is predictable with inverse-polynomial probability independently of the behaviour of the adversary. Observe that if the output space was not restricted in a set of polynomial size, this property was unachievable.

Lossy distributed sampler game

Initialisation: This procedure is run only once, at the beginning of the game.

1. $b \xleftarrow{\$} \{0,1\}$
2. $\mathsf{crs}_0 \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$
3. $(\mathsf{crs}_1, \zeta) \xleftarrow{\$} \mathsf{LossySetup}\big(1^\lambda, q(\lambda)\big)$
4. Activate $\mathcal{A}$ with 1^λ and crs_b
5. Receive a set of distinct identities $\mathsf{ID} := (\mathsf{id}_i)_{i\in[m]}$ from $\mathcal{A}$.

New session: This procedure can be queried multiple times and at any point of the game. Upon receiving any query $(\mathsf{NewSession}, \mathsf{tag}, \mathsf{id}_{j_1}, \ldots, \mathsf{id}_{j_n}, i)$ where the session identity $\mathsf{sid} := (\mathsf{tag}, \mathsf{id}_{j_1}, \ldots, \mathsf{id}_{j_n})$ has never been queried before, $\mathsf{id}_{j_l} \in \mathsf{ID}$ for every $l \in [n]$ and all $\mathsf{id}_{j_l} \neq \mathsf{id}_{j_k}$ for every $l \neq k$, compute the following.

1. $U_i^0 \xleftarrow{\$} \mathsf{Gen}\big(1^\lambda, \mathsf{sid}, i, \mathsf{crs}\big)$
2. $(U_i^1, \xi) \xleftarrow{\$} \mathsf{LossyGen}\big(1^\lambda, \mathsf{sid}, i, \zeta\big)$
3. Provide the adversary with $U_i := U_i^b$
4. Store $(\mathsf{sid}, i, U_i, \xi)$

Sample: This procedure can be queried multiple times and at any point of the game. Upon receiving any query $\big(\mathsf{Sample}, \mathsf{sid}, (U_j)_{j\neq i}\big)$ where sid denotes the identity of an already initiated session, compute the following. The same session identity can be queried multiple times.

1. Retrieve (sid, i, U_i)
2. $R_0 \leftarrow \mathsf{Sample}(U_1, \ldots, U_n, \mathsf{sid}, \mathsf{crs}_0)$
3. $R_1 \leftarrow \mathsf{Extract}\Big(\xi, \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)\Big)$
4. Provide the adversary with R_b.

Win: The adversary wins if it guesses b

Figure 4.26: Lossy distributed sampler game

Definition 4.6.2 (Regularity). We say that a lossy distributed sampler ($\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{LossySetup}, \mathsf{LossyGen}, \mathsf{Project}, \mathsf{Extract}$) is regular if there exists a uniform PPT algorithm $\mathcal{Z}$ and a polynomial $s(\lambda, q)$ such that, for every polynomial $q(\lambda)$, with overwhelming probability over the randomness of $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{LossySetup}\big(1^\lambda, q(\lambda)\big)$,

$$\Pr_{\mathcal{Z}}\left[\mathcal{Z}(\zeta) = \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)\right] \geq \frac{1}{s\big(\lambda, q(\lambda)\big)}$$

for every $\big(\mathsf{sid}, (U_i)_{i\in[n]}\big) \in \{0,1\}^*$, where the above probability is taken only over the randomness of $\mathcal{Z}$.

Formally, the above definition states that $\mathcal{Z}$ allows to predict the output of the projection with inverse-polynomial probability. Furthermore, the success probability is essentially only over the randomness of $\mathcal{Z}$. That immediately allows predicting the output thanks to $\mathsf{Extract}$.

Programmability. We finally formalise the notion of *programmable lossy distributed sampler*. This consists of a construction in which the lossy mode allows hiding an ideal sample R in the output space. In particular, there will be an element z such that executions that are projected to z will output R with high probability. Furthermore, the adversary will not be able to tell if one of the outputs was programmed even if we provide it with z and the trapdoor ζ. The extraction trapdoor ξ will instead remain secret.

Programmability Game

Each phase is run only once.

Initialisation Phase:

1. $b \xleftarrow{\$} \{0,1\}$
2. $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{LossySetup}\big(1^\lambda, q(\lambda)\big)$
3. Activate $\mathcal{A}$ with 1^λ, crs and ζ.

Generation Phase:

1. Receive $i \in [n]$, sid and z from the adversary.
2. $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$
3. $(U_i^0, \xi^0) \xleftarrow{\$} \mathsf{LossyGen}\big(1^\lambda, \mathsf{sid}, i, \zeta\big)$
4. $(U_i^0, \xi^1) \xleftarrow{\$} \mathsf{ProgGen}\big(1^\lambda, \mathsf{sid}, i, z, R, \zeta\big)$
5. Provide the adversary with $U_i := U_i^b$

Sampling Phase:

1. Receive $(U_j)_{j \neq i}$ from the adversary
2. $R_0 \leftarrow \mathsf{Extract}\Big(\xi^0, \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)\Big)$
3. $R_1 \leftarrow \mathsf{Extract}\Big(\xi^1, \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)\Big)$
4. If $\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big) = z$ and $z \neq \perp$, set $R_1 \leftarrow R$.
5. If $\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big) = \perp$, set $R_1 \leftarrow \perp$.
6. Provide the adversary with R_b.

Win: The adversary wins if it guesses b

Figure 4.27: Programmability game

Observe that programmability is the only property that guarantees that the outputs of the distributed sampler look like those of the targetted distribution $\mathcal{D}(1^\lambda)$ and no further information is leaked. In Section 4.8, we will show that lossy distributed samplers that are regular and programmable are hardness-preserving.

Definition 4.6.3 (Programmability). We say that a lossy distributed sampler ($\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{LossySetup}, \mathsf{LossyGen}, \mathsf{Project}, \mathsf{Extract}$) for $\mathcal{D}$ is programmable if there exists a uniform PPT algorithm $\mathsf{ProgGen}$ such that no PPT adversary in AClass can win the game in Figure 4.27 with non-negligible advantage.

4.7 Building Lossy Distributed Samplers

In this section, we present a lossy distributed sampler that is regular and programmable. In the non-uniform setting, the construction relies on a uniformly random CRS which can be reused multiple times. In the uniform setting, instead, we need no CRS. Security is based, among other primitives, on subexponentially secure indistinguishability obfuscation and multi-key FHE. We achieve security against any active adversary statically corrupting up to $n-1$ parties.

The construction of [ASY22]. Our starting point is the semi-malicious distributed sampler of [ASY22, Section 4], which achieves security against non-rushing adversaries in the plain model.

Our construction inherits the same structure: the distributed sampler message of each party P_i consists of two obfuscated programs. The purpose of the first one is to generate a pseudorandom string s_i and encrypt it under a multi-key FHE public key pk_i. The random string s_i will be P_i's share of the randomness input into $\mathcal{D}(1^\lambda)$. In other words, the output of the distributed sampler will be a sample R obtained by adding the strings $s_1, s_2, \dots, s_n$ and feeding the result as randomness for $\mathcal{D}(1^\lambda)$. We call this first program *the encryption program* of party P_i and we denote it by EP_i.

The second program instead has the purpose of applying homomorphic operations on the ciphertexts generated by the encryption programs, deriving an encryption C of the output R. The program terminates its execution outputting a partial decryption of C using the private counterpart of pk_i. We call this second program *the decryption program* of party P_i and we denote it by DP_i. The encryption of s_j and the public key pk_j will be derived running EP_j inside the code of DP_i. The encryption program EP_j will be given as input to DP_i for every $j \neq i$.

To summarise, in order to obtain a random sample R, the parties just feed each decryption program DP_i with the encryption programs $(\mathsf{EP}_j)_{j\neq i}$. In this way, they obtain the partial plaintext d_i. The output will be derived by performing the final decryption $R \leftarrow \mathsf{FinDec}(d_1, \dots, d_n)$.

Counteracting the residual function attack. A common issue of one-round MPC protocols is residual function attacks: the adversary can rerun the protocol in its head keeping the same messages for the honest parties but using different messages for the corrupted players. In this way, it obtains a different output that might be correlated to the original one. Observe that the adversary can repeat this attack as many times as it likes, potentially obtaining a lot of leakage.

In order to prevent this issue in their distributed sampler [ASY22], Abram, Scholl and Yakoubov made sure that EP_i encrypts an independent-looking s_i for every choice of $(\mathsf{EP}_j)_{j\neq i}$. They achieved this by letting every party P_i choose a hash key hk_i and providing EP_i with a digest y_i of $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ under hk_i (notice that we cannot directly input $(\mathsf{EP}_j)_{j\neq i}$ into EP_i as the former is significantly larger than the latter). The encryption program EP_i will derive s_i by feeding y_i into a puncturable PRF. The key used for the encryption will also change depending on $(\mathsf{EP}_j)_{j\neq i}$. The technique remains the same as before: by feeding y_i into another puncturable PRF, the program obtains randomness r_i and r'_i that will be used for the key generation and the encryption. The hash keys will be broadcast by the parties as part of their message.

Using this strategy, even if an adversary reruns the distributed sampler protocol in its head changing any $(\mathsf{hk}_j, \mathsf{EP}_j)$, the encryption program EP_i will generate an independent looking s_i and so the new output R' obtained by the adversary will look independent of the original one. Notice that changing any DP_j instead does not help in learning information about R.

The construction in this paper will keep using the technique of [ASY22]. We sketch the unobfuscated code of the encryption program EProg in Figure 4.28.

Adjustments in the decryption programs. The modifications to the encryptions programs we added in the previous paragraph require minor adjustments in the decryption programs. As we have mentioned, for every $j \in [n]$, each DP_i needs to evaluate the encryption program EP_j to obtain pk_j and the encryption of s_j. In order to do this, it needs to compute the digest y_j that will be fed into EP_j. For this reason, we need to provide DP_i not only with $(\mathsf{EP}_j)_{j\neq i}$ but also with all the hash keys $(\mathsf{hk}_j)_{j\neq i}$. The pair $(\mathsf{hk}_i, \mathsf{EP}_i)$ will instead be hardcoded into DP_i. In the decryption program, we also hardcode the PRF key that produced the randomness for the key generation in EP_i. This will allow DP_i to retrieve the secret key needed for the partial decryption.

We also introduce another modification to the decryption programs and the construction in general. The reason for this will be clearer after reading the next paragraphs. Along with hk_i, EP_i and DP_i, each party

```
EProg[$K_1^{(i)}, K_2^{(i)}, i$]

Hard-coded. The PPRF keys $K_1^{(i)}$ and $K_2^{(i)}$, the index $i$.
Input. A digest $y \in \{0,1\}^{t(\lambda)}$.

  1. $s_i \leftarrow F_1(K_1^{(i)}, y)$
  2. $(r_i, r'_i, r''_i, \eta_i, \eta'_i) \leftarrow F_2(K_2^{(i)}, y)$
  3. $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Gen}(1^\lambda, i;\, r_i)$
  4. $c_i \leftarrow \mathsf{mkFHE.Enc}(\mathsf{pk}_i, s_i;\, r'_i)$
  5. Output $(\mathsf{pk}_i, c_i)$.
```

Figure 4.28: The unobfuscated encryption program of party P_i

P_i will now broadcast an almost everywhere extractable NIZK π_i proving the well-formedness of $(\mathsf{hk}_i, \mathsf{EP}_i)$. In the non-uniform case, this NIZK will require a CRS. Luckily, the latter can be uniformly random (see Section 4.4.1 and [AWZ23, Section 10.1]). We denote the construction by NIZK. Each decryption program DP_i will now receive the proofs $(\pi_j)_{j\neq i}$ as input and will use them to check the pair $(\mathsf{hk}_j, \mathsf{EP}_j)$ for every $j \neq i$. If any of the NIZKs does not verify, the decryption program DP_i simply outputs $\perp$. We sketch the unobfuscated code of the decryption program DProg in Figure 4.29.

Circular dependencies between subexponentially secure primitives. Our construction can achieve security as long as at least one of the random strings s_i remains private. Since the encryption program EP_i always reveals an encryption of the latter, we need to rely on the security of multi-key FHE. Unfortunately, we cannot perform a direct reduction as the PRF key that allows retrieving the multi-key FHE private key is hardcoded into both the encryption and the decryption program. So, in the security proof, we need to somehow remove the information about sk_i from EP_i and DP_i first, and only at that point, we can apply the multi-key FHE security.

Our goal is to achieve this using subexponentially secure primitives, similarly to what Halevi *et al.* did in [HIJ+17]. Specifically, by repeating a hybrid argument for every tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ of well-formed elements, the programs EP_i and DP_i will gradually switch from performing the key generation, the encryptions and the partial decryptions to simulating them [AJJM20]. Notice that the multi-key FHE simulators need to know the randomness used by all the other parties. The program will extract it from the NIZKs $(\pi_j)_{j\neq i}$ that are given as input.

In order for our strategy to work, we need to rely on the subexponential security of multi-key FHE. In particular, if we denote the number of well-formed tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ by $N(\lambda)$ and the advantage of any PPT adversary $\mathcal{A}$ against the multi-key FHE scheme by $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{mkFHE}}(\lambda)$, we require that there exists a constant $e \in \mathbb{N}$ such that

$$N(\lambda) \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathsf{mkFHE}}(\lambda^e) = \mathsf{negl}(\lambda)^{9}.$$

This is because, every time we rely on the simulatability of the partial decryption, the advantage of the adversary increases by a negligible but non-zero amount. In our proof, we rely on this argument a super-polynomial number of times, namely at least $N(\lambda)$, so, at the end, the small advantages might add up to something non-negligible. If the e described above exists, however, we are sure that, by setting the security parameter of multi-key FHE to $\lambda' := \lambda^e$, this will not happen. The final stage will be indistinguishable from the initial one.

[9]To be precise, we will require a strictly stronger property: instead of N, we will use another function $M(\lambda) \gg N(\lambda)$.

$\mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the CRS for the extractable NIZK σ.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

```
1. $\forall j \neq i: \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
2. If $\exists j \neq i$ such that $b_j = 0$, output $\bot$
3. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
4. $\forall j \in [n]: \quad (\mathsf{pk}_j, c_j) \leftarrow \mathsf{EP}_j(y_j)$
5. $C \leftarrow \mathsf{mkFHE.Eval}\big(\tilde{\mathcal{D}}, \mathsf{pk}_1, c_1, \ldots, \mathsf{pk}_n, c_n\big)$ (see below)
6. $(r_i, r'_i, r''_i, \eta_i, \eta'_i) \leftarrow F_2(K_2^{(i)}, y_i)$
7. $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Gen}(1^\lambda, i;\, r_i)$
8. $d_i \leftarrow \mathsf{mkFHE.PartDec}\Big(C, (\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n), i, \mathsf{sk}_i;\, \eta_i\Big)$
9. Output $d_i$
```

The algorithm $\tilde{\mathcal{D}}$. On input n random strings $s_1, s_2, \ldots, s_n \in \{0,1\}^{m(\lambda)}$.

```
1. $s \leftarrow s_1 \oplus s_2 \oplus \cdots \oplus s_n$
2. $R \leftarrow \mathcal{D}(1^\lambda; s)$
3. Output $R$
```

Figure 4.29: The unobfuscated decryption program of party P_i

The issue is that $N(\lambda)$ already depends on λ'. Indeed, every encryption program generates a multi-key FHE key. We are therefore trapped in a circular dependency. It also turns out that this is not the only one, it is just the easiest to spot.

Decreasing the entropy of the corrupted messages. We solve our problems using an idea of [ASY22]: we decrease the entropy of $(\mathsf{hk}_j, \mathsf{EP}_j)$ generating it using a PRG. Each party P_j will now sample a random λ-bit seed and will use its expansion to generate the PRF keys hidden in EP_j, the hash key hk_j and to obfuscate EP_j. The NIZK π_j will guarantee that the pair $(\mathsf{hk}_j, \mathsf{EP}_j)$ is generated in this way. In other words, the adversary will be forced to output low-entropy messages. On the other hand, by leveraging the simulatability of the NIZK, we will be able to send full-entropy messages for the honest parties.

Thanks to this trick, the number of well-formed $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ will be independent of the multi-key FHE security parameter λ': the value of $N(\lambda)$ will be $2^{\lambda \cdot (n-1)}$. By choosing e sufficiently large and assuming subexponential security, we can finally make sure that

$$N(\lambda) \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathsf{mkFHE}}(\lambda^e) = \mathsf{negl}(\lambda).$$

This trick fixes all the other circular dependencies too.

Avoiding collisions between well-formed encryption programs. The technique described in the previous paragraph will also allow us to achieve a nice property: by taking a subexponentially collision resistant hash function, we can make sure that, with overwhelming probability over hk_i [10], there exist no hash collisions between well-formed tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$. In particular, we choose the hash function security parameter λ' so that, for every PPT adversary $\mathcal{A}$,

$$N(\lambda)^2 \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathsf{Hash}}(\lambda') = \mathsf{negl}(\lambda),$$

where $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Hash}}(\lambda')$ denotes the advantage of $\mathcal{A}$ against the collision resistance of Hash. Notice that $N(\lambda)^2$ upper-bounds the number of pairs of well-formed tuples. This of course will increase the size of the digests but they will still fit into EP_i. We will explain how this property is used in the security proof in Section 4.7.2.

Adding a final NIZK to achieve active security. The reader might have noticed that in our blueprint, nothing prevents an adversary to broadcast malformed decryption programs. In order to contrast this kind of malicious behaviour, we add a second NIZK to the construction proving the well-formedness of the programs and the hash key. We rely on a simulation-extractable NIZK, we denote it by NIZK'. Observe that the latter satisfied multi-theorem zero-knowledge. If we aim for security against non-uniform adversaries, NIZK' will require a CRS that can be small and uniformly random. In the uniform setting, if we use the construction in [AWZ23, Section 9.3], NIZK' has no CRS. We denote the new proof broadcast by party P_i by π'_i. To summarise, the distributed sampler message of P_i will consists of the tuple $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

4.7.1 Introducing ELFs to Achieve Lossy Properties

The construction we sketched above is not lossy: given the messages of the honest parties, the output still remains highly unpredictable, i.e. $\mathsf{H}\big(R|(U_i)_{i\in H}\big) = \omega(\log \lambda)$. For this reason, we introduce an ELF in the construction. When the latter is set in injective mode, the entropy of the output given the honest messages will remain high. When the ELF is instead in lossy mode, the messages of the honest players will restrict the output in a set of polynomial size, no matter what the adversary does. The properties of ELFs will also allow us to make the distinguishability advantage between injective mode and lossy mode arbitrarily small. That will be fundamental to achieve the first property of lossy distributed samplers (see Definition 4.6.1).

While integrating the ELF in the construction, we need to pay attention to particular conditions. As mentioned in the technical overview, we want the distributed sampler to be regular and programmable: our goal is to hide an ideal sample R among the small set of possible outputs allowed by the lossy mode. Any adversary must have a $1/\mathsf{poly}(\lambda)$ probability of obliviously selecting R as output of the protocol. We need also to focus on incorporating the ELF in the construction while keeping the CRS as simple as possible. Finally, we need to make sure that the protocol supports a polynomial number of parties instead of just a constant.

Where should we place the ELF? Satisfying all the conditions described above is not trivial. Our current construction allows the parties to produce a common string that looks random as long as one party is honest, i.e. the string $s := s_1 \oplus s_2 \oplus \cdots \oplus s_n$. In order to obtain a random looking sample from $\mathcal{D}(1^\lambda)$, we need a common source of entropy, i.e. entropy that can be accessed by all players. The CRS and s are the only sources of this kind we have at the moment.

After observing this, one could try to achieve the properties we need by feeding s into the ELF, convert the output into uniform randomness using an extractor and then input the result into the distribution $\mathcal{D}(1^\lambda)$. While adding two CRSs (one for the ELF, one for the extractor), this solution allows to restrict the output in a set of polynomial size when the ELF is set in lossy mode. However, it lacks programmability: how can we hide an ideal sample among the outputs without the adversary noticing it? Observe that, using just the CRSs, the adversary can compute the set of all possible outputs, so the ideal sample would stand out. In cryptography, programmability is often achieved using puncturable PRFs and obfuscation. The technique requires the PRF key K to be private and unpredictable. Hiding K in the CRS seems hard, perhaps even impossible. Another option would be to use s to generate K. The issue is that K needs high entropy, so we cannot use the output of the ELF, we are forced to use s itself. If we do that, however, the size of the output space would become superpolynomial.

[10] The probability is over full-entropy hash keys.

```
EProg_Ls[$K_2^{(i)}, i$]

Hard-coded. The PPRF key $K_2^{(i)}$ and the index $i$.
Input. A digest $y \in \{0,1\}^{t(\lambda)}$.

  1. $(r_i, r'_i, r''_i, \eta_i, \eta'_i) \leftarrow F_2(K_2^{(i)}, y)$
  2. $(\phi, \mathsf{pk}_i, c_i) \leftarrow \mathsf{mkFHE.Sim}_1(1^\lambda, i; r''_i)$
  3. Output $(\mathsf{pk}_i, c_i)$.
```

Figure 4.30: The unobfuscated encryption program for the lossy mode

The lossy mode of the distributed sampler. We solve all our problems by relying on subexponentially secure primitives and making the ELF appear only when the distributed sampler is set in lossy mode.

The lossy mode will produce programs EP_i and DP_i that differ from the ones in standard mode, we formally describe them in Figure 4.30 and Figure 4.31. The idea is that EP_i and DP_i will simulate the key generation, the encryptions and the partial decryptions. The security of multi-key FHE will guarantee that the output of the distributed sampler will be the sample given to the multi-key FHE simulator. Such sample will not coincide with the value hidden in the evaluated ciphertext C, it will be the element produced by $\mathsf{Project}$ and $\mathsf{Extract}$. The former will simply apply an ELF f on $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}$. The latter will use a puncturable PRF key K to deterministically map each projection into a pseudorandom string s, which will be used as randomness for $\mathcal{D}(1^\lambda)$. Notice that since the ELF is in lossy mode, the image of the projection will be polynomial in size.

Working out the rest of the details is now easy. The decryption program will be able to run $\mathsf{Project}$ and $\mathsf{Extract}$ inside its code as f and K will be hardcoded into it. It will also be able to perform the partial decryption as it will know the PRF keys hardcoded in the encryption programs that are given as input: it will extract them from the NIZKs that are provided along with $(\mathsf{EP}_j)_{j\neq i}$. When the extraction fails, the program will simply output $\perp$. In order for our strategy to succeed, the lossy setup will simulate the CRSs σ and σ'. The corresponding trapdoors will also allow us to generate proofs π_i and π'_i despite the fact that EP_i and DP_i are no longer well-formed. The lossy setup will also take care of generating the ELF f. The size of the image will be $q(\lambda)$ where q is the polynomial parametrising the lossy mode of the distributed sampler. A final minor issue is that the second multi-key FHE simulator needs to receive the state produced by the first simulator. The execution of the former takes place in DP_i, whereas the latter is run in EP_i. Thankfully, both executions are made deterministic using the outputs of a puncturable PRF. By storing the corresponding key in both EP_i and DP_i, we can rerun the first simulator inside DP_i to retrieve the state.

Regularity and programmability of the lossy mode. The above construction can easily be made regular. It is sufficient to use a regular ELF: by sampling a random element x in the domain $[M]$, $f(x)$ will hit all the elements in the support of the projection with inverse-polynomial probability.

The construction is also programmable. Thanks to obfuscation and the security of puncturable PRFs [SW14], we can easily hide an ideal sample in the output space of the lossy mode distributed sampler. All we need to do is to puncture K in the right position z. We then modify the decryption program by hardcoding an ideal sample R along with z and the punctured key. Differently from before, the new program will compare the output of the ELF with z. When the latter coincide, it will directly feed R to the partial decryption simulator. It is easy to prove that the adversary is not able to detect whether we hid an ideal sample in the output space or not.

```
DProg_Ls[$i$, sid, $K_2^{(i)}$, $\mathsf{EP}_i$, $\mathsf{hk}_i$, $\sigma$, $(\tau_e^j)_{j\neq i}$, $K$, $f$]

Hard-coded. The index $i$ of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program $\mathsf{EP}_i$, the hash key $\mathsf{hk}_i$, the extractable NIZK CRS $\sigma$ and the extraction trapdoors $(\tau_e^j)_{j\neq i}$, the PPRF key $K$, the ELF $f$.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$.

  1. $\forall j \neq i: \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
  2. $\forall j \neq i: \quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$ [a]
  3. If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \perp$, output $\perp$
  4. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l\neq j}\big)$
  5. $\forall j \neq i: \quad s_j \leftarrow F_1\big(K_1^{(j)}, y_j\big)$
  6. $\forall j \in [n]: \quad (r_j, r_j', r_j'', \eta_j, \eta_j') \leftarrow F_2\big(K_2^{(j)}, y_j\big)$
  7. $z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big)$
  8. $s \leftarrow F(K, z)$
  9. $\hat{R} \leftarrow \mathcal{D}(1^\lambda;\ s)$
  10. $(\phi, \mathsf{pk}_i, c_i) \leftarrow \mathsf{mkFHE.Sim}_1(1^\lambda, i;\ r_i'')$
  11. $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, \hat{R}, (s_j, r_j, r_j')_{j\neq i};\ \eta_i'\Big)$ (see bottom of Figure 4.29)
  12. Output $d_i$

[a] Here, we simplified the notation: the extractor would output only the PRG seed used to produce $(\mathsf{hk}_j, \mathsf{EP}_j)$. By the expanding that, it is straightforward to derive $K_1^{(j)}$ and $K_2^{(j)}$.
```

Figure 4.31: The unobfuscated decryption program for the lossy mode

4.7.2 Proving Security

We present a blueprint of the security proof. Seeing that, in our construction, the projection has small support is rather straightforward. We therefore focus on the first property of the lossy distributed sampler. The proof will hold independently of whether AClass represents the class of uniform adversaries or not.

Before starting, we recall our goal: we want to show that for every polynomial $p(\lambda)$ and inverse-polynomial function $\delta(\lambda)$, there exists a polynomial $q(\lambda)$ such that the advantage of all adversaries running in time at most $p(\lambda)$ in distinguishing the standard mode from the lossy mode parametrised by $q(\lambda)$ is asymptotically smaller than $\delta(\lambda)$. We prove the result through a series of hybrids, starting from the standard mode.

First step: simulating NIZK′. We start the proof by simulating the proof π_i' in every NewSession query. We recall that i denotes the index chosen by the adversary in each of these queries, π_i' denotes instead the simulation-extractable NIZK proving the well-formedness of the message of i-th party. We also modify the answer to the sampling queries: we start by extracting the witnesses from the NIZKs $(\pi_j')_{j\neq i}$ selected by the adversary. If the extraction fails, we answer with $\perp$, otherwise, we reply with the output of Sample. This hybrid is indistinguishable from the previous one due to the simulation-extractability of NIZK′.

$\mathsf{DProg}_1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the CRS for the extractable NIZK σ, the extraction trapdoors $(\tau_e^j)_{j\neq i}$.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$.

```
1. $\forall j \neq i: \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
2. $\forall j \neq i: \quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$[a]
3. If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \bot$, output $\bot$
4. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l\neq j}\big)$
5. $\forall j \in [n]: \quad (\mathsf{pk}_j, c_j) \leftarrow \mathsf{EP}_j(y_j)$
6. $C \leftarrow \mathsf{mkFHE.Eval}\big(\tilde{\mathcal{D}}, \mathsf{pk}_1, c_1, \ldots, \mathsf{pk}_n, c_n\big)$ (see Figure 4.29)
7. $(r_i, r'_i, r''_i, \eta_i, \eta'_i) \leftarrow F_2(K_2^{(i)}, y_i)$
8. $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Gen}(1^\lambda, i;\, r_i)$
9. $d_i \leftarrow \mathsf{mkFHE.PartDec}\Big(C, (\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n), i, \mathsf{sk}_i;\, \eta_i\Big)$
10. Output $d_i$
```

Figure 4.32: Second step: the unobfuscated decryption program of party P_i

Second step: witness extraction in the decryption programs. We proceed by simulating the proof π_i in every $\mathsf{NewSession}$ query. We recall that π_i is an almost everywhere extractable NIZK proving the well-fomredness of $(\mathsf{hk}_i, \mathsf{EP}_i)$. We also modify the decryption program DP_i. Specifically, we hardcode extraction trapdoors $(\tau_e^j)_{j\neq i}$ for the almost everywhere extractable NIZK. The label associated with τ_e^j will be (sid, j) where sid is the session identity queried by the adversary. The program DP_i will now try to extract the witness from the NIZK proofs that are given as input. If any extraction fails, DP_i will simply outputs $\bot$. Otherwise, it will perform the usual operations. Notice that now the decryption program will only accept well-formed inputs. We sketch the operations of the modified program DProg_1 in Figure 4.32.

We highlight that, compared to the previous step, the input-output behaviour of DP_i changed. However, the two hybrids will still be indistinguishable thanks to the almost-everywhere extractability of NIZK (see Lemma 4.4.4 and [AWZ23, Lemma 3]). In the uniform setting, this step requires additional attention. Indeed, in the reduction to almost-everywhere extractability, the uniform adversary needs to derive the trapdoor τ' for NIZK'. The latter cannot be computed in uniform polynomial time. We work around this problem by choosing NIZK, NIZK' and a superpolynomial function $S(\lambda)$ so that NIZK' is S-deterministic and NIZK is a-disclosed for every S-computable sequence a. The construction presented in [AWZ23, Section 10.1] allows this.

Third step: switching to full-entropy. Since the NIZKs are now simulated, we are free to switch to a full-entropy U_i. Specifically, we generate the hash key hk_i, the encryption program EP_i and the keys hard-coded into it using full-entropy randomness, instead of the output of a PRG. This stage is indistinguishable from the previous by the security of the PRG.

```
DProg_2[$i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}$]

Hard-coded. The index $i$ of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption
program $\mathsf{EP}_i$, the hash key $\mathsf{hk}_i$, the extractable NIZK CRS $\sigma$ and the extraction trapdoors $(\tau_e^j)_{j\neq i}$,
the PPRF key $K_1^{(i)}$.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$.

  1. $\forall j \neq i:\quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
  2. $\forall j \neq i:\quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$ [a]
  3. If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \perp$, output $\perp$
  4. $\forall j \in [n]:\quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l\neq j}\big)$
  5. $\forall j \in [n]:\quad s_j \leftarrow F_1\big(K_1^{(j)}, y_j\big)$
  6. $\forall j \in [n]:\quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2\big(K_2^{(j)}, y_j\big)$
  7. $\hat{R} \leftarrow \mathcal{D}(1^\lambda; s_1 \oplus \cdots \oplus s_n)$
  8. $(\phi, \mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Sim}_1(1^\lambda, i; r''_i)$
  9. $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, \hat{R}, (s_j, r_j, r'_j)_{j\neq i}; \eta'_i\Big)$ (see bottom of Figure 4.29)
  10. Output $d_i$
```

Figure 4.33: Forth step: the unobfuscated decryption program of party P_i

Fourth step: simulating key generation, ciphertexts and partial decryptions. We proceed by modifying both the encryption program EP_i and the decryption program DP_i. The new programs will not perform the multi-key FHE operations as usual, they will instead simulate them. In order to perform such operation, DP_i will use the PRF keys in the encryption programs of the other parties, which will be extracted from the NIZKs $(\pi_j)_{j\neq i}$ that are given as input. For the moment, the multi-key FHE simulator in DP_i will also need to know the sample $\hat{R}$ hidden into the joint ciphertext C. The program will reconstruct it using the PRF keys $(K_1^{(j)})_{j\neq i}$ hidden in the encryption programs $(\mathsf{EP}_j)_{j\neq i}$, those used to produce the pseudorandom strings $(s_j)_{j\neq i}$. In order to compute s_i, the new program will also have the key $K_1^{(i)}$ hardcoded. Finally, the simulator in DP_i will need to know that secret information output by the first simulator Sim_1, the one that produced a fake public key and a fake ciphertext in EP_i. The decryption program will obtain it by rerunning Sim_1 with the same randomness as in EP_i. It is easy to do that as the randomness is a PRF output and the corresponding key is hardcoded in both EP_i and DP_i. We sketch the unobfuscated version of the modified programs $\mathsf{EProg}_{\mathsf{Ls}}$ and DProg_2 in Figure 4.30 and Figure 4.33.

We prove that this step is indistinguishable from the previous one using an exponential number of hybrids. In particular, the number of reductions is proportional to the number of digests of the hash function, i.e. $2^{t(\lambda)}$. The proof relies on the reusable semi-malicious security of multi-key FHE, the collision resistance of the hash function, the security of iO and the one of the puncturable PRF F_2, all four subexponential. In order for the proof to go through, we need to use an injective obfuscator. In this way, we are sure that EP_j uniquely determines the PRF keys $K_1^{(j)}$ and $K_2^{(j)}$, so the NIZK extraction will always lead to the same values.

```
THE STANDARD MODE OF THE LOSSY DISTRIBUTED SAMPLER
Setup($1^\lambda$):
    1. $\sigma \xleftarrow{\$} \mathsf{NIZK.Gen}(1^\lambda)$
    2. $\sigma' \xleftarrow{\$} \mathsf{NIZK'.Gen}(1^\lambda)$
    3. Output $\mathsf{crs} := (\sigma, \sigma')$
Gen($1^\lambda, \mathsf{sid}, i, \mathsf{crs} := (\sigma, \sigma')$):
    1. $\rho_1 \xleftarrow{\$} \{0,1\}^{L_1(\lambda)}$
    2. $\rho_2 \xleftarrow{\$} \{0,1\}^{L_2(\lambda)}$
    3. $W \xleftarrow{\$} \{0,1\}^\lambda$
    4. $(K_1^{(i)}, K_2^{(i)}, u_1, u_2) \leftarrow \mathsf{PRG}(W)$
    5. $\mathsf{hk}_i \leftarrow \mathsf{Hash.Gen}(1^\lambda; u_1)$
    6. $\mathsf{EP}_i \leftarrow \mathsf{iO}(1^\lambda, \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i]; u_2)$ (see Figure 4.28)
    7. $\mathsf{DP}_i \leftarrow \mathsf{iO}(1^\lambda, \mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma]; \rho_1)$ (see Figure 4.29)
    8. $\pi_i \leftarrow \mathsf{NIZK.Prove}\big(\sigma, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i), W; \rho_2\big)$
    9. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.Prove}\big(\sigma', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma), (W, \rho_1, \rho_2)\big)$
    10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.
Sample$\Big(\big(U_j = (\mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \pi'_j)\big)_{j \in [n]}, \mathsf{sid}, \mathsf{crs} = (\sigma, \sigma')\Big)$
    1. $\forall j \in [n]: \quad b_j \leftarrow \mathsf{NIZK'.Verify}\big(\sigma', \pi'_j, (j, \mathsf{sid}, \mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \sigma)\big)$
    2. If there exists $j \in [n]$ such that $b_j = 0$, output $\perp$.
    3. $\forall j \in [n]: \quad d_j \leftarrow \mathsf{DP}_j\big((\mathsf{hk}_l, \mathsf{EP}_l, \pi_l)_{l \neq j}\big)$
    4. $R \leftarrow \mathsf{mkFHE.FinDec}(d_1, \ldots, d_n)$
    5. Output $R$
```

Figure 4.34: The standard mode of the lossy distributed sampler

Fifth step: embedding the ELF into the decryption program DP_i. In this step, we finally integrate the ELF in the construction. For the moment, the ELF will be set in injective mode.

We observe that at the end of step four, we have finally managed to unlink $K_1^{(i)}$ from $(s_j)_{j\neq i}$. Previously, indeed, it was impossible to modify $K_1^{(i)}$ without affecting $(s_j)_{j\neq i}$. By changing $K_1^{(i)}$, the encryption program EP_i would have become different, consequently all the digests $(y_j)_{j\neq i}$ would have changed and, at the end, we would end up with new PRF outputs $(s_j)_{j\neq i}$. Since all the strings $s_1, \dots, s_n$ where mutually dependent, it was hard to analyse how the adversary could affect the distribution of their sum. Now, instead, by the security of the puncturable PRF, we can finally say that s_i looks independent of $(s_j)_{j\neq i}$. So, we are sure that our construction generates pseudorandom samples.

We leverage this fact to modify the decryption program DP_i once again, switching to an obfuscation of $\mathsf{DProg}_{\mathsf{Ls}}$ (see Figure 4.31). The new program will ignore $(s_j)_{j\neq i}$. It will instead feed the inputs $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ along with the hardcoded pair $(\mathsf{hk}_i, \mathsf{EP}_i)$ into the injective-mode ELF. The result is then input into a puncturable PRF. The randomness produced in this way is given to $\mathcal{D}(1^\lambda)$. The generated sample will be input into the partial decryption simulator. We select the ELF so that its domain is sufficiently large to embed all $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}$ into it without causing any collision. In conclusion, in the new program, each tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ will be mapped to an independent-looking pseudorandom output.

Using a superpolynomial number of hybrids, we prove that step five and step four are indistinguishable. In particular, we repeat the hybrid arguments for every well-formed tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$, i.e. $2^{\lambda\cdot(n-1)}$ times. Observe that one way the adversary can try to distinguish between step four and step five is by finding two pairs of inputs $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$, $(\mathsf{hk}'_j, \mathsf{EP}'_j)_{j\neq i}$ having colliding digests under hk_i. Indeed, in step four, the two inputs would produce the same string s_i and therefore, the respective outputs would be correlated. In step five, instead, the adversary would end up with independent looking outputs. We prevent this attack by relying on the subexponential collision intractability of the hash function so that, with overwhelming probability over hk_i, there exist no collisions among the $2^{\lambda\cdot(n-1)}$ well-formed tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$. To summarise, we prove indistinguishability between step four and five by relying on the security of iO, the collision intractability of the hash function and the security of the puncturable PRFs F and F_1, all of them subexponential.

Final step: setting the ELF in lossy mode. At this point, we switch the ELF hidden in DP_i into lossy mode. Notice that step six can be distinguished from step five, however, by properly setting the parameters of the lossy mode, we can make the distinguishing advantage arbitrarily small. In particular, let $p'(\lambda)$ be the total time needed by the challenger and the adversary in step five. We can pick the polynomial $q(\lambda)$ parametrising the lossy mode so that no adversary running in time $p'(\lambda)$ can distinguish between injective mode and lossy mode with advantage greater than $\delta/2$. In this way, we are sure that the distinguishability advantage between step zero and step six is at most $\delta/2 + \mathsf{negl}(\lambda)$. This step corresponds to the lossy mode of the distributed sampler.

4.7.3 Formalising the Results

The full description of the standard mode of our distributed sampler is in Figure 4.34. In the construction, NIZK denotes an almost everywhere extractable NIZK. When we aim for security against non-uniform adversaries, NIZK will be chosen-ID zero-knowledge and almost everywhere extractable as in Definition 4.4.3. In the uniform setting instead, we will rely on simulation almost-everywhere extractability and zero-knowledge as in [AWZ23, Definition 31]. The NP relation underlying NIZK is

$$\mathcal{R}_1 := \left\{ \begin{matrix} (i, \mathsf{hk}_i, \mathsf{EP}_i), \\ W \end{matrix} \middle| \begin{array}{l} (K_1^{(i)}, K_2^{(i)}, u_1, u_2) := \mathsf{PRG}(W) \\ \mathsf{hk}_i = \mathsf{Hash.Gen}(1^\lambda;\, u_1) \\ \mathsf{EP}_i = \mathsf{iO}(1^\lambda, \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i];\, u_2) \end{array} \right\}$$

We also make use of a simulation-extractable NIZK, which we will denote by NIZK'. In the uniform setting, NIZK' will be $S(\lambda)$-deterministic whereas NIZK will be a-compatible for every $S(\lambda)$-computable sequence a.

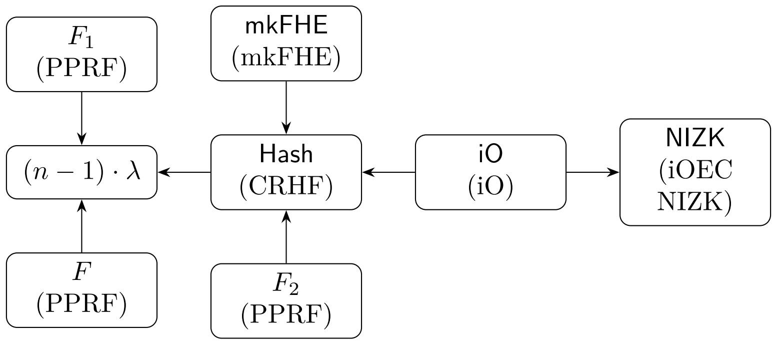


Figure 4.35: In this diagram, we describe the dependencies between the security parameters of the various subexponentially secure primitives. When a primitive is connected through an arrow to $(n-1)\cdot\lambda$, we mean that the advantage of any PPT adversary against the security of the primitive must be $\mathsf{negl}(\lambda)/2^{(n-1)\cdot\lambda}$. When a primitive is connected to Hash, we mean that the advantage of any PPT adversary against the security of the primitive must be $\mathsf{negl}(\lambda)/2^{t(\lambda)}$. We recall that $t(\lambda)$ denotes the length of the digests. When a primitive is connected to NIZK, we mean that the advantage of any PPT adversary against the security of the primitive must be $\mathsf{negl}(\lambda)/d(\lambda)$ where $d(\lambda)$ is the upper-bound on $|\mathsf{VPFE}_{\sigma,\tau_e}|$ in NIZK.

In other words, NIZK will be secure even if we leak the trapdoor τ' for NIZK'. The relation corresponding to NIZK' is the following.

$$\mathcal{R}_2 := \left\{ \begin{array}{c|l} \big((i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, & (K_1^{(i)}, K_2^{(i)}, u_1, u_2) := \mathsf{PRG}(W) \\ \mathsf{DP}_i, \pi_i, \sigma), & \big((i, \mathsf{hk}_i, \mathsf{EP}_i), W\big) \in \mathcal{R}_1 \\ (W, \rho_1, \rho_2)\big) & \mathsf{DP}_i = \mathsf{iO}(1^\lambda, \mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma];\ \rho_1) \\ & \pi_i = \mathsf{NIZK.Prove}\big(\sigma, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i), W;\ \rho_2\big) \end{array} \right\}$$

Above, we used L_1 and L_2 to denote the length of the randomness used to obfuscate DProg and to prove a statement using NIZK, respectively.

We rely on an injective and subexponentially secure indistinguishability obfuscator iO. We also use a multi-key FHE scheme mkFHE that satisfies subexponential reusable semi-malicious security. Let Hash be a subexponentially collision resistant hash function, outputting digests of length $t(\lambda)$. We use two subexponentially secure puncturable PRFs F_1 and F_2. The first one outputs a pseudorandom string of length equal to the randomness needed by $\mathcal{D}(1^\lambda)$. The second one outputs a pseudorandom string of length equal to the randomness needed by $\mathsf{mkFHE.Gen}$, $\mathsf{mkFHE.Enc}$, $\mathsf{mkFHE.Sim}_1$, $\mathsf{mkFHE.PartDec}$ and $\mathsf{mkFHE.Sim}_2$. Finally, we rely on a PRG mapping a λ-bit seed W into a pseudorandom string (K_1, K_2, u_1, u_2) where K_1 and K_2 are PRF keys for F_1 and F_2 respectively and u_1 and u_2 are as long as the randomness needed by $\mathsf{Hash.Gen}$ and the obfuscation of EProg respectively.

In Figure 4.36, we formalise the lossy mode of the distributed sampler. Notice that the construction relies on a subexponentially secure puncturable PRF F. Its outputs are pseudorandom strings that are as long as the randomness needed by $\mathcal{D}(1^\lambda)$. The construction relies also an ELF. We choose the domain of the latter so that all tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}$ fit into it. In Figure 4.37, we present the algorithms used for programmability and regularity. We describe the dependencies between the subexponentially secure primitives in Figure 4.35.

Theorem 4.7.1. Assume the existence of ELFs and the subexponential security of injective iO, multi-key FHE, collision resistant hash functions and puncturable PRFs. If AClass denotes the class of non-uniform adversaries, we also assume the existence of simulation-extractable NIZKs and almost everywhere extractable NIZKs with unstructured CRS. If instead AClass denotes the class of uniform adversaries, we additionally assume the existence of simulation-extractable NIZKs and simulation almost everywhere extractable NIZKs with no CRS.

Then, the construction in Figure 4.34 is a programmable lossy distributed sampler for $\mathcal{D}(1^\lambda)$ with security against AClass. If the ELF is regular, the lossy distributed sampler is also regular. If AClass denotes the

```
THE LOSSY MODE OF THE DISTRIBUTED SAMPLER
$\mathsf{LossySetup}\big(1^\lambda, q(\lambda)\big)$:
    1. $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{NIZK.SimSetup}(1^\lambda)$
    2. $(\sigma', \tau') \xleftarrow{\$} \mathsf{NIZK'.SimSetup}(1^\lambda)$
    3. $f \xleftarrow{\$} \mathsf{ELF.Gen}(M, q)$
    4. Output $\mathsf{crs} := (\sigma, \sigma')$ and $\zeta := (\sigma, \sigma', \tau_s, \tau_e, \tau', f)$.
$\mathsf{LossyGen}\big(1^\lambda, \mathsf{sid}, i, \zeta := (\sigma, \sigma', \tau_s, \tau_e, \tau', f)\big)$:
    1. $K \xleftarrow{\$} F.\mathsf{Gen}(1^\lambda)$
    2. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(1^\lambda)$
    3. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda)$
    4. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)
    5. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
    6. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f])$ (see Figure 4.31)
    7. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
    8. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
    9. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$ and $\xi_e := K$.
$\mathsf{Project}\Big(\zeta = (\sigma, \sigma', \tau_s, \tau_e, \tau', f), \big(U_j = (\mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \pi'_j)\big)_{j \in [n]}, \mathsf{sid}\Big)$:
    1. $\forall j \in [n]: \quad b_j \leftarrow \mathsf{NIZK'.Verify}\big(\sigma', \pi'_j, (j, \mathsf{sid}, \mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \sigma)\big)$
    2. If there exists $j \in [n]$ such that $b_j = 0$, output $\bot$.
    3. Output $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big)$.
$\mathsf{Extract}(\xi_e = K, z)$:
    1. If $z = \bot$, output $\bot$.
    2. $s \leftarrow F(K, z)$
    3. Output $\mathcal{D}(1^\lambda; s)$.
```

Figure 4.36: The lossy mode of the distributed sampler

```
PROGRAMMABILITY AND REGULARITY OF THE LOSSY DISTRIBUTED SAMPLER
$\mathsf{ProgGen}\big(1^\lambda, \mathsf{sid}, i, z, R, \zeta := (\sigma, \sigma', \tau_s, \tau_e, \tau', f)\big)$:
    1. $K \xleftarrow{\$} F.\mathsf{Gen}(1^\lambda)$
    2. $K^* \leftarrow F.\mathsf{Punct}(K, z)$
    3. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(1^\lambda)$
    4. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda)$
    5. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)
    6. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
    7. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_{\mathsf{Pr}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K^*, z, f, R])$ (see Figure 4.38)
    8. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
    9. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
    10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.
$\mathcal{Z}\big(\zeta = (\sigma, \sigma', \tau_s, \tau_e, \tau', f)\big)$:
    1. $b \xleftarrow{\$} \{0,1\}$
    2. If $b = 0$, output $\perp$.
    3. $x \xleftarrow{\$} [M]$
    4. Output $f(x)$.
```

Figure 4.37: Programmability and regularity of the lossy distributed sampler

class of non-uniform adversaries, the construction relies on an unstructured CRS whose size is independent of $\mathcal{D}(1^\lambda)$. If AClass denotes the class on uniform adversaries, the construction does not need any CRS.

Moreover, let $p'(\lambda)$ denote a polynomial upper-bounding the running times of $\mathsf{LossySetup}$, $\mathsf{LossyGen}$ and $\mathsf{LossySample}$. The advantage of an adversary $\mathcal{A}$ running in time at most $p(\lambda)$ in distinguishing between the lossy mode and the standard mode is

$$\mathsf{Adv}^{M,q}_{\mathsf{ELF},\mathcal{A}'}(\lambda) + \mathsf{negl}(\lambda),$$

where $\mathcal{A}'$ is an adversary running in time at most $p(\lambda)^2 \cdot p'(\lambda)$ and $\mathsf{Adv}^{M,q}_{\mathsf{ELF},\mathcal{A}'}(\lambda)$ denotes the advantage of $\mathcal{A}'$ in distinguishing between the injective mode of the ELF with domain size M and its lossy mode parametrised by $q(\lambda)$.

Proof. We observe that the second property of the lossy distributed sampler is a trivially implied by the ELF. Therefore, we focus on the first property, namely that for any polynomial $p(\lambda)$ and inverse polynomial function $\delta(\lambda)$, there exists a polynomial $q(\lambda)$ such that no adversary running in time at most $p(\lambda)$ can distinguish between the standard mode and the lossy mode parametrised by $q(\lambda)$ with advantage greater than $\delta(\lambda)$. We rely on an hybrid argument.

Hybrid 0. This hybrid corresponds to the game for lossy distributed samplers when the challenger uses the standard mode of operation.

We recap below the operations performed by $\mathsf{Setup}(1^\lambda)$ in this hybrid.

$\mathsf{DProg}_{\mathsf{Pr}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K^*, \hat{z}, f, R]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, a punctured PRF key K^*, the position $\hat{z}$, the ELF f, the ideal sample R.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

```
1. $\forall j \neq i: \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
2. $\forall j \neq i: \quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$[a]
3. If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \perp$, output $\perp$
4. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
5. $\forall j \neq i: \quad s_j \leftarrow F_1\big(K_1^{(j)}, y_j\big)$
6. $\forall j \in [n]: \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2\big(K_2^{(j)}, y_j\big)$
7. $z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big)$
8. $s \leftarrow F(K^*, z)$
9. $\hat{R} \leftarrow \mathcal{D}(1^\lambda; s)$
10. If $z = \hat{z}$, $\hat{R} \leftarrow R$
11. $(\phi, \mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Sim}_1(1^\lambda, i; r''_i)$
12. $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, \hat{R}, (s_j, r_j, r'_j)_{j \neq i}; \eta'_i\Big)$ (see bottom of Figure 4.29)
13. Output $d_i$
```

Figure 4.38: The unobfuscated decryption program for programmability

1. $\sigma \xleftarrow{\$} \mathsf{NIZK.Setup}(1^\lambda)$
2. $\sigma' \xleftarrow{\$} \mathsf{NIZK'.Setup}(1^\lambda)$
3. Output (σ, σ').

The operations used by $\mathsf{Gen}\big(1^\lambda, \mathsf{sid}, i, \mathsf{crs} = (\sigma, \sigma')\big)$ are instead the following.

1. $\rho_1 \xleftarrow{\$} \{0,1\}^{L_1(\lambda)}$
2. $\rho_2 \xleftarrow{\$} \{0,1\}^{L_2(\lambda)}$
3. $W \xleftarrow{\$} \{0,1\}^\lambda$
4. $(K_1^{(i)}, K_2^{(i)}, u_1, u_2) \leftarrow \mathsf{PRG}(W)$
5. $\mathsf{hk}_i \leftarrow \mathsf{Hash.Gen}(1^\lambda; u_1)$
6. $\mathsf{EP}_i \leftarrow \mathsf{iO}(1^\lambda, \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i]; u_2)$ (see Figure 4.28)

7. $\mathsf{DP}_i \leftarrow \mathsf{iO}(1^\lambda, \mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma];\ \rho_1)$ (see Figure 4.29)

8. $\pi_i \leftarrow \mathsf{NIZK.Prove}\big(\sigma, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i), W;\ \rho_2\big)$

9. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.Prove}\big(\sigma', (\mathsf{sid}, i, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma), (W, \rho_1, \rho_2)\big)$

10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i)$.

Hybrid 1. In this hybrid, we simulate the NIZK CRS σ' and the proof π'_i sent in each $\mathsf{NewSession}$ query. We recall that i is the index queried by the adversary. Formally, the CRS of the distributed sample is now generate as follows.

1. $\sigma \xleftarrow{\$} \mathsf{NIZK.Setup}(1^\lambda)$

2. $(\sigma', \tau') \xleftarrow{\$} \mathsf{NIZK'.SimSetup}(1^\lambda)$

3. Output (σ, σ').

The operations performed by the challenger in order to compute U_i in $\mathsf{NewSession}$ queries become the following.

1. $W \xleftarrow{\$} \{0, 1\}^\lambda$

2. $(K_1^{(i)}, K_2^{(i)}, u_1, u_2) \leftarrow \mathsf{PRG}(W)$

3. $\mathsf{hk}_i \leftarrow \mathsf{Hash.Gen}(1^\lambda;\ u_1)$

4. $\mathsf{EP}_i \leftarrow \mathsf{iO}(1^\lambda, \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i];\ u_2)$ (see Figure 4.28)

5. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma])$ (see Figure 4.29)

6. $\pi_i \xleftarrow{\$} \mathsf{NIZK.Prove}\big(\sigma, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i), W\big)$

7. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$

8. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

This hybrid is indistinguishable from the previous one due to the multi-theorem zero-knowledge of $\mathsf{NIZK'}$. In the reduction, we build a PPT adversary $\mathcal{B}$ that simulates the lossy distributed sampler game as in Hybrid 0 to an internal copy of $\mathcal{A}$. The adversary $\mathcal{B}$ models the CRS using the element σ' obtained from its challenger. In each $\mathsf{NewSession}$ query, it generates the proof π'_i by querying the corresponding statement $(i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)$ and the relative witness (W, ρ_1, ρ_2) to its challenger. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So if $\mathcal{A}$ succeeds at distinguishing between Hybrid 0 and Hybrid 1, $\mathcal{B}$ succeeds in breaking $\mathsf{NIZK'}$ too. Notice that if $\mathcal{A}$ is uniform $\mathcal{B}$ is uniform too.

Hybrid 2. In this hybrid, we change the reply to the sampling queries. In particular, for every U_l provided by the adversary in session sid, we compute

$$b_l \leftarrow \mathsf{NIZK'.Verify}\big(\sigma', \pi'_l, (l, \mathsf{sid}, \mathsf{hk}_l, \mathsf{EP}_l, \mathsf{DP}_l, \pi_l, \sigma)\big),$$
$$w_l \leftarrow \mathsf{NIZK'.Extract}\big(\tau', \pi'_l, (l, \mathsf{sid}, \mathsf{hk}_l, \mathsf{EP}_l, \mathsf{DP}_l, \pi_l, \sigma)\big).$$

If $b_l = 0$ or $w_l = \perp$, we provide the adversary with $\perp$. In all other cases, we provided it with $\mathsf{Sample}(U_1, \ldots, U_n, \mathsf{sid}, \mathsf{crs})$.

This hybrid is indistinguishable from the previous one due to the simulation-extractability of $\mathsf{NIZK'}$. Let $M(\lambda)$ denote a polynomial upper-bounding the number of sampling queries issued by the adversary. In the reduction, we build a PPT adversary $\mathcal{B}$ that simulates the lossy distributed sampler game as in Hybrid 0 to an internal copy of $\mathcal{A}$. The adversary $\mathcal{B}$ starts its execution by sampling $\iota \xleftarrow{\$} [M]$. It models the distributed sampler CRS using the element σ' obtained from its challenger. In each $\mathsf{NewSession}$ query, it generates the proof π'_i by querying the corresponding statement $(i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)$ to the simulation oracle. It replies to the first $\iota - 1$ sampling queries as in Hybrid 1. At the ι-th sampling query $\big(\mathsf{Sample}, \mathsf{sid}, (U_l)_{l \neq i}\big)$, however, $\mathcal{B}$ samples $j \xleftarrow{\$} [n] \setminus \{i\}$ and outputs $\pi'_j, (j, \mathsf{sid}, \mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \sigma)$ where $\mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j$ and π'_j are the elements in U_j.

If $\mathcal{A}$ succeeds at distinguishing, it must be that, with non-negligible probability $\epsilon(\lambda)$, one of its sampling queries contains a proof that verifies but cannot be extracted. With $1/M(\lambda)$ probability, the first proof of this kind will appear in the ι-th sampling query. So, $\mathcal{B}$ will succeed with probability at least $\epsilon(\lambda)/(M \cdot n)$. This contradicts the simulation-extractability of NIZK'. Notice that if $\mathcal{A}$ is uniform $\mathcal{B}$ is uniform too.

Hybrid 3. In this hybrid, we simulate the NIZK CRS σ and the proof π_i sent in each $\mathsf{NewSession}$ query. We also modify the decryption program DP_i. Instead of verifying the NIZKs that are given as input, the program extracts the witness from them using trapdoors $(\tau_e^j)_{j \neq i}$ we hardcode into it. When extraction of any NIZK fails, the program outputs $\perp$. Each of the trapdoor is obtained as $\tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$.

The distributed sampler CRS is now computed as follows.

1. $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{NIZK.SimSetup}(1^\lambda)$
2. $(\sigma', \tau') \xleftarrow{\$} \mathsf{NIZK'.SimSetup}(1^\lambda)$
3. Output $\mathsf{crs} := (\sigma, \sigma')$

The operations performed by the challenger in order to compute U_i in $\mathsf{NewSession}$ queries become the following.

1. $W \xleftarrow{\$} \{0,1\}^\lambda$
2. $(K_1^{(i)}, K_2^{(i)}, u_1, u_2) \leftarrow \mathsf{PRG}(W)$
3. $\mathsf{hk}_i \leftarrow \mathsf{Hash.Gen}(1^\lambda; u_1)$
4. $\mathsf{EP}_i \leftarrow \mathsf{iO}(1^\lambda, \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i]; u_2)$ (see Figure 4.28)
5. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
6. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}])$ (see Figure 4.32)
7. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
8. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
9. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

Claim 4.7.2. If AClass denotes the class of non-uniform adversaries, Hybrid 3 is indistinguishable from Hybrid 2 due to the subexponential security of iO and almost-everywhere extractability and the chosen-ID zero-knowledge of NIZK.

Proof of the claim. We proceed by means of a sequence of indistinguishable hybrids.

Hybrid' 0. This hybrid corresponds to the game in Hybrid 2.

Hybrid' 1. In this hybrid, we generate the CRS σ using $\mathsf{SimSetup}$. In the process, we obtain also the trapdoors τ_e and τ_s. Hybrid' 1 and Hybrid' 0 are indistinguishable thanks to the first property of almost everywhere extractable NIZKs.

Let $M(\lambda)$ be a polynomial upper-bound on the number of $\mathsf{NewSession}$ queries issued by $\mathcal{A}$. We proceed with $M(\lambda) + 1$ hybrids indexed by $\iota = 0, 1, \ldots, M(\lambda)$.

Hybrid' 2.ι. In this hybrid, we reply to the first ι $\mathsf{NewSession}$ queries using an obfuscation of DProg_1 (see Figure 4.32). Starting from the $(\iota+1)$-th query, we instead send an obfuscation of DProg (see Figure 4.29). We observe that Hybrid' 2.0 is identical to Hybrid' 1. For every $\iota \in [M]$, Hybrid' 2.ι is indistinguishable from Hybrid' 2.$(\iota - 1)$ thanks to Lemma 4.4.4. In the reduction, we build adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ that contradict Lemma 4.4.4. The adversary $\mathcal{B}_1$ receives σ and τ_e from the challenger. Then, it simulates the lossy distributed sampler game as in Hybrid' 2.$(\iota - 1)$ to an internal copy of $\mathcal{A}$. At the ι-th $\mathsf{NewSession}$ query, $\mathcal{B}$ generates hk_i and EP_i as usual, then, it outputs the circuit $\mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma]$ erasing the first two lines (i.e. the NIZKs verification) along with the identities $(\mathsf{sid}, l)_{l \neq i}$ and its internal state. The adversary $\mathcal{B}_2$ after receiving the obfuscated program DP_i and the internal state of $\mathcal{B}_1$ resumes the simulation of the lossy distributed sampler game with $\mathcal{A}$. It includes DP_i as part of the answer U_i to the ι-th $\mathsf{NewSession}$ query of $\mathcal{A}$. It produces the proofs π_i and π'_i in U_i as in Hybrid 2. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. Observe that if $\mathcal{A}$ distinguishes, then $(\mathcal{B}_1, \mathcal{B}_2)$ breaks Lemma 4.4.4.

Hybrid' 3. In this hybrid, we reply to all $\mathsf{NewSession}$ queries using an obfuscation of DProg_1 (see Figure 4.32). Notice that Hybrid' 3 is identical to Hybrid' 2.M.

Hybrid' 4. In this hybrid, we simulate the proof π_i in each $\mathsf{NewSession}$ query. Hybrid' 3 and Hybrid' 4 are indistinguishable under the chosen-ID zero-knowledge of NIZK. In the reduction, we build a PPT adversary $\mathcal{B}$ that simulates the game as in Hybrid' 3 to an internal copy of $\mathcal{A}$. The NIZK CRS σ is given by the zero-knowledge challenger. In every execution of $\mathsf{NewSession}$, the adversary $\mathcal{B}$ generates $\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi'_i$ as in Hybrid' 3. The only difference is that it obtains the extraction trapdoors hardcoded into DP_i by querying $\big(\mathsf{Trap}, (\mathsf{sid}, j)\big)$ for every $j \neq i$ to the chosen-ID zero-knowledge challenger. The proof π_i is obtained by querying

$$\big(\mathsf{Prove}, (\mathsf{sid}, i), x := (i, \mathsf{hk}_i, \mathsf{EP}_i), w := W\big)$$

to the chosen-ID zero-knowledge challenger. Observe that $\mathcal{B}$ never queries proofs and trapdoors for the same identity. At the end of its execution $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ breaks the chosen-ID zero-knowledge property of NIZK.

Observe that Hybrid' 4 is identical to Hybrid 3. This terminates the proof of the claim. ■

Claim 4.7.3. If AClass denotes the class of uniform adversaries, Hybrid 3 is indistinguishable from Hybrid 2 due to the subexponential security of iO and the τ'-disclosed simulation almost-everywhere extractability and zero-knowledge of NIZK.

Proof of the claim. We proceed by means of a sequence of indistinguishable hybrids.

Hybrid' 0. This hybrid corresponds to the game in Hybrid 2.

Hybrid' 1. In this hybrid, we generate the CRS σ using $\mathsf{SimSetup}$. In the process, we obtain also the trapdoors τ_e and τ_s. Hybrid' 1 and Hybrid' 0 are indistinguishable thanks to the first property of simulation almost everywhere extractable NIZKs.

Hybrid' 2. In this hybrid, we simulate the proof π_i in each $\mathsf{NewSession}$ query. Hybrid' 1 and Hybrid' 2 are indistinguishable under zero-knowledge of NIZK. In the reduction, we build a PPT adversary $\mathcal{B}$ that simulates the game as in Hybrid' 1 to an internal copy of $\mathcal{A}$. The NIZK CRS σ and the value $a_\lambda = \tau'$ is given by the zero-knowledge challenger. In every execution of $\mathsf{NewSession}$, the adversary $\mathcal{B}$ generates $\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi'_i$ as in Hybrid' 1. The proof π_i is obtained by querying

$$\big(\mathsf{Prove}, (\mathsf{sid}, i), x := (i, \mathsf{hk}_i, \mathsf{EP}_i), w := W\big)$$

to the zero-knowledge challenger. At the end of its execution $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ breaks the zero-knowledge property of NIZK.

Let $M(\lambda)$ be a polynomial upper-bound on the number of $\mathsf{NewSession}$ queries issued by $\mathcal{A}$. We proceed with $M(\lambda) + 1$ hybrids indexed by $\iota = 0, 1, \ldots, M(\lambda)$.

Hybrid' 3.ι. In this hybrid, we reply to the first ι $\mathsf{NewSession}$ queries using an obfuscation of DProg_1 (see Figure 4.32). Starting from the $(\iota + 1)$-th query, we instead send an obfuscation of DProg (see Figure 4.29). We observe that Hybrid' 3.0 is identical to Hybrid' 2. We show that for a random $\iota \xleftarrow{\$} [M]$, Hybrid' 3.$\iota$ is indistinguishable from Hybrid' 3.$(\iota - 1)$ thanks to [AWZ23, Lemma 3]. In the reduction, we build an adversary $\mathcal{B}$ that contradicts [AWZ23, Lemma 3]. The adversary $\mathcal{B}$ receives σ, τ_e and $a_\lambda = \tau'$ from the challenger. Then, it simulates the lossy distributed sampler game as in Hybrid' 3.$(\iota - 1)$ to an internal copy of $\mathcal{A}$. In each $\mathsf{NewSession}$ query, $\mathcal{B}$ generates the simulated proof π_i by querying its challenger. At the ι-th $\mathsf{NewSession}$ query, $\mathcal{B}$ generates hk_i and EP_i as usual, then, it provides its challenger with the circuit $\mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma]$ erasing the first two lines (i.e. the NIZKs verification) along with the identities

$(\mathsf{sid}, l)_{l \neq i}$. Then, it includes the answer DP_i from its challenger as part of the distributed sampler message U_i. Notice that $\mathcal{B}$ never queries for a simulated proof with identity (sid, j) for any $j \neq i$. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. Observe that $\mathcal{B}$ is uniform so if $\mathcal{A}$ distinguishes, then $\mathcal{B}$ breaks [AWZ23, Lemma 3].

Hybrid' 4. In this hybrid, we reply to all NewSession queries using an obfuscation of DProg_1 (see Figure 4.32). Notice that Hybrid' 4 is identical to Hybrid' 3.M.

Observe that Hybrid' 4 is identical to Hybrid 3. This terminates the proof of the claim. ■

We now proceed by repeating the following sequence of hybrids for $\iota = 1, \dots, M(\lambda), M(\lambda) + 1$ where $M(\lambda)$ is a polynomial upper-bound on the number of NewSession queries issued by the adversary. From now on, all pairs of hybrids can be proven indistinguishable by means of reductions to primitives that are secure against non-uniform adversaries. For this reason, we do not need to worry anymore on how the reduction obtains τ_e and τ_s.

Hybrid 4.ι.0. In this hybrid, the challenger starts its execution by generating a ELF $f \xleftarrow{\$} \mathsf{ELF.Gen}(M, M)$. Notice that the latter is in injective mode. The input space is chosen sufficiently big to embed all tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ into it without collisions.

The challenger generates the answer U_i to the first $\iota - 1$ NewSession queries as follows.

1. $K_1^{(i)} \xleftarrow{\$} F_1.\mathsf{Gen}(1^\lambda)$
2. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(1^\lambda)$
3. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda)$
4. $K \xleftarrow{\$} F.\mathsf{Gen}(1^\lambda)$
5. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)
6. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
7. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f])$ (see Figure 4.31)
8. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
9. $\pi_i' \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

All the remaining NewSession queries are answered as in Hybrid 3. Notice that when $\iota = 0$, Hybrid 4.ι.0 is identical to Hybrid 2. In all other cases, Hybrid 4.ι.0 is identical to Hybrid 4.$(\iota - 1)$.

Hybrid 4.ι.1. In this hybrid, we generate the pair $(\mathsf{hk}_i, \mathsf{EP}_i)$ in the ι-th NewSession query using full-entropy randomness instead of by expanding a PRG seed. All the rest remains as in the previous hybrid. This hybrid is indistinguishable from the previous one by the security of the PRG (the proof is an easy reduction). The operations performed by the challenger in order to compute U_i in the ι-th NewSession query become the following.

1. $K_1^{(i)} \xleftarrow{\$} F_1.\mathsf{Gen}(1^\lambda)$
2. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(1^\lambda)$
3. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda)$
4. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i])$ (see Figure 4.28)
5. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$

6. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}])$ (see Figure 4.32)

7. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$

8. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$

9. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

Remark 4.7.4. From now on, we will keep the generation of $K_1^{(i)}$, $K_2^{(i)}$ and hk_i implicit. Indeed, the procedure will remain as in Hybrid 4.ι.1. We do the same for π_i and π'_i and $(\tau_e^j)_{j\neq i}$.

Hybrid 4.ι.2. In this hybrid, we modify the answer to the sampling queries concerning the ι-th session. In particular, when the NIZKs verify and we succeed in extracting the witnesses $(W_j)_{j\neq i}$ from the messages $(U_j)_{j\neq i}$ provided by the adversary, we answer the query as follows.

1. $\forall j \in [n] : y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l\neq j}\big)$

2. $\forall j \neq i : (K_1^{(j)}, K_2^{(j)}, u_1^j, u_2^j) \leftarrow \mathsf{PRG}(W_j)$

3. $\forall j \in [n] : s_j \leftarrow F_1(K_1^{(j)}, y_j)$

4. $R \leftarrow \mathcal{D}(1^\lambda; s_1 \oplus \cdots \oplus s_n)$

5. Provide R to the adversary.

Observe that this hybrid is identical to the previous one by the perfect correctness of multi key FHE and the perfect correctness and injectivity of iO. The latter is needed to argue that EP_j univocally determines $K_1^{(j)}$ and $K_2^{(j)}$.

Hybrid 4.ι.3. In this hybrid, we change both the encryption program EP_i and the decryption program DP_i generated for the ι-th $\mathsf{NewSession}$ query, switching to an obfuscation of $\mathsf{EProg}_{\mathsf{Ls}}$ (see Figure 4.30) and DProg_2 (see Figure 4.33). All the rest remains as in the previous hybrid. Notice that, at this point, we removed $K_1^{(i)}$ from the code of EP_i. The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following.

1. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)

2. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_2[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}])$ (see Figure 4.33)

3. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

Claim 4.7.5. Assuming the subexponential security of iO, of Hash, of the puncturable PRF F_2 and of multi-key FHE, no PPT adversary can distinguish between Hybrid 4.ι.2 and Hybrid 4.ι.3.

Proof of the claim. We select the security parameter of the subexponentially collision resistance hash function so that, for any PPT adversary,

$$2^{2\lambda\cdot(n-1)} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathsf{CR}}(\lambda) = \mathsf{negl}(\lambda).$$

Let Ω be the set of all the tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ such that each $(\mathsf{hk}_j, \mathsf{EP}_j)$ is generated by expanding a λ-bit PRG seed as in Figure 4.34. Observe that $|\Omega| \leq 2^{\lambda\cdot(n-1)}$. We conclude that with overwhelming probability over hk_i, there exist no collisions in Ω. Otherwise, the adversary that simply outputs two random elements in Ω would break the above assumption. We can therefore, prove indistinguishability conditioned on this event occurring.

We proceed once again by means of a series of hybrids. Their number will however be superpolynomial. Specifically, we consider the set of all possible digests $\{0,1\}^{t(\lambda)}$ and we set $\hat{y}$ to its minimum according to the lexicographical order. We proceed through the following sequence of hybrids gradually increasing $\hat{y}$ until it reaches the maximum in $\{0,1\}^{t(\lambda)}$.

$\mathsf{EProg}^0[K_1^{(i)}, K_2^{(i)}, i, \hat{y}]$

Hard-coded. The PPRF keys $K_1^{(i)}$ and $K_2^{(i)}$, the index i, the hybrid index $\hat{y}$.
Input. A digest $y \in \{0,1\}^{t(\lambda)}$.

1. If $y <_{\mathsf{lex}} \hat{y}$: $(\mathsf{pk}_i, c_i) \leftarrow \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i](y)$ (see Figure 4.30)
2. Otherwise, $(\mathsf{pk}_i, c_i) \leftarrow \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i](y)$ (see Figure 4.28)
3. Output (pk_i, c_i)

Figure 4.39: Hybrid $\hat{y}$.0: the unobfuscated encryption program of party P_i

Hybrid $\hat{y}$.0. In this hybrid, we modify the programs EP_i and DP_i sent in the ι-th NewSession query. In particular, instead of obfuscating EProg (see Figure 4.28), we obfuscate EProg^0 (see Figure 4.39). Similarly, instead of obfuscating DProg_1 (see Figure 4.32), we obfuscate DProg_1^0 (see Figure 4.40). In both these programs, we hardcode $\hat{y}$. If the digest input in EP_i is strictly lexicographically smaller than $\hat{y}$, the encryption program performs the same operations as $\mathsf{EProg}_{\mathsf{Ls}}$ (see Figure 4.30), otherwise, it behaves as EProg (see Figure 4.28). Similarly, if the hash y_i of the tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ input in DP_i is strictly lexicographically smaller than $\hat{y}$, the decryption program performs the same operations as DProg_2 (see Figure 4.33), otherwise, it behave as DProg_1 (see Figure 4.32).

Notice that when $\hat{y}$ is the minimum of $\{0,1\}^{t(\lambda)}$, the new programs have exactly the same input-output behaviour as EProg and DProg_1. We conclude that, when $\hat{y}$ is minimum, Hybrid $\hat{y}$.0 is indistinguishable from Hybrid 4.ι.1, by the security of iO. If $\hat{y}$ is not the minimum, this hybrid will be identical to the previous one (i.e. Hybrid $\hat{y}'$.7 where $\hat{y}'$ is the previous value of $\hat{y}$).

The operations performed by the challenger in order to compute U_i in the ι-th NewSession query become the following.

1. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}^0[K_1^{(i)}, K_2^{(i)}, i, \hat{y}])$ (see Figure 4.39)
2. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_1^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}])$ (see Figure 4.40)
3. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Hybrid $\hat{y}$.1. In this hybrid, we modify the encryption program EP_i sent in the ι-th NewSession query. In particular, instead of obfuscating EProg^0 (see Figure 4.39), we obfuscate EProg^1 (see Figure 4.41). In the latter, the PPRF key $K_2^{(i)}$ will now be punctured in position $\hat{y}$. Furthermore, we store into EP_i, the pair

$$(\widehat{\mathsf{pk}}_i, \hat{c}_i) \leftarrow \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i](\hat{y}).$$

When $\hat{y}$ is provided as input, EP_i will directly output $(\widehat{\mathsf{pk}}_i, \hat{c}_i)$. The rest remains as before. Since the input-output behaviour of EProg^1 is the same as for EProg^0, no adversary can distinguish between this hybrid and the previous one under the security of iO.

The operations performed by the challenger in order to compute U_i in the ι-th NewSession query become the following.

1. $K_2^* \leftarrow F_2.\mathsf{Punct}(K_2^{(i)}, \hat{y})$
2. $s_i \leftarrow F_1(K_1^{(i)}, \hat{y})$
3. $(r_i, r_i', r_i'', \eta_i, \eta_i') \leftarrow F_2(K_2^{(i)}, \hat{y})$
4. $(\widehat{\mathsf{pk}}_i, \widehat{\mathsf{sk}}_i) \leftarrow \mathsf{mkFHE.Gen}(1^\lambda, i; r_i)$

$\mathsf{DProg}_1^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the PPRF key $K_1^{(i)}$, the hybrid index $\hat{y}$.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

1. If $\mathsf{Hash}\big(\mathsf{hk}_i, (\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}\big) <_{\mathsf{lex}} \hat{y}$:
   $d_i \leftarrow \mathsf{DProg}_2[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\Big)$ (see Figure 4.33)
2. Otherwise,
   $d_i \leftarrow \mathsf{DProg}_1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\Big)$ (see Figure 4.32)
3. Output d_i

Figure 4.40: Hybrid $\hat{y}$.0: the unobfuscated decryption program of party P_i

$\mathsf{EProg}^1[K_1^{(i)}, K_2^{(i)}, i, \hat{y}, \widehat{\mathsf{pk}}, \hat{c}]$

Hard-coded. The PPRF keys $K_1^{(i)}$ and $K_2^{(i)}$, the index i, the hybrid index $\hat{y}$, the public key $\widehat{\mathsf{pk}}$ and the ciphertext $\hat{c}$.
Input. A digest $y \in \{0,1\}^{t(\lambda)}$.

1. If $y <_{\mathsf{lex}} \hat{y}$: $(\mathsf{pk}_i, c_i) \leftarrow \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i](y)$ (see Figure 4.30)
2. If $y = \hat{y}$: $(\mathsf{pk}_i, c_i) \leftarrow (\widehat{\mathsf{pk}}, \hat{c})$
3. Otherwise, $(\mathsf{pk}_i, c_i) \leftarrow \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i](y)$ (see Figure 4.28)
4. Output (pk_i, c_i)

Figure 4.41: Hybrid $\hat{y}$.1: the unobfuscated encryption program of party P_i

5. $\hat{c}_i \leftarrow \mathsf{mkFHE.Enc}(\widehat{\mathsf{pk}}_i, s_i; r_i')$
6. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}^1[K_1^{(i)}, K_2^*, i, \hat{y}, \widehat{\mathsf{pk}}_i, \hat{c}_i])$ (see Figure 4.41)
7. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_1^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}])$ (see Figure 4.40)
8. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Hybrid $\hat{y}$.2. In this hybrid, we modify the decryption program DP_i sent in the ι-th $\mathsf{NewSession}$ query. In particular, instead of obfuscating DProg_1^0 (see Figure 4.40), we obfuscate DProg_1^1 (see Figure 4.42). In the latter, the PPRF key $K_2^{(i)}$ will now be punctured in position $\hat{y}$. Now, there are two cases: if there exists a tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} \in \Omega$ such that $\mathsf{Hash}\big(\mathsf{hk}_i, (\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}\big) = \hat{y}$, then, we store into DP_i, the partial decryption

$$\hat{d}_i \leftarrow \mathsf{DProg}_1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}]\Big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}\Big).$$

Notice that, thanks to the subexponential security of Hash, the tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ is univocally defined. If instead, the tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ we are looking for does not exist, we set $\hat{d}_i \leftarrow \bot$. When the hash of the input collides with $\hat{y}$, DP_i will now directly output $\hat{d}_i$. The rest remains as before. Observe that the input-output behaviour of DProg_1^1 is the same as for DProg_1^0. Indeed, the input-output behaviour can change only if the input consists of a tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ that hashes to $\hat{y}$. We know that there exists at most one such tuple in Ω and, in that case, the output of both DProg_1^1 and DProg_1^0 is the hardcoded value $\hat{d}_i$. When $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ in not in Ω, then both DProg_1^1 and DProg_1^0 output $\perp$ as the extraction of the witness from the NIZKs will always fails. We conclude that no adversary can distinguish between this hybrid and the previous one under the security of iO.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ denotes the tuple in Ω that hashes to $\hat{y}$ under hk_i. If such tuple does not exist, we ignore steps 7-10 below, and we simply set $\hat{d}_i \leftarrow \perp$.

1. $K_2^* \leftarrow F_2.\mathsf{Punct}(K_2^{(i)}, \hat{y})$
2. $s_i \leftarrow F_1(K_1^{(i)}, \hat{y})$
3. $(r_i, r_i', r_i'', \eta_i, \eta_i') \leftarrow F_2(K_2^{(i)}, \hat{y})$
4. $(\widehat{\mathsf{pk}}_i, \widehat{\mathsf{sk}}_i) \leftarrow \mathsf{mkFHE.Gen}(1^\lambda, i;\, r_i)$
5. $\hat{c}_i \leftarrow \mathsf{mkFHE.Enc}(\widehat{\mathsf{pk}}_i, s_i;\, r_i')$
6. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}^1[K_1^{(i)}, K_2^*, i, \hat{y}, \widehat{\mathsf{pk}}_i, \hat{c}_i])$ (see Figure 4.41)
7. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
8. $\forall j \in [n]: \quad (\mathsf{pk}_j, c_j) \leftarrow \mathsf{EP}_j(y_j)$
9. $C \leftarrow \mathsf{mkFHE.Eval}\big(\tilde{\mathcal{D}}, \mathsf{pk}_1, c_1, \ldots, \mathsf{pk}_n, c_n\big)$ (see bottom of Figure 4.29)
10. $\hat{d}_i \leftarrow \mathsf{mkFHE.PartDec}\Big(C, (\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n), i, \widehat{\mathsf{sk}}_i;\, \eta_i\Big)$
11. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_1^1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}, \hat{d}_i])$ (see Figure 4.42)
12. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Hybrid $\hat{y}$.3. In this hybrid, in the ι-th $\mathsf{NewSession}$ query, we generate $\widehat{\mathsf{pk}}_i$, $\hat{c}_i$ and $\hat{d}_i$ by inputting full-entropy randomness r_i, r_i' and η_i into $\mathsf{mkFHE.Gen}$, $\mathsf{mkFHE.Enc}$ and $\mathsf{mkFHE.PartDec}$ instead of producing it using $F_2(K_2^{(i)}, \hat{y})$. This hybrid is indistinguishable from the previous one by the security of puncturable PRFs.

Remark 4.7.6. Observe that since the number of pairs (i, hk_i) is finite, for every $\lambda \in \mathbb{N}$, there exists one that maximises the advantage of the adversary in distinguishing this hybrid from the previous one. We call the corresponding hash key "the worst hash key". Of course the worst hash key is chosen among those for which there exist no collisions in Ω. In the reduction to the security of puncturable PRFs, we can assume that the new adversary (the one attacking F_2) obtains $\hat{y}$, the worst hash key $\widehat{\mathsf{hk}}_i$ and the tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ in Ω that is hashed to $\hat{y}$ (if such tuple exists) as part of its non-uniform advice. The new adversary will simulate the indistinguishability game between Hybrid $\hat{y}$.2 and Hybrid $\hat{y}$.3 using these values.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ denotes the tuple in Ω that hashes to $\hat{y}$ under hk_i. If such tuple does not exist, we ignore steps 6-9 below, and we simply set $\hat{d}_i \leftarrow \perp$.

1. $K_2^* \leftarrow F_2.\mathsf{Punct}(K_2^{(i)}, \hat{y})$
2. $s_i \leftarrow F_1(K_1^{(i)}, \hat{y})$

$\mathsf{DProg}_1^1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \omega, \hat{d}_i]$

Hard-coded. The index i of the party, the session identity sid, a punctured PRF key K_2^*, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the PPRF key $K_1^{(i)}$, the hybrid index $\hat{y}$, the partial decryption $\hat{d}_i$.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

```
1. If $\mathsf{Hash}\big(\mathsf{hk}_i, (\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}\big) <_{\mathsf{lex}} \hat{y}$:
   $d_i \leftarrow \mathsf{DProg}_2[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\Big)$ (see Figure 4.33)
2. If $\mathsf{Hash}\big(\mathsf{hk}_i, (\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}\big) = \hat{y}$ :
   (a) $\forall j \neq i : \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
   (b) $\forall j \neq i : \quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$ [a]
   (c) If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \perp$, output $\perp$
   (d) $d_i \leftarrow \hat{d}_i$
3. Otherwise,
   $d_i \leftarrow \mathsf{DProg}_1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\Big)$ (see Figure 4.32)
4. Output $d_i$
```

Figure 4.42: Hybrid $\hat{y}$.2: the unobfuscated decryption program of party P_i

3. $(\widehat{\mathsf{pk}}_i, \widehat{\mathsf{sk}}_i) \xleftarrow{\$} \mathsf{mkFHE.Gen}(1^\lambda, i)$
4. $\hat{c}_i \xleftarrow{\$} \mathsf{mkFHE.Enc}(\widehat{\mathsf{pk}}_i, s_i)$
5. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}^1[K_1^{(i)}, K_2^*, i, \hat{y}, \widehat{\mathsf{pk}}_i, \hat{c}_i])$ (see Figure 4.41)
6. $\forall j \in [n] : \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
7. $\forall j \in [n] : \quad (\mathsf{pk}_j, c_j) \leftarrow \mathsf{EP}_j(y_j)$
8. $C \leftarrow \mathsf{mkFHE.Eval}\big(\tilde{\mathcal{D}}, \mathsf{pk}_1, c_1, \ldots, \mathsf{pk}_n, c_n\big)$ (see bottom of Figure 4.29)
9. $\hat{d}_i \xleftarrow{\$} \mathsf{mkFHE.PartDec}\Big(C, (\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n), i, \widehat{\mathsf{sk}}_i\Big)$
10. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_1^1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}, \hat{d}_i])$ (see Figure 4.42)
11. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Hybrid $\hat{y}$.4. In this hybrid, in the ι-th $\mathsf{NewSession}$ query, instead of computing $\widehat{\mathsf{pk}}_i$, $\hat{c}_i$ and $\hat{d}_i$ using $\mathsf{mkFHE.Gen}$, $\mathsf{mkFHE.Enc}$ and $\mathsf{mkFHE.PartDec}$, we simulate them. Notice that the multi-key FHE simulator $\mathsf{mkFHE.Sim}_2$ needs to receive the inputs and the randomness used by the other parties. We retrieve the latter by expanding the PRF keys $K_1^{(j)}$ and $K_2^{(j)}$ hidden in EP_j (we recall that $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ denotes the only tuple in Ω that is hashed to $\hat{y}$ under hk_i). Since the obfuscation scheme is injective, $K_1^{(j)}$ and $K_2^{(j)}$ are univocally defined.

This hybrid is indistinguishable from the previous one under the reusable semi-malicious security of multi-key FHE. For the reduction, we use the same trick as in Hybrid $\hat{y}$.3, i.e. we provide the adversary with

$\hat{y}$, the worst hash key, the only preimage $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ of $\hat{y}$ in Ω along with the PRF keys $K_1^{(j)}, K_2^{(j)}$ hidden in each EP_j as part of the non-uniform advice string.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ denotes the tuple in Ω that hashes to $\hat{y}$ under hk_i. For every $j \neq i$, we use $K_1^{(j)}$ and $K_2^{(j)}$ to denote the PPRF keys hidden in EP_j. If such tuple does not exist, we ignore steps 4-8 below, and we simply set $\hat{d}_i \leftarrow \perp$.

1. $K_2^* \leftarrow F_2.\mathsf{Punct}(K_2^{(i)}, \hat{y})$
2. $(\phi, \widehat{\mathsf{pk}}_i, \hat{c}_i) \xleftarrow{\$} \mathsf{mkFHE.Sim}_1(1^\lambda, i)$
3. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}^1[K_1^{(i)}, K_2^*, i, \hat{y}, \widehat{\mathsf{pk}}_i, \hat{c}_i])$ (see Figure 4.41)
4. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l\neq j}\big)$
5. $\forall j \in [n]: \quad s_j \leftarrow F_1(K_1^{(j)}, y_j)$
6. $\forall j \neq i: \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2(K_2^{(j)}, y_j)$
7. $\hat{R} \leftarrow \mathcal{D}(1^\lambda; s_1 \oplus \cdots \oplus s_n)$
8. $\hat{d}_i \xleftarrow{\$} \mathsf{mkFHE.Sim}_2\Big(\phi, d\tilde{i}st, \hat{R}, (s_j, r_j, r'_j)_{j\neq i}\Big)$
9. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_1^1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}, \hat{y}, \hat{d}_i])$ (see Figure 4.42)
10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

Hybrid $\hat{y}$.5. In this hybrid, in the ι-th $\mathsf{NewSession}$ query, we generate $\widehat{\mathsf{pk}}_i$, $\hat{c}_i$ and $\hat{d}_i$ using the randomness generated by $F_2(K_2^{(i)}, \hat{y})$. This hybrid is indistinguishable from the previous one under the security of the puncturable PRF.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ denotes the tuple in Ω that hashes to $\hat{y}$ under hk_i. For every $j \neq i$, we use $K_1^{(j)}$ and $K_2^{(j)}$ to denote the PPRF keys hidden in EP_j. If such tuple does not exist, we ignore steps 5-9 below, and we simply set $\hat{d}_i \leftarrow \perp$.

1. $K_2^* \leftarrow F_2.\mathsf{Punct}(K_2^{(i)}, \hat{y})$
2. $(r_i, r'_i, r''_i, \eta_i, \eta'_i) \leftarrow F_2(K_2^{(i)}, \hat{y})$
3. $(\phi, \widehat{\mathsf{pk}}_i, \hat{c}_i) \leftarrow \mathsf{mkFHE.Sim}_1(1^\lambda, i;\ r''_i)$
4. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}^1[K_1^{(i)}, K_2^*, i, \hat{y}, \widehat{\mathsf{pk}}_i, \hat{c}_i])$ (see Figure 4.41)
5. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l\neq j}\big)$
6. $\forall j \in [n]: \quad s_j \leftarrow F_1(K_1^{(j)}, y_j)$
7. $\forall j \neq i: \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2(K_2^{(j)}, y_j)$
8. $\hat{R} \leftarrow \mathcal{D}(1^\lambda; s_1 \oplus \cdots \oplus s_n)$
9. $\hat{d}_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, d\tilde{i}st, \hat{R}, (s_j, r_j, r'_j)_{j\neq i};\ \eta'_i\Big)$
10. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_1^1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}, \hat{y}, \hat{d}_i])$ (see Figure 4.42)

11. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

Hybrid $\hat{y}$.6. In this hybrid, we change the encryption program EP_i sent in the ι-th $\mathsf{NewSession}$ query. In particular, we switch back to an obfuscation of EProg^0 (see Figure 4.39). This time, however, instead of hardcoding $\hat{y}$, we hardcode the next element in $\{0,1\}^{t(\lambda)}$. We denote it by $\hat{y}'$[11]. The input-output behaviour of EP_i remains the same as in the previous hybrid, so we can argue for indistinguishability based on the security of iO.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ denotes the tuple in Ω that hashes to $\hat{y}$ under hk_i. For every $j \neq i$, we use $K_1^{(j)}$ and $K_2^{(j)}$ to denote the PPRF keys hidden in EP_j. If such tuple does not exist, we ignore steps 5-9 below, and we simply set $\hat{d}_i \leftarrow \perp$.

1. $K_2^* \leftarrow F_2.\mathsf{Punct}(K_2^{(i)}, \hat{y})$
2. $(r_i, r'_i, r''_i, \eta_i, \eta'_i) \leftarrow F_2(K_2^{(i)}, \hat{y})$
3. $(\phi, \widehat{\mathsf{pk}}_i, \hat{c}_i) \leftarrow \mathsf{mkFHE.Sim}_1(1^\lambda, i;\, r''_i)$
4. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}^0[K_1^{(i)}, K_2^{(i)}, i, \hat{y}'])$ (see Figure 4.39)
5. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
6. $\forall j \in [n]: \quad s_j \leftarrow F_1(K_1^{(j)}, y_j)$
7. $\forall j \neq i: \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2(K_2^{(j)}, y_j)$
8. $\hat{R} \leftarrow \mathcal{D}(1^\lambda;\, s_1 \oplus \cdots \oplus s_n)$
9. $\hat{d}_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{dist}, \hat{R}, (s_j, r_j, r'_j)_{j \neq i};\, \eta'_i\Big)$
10. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_1^1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}, \hat{d}_i])$ (see Figure 4.42)
11. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

Hybrid $\hat{y}$.7. In this hybrid, we change the decryption program DP_i sent in the ι-th $\mathsf{NewSession}$ query. In particular, we switch back to an obfuscation of DProg_1^0 (see Figure 4.40). This time, however, instead of hardcoding $\hat{y}$, we hardcode $\hat{y}'$[12]. The input-output behaviour of DP_i remains the same as in the previous hybrid, so we can argue for indistinguishability based on the security of iO.

We observe that the differing-inputs can only consist of tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ that hash to $\hat{y}$. Any of these tuples that does not belong to Ω is mapped to $\perp$ by DP_i in both hybrids. Indeed, the extraction of the witnesses from the proofs will always fail. The only differing input can therefore be the only preimage of $\hat{y}$ in Ω, if this exists. However, even for such input, DP_i behaves the same in the two hybrids.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following.

1. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}^0[K_1^{(i)}, K_2^{(i)}, i, \hat{y}'])$ (see Figure 4.39)
2. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_1^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}'])$ (see Figure 4.40)
3. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

[11] If $\hat{y}$ is already the maximum of $\{0,1\}^{t(\lambda)}$, we augment $\{0,1\}^{t(\lambda)}$ with an imaginary element that is strictly greater than all other values. Let $\hat{y}'$ be such value.

[12] If $\hat{y}$ is already the maximum of $\{0,1\}^{t(\lambda)}$, we augment $\{0,1\}^{t(\lambda)}$ with an imaginary element that is strictly greater than all other values. Let $\hat{y}'$ be such value.

When $\hat{y}$ reaches the maximum in $\{0,1\}^{t(\lambda)}$, Hybrid $\hat{y}.7$ is indistinguishable from Hybrid $4.\iota.3$ due to the security of iO. Indeed, in Hybrid $\hat{y}.7$, for any input, EP_i computes the output using $\mathsf{EProg}_{\mathsf{Ls}}$, whereas DP_i computes the output using DProg_2. This terminates the proof of the claim. ■

Hybrid 4.ι.4. In this hybrid, we change the decryption program DP_i generated in the ι-th NewSession query, switching to an obfuscation of $\mathsf{DProg}_{\mathsf{Ls}}$ (see Figure 4.31). In the latter, we embed the ELF f used to reply to the first $\iota - 1$ NewSession queries, and a random PPRF key K.

The operations performed by the challenger in order to compute U_i in the ι-th NewSession query become the following.

1. $K \xleftarrow{\$} F.\mathsf{Gen}(1^\lambda)$
2. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)
3. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K, f])$ (see Figure 4.31)
4. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

In the hybrid, we also change the reply to the sampling queries concerning the ι-th session.In particular, when the NIZKs verify and we succeed in extracting the witnesses from the messages $(U_j)_{j\neq i}$ provided by the adversary, we answer the sampling query as follows.

1. $z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big)$
2. $s \leftarrow F(K, z)$
3. $R \leftarrow \mathcal{D}(1^\lambda;\, s)$
4. Provide R to the adversary.

Claim 4.7.7. Assuming the subexponential security of iO, of the puncturable PRFs F and F_1 and the subexponential collision intractability of the hash function, no PPT adversary can distinguish between Hybrid $4.\iota.3$ and Hybrid $4.\iota.4$.

Proof of the claim. We select the security parameter of the subexponentially collision resistance hash function so that, for any PPT adversary,

$$2^{2\lambda\cdot(n-1)} \cdot \mathsf{Adv}_{\mathsf{CR}}^{\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda).$$

Observe that $|\Omega| = 2^{\lambda\cdot(n-1)}$. We conclude that with overwhelming probability over hk_i, there exist no collisions in Ω. Otherwise, the adversary that simply outputs two random elements in Ω would break the above assumption. We can therefore, prove indistinguishability conditioned on this event occurring.

We proceed once again through a series of indistinguishable hybrids. Their number will be superpolynomial. In particular, we repeat the following sequence for every $\omega \in \Omega$ (Ω was defined in the proof of Claim 4.7.5). We initially set ω to be the minimum in Ω according to the lexicographical order. Then, we gradually increment it until we reach the maximum. In the proof, we use ϖ to denote the tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}$ where $(\mathsf{hk}_i, \mathsf{EP}_i)$ are the hash key and the encryption program chosen by party P_i, and $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i} = \omega$.

Hybrid ω.0. In this hybrid, we modify the decryption program DP_i sent in the ι-th NewSession query, switching to an obfuscation of DProg_2^0 (see Figure 4.43). The new program will have the hybrid index ω hardcoded. Whenever the input $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ is strictly smaller than ω according to the lexicographical order, DP_i will compute the output using $\mathsf{DProg}_{\mathsf{Ls}}$ (see Figure 4.31), otherwise it will use DProg_2 (see Figure 4.33). Also the answer to the sampling queries is modified: if the adversary queries messages $(U_j)_{j\neq i}$ for the ι-th session such that $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ is strictly smaller than ω, the challenger replies as in Hybrid $4.\iota.4$. In the other cases, it replies as in Hybrid $4.\iota.3$.

When ω is the minimum in Ω, Hybrid $\omega.0$ is indistinguishable from Hybrid $4.\iota.3$ by the security of obfuscation. Indeed, all $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ that are strictly smaller than ω contain a malformed pair $(\mathsf{hk}_j, \mathsf{EP}_j)$. Since the NIZK extraction fails, DProg_2 always outputs $\perp$ in these cases. The same does DP_i in Hybrid

$\mathsf{DProg}_2^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}, K, f, \omega]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key K_2, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j\neq i}$, the PPRF key $K_1^{(i)}$, the PPRF key K, the ELF f, the hybrid index ω.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$.

1. If $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i} <_{\mathsf{lex}} \omega$:
   $d_i \leftarrow \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K, f]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}\Big)$ (see Figure 4.31)
2. Otherwise,
   $d_i \leftarrow \mathsf{DProg}_2[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}\Big)$ (see Figure 4.33)
3. Output d_i

Figure 4.43: Hybrid $\omega.0$: the unobfuscated decryption program of party P_i

4.ι.4. Clearly, the programs behave identically when $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i} \geq_{\mathsf{lex}} \omega$. When ω in not the minimum in Ω, instead, this hybrid is identical to the previous one, i.e. Hybrid $\widehat{\omega}.4$ where $\widehat{\omega}$ is the previous value of ω.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following.

1. $K \overset{\$}{\leftarrow} F.\mathsf{Gen}(1^\lambda)$
2. $\mathsf{EP}_i \overset{\$}{\leftarrow} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)
3. $\mathsf{DP}_i \overset{\$}{\leftarrow} \mathsf{iO}(1^\lambda, \mathsf{DProg}_2^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}, K, f, \omega])$ (see Figure 4.43)
4. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Hybrid ω.1. In this hybrid, we modify the decryption program DP_i sent in the ι-th $\mathsf{NewSession}$ query, switching to an obfuscation of DProg_2^1 (see Figure 4.44). The keys $K_1^{(i)}$ and K stored in DP_i will now be punctured in $y_i = \mathsf{Hash}(\mathsf{hk}_i, \omega)$ and $f(\overline{\omega})$, respectively. We also hardcode the value $\hat{R}$ that DProg_2 feeds into the partial decryption simulator when ω is given as input. The behaviour of DP_i remains as in the previous hybrid, with the exception that when $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i} = \omega$ and the NIZK extraction succeeds, the program directly feeds the hardcoded $\hat{R}$ into the partial decryption simulator. We highlight that, the modified program will never need to evaluate $K_1^{(i)}$ in y_i. Indeed, by the subexponential collision resistance of the hash function, the only preimage of y_i in Ω will be ω. Moreover, K will never be evaluated in $f(\overline{\omega})$ as the ELF is set in injective mode. Since the input-output behaviour of DP_i has not changed (notice that all the witnesses for the NIZK statement lead to the same sk_i by the injectivity of iO), this hybrid and the previous one are indistinguishable under the security of iO.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, we rewrite ω as $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$. We denote the first PRF key hardcoded in EP_j by $K_1^{(j)}$ (we recall that we are only considering EP_js that are well-formed). For the reduction, since ω is fixed and the adversary is non-uniform, we can assume it knows $K_1^{(j)}$ for every $j \neq i$ (the latter is uniquely determined by ω by the injectivity of iO).

1. $K \overset{\$}{\leftarrow} F.\mathsf{Gen}(1^\lambda)$
2. $\mathsf{EP}_i \overset{\$}{\leftarrow} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)
3. $z \leftarrow f(\overline{\omega})$

```
$\mathsf{DProg}_2^1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^*, K^*, f, \omega, \hat{R}]$

Hard-coded. The index $i$ of the party, the session identity $\mathsf{sid}$, a PPRF key $K_2^{(i)}$, the encryption program $\mathsf{EP}_i$, the hash key $\mathsf{hk}_i$, the extractable NIZK CRS $\sigma$ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the punctured PRF keys $K_1^*$ and $K^*$, the ELF $f$, the hybrid index $\omega$, the sample $\hat{R}$.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

  1. If $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} <_{\mathsf{lex}} \omega$:
     $d_i \leftarrow \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K^*, f]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\Big)$ (see Figure 4.31)
  2. If $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} = \omega$ :
     (a) $\forall j \neq i :\quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
     (b) $\forall j \neq i :\quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$ [a]
     (c) If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \perp$, output $\perp$
     (d) $\forall j \in [n] :\quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
     (e) $\forall j \neq i :\quad s_j \leftarrow F_1(K_1^{(j)}, y_j)$
     (f) $\forall j \in [n] :\quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2(K_2^{(j)}, y_j)$
     (g) $(\phi, \mathsf{pk}_i, c_i) \leftarrow \mathsf{mkFHE.Sim}_1(1^\lambda, i; r''_i)$
     (h) $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, \hat{R}, (s_j, r_j, r'_j)_{j \neq i}; \eta'_i\Big)$ (see bottom of Figure 4.29)
  3. Otherwise,
     $d_i \leftarrow \mathsf{DProg}_2[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^*]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\Big)$ (see Figure 4.33)
  4. Output $d_i$
```

Figure 4.44: Hybrid ω.1: the unobfuscated decryption program of party P_i

4. $K^* \leftarrow F.\mathsf{Punct}(K, z)$
5. $\forall j \in [n] :\quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
6. $K_1^* \leftarrow F_1.\mathsf{Punct}(K_1^{(i)}, y_i)$
7. $\forall j \in [n] :\quad s_j \leftarrow F_1(K_1^{(j)}, y_j)$
8. $\hat{R} \leftarrow \mathcal{D}(1^\lambda; s_1 \oplus s_2 \oplus \cdots \oplus s_n)$
9. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_2^1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^*, K^*, f, \omega, \hat{R}])$ (see Figure 4.44)
10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

Hybrid ω.2. In this hybrid, in the ι-th $\mathsf{NewSession}$ query, we generate $\hat{R}$ using true randomness instead of using $s_1 \oplus \cdots \oplus s_n$ where $s_j \leftarrow F_1(K_1^{(j)}, y_j)$. Furthermore, if the adversary issues any sampling queries $(U_j)_{j \neq i}$ for the ι-th session where the NIZKs verify, the extraction succeeds and $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ coincides with ω, the challenger replies with $\hat{R}$.

Indistinguishability between this hybrid and the previous one is a consequence of the security of the puncturable PRF F_1. Indeed, we are able to substitute s_i with a truly random string without the adversary noticing it. Furthermore, observe that the challenger never needs to evaluate F_1 over $y_i := \mathsf{Hash}(\mathsf{hk}_i, \omega)$. Indeed, with overwhelming probability, there exists no pair of well-formed tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ having y_i as digest.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, we rewrite ω as $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$. We denote the first PRF key hardcoded in EP_j by $K_1^{(j)}$ (we recall that we are only considering EP_js that are well-formed). For the reduction, since ω is fixed and the adversary is non-uniform, we can assume it knows $K_1^{(j)}$ for every $j \neq i$ (the latter is uniquely determined by ω by the injectivity of iO).

1. $K \xleftarrow{\$} F.\mathsf{Gen}(1^\lambda)$
2. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)
3. $z \leftarrow f(\varpi)$
4. $K^* \leftarrow F.\mathsf{Punct}(K, z)$
5. $y_i \leftarrow \mathsf{Hash}(\mathsf{hk}_i, \omega)$
6. $K_1^* \leftarrow F_1.\mathsf{Punct}(K_1^{(i)}, y_i)$
7. $\hat{R} \xleftarrow{\$} \mathcal{D}(1^\lambda)$
8. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_2^1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^*, K^*, f, \omega, \hat{R}])$ (see Figure 4.44)
9. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Hybrid ω.3. In this hybrid, in the ι-th $\mathsf{NewSession}$ query, we generate the randomness of $\hat{R}$ using $F(K, z)$ where $z = f(\omega)$. Indistinguishability is a consequence of the security of the puncturable PRF F.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, we rewrite ω as $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$.

1. $K \xleftarrow{\$} F.\mathsf{Gen}(1^\lambda)$
2. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)
3. $z \leftarrow f(\varpi)$
4. $K^* \leftarrow F.\mathsf{Punct}(K, z)$
5. $y_i \leftarrow \mathsf{Hash}(\mathsf{hk}_i, \omega)$
6. $K_1^* \leftarrow F_1.\mathsf{Punct}(K_1^{(i)}, y_i)$
7. $s \leftarrow F(K, z)$
8. $\hat{R} \leftarrow \mathcal{D}(1^\lambda; s)$
9. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_2^1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^*, K^*, f, \omega, \hat{R}])$ (see Figure 4.44)
10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

When the adversary issues any sampling queries $(U_j)_{j \neq i}$ for the ι-th session where the NIZKs verify, the extraction succeeds and $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ coincides with ω, the challenger replies as follows.

1. $z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big)$
2. $s \leftarrow F(K, z)$

3. $R \leftarrow \mathcal{D}(1^\lambda; s)$

Hybrid ω.4. In this hybrid, we modify the decryption program DP_i sent in the ι-th $\mathsf{NewSession}$ query, switching back to an obfuscation of DProg_2^0 (see Figure 4.43). This time, however, the we do not hardcode ω into it, but the next element in Ω. We denote it by ω'[13]. The input-output behaviour of DP_i has not changed since the last hybrid, so, we can argue for indistinguishability under the security of iO.

We observe, indeed, that the behaviour of the program can change only if the input satisfies $\omega \leq_{\mathsf{lex}} (\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} <_{\mathsf{lex}} \omega'$. If $\omega <_{\mathsf{lex}} (\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} <_{\mathsf{lex}} \omega'$, the programs always output $\perp$ because one $(\mathsf{hk}_j, \mathsf{EP}_j)$ must be malformed, so the NIZK extraction always fails.

We therefore focus on the case $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} = \omega$. We observe that in this case, the behaviour of the new DP_i is the same as in the previous hybrid. In particular, if the NIZK is not rejected, the value $\hat{R}$ computed by the new DP_i was the same that was previously hardcoded.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, we rewrite ω as $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$.

1. $K \overset{\$}{\leftarrow} F.\mathsf{Gen}(1^\lambda)$
2. $\mathsf{EP}_i \overset{\$}{\leftarrow} \mathsf{iO}(1^\lambda, \mathsf{EProg}_1[K_2^{(i)}, i])$ (see Figure 4.41)
3. $\mathsf{DP}_i \overset{\$}{\leftarrow} \mathsf{iO}(1^\lambda, \mathsf{DProg}_2^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, K, f, \omega'])$ (see Figure 4.43)
4. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

We conclude the proof of the claim by observing that when ω reaches the maximum in Ω, Hybrid ω.4 is indistinguishable from Hybrid 4.ι.4 under the security of iO. Indeed, in Hybrid ω.4, DP_i computed all the outputs running $\mathsf{DProg}_{\mathsf{Ls}}$. ■

Hybrid 5. In this hybrid, we modify the sampling queries. In particular, we do not try anymore to extract the witnesses from the NIZKs provided by the adversary, we simply verify the proofs. If the check succeeds, we proceed by inputting $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}$ in the ELF, we feed the result into F and we use the output as randomness for $\mathcal{D}(1^\lambda)$. If the verification fails, we reply with $\perp$.

This hybrid is indistinguishable from hybrid 4.$(M+1)$.4. Indeed, an adversary can distinguish if and only if, in Hybrid 4.$(M+1)$.4, it can generate a proof that verifies but cannot be extracted. Such adversary would also be able to distinguish between Hybrid 0 and Hybrid 4.$(M+1)$.4. However, we proved that such adversary cannot exist.

Hybrid 6. In this hybrid, we switch the ELF f to lossy mode. Let $p'(\lambda)$ be a polynomial upper bound on the running time of the lossy distributed sampler challenger in Hybrid 5 when it interacts with an adversary running in time a most $p(\lambda)$. We choose the polynomial $q(\lambda)$ parametrising the lossy mode so that no adversary running in time at most $p(\lambda) + p'(\lambda)$ can distinguish between the injective mode and the lossy mode with advantage greater than $\delta/2$.

We highlight that Hybrid 5 and Hybrid 6 can be distinguished with non-negligible advantage. However, by the security of ELFs, no adversary running in time at most $p(\lambda)$ can distinguish between them with advantage greater than $\delta/2$.

In order to conclude the proof, we show that it is possible to choose the security parameters of the subexponentially secure primitives so that Claim 4.7.5 and Claim 4.7.7 are all true. This is an immediate consequence of the fact that the dependency graph among subexponentially secure primitives in Figure 4.35 contains no cycles.

Regularity.

Assume that the ELF is regular. We observe that the output of $\mathsf{Project}$ is either $\perp$ or an element in the image of f. The output of $\mathcal{Z}(\zeta)$ is $\perp$ with probability 1/2. Otherwise, the output is $f(x)$ where x is uniformly

[13]If ω is already the maximum of Ω, we augment Ω with an imaginary element that is strictly greater than all other values. Let ω' be such value.

sampled over the domain of f. By the regularity of the ELF, we know that there exists a polynomial $s(\log M, q)$ such that, with overwhelming probability over $\mathsf{ELF.Gen}(M, q)$,

$$\Pr_x[f(x) = z] \geq \frac{1}{s(\log M, q)}$$

for every element z in the image of f, where $\Pr_x$ denotes the probability over the randomness of x. Since $\log M$ is polynomial in λ, we conclude that our lossy distributed sampler is regular.

Programmability.

We prove the property by means of a series of indistinguishable hybrids.

Hybrid 0. This hybrid corresponds to the programmability game in which $b = 0$. In particular, the distributed sampler message U_i received by the adversary as computed as follows.

1. $K \xleftarrow{\$} F.\mathsf{Gen}(1^\lambda)$
2. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(1^\lambda)$
3. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda)$
4. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)
5. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
6. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K, f])$ (see Figure 4.31)
7. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
8. $\pi_i' \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
9. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$ and $\xi_e := K$.

During the sampling phase, after querying $U_j := (\mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \pi_j')$ for every $j \neq i$, the adversary is provided with a value R computed as follows:

1. $\forall j \in [n]: \quad b_j \leftarrow \mathsf{NIZK'.Verify}\big(\sigma', \pi_j', (j, \mathsf{sid}, \mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \sigma)\big)$
2. If there exists $j \in [n]$ such that $b_j = 0$, output $\perp$.
3. $z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big)$
4. $s \leftarrow F(K, z)$
5. Output $\mathcal{D}(1^\lambda; s)$.

Hybrid 1. In this hybrid, we modify the decryption program DP_i switching to an obfuscation of $\mathsf{DProg}_{\mathsf{Pr}}$ (see Figure 4.38). In particular, the PRF key K hardcoded in the program will be punctured in the position z chosen by the adversary. Moreover, we hardcode into the program the value $R := \mathcal{D}(1^\lambda; s)$ where $s = F(K, z)$. When the output of the ELF in the modified decryption program coincides with z, DP_i will directly input R in the partial decryption simulator. We formalise below the operations used for the generation of U_i.

1. $K \xleftarrow{\$} F.\mathsf{Gen}(1^\lambda)$
2. $K^* \leftarrow F.\mathsf{Punct}(K, z)$
3. $s \leftarrow F(K, z)$

4. $R \leftarrow \mathcal{D}(1^\lambda; s)$

5. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(1^\lambda)$

6. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda)$

7. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)

8. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$

9. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_{\mathsf{Pr}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K^*, z, f, R])$ (see Figure 4.38)

10. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$

11. $\pi_i' \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$

12. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$ and $\xi_e := K$.

Observe that this hybrid is indistinguishable from the previous one thanks to the security of iO.

Hybrid 2. In this hybrid, instead of generating R using the randomness output by F, we use an ideal sample. If, in the sampling phase, the adversary selects values $(U_j)_{j\neq i}$ such that $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = z \neq \perp$ and, for every $j \neq i$,

$$\mathsf{NIZK'.Verify}\big(\sigma', \pi_j', (j, \mathsf{sid}, \mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \sigma)\big) = 1,$$

the challenger immediately provides R to the adversary.

This hybrid is indistinguishable from the previous one by the security of the puncturable PRF F. We formalise below the operations used for the generation of U_i.

1. $K \xleftarrow{\$} F.\mathsf{Gen}(1^\lambda)$

2. $K^* \leftarrow F.\mathsf{Punct}(K, z)$

3. $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$

4. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(1^\lambda)$

5. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda)$

6. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)

7. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$

8. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_{\mathsf{Pr}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K^*, z, f, R])$ (see Figure 4.38)

9. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$

10. $\pi_i' \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$

11. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$ and $\xi_e := K$.

We observe that the last hybrid is identical to the programmability game with $b = 1$. □

THE HARDNESS-PRESERVING SIMULATORS.

Let $q(\lambda)$ be the polynomial associated with $\mathcal{A}$.

$\mathsf{SimSetup}_{\mathcal{A}}(1^\lambda)$:

1. $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{LossySetup}(1^\lambda, q)$
2. Output crs and ζ.

$\mathsf{SimGen}_{\mathcal{A}}(1^\lambda, \mathsf{sid}, i, \zeta, R)$:

1. $z \xleftarrow{\$} \mathcal{Z}(\zeta)$
2. $(U_i, \xi) \xleftarrow{\$} \mathsf{ProgGen}(1^\lambda, \mathsf{sid}, i, z, R, \zeta)$
3. Output U_i.

Figure 4.45: The hardness-preserving simulators.

4.8 Building Hardness-Preserving Distributed Samplers

We explain the idea behind our result. Consider a PPT adversary $\mathcal{A}$ that outputs 1 with non-negligible probability $\epsilon(\lambda)$ in the real-world execution of the regular and programmable lossy distributed sampler (see Figure 4.20). In such execution, the distributed sampler will be in standard mode. We recall that our goal is to show the existence of a simulator, which depends on $\mathcal{A}$, such that, even in the simulated execution, the adversary $\mathcal{A}$ still outputs 1 with non-negligible probability.

We use a hybrid argument. In the first stage, we switch our distributed sampler to lossy mode. The new setting is clearly distinguishable from the initial one but, by choosing the polynomial q parametrising the lossy mode properly, we can make sure that the adversary $\mathcal{A}$ still outputs 1 with probability at least $\epsilon(\lambda)/2$.

In the next hybrid, we use the regularity of the lossy distributed sampler to argue that the probability that $\mathcal{A}$ outputs 1 and $\mathcal{Z}$ guesses the output chosen by the adversary is also non-negligible.

In the final hybrid, we rely on the programmability properties to hide an ideal sample R in the position guessed by $\mathcal{Z}$. Since the adversary cannot detect any change, $\mathcal{A}$ will still have a non-negligible probability of outputting 1 while picking R as output of the protocol.

From the last hybrid, we can easily obtain the simulators we are looking for. We simulate the CRS using $\mathsf{LossySetup}\big(1^\lambda, q(\lambda)\big)$. The choice of $q(\lambda)$ depends on $\mathcal{A}$. In particular, $q(\lambda)$ needs to be sufficiently large so that $\mathcal{A}$ cannot distinguish between the first two hybrids with advantage greater than $\epsilon(\lambda)/2$. The simulation of the distributed sampler message is instead performed using $\mathsf{ProgGen}$. The programmed position is sampled using $\mathcal{Z}$. We formalise the construction in Figure 4.45.

Theorem 4.8.1 (Hardness-preserving distributed sampler). Let $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{SimSetup}_{\mathcal{A}}, \mathsf{SimGen}_{\mathcal{A}})$ be a regular and programmable lossy distributed sampler for $\mathcal{D}(1^\lambda)$ against AClass. Then, the construction described in Figure 4.34 and Figure 4.45 is an n-party hardness-preserving distributed sampler for $\mathcal{D}$ against AClass.

Proof. Let $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{LossySetup}, \mathsf{LossyGen}, \mathsf{Project}, \mathsf{Extract})$ be our n-party regular and programmable lossy distributed sampler for the distribution $\mathcal{D}(1^\lambda)$. Let $\mathcal{A} \in \mathsf{AClass}$ be any PPT adversary such that, in the hardness-preserving game $\mathcal{G}$ in Figure 4.20,

$$\Pr\left[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}}(1^\lambda) = 1 \middle| b = 0\right] = \mathsf{nonegl}(\lambda).$$

Our goal is to prove that

$$\Pr\left[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}}(1^\lambda) = 1 \middle| b = 1\right] = \mathsf{nonegl}(\lambda).$$

We define $\epsilon(\lambda) := \Pr\left[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}}(1^\lambda) = 1 \middle| b = 0\right]$. Since $\epsilon(\lambda)$ is non-negligible, we know that there exists a polynomial $e(\lambda)$ such that for every $\overline{\lambda} \in \mathbb{N}$, there is a $\lambda \geq \overline{\lambda}$ such that $\epsilon(\lambda) \geq 1/e(\lambda)$. Let $p(\lambda)$ be a polynomial upper-bounding twice the running times of $\mathcal{A}$.

We proceed by means of a Hybrid argument.

Hybrid 0. This stage corresponds to $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}}$. In particular, the challenger provides the adversary with a pair (crs, U_i) generated using the algorithms $\mathsf{Setup}(1^\lambda)$ and $\mathsf{Gen}(1^\lambda, \mathsf{sid}, i, \mathsf{crs})$. The sample given to $\mathcal{A}$ is instead computed using Sample.

Hybrid 1. In this hybrid, we change the distribution of crs, U_i and R. Specifically, we use the algorithms $\mathsf{LossySetup}\big(1^\lambda, q(\lambda)\big)$, $\mathsf{LossyGen}(1^\lambda, \mathsf{sid}, i, \zeta)$, $\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)$ and $\mathsf{Extract}(\xi, z)$. The polynomial $q(\lambda)$ is chosen so that all adversaries running in time at most $p(\lambda)$ distinguish between the standard mode and the lossy mode parametrised by $q(\lambda)$ with advantage asymptotically smaller than $1/\big(2e(\lambda)\big)$. We denote the output of the adversary $\mathcal{A}$ after the interaction with the modified challenger by $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}$.

Claim 4.8.2. In Hybrid 1, $\Pr\left[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(1^\lambda) = 1\right] = \mathsf{nonegl}(\lambda)$.

Proof of the claim. Assume, by contradiction, that our claim is false. We construct an adversary that runs in time at most $p(\lambda)$ distinguishing between the standard mode and the lossy mode with advantage that is not asymptotically smaller than $1/\big(2e(\lambda)\big)$.

Our new adversary, denoted by $\mathcal{B}$, runs an internal copy of $\mathcal{A}$. The adversary $\mathcal{B}$ provides $\mathcal{A}$ with the value crs obtained from its challenger, after which, it obtains $i \in [n]$ and $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$. The adversary $\mathcal{B}$ forwards $\mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n}$ to its challenger. Next, it issues a $\mathsf{NewSession}$ query with identity (sid, i). The answer U_i is forwarded to $\mathcal{A}$. When $\mathcal{A}$ replies with $(U_j)_{j\neq i}$, the adversary $\mathcal{B}$ queries $\big(\mathsf{Sample}, \mathsf{sid}, (U_j)_{j\neq i}\big)$ to its challenger and relays the result to $\mathcal{A}$. Finally, $\mathcal{B}$ outputs 1 if and only $\mathcal{A}$ outputs 1 and the distributed sampler output is not $\perp$.

The distinguishing advantage $\mathsf{Adv}_{\mathcal{B}}(\lambda)$ of $\mathcal{B}$ is $|\epsilon(\lambda) - \mathsf{negl}(\lambda)|$. For λ sufficiently big, we have that $\mathsf{Adv}_{\mathcal{B}}(\lambda)$ is greater than $\epsilon(\lambda) - 1/\big(4e(\lambda)\big)$. So, we conclude that for every $\overline{\lambda} \in \mathbb{N}$, there exists a $\lambda \geq \overline{\lambda}$ such that $\mathsf{Adv}_{\mathcal{B}}(\lambda) \geq 3/\big(4e(\lambda)\big)$. Since $\mathcal{B}$ at most in time $p(\lambda)$, we reached a contradiction. Notice that $\mathcal{B}$ is uniform if and only $\mathcal{A}$ is uniform. ■

Claim 4.8.3. Let E be the event in which $\mathcal{Z}(\zeta) = \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)$. In Hybrid 1, we have

$$\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(1^\lambda) = 1, E] = \mathsf{nonegl}(\lambda).$$

Proof of the claim. Let V denote the event in which

$$\Pr_{\mathcal{Z}}\left[\mathcal{Z}(\zeta) = \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)\right] < \frac{1}{s\big(\lambda, q(\lambda)\big)}$$

for some $(U_j)_{j\in[n]} \in \{0,1\}^*$ where the above probability is taken only over the randomness of $\mathcal{Z}$. By the regularity of the lossy distributed sampler $\Pr[V] = \mathsf{negl}(\lambda)$. We conclude that

$$\begin{aligned}
&\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(1^\lambda) = 1, E] \geq \\
\geq &\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(1^\lambda) = 1, E, V] + \mathsf{negl}(\lambda) = \\
= &\Pr\left[E \middle| \mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(1^\lambda) = 1, V\right] \cdot \Pr\left[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(1^\lambda) = 1, V\right] + \mathsf{negl}(\lambda) \geq \\
\geq &\frac{1}{s\big(\lambda, q(\lambda)\big)} \cdot \Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(1^\lambda) = 1, V] + \mathsf{negl}(\lambda) \geq \\
\geq &\frac{1}{s\big(\lambda, q(\lambda)\big)} \cdot \Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(1^\lambda) = 1] + 2 \cdot \mathsf{negl}(\lambda).
\end{aligned}$$

We conclude the proof of the claim by observing that $\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(1^\lambda) = 1]$ is non-negligible by Claim 4.8.2. ■

Hybrid 2. In this hybrid, we generate U_i and ξ using $\mathsf{ProgGen}(1^\lambda, \mathsf{sid}, i, z, R, \zeta)$ where $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$ and $z \xleftarrow{\$} \mathcal{Z}(\zeta)$. If the adversary selects $(U_j)_{j\neq i}$ such that $\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big) = z$ and $z \neq \perp$, we provide the adversary with R. If instead $\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big) = \bot$, we provide the adversary with $\bot$. The rest remains as in the previous hybrid. We denote the output of the adversary $\mathcal{A}$ after the interaction with the modified challenger by $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}$.

Claim 4.8.4. In the new game $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}$, we have

$$\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}(1^\lambda) = 1, E] = \mathsf{nonegl}(\lambda).$$

Proof of the claim. Suppose that our claim is false. Then, we can find a PPT adversary $\mathcal{B}$ that breaks the programmability of the lossy distributed sampler. The adversary $\mathcal{B}$ provides the CRS it receives from its challenger to an internal copy of $\mathcal{A}$. Then, it samples $z \overset{\$}{\leftarrow} \mathcal{Z}(\zeta)$. Notice that ζ is given to $\mathcal{B}$ by its challenger. When $\mathcal{A}$ selects sid and $i \in [n]$, $\mathcal{B}$ sends sid, i, z to its challenger. It then proceeds by relaying all the communications between $\mathcal{A}$ and its challenger. At the end of its execution, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ outputs 1, the distributed sampler output is not $\bot$ and $\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big) = z$.

Notice that the advantage of $\mathcal{B}$ is

$$\Big|\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(1^\lambda) = 1, E] - \Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}(1^\lambda) = 1, E]\Big| = \mathsf{nonegl}(\lambda) - \mathsf{negl}(\lambda).$$

Observe also that $\mathcal{B}$ is uniform if and only $\mathcal{A}$ is uniform. We reached a contradiction. ■

Hybrid 3. In this hybrid, we modify $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}$. Instead of providing $\mathcal{A}$ with $\mathsf{Extract}\Big(\xi, \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)\Big)$, we now provide it with the value R hidden in U_i. We call the resulting game $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}$.

Claim 4.8.5. In the new game $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}$, we have $\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}(1^\lambda) = 1] = \mathsf{nonegl}(\lambda)$.

Proof of the claim. We prove that $\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}(1^\lambda) = 1, E] = \mathsf{nonegl}(\lambda)$. The result follows from the fact that $\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}(1^\lambda) = 1] \geq \Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}(1^\lambda) = 1, E]$.

We notice that in both $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}$ and $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}$, when E occurs, the adversary is always provided with either R or $\bot$. In the second case, both $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}(1^\lambda) = 0$ and $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}(1^\lambda) = 0$. We conclude that, by Claim 4.8.4,

$$\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}(1^\lambda) = 1, E] = \Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}(1^\lambda) = 1, E] = \mathsf{nonegl}(\lambda).$$

■

Hybrid 3 corresponds to the ideal world execution of the hardness-preserving distributed sampler. In particular, $\mathsf{SimSetup}_{\mathcal{A}}$ just performs the same operations as $\mathsf{LossySetup}$. The simulator $\mathsf{SimGen}_{\mathcal{A}}(1^\lambda, \mathsf{sid}, i, \zeta, R)$ instead outputs the message U_i generate by $\mathsf{ProgGen}(1^\lambda, \mathsf{sid}, i, z, R, \zeta)$ where $z \overset{\$}{\leftarrow} \mathcal{Z}(\zeta)$. Notice that the polynomial $q(\lambda)$ used for the ELF depends on the running time of $\mathcal{A}$. Claim 4.8.5 proves that

$$\Pr\left[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}}(1^\lambda) = 1 \middle| b = 1\right] = \mathsf{nonegl}(\lambda).$$

□

4.8.1 Building Indistinguishability Preserving Distributed Samplers

We now explain why the distributed sampler presented in Section 4.7 is indistinguishability preserving.

Consider any pair of chosen-sample indistinguishable games $\mathcal{G}_0$ and $\mathcal{G}_1$ where $\mathcal{G}_0 = (\mathcal{D}, \mathsf{Ch}_0)$ is a game with oracle distribution and $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$ is a game with trapdoored oracle distribution satisfying trapdoor security. We start by considering any PPT adversary $\mathcal{A}$ whose goal is to distinguish between the compiled games $\mathcal{G}'_0$ and $\mathcal{G}'_1$. The proof relies on a hybrid argument beginning from $\mathcal{G}'_0$. We will explain the distributed sampler simulator for the trapdoored mode in the last hybrid.

```
THE INDISTINGUISHABILITY-PRESERVING SIMULATORS
$\mathsf{SimSetup}(1^\lambda)$:
    1. $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{NIZK.SimSetup}(1^\lambda)$
    2. $(\sigma', \tau') \xleftarrow{\$} \mathsf{NIZK'.SimSetup}(1^\lambda)$
    3. $f \xleftarrow{\$} \mathsf{ELF.Gen}(M, M)$
    4. Output $\mathsf{crs} := (\sigma, \sigma')$ and $\zeta := (\sigma, \sigma', \tau_s, \tau_e, \tau', f)$
$\mathsf{SimGen}\big(1^\lambda, \mathsf{sid}, i, \zeta := (\sigma, \sigma', \tau_s, \tau_e, \tau', f), \mathsf{aux}\big)$:
    1. $K \xleftarrow{\$} F.\mathsf{Gen}(1^\lambda)$
    2. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(1^\lambda)$
    3. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(1^\lambda)$
    4. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Figure 4.30)
    5. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
    6. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathsf{DProg}_{\mathsf{IP}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f, \mathsf{aux}])$ (see Figure 4.47)
    7. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
    8. $\pi_i' \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
    9. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$ and $\xi := (\mathsf{sid}, \sigma', \sigma, f, K, \mathsf{aux})$.
$\mathsf{Trap}\big(\xi = (\mathsf{sid}, \sigma', \sigma, f, K, \mathsf{aux}), \big(U_j = (\mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \pi_j')\big)_{j \in [n]}\big)$:
    1. $\forall j \in [n]: \quad b_j \leftarrow \mathsf{NIZK'.Verify}\big(\sigma', \pi_j', (j, \mathsf{sid}, \mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \sigma)\big)$
    2. If $\exists j \in [n]$ such that $b_j = 0$, output $(\perp, \perp)$.
    3. $z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big)$
    4. $s \leftarrow F(K, z)$
    5. Output $(R, T) \leftarrow \mathcal{D}'(1^\lambda, \mathsf{aux};\ s)$
```

Figure 4.46: The indistinguishability-preserving simulators

The hybrids. In the first stage, we activate the lossy mode of the distributed sampler using some polynomial $q(\lambda)$. At this point, the output of the construction is restricted in a set of polynomial size. Notice, however, that we have given the adversary non-negligible distinguishability advantage $\epsilon_1(\lambda)$. We will argue later why this will not constitute a problem.

In the next hybrid, we proceed by switching from the challenger Ch_0 to the challenger Ch_1 without providing the latter with any trapdoor T. The modification cannot be detected by the adversary due to the chosen-sample indistinguishability between $\mathcal{G}_0$ and $\mathcal{G}_1$.

Next, using obfuscation and puncturable PRFs, we will gradually change the distribution of the outputs of the distributed sampler, switching from $\mathcal{D}$ to the trapdoored distribution $\mathcal{D}'$. The technique is similar to the one we used to prove programmability. The main difference is that we repeat the procedure many times, once for each element in the image of the ELF. Simultaneously, we will start providing Ch_1 with the trapdoors T. Specifically, there will be some hybrids in which part of the distributed sampler outputs are produced using $\mathcal{D}$ whereas the rest is generated using $\mathcal{D}'$. When the distributed sampler output chosen by the adversary is generated using $\mathcal{D}'$, we provide the corresponding trapdoor T to Ch_1 otherwise, we will not. We will be able to retrieve the trapdoors leveraging the knowledge of the ELF f and the PPRF key K hardcoded into the lossy-mode messages. The randomness fed into $\mathcal{D}'$ will indeed be $F(K, z)$ where $z = f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big)$, similarly to what happened in $\mathsf{DProg}_{\mathsf{Ls}}$ (see Figure 4.31). To prove that this stage is indistinguishable from the previous one, we use a hybrid argument that is iterated over the image of the ELF. Since the latter has polynomial cardinality, we do not need to assume that $\mathcal{G}_1$ satisfies subexponential trapdoor security.

In the last stage, which will correspond to $\mathcal{G}_1'$, we switch back to a construction where the outputs have high entropy. This will be done by setting the ELF in the construction back to injective mode. The distributions of the outputs will remain as in the previous hybrid, namely, with a trapdoor embedded in them. In the process, however, we will give the adversary other non-negligible advantage $\epsilon_2(\lambda)$. Notice anyway, that this stage is independent of the polynomial $q(\lambda)$.

Why are $\mathcal{G}_0'$ and $\mathcal{G}_1'$ indistinguishable? Suppose that our adversary $\mathcal{A}$ can distinguish between the initial and the final stage with non-negligible advantage $\epsilon(\lambda)$. By choosing the polynomial $q(\lambda)$ in the lossy mode properly, we can make $\epsilon_1(\lambda)$ and $\epsilon_2(\lambda)$ arbitrarily small non-negligible functions. In particular, we can make sure that no adversary running in the same time as $\mathcal{A}$ can distinguish between $\mathcal{G}_0'$ and $\mathcal{G}_1'$ with advantage greater than $\epsilon(\lambda)/2$. In this way, we reach a contradiction.

The simulators. From the last stage of our hybrid argument, we can easily derive the simulators for the indistinguishability-preserving distributed sampler. The algorithm $\mathsf{SimSetup}$ will simulate the CRSs for NIZK and NIZK' as $\mathsf{LossySetup}$ did (see Figure 4.36). Furthermore, it will generate an injective-mode ELF f. The simulator SimGen will behave exactly as $\mathsf{LossyGen}$ (see Figure 4.36) with the exception that, in DP_i, we substitute $\mathcal{D}(1^\lambda)$ with $\mathcal{D}'(1^\lambda, \mathsf{aux})$. The trapdoor information ξ will contain the ELF f, the PPRF key K and aux. This information is sufficient to retrieve the trapdoors hidden in the distributed sampler outputs. We formalise the construction in Figure 4.46.

Theorem 4.8.6 (Indistinguishability-preserving distributed sampler). Let $\mathcal{D}$ be an efficient distribution and let $\mathcal{D}'$ be a trapdoored distribution for $\mathcal{D}$. Assume that ELF is a regular extremely lossy function. Under the hypothesis of Theorem 4.7.1, the construction $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{SimSetup}, \mathsf{SimGen}, \mathsf{Trap})$ described in Figure 4.34 and Figure 4.46 is an n-party indistinguishability-preserving distributed sampler for $(\mathcal{D}, \mathcal{D}')$ against AClass.

Observe that in the non-uniform setting, we can instantiate the construction so that the CRS is statistically close to uniform and its length depends only on the security parameter. In the uniform setting, instead, we do not need any CRS.

Proof. Let $\mathcal{A}$ be any PPT adversary in AClass that distinguishes between $\mathcal{G}_0'$ and $\mathcal{G}_1'$ with non-negligible advantage $\epsilon(\lambda)$. In particular, we know that there exists a polynomial $e(\lambda)$ such that, for every $\overline{\lambda} \in \mathbb{N}$, there exists a $\lambda \geq \overline{\lambda}$ such that $\epsilon(\lambda) \geq 1/e(\lambda)$. We proceed by means of a hybrid argument starting from

$\mathsf{DProg}_{\mathsf{IP}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f, \mathsf{aux}]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the PPRF key K, the ELF f, the auxiliary information aux.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

```
1. $\forall j \neq i: \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
2. $\forall j \neq i: \quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$[a]
3. If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \bot$, output $\bot$
4. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
5. $\forall j \neq i: \quad s_j \leftarrow F_1\big(K_1^{(j)}, y_j\big)$
6. $\forall j \in [n]: \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2\big(K_2^{(j)}, y_j\big)$
7. $z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big)$
8. $s \leftarrow F(K, z)$
9. $(\hat{R}, \hat{T}) \leftarrow \mathcal{D}'(1^\lambda, \mathsf{aux}; s)$
10. $(\phi, \mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Sim}_1(1^\lambda, i; r''_i)$
11. $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, \hat{R}, (s_j, r_j, r'_j)_{j \neq i}; \eta'_i\Big)$ (see bottom of Figure 4.29)
12. Output $d_i$
```

Figure 4.47: The unobfuscated decryption program for the indistinguishability-preserving simulator

$\mathcal{G}'_0$. Let i be the index of a honest party. Let $p(\lambda)$ be a polynomial upper-bounding the running time of $\mathcal{A}$. Let $p'(\lambda)$ be a polynomial upper-bounding the running time of the challengers in $\mathcal{G}'_0$ and $\mathcal{G}'_1$ when the adversary runs in time at most $p(\lambda)$. We select the polynomial $q(\lambda)$ so that every adversary running in time at most $p(\lambda) + p'(\lambda)$ distinguishes between the standard mode and the lossy mode of both the ELF and the distributed sampler with advantage definitively smaller than $1/(4e(\lambda))$. Notice that by Theorem 4.7.1, such polynomial $q(\lambda)$ exists.

Hybrid 0. This hybrid corresponds to $\mathcal{G}'_0$

Hybrid 1. In this hybrid, the challenger witched the distributed sampler to lossy mode. Specifically, it generates the distributed sampler CRS crs using $\mathsf{LossySetup}\big(1^\lambda, q(\lambda)\big)$. Furthermore, in every $\mathsf{NewSession}$ query, it generates the last distributed sampler message sent by a honest party using the lossy mode of the distributed sampler, i.e.,

$$U_i \xleftarrow{\$} \mathsf{LossyGen}(1^\lambda, \mathsf{sid}, i, \zeta).$$

Finally, when all the distributed sampler messages have been exchanged, the challenger computes the output R using $\mathsf{Project}$ and $\mathsf{Extract}$ instead of Sample. The rest remains exactly as in $\mathcal{G}'_0$.

Notice that, by the first property of lossy distributed samplers, the distinguishable advantage of $\mathcal{A}$ between Hybrid 0 and Hybrid 1 is asymptotically smaller than $1/(4e(\lambda))$. The reduction is pretty straightforward. The new adversary $\mathcal{B}$ receives the CRS crs from the lossy distributed sampler challenger. It uses the latter to simulate $\mathcal{G}'_0$ to an internal copy of $\mathcal{A}$. When $\mathcal{A}$ sends a new session identity sid, $\mathcal{B}$ performs the same operations as the challenger in $\mathcal{G}'_0$. Things change when the last honest party sends its distributed sampler message. Let id_{j_i} be the identity of the corresponding party. The adversary $\mathcal{B}$ queries $(\mathsf{NewSession}, \mathsf{sid}, i)$ to its challenger and relays the answer to $\mathcal{A}$. Then, when all the distributed sampler messages $(U_j)_{j\in[n]}$ have been exchanged, $\mathcal{B}$ queries $\big(\mathsf{Sample}, \mathsf{sid}, (U_j)_{j\neq i}\big)$ to its challenger and provides the answer to the copy of Ch_0. The new adversary $\mathcal{B}$ outputs the same value as $\mathcal{A}$. We observe that if $\mathcal{A}$ is uniform, $\mathcal{B}$ is uniform too. Furthermore, the running time of $\mathcal{B}$ is at most $p(\lambda) + p'(\lambda)$. By the first property of lossy distributed samplers, the advantage of $\mathcal{B}$ is asymptotically smaller than $1/(4e(\lambda))$.

Hybrid 2. In this hybrid, the challenger uses Ch_1 instead of Ch_0 in every $\mathsf{NewSession}$ query. Notice that Ch_1 is just provided with a sample R, but not with any trapdoor T.

Claim 4.8.7. No PPT adversary $\mathcal{A}$ can distinguish between Hybrid 1 and Hybrid 2.

Proof of the claim. Let $M(\lambda)$ be a polynomial upper-bound on the number of $\mathsf{NewSession}$ queries issued by the adversary $\mathcal{A}$. For every $\iota \in [M] \cup \{0\}$, we define Hybrid' ι in which the first ι $\mathsf{NewSession}$ queries are dealt using Ch_1, whereas the rest are dealt using Ch_0. We prove that, for every $\iota \in [M]$, no PPT adversary can distinguish between Hybrid' $\iota - 1$ and Hybrid' ι.

We do this by means of a reduction to the chosen-sample indistinguishability of $\mathcal{G}_0$ and $\mathcal{G}_1$. In the reduction, we build a new adversary $\mathcal{B}$ having a copy of $\mathcal{A}$. The adversary $\mathcal{B}$ starts its execution by producing a distributed sampler CRS crs using $\mathsf{LossySetup}$. In the process, it obtains also ζ. In the first $\iota-1$ $\mathsf{NewSession}$ queries, $\mathcal{B}$ simulates the game in Hybrid 1 using Ch_1 as challenger. Starting from the $(\iota+1)$-th query, $\mathcal{B}$ uses instead Ch_0. For the ι-th session, $\mathcal{B}$ sends the corresponding auxiliary information aux and the set of honest parties $H' := \{l \in [n] | j_l \in H\}$ to its challenger. It then relays the messages between its challenger and $\mathcal{A}$. In all the sessions, including the ι-th one, $\mathcal{B}$ generates the distributed sampler messages as in Hybrid 1. In particular, the last honest distributed sampler message sent in every session is produced using $\mathsf{LossyGen}$ and ζ. Furthermore, the output of the distributed sampler is computed using $\mathsf{Project}$ and $\mathsf{Extract}$. In the ι-th session, $\mathcal{B}$ gives the output of $\mathsf{Extract}$ to its challenger.

We have just proven that Hybrid' $\iota-1$ and Hybrid' ι are indistinguishable for every $\iota \in [M]$. We conclude that Hybrid' 0 and Hybrid' M are indistinguishable too. The latter are identical to Hybrid 1 and Hybrid 2 respectively. That ends the proof of the claim. ■

Hybrid 3. For any session of identity $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$, let j_i be the index of the last honest party sending a distributed sampler message. In this hybrid, the challenger generates the decryption program DP_i in U_i by obfuscating the program $\mathsf{DProg}_{\mathsf{IP}}$ (see Figure 4.47) instead of $\mathsf{DProg}_{\mathsf{Ls}}$ (see Figure 4.31). In $\mathsf{DProg}_{\mathsf{IP}}$ we hardcode the auxiliary information aux' given by Ch_1. Notice that we now generate the outputs of the distributed sampler using the trapdoored distribution $\mathcal{D}'(1^\lambda, \mathsf{aux}')$.

After computing the output of the distributed sampler $\hat{R}$, the challenger provides $\mathcal{G}_1$ with a trapdoor $\hat{T}$. The latter is retrieved by rerunning the computations of DP_i in clear. Specifically, we perform the following operations

1. $z \leftarrow \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)$
2. $s \leftarrow F(\xi, z)$
3. $(\hat{R}, \hat{T}) \leftarrow \mathcal{D}'(1^\lambda, \mathsf{aux}';\, s)$

We recall that ξ is computed by $\mathsf{LossyGen}$ together with U_i and consists of the PPRF key K hardcoded in DP_i.

Claim 4.8.8. Hybrid 3 is computationally indistinguishable from Hybrid 2.

Proof of the claim. Let $N(\lambda)$ be a polynomial upper-bound on the number of $\mathsf{NewSession}$ queries issued by the adversary $\mathcal{A}$. For every $\iota \in [N] \cup \{0\}$, we define Hybrid' ι in which the first ι $\mathsf{NewSession}$ queries are answered as in Hybrid 3. The remaining sessions are answered as in Hybrid 2. Notice that Hybrid' 0 is identical to Hybrid 2. Similarly, Hybrid' N is identical to Hybrid 3. We show that, for every $i \in [N]$, Hybrid' ι and Hybrid' $\iota - 1$ are computationally indistinguishable. That will immediately imply our claim.

```
$\mathsf{DProg}^0_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K^*, f, \hat{z}, R]$

**Hard-coded.** The index $i$ of the party, the session identity $\mathsf{sid}$, a PPRF key $K_2^{(i)}$, the encryption program $\mathsf{EP}_i$, the hash key $\mathsf{hk}_i$, the extractable NIZK CRS $\sigma$ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the punctured PRF key $K^*$, the ELF $f$, the value $\hat{z}$, the sample $R$.
**Input.** Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

  1. If $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) \neq \hat{z}$, output
     $d_i \leftarrow \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K^*, f]\big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\big)$ (see Figure 4.31)
  2. If $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) = \hat{z}$, perform the following operations:
     (a) $\forall j \neq i: \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (\mathsf{hk}_j, \mathsf{EP}_j)\big)$
     (b) $\forall j \neq i: \quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$[a]
     (c) If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \perp$, output $\perp$
     (d) $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
     (e) $\forall j \neq i: \quad s_j \leftarrow F_1\big(K_1^{(j)}, y_j\big)$
     (f) $\forall j \in [n]: \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2\big(K_2^{(j)}, y_j\big)$
     (g) $(\phi, \mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Sim}_1(1^\lambda, i; r''_i)$
     (h) $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, R, (s_j, r_j, r'_j)_{j \neq i}; \eta'_i\Big)$ (see bottom of Figure 4.29)
     (i) Output $d_i$
```

Figure 4.48: The unobfuscated decryption program for Hybrid" 1

We prove that Hybrid' ι and Hybrid' $\iota-1$ are indistinguishable by means of a sequence of indistinguishable hybrids.

Hybrid" 0. In this hybrid, we answer the first $\iota - 1$ NewSession queries as in Hybrid" 3. Starting from the $(\iota + 1)$-th query, we instead answer as in Hybrid" 2. We deviate from the usual behaviour in the ι-th query. We sample $x \xleftarrow{\$} [M]$ and we compute $\hat{z} \leftarrow f(x)$. Then, if $\hat{z} \neq f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big)$, where $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}$ are the hash keys and encryption programs exchanged in the ι-th session, we rewind the adversary and we retry. Observe that, with overwhelming probability, we succeed within $t(\lambda)$ tries for some polynomial t. This hybrid is perfectly indistinguishable from Hybrid' $\iota - 1$.

Hybrid" 1. In this hybrid, we behave as in Hybrid" 0 with minor changes in the answer to the ι-th NewSession query. Let id_{j_i} be the identity of the last honest party sending a distributed sampler message in the ι-th session. We generate the decryption program DP_i by obfuscating $\mathsf{DProg}^0_{\mathsf{Ls}}$ (see Figure 4.48): after receiving the input $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$, $\mathsf{DProg}^0_{\mathsf{Ls}}$ immediately checks if $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) = \hat{z}$. If that is the case, instead of inputting $\hat{R}$ into the partial decryption simulator, the program inputs $R := \mathcal{D}(1^\lambda; s)$ where $s = F(K, \hat{z})$. Such value R is hardcoded into $\mathsf{DProg}^0_{\mathsf{Ls}}$. If instead $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) \neq \hat{z}$, the program computes the output using K^* instead of K, where K^* denotes the puncturing of K in position $\hat{z}$. Observe that the input-output behaviour of the program remains the same as in the previous hybrid. We conclude that Hybrid" 0 and Hybrid" 1 are indistinguishable due to the security of iO.

Hybrid" 2. In this hybrid, in the ι-th session, instead of generating the sample R hardcoded into DP_i using the randomness produced by $F(K, \hat{z})$, the challenger simply samples $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$. When all the distributed sampler messages have been exchanged, the challenger verifies the NIZKs. If any check fails or $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) \neq \hat{z}$, the challenger behaves as before. Otherwise, it directly provides R to Ch_1. Since we provide the adversary only with K^*, this hybrid is indistinguishable from the previous one by the security of the puncturable PRF F.

Hybrid" 3. In this hybrid, in the ι-th session, the challenger generates the sample R hardcoded in DP_i using $(R, T) \xleftarrow{\$} \mathcal{D}'(1^\lambda, \mathsf{aux}')$. This hybrid is indistinguishable from the previous one since $\mathcal{D}'$ is a trapdoored distribution for $\mathcal{D}$.

Hybrid" 4. In this hybrid, in the ι-th session, if all the NIZKs in the distributed sampler messages verify and $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = \hat{z}$, the challenger provides Ch_1 with the trapdoor T produced by $\mathcal{D}'$ along with R. This hybrid is indistinguishable from the previous one by the trapdoor security of $\mathcal{G}_1$.

In the reduction, we build an adversary $\mathcal{B}$ holding a copy of $\mathcal{A}$. Upon activation, $\mathcal{B}$ simulates the $\mathcal{G}_0'$ as in Hybrid" 3 to $\mathcal{A}$. It behaves differently in the ι-th session. Let aux be corresponding auxiliary input and let $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$ be the corresponding session identity. Let id_{j_i}, be the identity of the last honest party sending a distributed sampler message. The adversary $\mathcal{B}$ provides its challenger with aux and $H' := \{l \in [n] | j_l \in H\}$. Then, it keeps relaying the messages between its challenger and $\mathcal{A}$. The distributed sampler messages in the ι-th session are produced by $\mathcal{B}$ as in Hybrid" 3, except for the fact that $\mathcal{B}$ uses the sample R provided by its challenger when it is time to generate DP_i. When all the distributed sampler messages have been sent, $\mathcal{B}$ checks whether the NIZKs verify and $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = \hat{z}$. If that is the case, $\mathcal{B}$ keeps relaying the messages between its challenger and $\mathcal{A}$ and outputs the same value as $\mathcal{A}$. In the other cases, $\mathcal{B}$ simply outputs a random bit. Observe that the advantage of $\mathcal{B}$ against the trapdoor security of $\mathcal{G}_1$ is the same as the advantage of $\mathcal{A}$ in distinguishing between Hybrid" 3 and 4.

Hybrid" 5. In this hybrid, in the ι-th session, we generate the sample R hardcoded in DP_i using $(R, T) \xleftarrow{\$} \mathcal{D}'(1^\lambda, \mathsf{aux}'; s)$ where $s = F(K, \hat{z})$. All the rest remains as in the previous hybrid. Since we provide the adversary only with K^*, this hybrid is indistinguishable from the previous one by the security of the puncturable PRF F.

We now proceed with a sequence of $q(\lambda)$ hybrids, where $q(\lambda)$ is the polynomial given as input to $\mathsf{LossySetup}$ (the total number of hybrids is polynomial). Let γ denote the ω-th element in the image of f that differs from $\hat{z}$, if we order the latter according to the lexicographical order. Notice that since the ELF is regular, it is also strongly efficiently enumerable [Zha16], so, given f, we can efficiently compute γ. Throughout the proof, we assume that f has an image with at most $q(\lambda)$ elements and that the challenger successfully retrieves the whole image of f. This is enough to prove our claim as these events occur with overwhelming probability.

Hybrid" ω.0. We behave as in Hybrid" 5, except in the ι-th $\mathsf{NewSession}$ query. Let id_{j_i} be the identity of the last honest party sending a distributed sampler message in the ι-th session. We generate the decryption program DP_i by obfuscating $\mathsf{DProg}^1_{\mathsf{Ls}}$ (see Figure 4.49): after receiving the input $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$, $\mathsf{DProg}^1_{\mathsf{Ls}}$ immediately checks if $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) <_{\mathsf{lex}} \gamma$ or $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = \hat{z}$. If that is the case, it behaves as $\mathsf{DProg}_{\mathsf{IP}}$ (see Figure 4.47), otherwise, it performs the same operations as $\mathsf{DProg}_{\mathsf{Ls}}$ (see Figure 4.31).

We observe that if $\omega = 1$, this hybrid is indistinguishable from Hybrid" 5 by the security of iO. Indeed, x is the minimum in the image of f, so $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big)$ will always be greater or equal to x. In other words, DP_i will always behave as $\mathsf{DProg}^0_{\mathsf{Ls}}$. If instead $\omega > 1$, this hybrid is identical to the previous one, i.e. Hybrid" $(\omega - 1)$.5.

Hybrid" ω.1. In this hybrid, we change the decryption program DP_i of the last honest party sending a distributed sampler message in the ι-th session. Specifically, instead of obfuscating $\mathsf{DProg}^1_{\mathsf{Ls}}$, we obfuscate $\mathsf{DProg}^2_{\mathsf{Ls}}$ (see Figure 4.50). In the latter, the PRF key K is punctured in position γ, we denote it by K^*. After receiving the input $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$, $\mathsf{DProg}^2_{\mathsf{Ls}}$ immediately checks if $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) <_{\mathsf{lex}} \gamma$ or $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = \hat{z}$. If that is the case, it behaves as $\mathsf{DProg}_{\mathsf{IP}}$ (see Figure 4.47). Otherwise, $\mathsf{DProg}^2_{\mathsf{Ls}}$ performs the same operations as $\mathsf{DProg}_{\mathsf{Ls}}$ (see Figure 4.31) with only one exception: when $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = \gamma$, instead of inputting $\hat{R}$ into the partial decryption simulator, the program inputs $R := \mathcal{D}(1^\lambda; s)$ where $s = F(K, \gamma)$. Such value R is hardcoded into $\mathsf{DProg}^2_{\mathsf{Ls}}$.

All the rest remains as in the previous hybrid. Since the input-output behaviour of DP_i has not changed, this hybrid is indistinguishable from the previous one by the security of iO.

Hybrid" ω.2. In this hybrid, in the ι-th session, instead of generating the sample R hardcoded into DP_i using the randomness produced by $F(K, \gamma)$, the challenger simply samples $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$. Since we provide

$\mathsf{DProg}^1_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f, \hat{z}, \mathsf{aux}', \gamma]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the PPRF key K, the ELF f, the value $\hat{z}$, the auxiliary information aux', the hybrid index γ.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

1. If $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) <_{\mathsf{lex}} \gamma$ or $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) = \hat{z}$, output
   $d_i \leftarrow \mathsf{DProg}_{\mathsf{IP}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f, \mathsf{aux}']\big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\big)$ (see Figure 4.47)
2. Otherwise, output
   $d_i \leftarrow \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f]\big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\big)$ (see Figure 4.31)

Figure 4.49: The unobfuscated decryption program for Hybrid" $\omega.0$

the adversary only with K^*, this hybrid is indistinguishable from the previous one by the security of the puncturable PRF F.

Hybrid" ω.3. In this hybrid, in the ι-th session, the challenger generates the sample R hardcoded in DP_i using $(R, T) \xleftarrow{\$} \mathcal{D}'(1^\lambda, \mathsf{aux}')$. This hybrid is indistinguishable from the previous one since $\mathcal{D}'$ is a trapdoored distribution for $\mathcal{D}$.

Hybrid" ω.4. In this hybrid, in the ι-th session, we generate the sample R hardcoded in DP_i using $(R, T) \xleftarrow{\$} \mathcal{D}'(1^\lambda, \mathsf{aux}'; s)$ where $s = F(K, \gamma)$. All the rest remains as in the previous hybrid. Since we provide the adversary only with K^*, this hybrid is indistinguishable from the previous one by the security of the puncturable PRF F.

Hybrid" ω.5. In this hybrid, in the ι-th session, the challenger generates the decryption program DP_i by obfuscating $\mathsf{DProg}^1_{\mathsf{Ls}}$ (see Figure 4.49), however, instead of hardcoding γ, it will hardcode the next value in the image of f that differs from $\hat{z}$[14]. In other words, DP_i will behave as $\mathsf{DProg}_{\mathsf{IP}}$ (see Figure 4.47) whenever $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) \leq_{\mathsf{lex}} \gamma$ or $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) = \hat{z}$. In the other cases, it will behave as $\mathsf{DProg}_{\mathsf{Ls}}$ (see Figure 4.31). Notice that the input-output behaviour of DP_i has not changed. We conclude that this hybrid is indistinguishable from the previous one by the security of iO.

When $\omega = q(\lambda)$, the last hybrid is indistinguishable from Hybrid 3 by the security of iO. Indeed, DP_i always behaves as $\mathsf{DProg}_{\mathsf{IP}}$ as there are no elements in the image of f such that $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) >_{\mathsf{lex}} \gamma$.

This ends the proof of the claim. ■

Hybrid 4. In this hybrid, we switch the ELF to injective mode. Observe that this stage corresponds to game $\mathcal{G}'_1$. Observe that the distinguishability advantage of $\mathcal{A}$ against Hybrid 3 and Hybrid 4 is at most $1/(4e(\lambda))$.

We conclude that the distinguishability advantage of $\mathcal{A}$ between $\mathcal{G}'_0$ and $\mathcal{G}'_1$ is at most $1/(2e(\lambda)) + \mathsf{negl}(\lambda)$. The latter is asymptotically smaller than $1/e(\lambda)$. We reached a contradiction, so no PPT adversary can distinguish between $\mathcal{G}'_0$ and $\mathcal{G}'_1$. □

Bibliography

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572. Springer, Heidelberg, May / June 2010.

[14]If γ is already the maximum in the image of f, we augment the latter with an imaginary element that is strictly greater than all other values and we hardcode it into $\mathsf{DProg}^1_{\mathsf{Ls}}$.

```
$\mathsf{DProg}^2_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K^*, f, \hat{z}, \mathsf{aux}', \gamma, R]$

Hard-coded. The index $i$ of the party, the session identity $\mathsf{sid}$, a PPRF key $K_2^{(i)}$, the encryption
program $\mathsf{EP}_i$, the hash key $\mathsf{hk}_i$, the extractable NIZK CRS $\sigma$ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$,
the punctured PRF key $K^*$, the ELF $f$, the value $\hat{z}$, the auxiliary information $\mathsf{aux}'$, the hybrid index
$\gamma$, the sample $R$.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

  1. If $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) <_{\mathsf{lex}} \gamma$ or $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) = \hat{z}$, output
     $d_i \leftarrow \mathsf{DProg}_{\mathsf{IP}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K^*, f, \mathsf{aux}']\big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\big)$ (see Figure 4.47)
  2. If $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) = \gamma$, perform the following operations:
     (a) $\forall j \neq i :\quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (\mathsf{hk}_j, \mathsf{EP}_j)\big)$
     (b) $\forall j \neq i :\quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$[a]
     (c) If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \perp$, output $\perp$
     (d) $\forall j \in [n] :\quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
     (e) $\forall j \neq i :\quad s_j \leftarrow F_1\big(K_1^{(j)}, y_j\big)$
     (f) $\forall j \in [n] :\quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2\big(K_2^{(j)}, y_j\big)$
     (g) $(\phi, \mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Sim}_1(1^\lambda, i;\, r''_i)$
     (h) $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, R, (s_j, r_j, r'_j)_{j \neq i};\, \eta'_i\Big)$ (see bottom of Figure 4.29)
     (i) Output $d_i$
  3. Otherwise, output
     $d_i \leftarrow \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K^*, f]\big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\big)$ (see Figure 4.31)
```

Figure 4.50: The unobfuscated decryption program for Hybrid” $\omega.1$

[AJJM20] Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 28–57. Springer, Heidelberg, November 2020.

[AOS23] Damiano Abram, Maciej Obremski, and Peter Scholl. On the (Im)possibility of Distributed Samplers: Lower Bounds and Party-Dynamic Constructions, 2023.

[ASY22] Damiano Abram, Peter Scholl, and Sophia Yakoubov. Distributed (correlation) samplers: How to remove a trusted dealer in one round. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 790–820. Springer, Heidelberg, May / June 2022.

[AWZ23] Damiano Abram, Brent Waters, and Mark Zhandry. Security-Preserving Distributed Samplers: How to Generate Any CRS in One Round Without Random Oracles. Cryptology ePrint Archive, Report 2023/860, 2023. https://eprint.iacr.org/2023/860.

[BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 443–459. Springer, Heidelberg, August 2004.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 52–73. Springer, Heidelberg, February 2014.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

[BL18a] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532. Springer, Heidelberg, April / May 2018.

[BL18b] Nir Bitansky and Huijia Lin. One-message zero knowledge and non-malleable commitments. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 209–234. Springer, Heidelberg, November 2018.

[BOV03] Boaz Barak, Shien Jin Ong, and Salil P. Vadhan. Derandomization in cryptography. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 299–315. Springer, Heidelberg, August 2003.

[BP04] Boaz Barak and Rafael Pass. On the possibility of one-message weak zero-knowledge. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 121–132. Springer, Heidelberg, February 2004.

[BP15] Nir Bitansky and Omer Paneth. ZAPs and non-interactive witness indistinguishability from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 401–427. Springer, Heidelberg, March 2015.

[BSW16] Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 792–821. Springer, Heidelberg, May 2016.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

[CCK$^+$22] Ran Canetti, Suvradip Chakraborty, Dakshita Khurana, Nishant Kumar, Oxana Poburinnaya, and Manoj Prabhakaran. COA-secure obfuscation and applications. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 731–758. Springer, Heidelberg, May / June 2022.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 255–271. Springer, Heidelberg, May 2003.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, March 2015.

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 630–656. Springer, Heidelberg, August 2015.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Heidelberg, August 2016.

[Gen06] Craig Gentry. Practical identity-based encryption without random oracles. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 445–464. Springer, Heidelberg, May / June 2006.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 518–535. Springer, Heidelberg, August 2014.

[GKLW21] Rachit Garg, Dakshita Khurana, George Lu, and Brent Waters. Black-box non-interactive non-malleable commitments. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 159–185. Springer, Heidelberg, October 2021.

[GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 323–341. Springer, Heidelberg, August 2007.

[GOS06a] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 97–111. Springer, Heidelberg, August 2006.

[GOS06b] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Heidelberg, May / June 2006.

[HIJ$^+$17] Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 181–211. Springer, Heidelberg, December 2017.

[HV16] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. What security can we achieve within 4 rounds? In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 486–505. Springer, Heidelberg, August / September 2016.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Heidelberg, May / June 2022.

[KK19] Yael Tauman Kalai and Dakshita Khurana. Non-interactive non-malleability from quantum supremacy. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 552–582. Springer, Heidelberg, August 2019.

[KNYY21] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Round-optimal blind signatures in the plain model from classical and quantum standard assumptions. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 404–434. Springer, Heidelberg, October 2021.

[KOR05] Jonathan Katz, Rafail Ostrovsky, and Michael O. Rabin. Identity-based zero knowledge. In Carlo Blundo and Stelvio Cimato, editors, *SCN 04*, volume 3352 of *LNCS*, pages 180–192. Springer, Heidelberg, September 2005.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

[KS17] Dakshita Khurana and Amit Sahai. How to achieve non-malleability in one or two rounds. In Chris Umans, editor, *58th FOCS*, pages 564–575. IEEE Computer Society Press, October 2017.

[LPS17] Huijia Lin, Rafael Pass, and Pratik Soni. Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In Chris Umans, editor, *58th FOCS*, pages 576–587. IEEE Computer Society Press, October 2017.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Heidelberg, October 2021.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer, Heidelberg, August 1984.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127. Springer, Heidelberg, May 2005.

[Zha16] Mark Zhandry. The magic of ELFs. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 479–508. Springer, Heidelberg, August 2016.

Chapter 5

Constant-Round Simulation-Secure Coin Tossing Extension with Guaranteed Output

Damiano Abram, Jack Doerner, Yuval Ishai, Varun Narayanan

Abstract. Common randomness is an essential resource in many applications. However, Cleve (STOC 86) rules out the possibility of tossing a fair coin from scratch in the presence of a dishonest majority. A second-best alternative is a *Coin Tossing Extension* (CTE) protocol, which uses an "online" oracle that produces a few common random bits to generate many common random-looking bits. We initiate the systematic study of *fully-secure* CTE, which guarantees output even in the presence of malicious behavior. A fully-secure two-party statistical CTE protocol with black-box simulation was implicit in Hofheinz et al. (Eurocrypt 06), but its round complexity is nearly linear in its output length. The problem of constant-round CTE with superlogarithmic stretch remained open.

We prove that *statistical* CTE with full black-box security and superlogarithmic stretch must have superconstant rounds. In the *computational* setting we prove that with N parties and polynomial stretch:

- *One round* suffices for CTE under subexponential LWE, even with Universally Composable security against adaptive corruptions.
- One-round CTE is implied by DDH or the hidden subgroup assumption in class groups, with a short, reusable Uniform Random String, and by DCR and QR, with a reusable *Structured* Reference String.
- One-way functions imply CTE with $O(N)$ rounds, and thus constant-round CTE for any constant number of parties.

Such results were not previously known even in the two-party setting with standalone, static security. We also extend one-round CTE to sample from *any* efficient distribution, via strong assumptions including IO.

Our one-round CTE protocols can be interpreted as *explainable* variants of classical randomness extractors, wherein a (short) seed and a source instance can be efficiently reverse-sampled given a random output. Such explainable extractors may be of independent interest.

5.1 Introduction

Common randomness is a crucial resource in many applications, yet after 40 years of research, it remains difficult and sometimes even impossible to generate in many settings. The problem of flipping common coins was first posed and solved by Blum [Blu82] in the two-party context,[1] but his protocol did not ensure that both parties received an output in the case that one of them deviated from the protocol instructions. Shortly thereafter, Cleve [Cle86] proved that this "fairness" problem is inherent and unconditional: *any* r-round coin-tossing protocol that is guaranteed to output a common bit must suffer an inverse-polynomial bias of $\Omega(1/r)$ if a majority of the participants may be corrupted. In any round of interaction, corrupted parties may *rush* to see the messages of the honest parties before they transmit their own, and condition their responses (or choose not to respond) on the honest parties' contributions.

Cleve's bound has left a dichotomy in the plain model: in the face of a dishonest majority, either one must accept biased coins, or one must accept that the adversary can block the sampling of common coins entirely. Follow-up works have followed both pathways; for example, Buchbinder et al. [BHLT17] finally achieved guaranteed output with optimal bias for any constant number of parties, while Lindell [Lin03] showed that polynomially-many coins can be tossed in constant rounds without guaranteed output, under the minimal assumption of one-way functions in the plain model. The impossibility of reliable unbiased coin tossing remains barrier to constructing a wide swath of essential primitives with guarantees against adversarial denial-of-service in the plain model, such as broadcast and consensus protocols, election protocols, lotteries, *Common Random String* (CRS) setup for cryptographic protocols, and many others.

The only means to evade Cleve's bound is to assume the impossibility away. Suppose there exists an oracle that outputs a small number n of common random coins when invoked (n can be thought of as a security parameter). This oracle might be implemented, for example, by a natural process or by an expensive honest-majority protocol. From such an ideal object, a protocol to flip at least one unbiased coin is trivial. We call this oracle $\mathcal{F}^n_{\mathsf{Coin}}$.

Even in settings where unbiased common coins can be found, it is prudent to assume that they are few or expensive to access. If an ideal n-coin oracle allows one to evade Cleve for the flipping of one coin, can it do so for $n+1$ coins, or more? In other words, can the oracle's output serve as a seed? A multiparty protocol that uses an n-coin seed oracle to sample m apparently-uniform common coins for $m > n$ is known as a *Coin Tossing Extension* (CTE) protocol; if it invokes the seed oracle t times, we say that it has a *stretch* of $m - t \cdot n$. Bellare et al. [BGR96] originally introduced the notion of CTE, motivated primarily by concrete efficiency and *without* any guarantee that output will be produced in the presence of malicious behavior. CTE protocols can be thought of as the distributed analogue of classical randomness extractors, and like classical extractors they require that the uniform seed be independent of the entropy source. This can be ensured by revealing the seed at the *end* of the protocol, after the corrupt parties (who exert partial control over the source) are committed to their contributions, which necessitates an "online" seed oracle, rather than a seed established ahead of time.

Game-Based Security is not Enough. If we insist only that a coin tossing extension protocol produce a common output that is indistinguishable from a uniform bit string, then optimal CTE protocols with guaranteed output and unconditional security are easy to construct. If one way functions exist, then so does a trivial protocol: the parties simply use the n-bit output of $\mathcal{F}^n_{\mathsf{Coin}}$ to seed a *Pseudorandom Generator* (PRG), which can produce a string of $m(n)$ bits for any polynomial m that is indistinguishable from uniform to all adversaries whose runtimes are also bounded polynomially in n. Notice that this protocol does not involve any communication at all, apart from invoking the oracle! If one way functions do not exist or the adversary is unbounded, then the solution need not be much more complicated. Consider the following simple protocol for N parties:

[1]With proposed applications to the equitable distribution of formerly-shared property among recent divorcées.

Protocol 5.1.1. Game-based Statistical Coin Tossing Extension

1. The parties exchange ℓ-bit uniformly-sampled strings over a broadcast channel; if any party fails to send a string then its string is taken to be all zeroes.
2. The parties query $\mathcal{F}^n_{\mathsf{Coin}}$, receiving $\boldsymbol{u} \in \{0,1\}^n$ in response.
3. They concatenate their strings to form $\boldsymbol{x} \in \{0,1\}^{\ell \cdot N}$.
4. The parties individually apply a classical randomness extractor: $\boldsymbol{s} \leftarrow \mathsf{Extract}(\boldsymbol{x}, \boldsymbol{u})$, and output $\boldsymbol{s}$.

This protocol produces an output ε-close to uniform if there exists an (ℓ, ε)-extractor $\mathsf{Extract} : \{0,1\}^{\ell \cdot N} \times \{0,1\}^n \to \{0,1\}^m$ [NZ96]. For $\varepsilon \in \mathsf{negl}(n)$ and $\ell \in O(m+n)$, there is a construction of such extractors from universal hashing [ILL89].

The simplicity of these protocols comes with a shortcoming: though they may produce outputs indistinguishable from uniform, their outputs *cannot* be used to replace a truly uniform string in any given context. This can be seen by means of a simple example: suppose that the adversary is challenged to produce an n-bit representation of m uniformly sampled bits, for $n < m$. If these m bits are sampled by $\mathcal{F}^m_{\mathsf{Coin}}$ directly, then the adversary will fail with probability overwhelming in $m - n$. On the other hand, if the bits are produced by the simple computational CTE protocol we have just described, then the n-bit intermediate output of the seed oracle $\mathcal{F}^n_{\mathsf{Coin}}$ is *exactly* such a representation, and thus the protocol cannot be used in place of the oracle, even though their outputs are indistinguishable from one another. This is a simple and specific example of the general separation between *game*-based security and *simulation*-based security.[2]

Simulation-Secure CTE. Simulation-based security—otherwise known as security in the real/ideal-paradigm [Can00, Gol04]—insists that there exist an efficient simulator algorithm to produce a protocol transcript given any protocol output, and that the simulated transcript must be indistinguishable from a transcript of the real protocol. If the oracle $\mathcal{F}^m_{\mathsf{Coin}}$ in the *ideal* world is replaced by some simulation-secure protocol in the *real* world, then the simulator ensures that for any attack on the protocol, an equivalent attack can be mounted against $\mathcal{F}^m_{\mathsf{Coin}}$; thus, the protocol can be used in any context $\mathcal{F}^m_{\mathsf{Coin}}$ can be. A protocol is said to *black-box* simulation-secure if the simulator does not need to inspect the code of the protocol's adversary, but only to run the adversary as a subroutine. It is said to be *Universally Composable* (UC) [Can01] if the simulation property is guaranteed to hold in an arbitrary protocol context defined by an adversarial environment,[3] and it is said to have *standalone* security if simulatability is only guaranteed to hold for one instance at a time. We refer to any protocol that has both guaranteed output and black-box standalone-simulatable (or universally composable) security against a malicious adversary corrupting a dishonest majority of participants as *fully secure*.

Hofheinz et al. [HMU06] gave a thorough treatment of simulation-secure CTE *without* guaranteed output (i.e. permitting the adversary to force the protocol to abort), with a specific focus on the *feasibility* of CTE for two parties in a set of six contexts comprising each combination of perfect, statistical, or computational security, and either black-box standalone security or universally composable security. They proved that perfectly secure CTE and statistically universally composable CTE are both impossible even for *one bit* of stretch. On the other hand, they proved that it is possible to achieve *polynomial* stretch in the computational universally composable case via a constant-round construction based on commitments; their protocol explicitly allows aborts to occur. They also proved that it is possible to achieve polynomial strech in the statistical standalone case via a protocol with a round count linear in the number of output bits. Although they make no explicit claim about guaranteed output, this protocol does not in fact contain an abort: to our knowledge this makes it the *first* fully-secure CTE protocol, and the first to evade Cleve's bound.

[2]Hofheinz et al. [HMU06] proved that Protocol 5.1.1 can in fact be simulated for some parameterizations, but not all.
[3]Other models exist that achieve a similar goal. UC security is always black-box.

5.1.1 Our Contributions

Evading Cleve with Fully-Secure CTE. In this work, we initiate the systematic study of fully-secure coin tossing extension, and pose the question:

> How many rounds does fully-secure coin tossing extension require?

It is clear that there must be at least one invocation of an "online" oracle, or else Cleve's bound would apply and we would be forced to sacrifice either guaranteed output or unbiased output. It is also clear that there must be at least one round of communication among the parties, or else there could be no additional entropy with which to extend the coins that the oracle produces. We begin our study by proving that any interaction after the coin-tossing oracle $\mathcal{F}^n_{\mathsf{Coin}}$ is invoked is useless. This implies that if the adversary is given the (standard) power to rush, then the communication of the parties must occur *before* the invocation of $\mathcal{F}^n_{\mathsf{Coin}}$ and thus the minimal interaction for *any* fully-secure CTE protocol is one round, followed by one oracle invocation. This result is proven in Section 5.4.1.

In the black-box standalone statistical setting, we prove much more interaction than this is required: specifically, we prove that a protocol with r rounds of interaction before the oracle's invocation cannot output more than $O(r \cdot \log \lambda)$ bits, where λ is a security parameter. This implies that the protocol of Hofheinz et al. is *nearly* optimal with linear stretch, and with a simple adjustment optimality can easily be achieved. Since Hofheinz et al. have already proved that statistically universally composable CTE is impossible, even without guaranteed output or polynomial stretch, this effectively closes the question of the round complexity of statistical CTE with black-box simulation. This result is proven in Section 5.8.

To evade the bound we have just proved, we turn to computational security and cryptographic assumptions. Under the subexponential *Learning with Errors* (LWE) Assumption [Reg05], we construct an N-party protocol that achieves polynomial stretch with the optimal interaction pattern: *one* round, followed by an invocation of $\mathcal{F}^n_{\mathsf{Coin}}$. Furthermore, this protocol is universally composable and secure against malicious adversaries *adaptively* corrupting up to $N-1$ parties. Due to its basis in a lattice assumption it is also plausibly post-quantum secure. Indeed, we do not know of a stronger bounded adversary than the one implied by this combination of attributes. This result is proven in Section 5.5.

By allowing a trusted setup and settling for weaker notions of security, we expand the set of assumptions we can rely on: using a general *hidden subgroup* framework, we devise a one-round fully-secure and universally composable N-party CTE protocol in the Common Reference String (CRS) model. Unlike our previous construction, this one is proven only to have security against the *static* corruption of $N-1$ participants. We show how to instantiate our framework from the *Decisional Diffie-Hellman* (DDH) assumption [DH76] and the *Hard Subgroup Membership* (HSM) assumption[4] on class groups [CL15]: both of these assumptions yield short, reausable, *uniform* CRSes that can be sampled via one extra call the very same coin oracle $\mathcal{F}^n_{\mathsf{Coin}}$ that the protocol extends. We also give an instantiation of the framework using Paillier encryption [Pai99] (and thus under both the *Decisional Composite Residuosity* (DCR) and *Quadratic Residuosity* (QR) assumptions); this instantiation requires a reusable *structured* CRS that cannot be so trivially sampled. These results are proven in Section 5.6.

We find a result in even the weakest of computational assumptions: *One-Way Functions* (OWFs). A close inspection of the OWF-based constant-round N-party coin tossing protocol of Goyal et al. [GLOV12] reveals that it can achieve *Identifiable Abort* (IA) [IOZ14], though the authors did not claim this. This type of security does not guarantee an output, but does ensure that if no output is produced, then the honest parties can agree upon the identity of at least one malicious party. Simple iteration of this protocol to eliminate cheating parties one-by-one can ensure that an output is produced in $O(N)$ rounds, even if $N-1$ parties are corrupt, but cannot prevent the adversary from biasing that output. We describe a simple way to eliminate this bias using a sample from $\mathcal{F}^n_{\mathsf{Coin}}$, yielding a constant-round fully-secure coin tossing protocol for any constant number of parties. This result is proven in Section 5.7.

Fully-Secure Distributed Sampling. Abram et al. [ASY22, AWZ23] recently introduced the notion of *Distributed Sampling*, which can be thought of as a generalization of coin tossing to any (efficiently-samplable)

[4] Also known as the Hidden Subgroup Assumption.

distribution, and they proved the existence of one-round distributed samplers in the malicious, dishonest-majority setting under a set of strong assumptions, including *Indistinguishability Obfuscation* (iO) [BGI+01]. They could not, however, construct one-round unbiased distributed samplers with guaranted output without running afoul of Cleve [Cle86], and settled instead for a notion they referred to as *indistinguishability-preservingness*. In this work, we augment their techniques with a single call to the coin tossing oracle $\mathcal{F}^{n}_{\mathsf{Coin}}$ to achieve what they could not. This result is proven in Section 5.9.

Explainable Randomness Extractors. At the beginning of this section, we explained coin tossing extension as the multiparty analogue of a randomness extractor. This analogy is not an accident: when the parties in a coin tossing extension protocol perform a final local computation to recover the protocol's output from the coin oracle's output and the transcript of the protocol before the oracle was invoked, they are *precisely* invoking a randomness extractor with the pre-oracle transcript as the source and the oracle's output as the seed. For one-round protocols as several of ours are, the source is simply the concatenation of the parties' single messages. The black-box simulatability of our protocols suggests a heretofore-unidentified property of certain extractors: it is efficient to find a seed and source that yield a chosen output under the extractor, such that the distribution of seeds is independent of the output, and the sources belong to some well-defined distribution that is tolerated by the extractor. We refer to this property as *explainability*, and formally prove the correspondance; as corollaries our other results imply explainable *computational* extractors for various source distributions under various assumptions. Correspondingly, the statistical CTE protocol of Hofheinz et al. [HMU06] implies an explainable *statistical* extractor for a particular source distribution. This result is proven in Section 5.4.2.

Open Questions.

This work leaves an interesting question open:

Do there exist constant-round and statistically simulation-secure coin tossing extension protocols with $\omega(\log \lambda)$ stretch?

This paper shows that if we relax this question by considering computational security, the answer is *yes* (under standard cryptographic assumptions). On the other hand, for the more stringent variant of the question that requires *black-box simulation*, the answer is unconditionally *no*. Thus, an affirmative answer to our question would separate black-box and non-black-box simulation in the statistical security setting. We are not aware of such a separation in the literature.

5.2 Technical Overview

Notation. Let λ be the security parameter. For any $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, \ldots, n\}$. We use $\lfloor x \rfloor$ to denote the integral part of the number x. We use bold font to denote vectors and capital letters for matrices. We use $\|\cdot\|_\infty$ to denote the ℓ_∞-norm (i.e. the maximum magnitude of the entries of the vector). We use $=$ to denote equality, $:=$ to denote equality by definition of the left hand side by the right hand side, $\equiv$ to denote congruence, and $\leftarrow$ to denote deterministic assignment. We use $\overset{\$}{\leftarrow}$ in an overloaded fashion to denote sampling from a distribution, assignment using a randomized algorithm, or, if the right hand side is a domain, uniform sampling from that domain. We use $\sim$ to denote that two scalars approximate each other, or in some contexts that they are are negligibly close in the security parameter, and we use $\approx_{\mathrm{s}}$ and $\approx_{\mathrm{c}}$ to denote that distributions are statistically or computationally indistinguishable, respectively.

A *Coin Tossing Extension* (CTE) protocol consists of an N-party protocol that produces m unbiased random bits given a source of $n < m$ unbiased random coins. Formally, our security definition is based on black-box simulation: we insist that a CTE protocol *realizes* the functionality $\mathcal{F}^{m}_{\mathsf{Coin}}$ that provides all parties with a random string $\boldsymbol{s} \overset{\$}{\leftarrow} \{0,1\}^m$ in the presence of a malicious PPT adversary corrupting up to $N-1$ parties, in the *hybrid model* of the functionality $\mathcal{F}^{n}_{\mathsf{Coin}}$ (otherwise known as the *seed oracle*) that supplies a random string $\boldsymbol{u} \overset{\$}{\leftarrow} \{0,1\}^n$. $\mathcal{F}^{n}_{\mathsf{Coin}}$ and $\mathcal{F}^{m}_{\mathsf{Coin}}$ are identical, apart from their parametrization.

Functionality 5.2.1. $\mathcal{F}^n_{\mathsf{Coin}}$. The Coin Tossing Functionality

Initialisation: On `init` from all parties, the functionality activates.

Coins: On receiving $(\mathtt{flip}, \mathsf{sid})$ from all parties, the functionality samples $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$ and sends $(\mathtt{coins}, \mathsf{sid}, \boldsymbol{s})$ to all parties.

We highlight that $\mathcal{F}^m_{\mathsf{Coin}}$ has *guaranteed output*, and thus our CTE protocols achieve *full security*: even if the corrupted parties deviate from the protocol, the honest parties will always agree on an unbiased random output. In general, both n and m may be polynomials in the security parameter λ, but we often leave this implicit. We define the round complexity of the protocol to be the number of rounds of interaction between the parties before the final invocation of $\mathcal{F}^n_{\mathsf{Coin}}$, the sampling complexity t to be the number of times $\mathcal{F}^n_{\mathsf{Coin}}$ is invoked, the additive stretch to be $m - t \cdot n$, and the multiplicative stretch to be $m/(t \cdot n)$.

Throughout this paper, we consider different flavours of security: most of our constructions achieve computational security in the UC model, and one is only standalone-secure. One construction achieves security against adaptive corruptions. Our lower bound applies to the information-theoretic case with black-box simulation. We always assume that the parties are connected by authenticated, private, point-to-point channels, and by an authenticated broadcast medium.

5.2.1 The Round Structure of CTE Protocols

In a coin tossing extension protocol, the seed oracle behaves very differently from a typical *trusted setup*: in order to evade Cleve's impossibility [Cle86], the random coins must be delivered throughout the execution of the protocol and not at the beginning. We begin by proving that any round of interaction after the last call to the seed oracle is essentially useless. Immediately after the last call, the parties must already agree on an unbiased random string of the right length. This fact holds even for CTE protocols that rely on non-black-box simulators.

Theorem 5.2.2 (Informal Version of Theorem 5.4.3). Let Π be a CTE protocol producing m random bits. Let Π' be the protocol in which the parties behave exactly as in Π until the last call to the seed oracle, after which they stop interacting. Π' is a secure coin tossing extension protocol producing m random bits.

Proving the above theorem begins with the simple observation that the output of any honest party P_i in Π' is the value it would return if it were executing Π and all other parties ceased interacting (as they might, if they are corrupt) after last call to the entropy source. This implies that P_i's output in Π' cannot be noticeably biased by the adversary. It is more challenging to prove that the outputs of the honest parties in Π' all coincide. We show this by following the blueprint of Cleve's impossibility result [Cle86] and applying a similar argument in the N-party setting. Let α be the index of one (arbitrary) bit of the output, and let P_i and P_j be two honest parties. Let the random variables $b_{i,r}$ and $b_{j,r}$ be the α^{th} bit that P_i and P_j would output in Π if all other parties ceased interacting in the r^{th} round *after* the last call to the seed oracle. By the security of Π, the bits $b_{i,r}$ and $b_{j,r}$ must be equal for sufficiently large r, because, at the end of the protocol, P_i and P_j are guaranteed to agree. Our goal is to prove that, whatever the value of α and the behaviour of the adversary $\mathcal{A}$, the bits $b_{i,0}$ and $b_{j,0}$ are equal. For every adversary $\mathcal{A}$, every round r, and $b \in \{0,1\}$, we define two modified adversaries, $\mathcal{A}^b_{r,0}$ and $\mathcal{A}^b_{r,1}$. The adversary $\mathcal{A}^b_{r,0}$ corrupts all parties except P_i. Any parties that $\mathcal{A}$ would corrupt behave when corrupted by $\mathcal{A}^b_{r,0}$ as they would when corrupted by $\mathcal{A}$; for all other parties, $\mathcal{A}^b_{r,0}$ follows the protocol. At the r^{th} round after the last call to the seed oracle, $\mathcal{A}^b_{r,0}$ simulates the execution of the round in its head using P_i's message ($\mathcal{A}^b_{r,0}$ is a rushing adversary, and thus has this message before the round completes). If it predicts that $b_{j,r+1} = b$, then $\mathcal{A}^b_{r,0}$ sends no further messages; otherwise, it sends the messages of the execution it simulated in its head and ceases communication thereafter. The adversary $\mathcal{A}^b_{r,1}$ is slightly simpler: it corrupts all parties except P_j and determines their behavior just like $\mathcal{A}^b_{r,0}$ did. At the r^{th} round after the last call to the seed oracle, $\mathcal{A}^b_{r,1}$ checks whether $b_{i,r} = b$, and sends no further messages if so; otherwise, it ceases communication at the end of the following round. Using Cleve's argument [Cle86], we can prove that unless $b_{i,0} = b_{j,0}$, at least one of the adversaries $(\mathcal{A}^b_{r,0}, \mathcal{A}^b_{r,1})_{r,b}$ must bias the α^{th} bit of the output of Π by a non-negligible amount, with overwhelming probability.

This result is formalized in Section 5.4.1.

5.2.2 Coin Tossing Extension and Explainable Extractors

A randomness extractor is a primitive that converts samples from a high-entropy source into true randomness, with the aid of an auxiliary source of truly random bits, referred to as a *seed*. The samples provided by the high-entropy source should be independent of the seed, and the seed is typically required to be much shorter than the output. Formally, we require that for every source of sufficiently high entropy S, the following distribution is indistinguishable from the uniform distribution:

$$\left\{\mathsf{Extract}(\boldsymbol{x},\boldsymbol{u})\middle|\boldsymbol{x}\xleftarrow{\$}S,\boldsymbol{u}\xleftarrow{\$}\{0,1\}^n\right\}$$

In Section 5.4.2, we introduce the notion of an *explainable extractor*: a randomness extractor that satisfies a stronger, simulation-based security definition relative to some class of entropy sources $\mathcal{S}$. Specifically, for every source $S \in \mathcal{S}$, there must exist a PPT simulator $\mathsf{Explain}_S$ such that the following distributions are indistinguishable:

$$\left\{\mathsf{aux},\boldsymbol{x},\boldsymbol{u},\boldsymbol{s}\middle|\begin{array}{l}(\boldsymbol{x},\mathsf{aux})\xleftarrow{\$}S\\ \boldsymbol{u}\xleftarrow{\$}\{0,1\}^n\\ \boldsymbol{s}\leftarrow\mathsf{Extract}(\boldsymbol{x},\boldsymbol{u})\end{array}\right\}\qquad\left\{\mathsf{aux},\boldsymbol{x},\boldsymbol{u},\boldsymbol{s}\middle|\begin{array}{l}\boldsymbol{s}\xleftarrow{\$}\{0,1\}^m\\ (\mathsf{aux},\boldsymbol{x},\boldsymbol{u})\xleftarrow{\$}\mathsf{Explain}_S(\boldsymbol{s})\end{array}\right\}$$

We prove that fully-secure CTE protocols imply explainable extractors. In particular, any CTE protocol can be viewed as an explainable extractor for the class of entropy sources that is generated by running the CTE protocol with an adversary corrupting at most $N-1$ parties, and then outputting the transcript $\boldsymbol{x}$ and the view of the adversary aux just before the last call to the seed oracle. The extractor $\mathsf{Extract}(\boldsymbol{x},\boldsymbol{u})$ simply emits an honest party's (truly random) output using the transcript, simulating the seed oracle via $\boldsymbol{u}$, and performing any final computations via the honest party's code. The algorithm $\mathsf{Explain}_S$ can easily be derived from the simulator of the CTE protocol.

In Section 5.5, we use this fact to prove the existence of an explainable extractor for the class of entropy sources that produce N blocks of $\mathsf{poly}(\lambda)$ bits $\boldsymbol{x_1},\ldots,\boldsymbol{x_N}$, such that one block (say $\boldsymbol{x_i}$) is truly random, and the other blocks and aux are produced by any PPT algorithm receiving $\boldsymbol{x_i}$ as input. See Corollary 5.5.7.

5.2.3 Computational Coin Tossing Extension with Long Stretch

We now describe our CTE constructions starting from the simplest to the most sophisticated. All of our schemes achieve security against adversaries corrupting up to $N-1$ parties and require a single call to the seed oracle.

On UC-Security and Arbitrary Polynomial Stretch at No Round Cost. If a UC-secure coin tossing extension protocol generates $m > n$ random bits using a single call to the seed oracle, then we immediately obtain a UC-secure coin tossing extension protocol with the same round complexity, a single call to the seed oracle, and arbitrary polynomial stretch: all we must do is run the original CTE protocol many times in parallel. We use the coins produced by the seed oracle in the first execution, and then use n bits of the resulting output as the seed for the second execution, reserving at least one bit for the final output, and so on. We obtain at least one additional bit per instance.

From One-Way Functions via Coin Tossing with Identifiable Abort.

We start from the simplest construction: we consider a secondary coin tossing functionality with identifiable abort and *not* guaranteed output. We refer to this functionality as $\mathcal{F}^m_{\mathsf{Coin+IA}}$; it can be realized by the protocol of Goyal et al. [GLOV12] in the standalone setting assuming one-way functions exist. We might naïvely hope to build a fully-secure CTE protocol via player elimination. The parties invoke $\mathcal{F}^m_{\mathsf{Coin+IA}}$, and if the invocation succeeds, then they output the random string produced by it; otherwise, they repeat the invocation of $\mathcal{F}^{m}_{\mathsf{Coin+IA}}$ *without* the party that cause the abort. After at most $N-1$ attempts, the honest parties are guaranteed to agree on a random string. Unfortunately, this construction allows the adversary to bias the output. Remember, we have not relied on the seed oracle, so Cleve's impossibility [Cle86] must apply. During each invocation of $\mathcal{F}^{m}_{\mathsf{Coin+IA}}$, the adversary learns the candidate output before the honest parties, and can choose to abort or accept the result; conditioning this choice on the candidate output allows bias to be injected. For example, if the adversary desires the initial bit of the output to be 0, then it can abort the invocation of $\mathcal{F}^{m}_{\mathsf{Coin+IA}}$ only when the candidate output starts with a 1.

We prevent this bias attack by "encrypting" the output of the CTE protocol using a PRG. Specifically, after $\mathcal{F}^{m}_{\mathsf{Coin+IA}}$ produces an output, the parties invoke the seed oracle $\mathcal{F}^{\lambda}_{\mathsf{Coin}}$ to obtain a λ-bit seed, where λ denotes the security parameter. They expand this seed with a PRG, and output the XOR of the expansion and the output of $\mathcal{F}^{m}_{\mathsf{Coin+IA}}$.

We highlight that even with this modification, the adversary still has the ability to cause an abort and force the honest parties to invoke $\mathcal{F}^{m}_{\mathsf{Coin+IA}}$ repeatedly. Compared to the naïve protocol, the adversary must now face the choice of whether to abort blindly: even given the candidate output of $\mathcal{F}^{m}_{\mathsf{Coin+IA}}$, the adversary cannot predict the output of the CTE protocol. At the time the adversary must make the decision, the privacy of the final output is guaranteed by the security of the PRG, because the λ-bit seed chosen by the seed oracle is not yet revealed. This result is expounded in Section 5.7. Ultimately, we prove:

Corollary 5.2.3. If one-way functions exist, then for any constant number of parties there is a constant-round fully-secure CTE protocol in the plain model, with standalone black-box simulatability against a malicious PPT adversary statically corrupting all parties but one. This construction is black-box in the OWF.

An Algebraic Framework for Coin-Tossing Extension.

After discovering a fully-secure CTE protocol from coin tossing with identifiable abort, we wondered whether it is possible to guarantee output without restarting in response to malicious behavior. We answer this question affirmatively, and moreover demonstrate that only a *single* round of interaction followed by a single call to the seed oracle is required in the CRS model. Our construction is an *algebraic framework* for coin tossing extension that can be instantiated using DDH groups, class groups, or Paillier Encryption.

One-Round CTE against Rushing Adversaries. One of the main challenges in designing a one-round CTE protocol is dealing with rushing behaviour. For the moment, imagine that our goal is to construct a protocol with black-box simulation and *no* CRS. In the ideal world, the corresponding functionality provides the simulator with the output of the protocol, and then the simulator must generate fake but consistent messages for the honest parties. The simulator must do this without knowing the messages of the corrupted players, since the honest party is assumed to speak first. In principle, the adversary might be able to inject so much randomness in the protocol that the Shannon entropy of the output $\boldsymbol{s}$ conditioned on the messages generated by the simulator U_H is greater than $n+\omega(\log\lambda)$; specifically

$$\mathsf{H}(\boldsymbol{s}|U_H) \geq n+\omega(\log\lambda). \tag{5.1}$$

Indeed, what happens if the adversary is the simulator itself, using a random string in place of the functionality's output? Without a CRS, this is a possibility! If (5.1) holds, the simulator is doomed to fail: the only power it has is to rewind, or generate a random-looking n-bit response $\boldsymbol{u}$ on behalf of the seed oracle. The latter will not help because the entropy of $\boldsymbol{u}$ is bounded by n; it is too low to fully correct the bias induced by the adversary. Rewinding will not help, since it is equivalent to restarting: each execution is overwhelmingly likely to fail.

Relying on a CRS: the Hidden Subgroup Framework. Now that we have understood what the main challenges are, we relax our requirements to permit a CRS. We hope that by relying on the common reference string, we can restrict the influence of the adversary while allowing full freedom to the simulator. We demonstrate that this is possible using an algebraic framework inspired by the work of Abram et al. [ADOS22], which we refer to as the *the hidden subgroup framework*.

Theorem 5.2.4 (Informal Version of Theorem 5.6.4). Given any instantiation of the hidden subgroup framework, there exists a one-round N-party fully-secure protocol in the $\mathcal{F}^n_{\mathsf{Coin}}$-hybrid CRS model that UC-realizes $\mathcal{F}^m_{\mathsf{Coin}}$ in the presence of a malicious PPT adversary statically corrupting up to $N-1$ parties.

Consider a multiplicative group G with a smaller subgroup H. Suppose that we can efficiently sample uniformly random elements from both groups, but the two distributions are computationally indistinguishable. Our CRS consists of a description of G and H along with the CRS for a simulation-extractable NIZK. Each party P_i broadcasts a random sample $h_i \in H$, along with a NIZK proving that the sample belongs to H. If any NIZK does not verify, the party that generated it is excluded from the execution of the protocol (without restarting). Next, the seed oracle provides the parties with the description of another random element $h \in H$. The parties output the product $h \cdot \prod_i h_i$, ignoring every h_j for which the corresponding NIZK does not verify.

We argue that from the adversary's perspective, the output is indistinguishable from a random element in G. Indeed, a simulator that receives $g \xleftarrow{\$} G$ from the functionality can generate a trapdoored CRS for the NIZK and send $g \cdot h_\iota$ instead of h_ι on behalf of an arbitrary honest party P_ι. The corresponding NIZK is simulated using the trapdoor, and the simulator sends a correcting term $h \leftarrow \prod_i h_i^{-1}$ on behalf of the seed oracle (where, again, the product ignores all indices corresponding to non-verifying NIZKs). Notice that h is a random element in H due to h_ι. The output of this protocol execution is g, and the transcript is indistinguishable from a real protocol transcript due to the security of the NIZK and the indistinguishability of uniform elements in G and H.

The Representation of the Group Elements. There is a problem yet to be solved in the blueprint we have just given: a CTE protocol must produce more random bits than those provided by the seed oracle. The entropy of a random element in G is higher than the entropy of random elements in H, but since the two distributions are indistinguishable, it would seem that the representations of elements in G and H require strings of the same length. In other words, it seems that in the foregoing construction, the seed oracle provides as many bits as are produced by the protocol. This is not the case, however: the stretch depends on how we represent the response of the seed oracle. This representation can be compressed because it is known to *always* be in H. If H is a cyclic group of order q and h_0 is a generator, then the seed oracle can represent any $h = h_0^r$ as $r \bmod q$. This representation is optimal as it requires roughly $\log q = \log|H| < \log|G|$ bits, and it yields a protocol with positive stretch.

Using such a representation introduces another issue, though: how can the simulator obtain a succinct representation of $\prod_i h_i^{-1}$? Doing so implies computing a discrete logarithm. This is the purpose of the simulation-extractable NIZKs: the simulator samples the discrete logarithms of the elements chosen by the honest parties (including h_ι), and extracts the discrete logarithms of the elements of the corrupted parties using their NIZKs.

We require three additional properties from our framework:

- There should exist a succinct representation for the elements of H. For any $h_i \in H$, we denote the representation by ρ_i. This representation may not be unique.
- Given elements $h_1, \ldots, h_\ell \in H$ and corresponding succinct representations $\rho_1, \ldots, \rho_\ell$, we must be able to obtain a succinct representation of $\prod_{i\in[\ell]} h_i^{-1}$
- There must be a way to sample h_1 and a corresponding succinct representation ρ_1 such that, for every $h_2, \ldots, h_\ell \in H$ and corresponding $\rho_2, \ldots, \rho_\ell$, the succinct representation of $\prod_{i\in[\ell]} h_i^{-1}$ derived from $\rho_1, \rho_2 \ldots, \rho_\ell$ is indistinguishable from the succinct representation of a uniform element in H.

One Last Property: Converting the Output into Bits. A CTE protocol is supposed to produce random bits. The protocol we described above outputs a random group element $g \in G$ instead. How do we convert this into a random string? This can be surprisingly challenging! We cannot simply apply an arbitrary hash function f; the procedure must be explainable. There must be a way for the simulator to convert the random string $\boldsymbol{s}$ obtained from the functionality into a random group element g such that $f(g) = \boldsymbol{s}$. In other words, our framework also requires the existence of an efficiently invertible deterministic function f such that the following distributions are indistinguishable.[5]

$$\left\{g, f(g) \middle| g \xleftarrow{\$} G\right\} \qquad \left\{f^{-1}(\boldsymbol{s}), \boldsymbol{s} \middle| \boldsymbol{s} \xleftarrow{\$} \{0,1\}^m\right\}$$

Instantiations of the Framework. We present three instantiations of the hidden subgroup framework: one based on DDH over cyclic groups, one based on Paillier Encryption [Pai99], and one based on class groups of imaginary quadratic fields [CL15].

- Let $\widehat{G}$ be a cyclic group of prime order p wherein DDH is hard. Inspired by Peikert et al. [PVW08], we choose G to be the product $\widehat{G} \times \widehat{G}$. The subgroup H consists of all pairs $(g_1, g_2) \in G$ such that $g_2 = g_1^\alpha$ for a randomly sampled, secret $\alpha \xleftarrow{\$} \mathbb{Z}_p$. It is easy to see that H is a proper subgroup of G. Furthermore, it is hard to distinguish between the uniform distributions over G and H under the DDH assumption. Since H is cyclic with order p, we can also succinctly represent any element in H as the usual discrete logarithm of (g_1, g_2) with respect to some generator (g_0, g_0^α). This succinct representation satisfies the properties required by the framework. Finally, the matter of conversion from G into random bits depends greatly on the choice of $\widehat{G}$. If the latter is a cyclic subgroup of $\mathbb{F}_q^*$ for some power of a prime q or of an elliptic curve, conversion is usually easy as long as the cofactor is small.

- Consider the Paillier group $G := \mathbb{Z}_{N^2}^*$ where N is the product of two large, random, safe primes $p = 2p' + 1$ and $q = 2q' + 1$. The subgroup H will consist of all $2N^{\text{th}}$ powers of G. H is a subgroup of order $p' \cdot q'$, and since $p' \neq q'$ are primes, all abelian groups of this order are cyclic. Under the QR assumption and the DCR assumption, no PPT adversary can distinguish between a random element in G and a random $2N^{\text{th}}$ power. As before, we can succinctly represent any element in H via the discrete logarithm with respect to a fixed generator h_0 (h_0 can be a random $2N^{\text{th}}$ power). This instantiation differs from the previous one only in that the order of H is unknown: to sample a random element in H with a known succinct representation, we must sample $\rho \xleftarrow{\$} [N]$ and set $h \leftarrow h_0^\rho$. We use flooding to ensure the third property of succinct representations.

- Consider a class group $\widehat{G}$ and let F denote the cyclic subgroup of prime order q where the discrete logarithm problem is easy. Let h_0 be a random element in $\widehat{G}$ of order coprime with q. Let ℓ be 2^λ times greater than an upper-bound on the order of h_0. The subgroup H is generated by h_0 and G is $F \times H$. Under the hidden subgroup membership assumption, the uniform distributions over H and G are indistinguishable. Once again, we can succinctly represent the elements of H through their discrete logarithm with respect to h_0, and as in the Paillier case, the order of H is unknown, so we generate random elements with a known succinct representation by sampling $\rho \xleftarrow{\$} [\ell]$ and computing $h \leftarrow h_0^\rho$. Also as in the Paillier case, we use flooding to ensure the third property of succinct representations. While our class group instantiation has an advantage over our Paillier instantiation in that we can generate the parameters of the group transparently, there is also an important disadvantage: as far as we know, there exists no explainable procedure that converts random elements in G into random strings of bits. In other words, we do not know how to ensure the last property of our framework.

These results are formalized in Section 5.6, with the three instantiations of our framework being presented in subsections 5.6.1, 5.6.2, and 5.6.3 respectively.

One-Round Setup-Free Adaptively-Secure Coin Tossing Extension.

We return to the question of whether it is possible to built one-round CTE protocols *without* a CRS. As we explained in the context of our construction from the hidden subgroup framework, the goal of using a CRS is to limit adversarial influence on the output, while giving the simulator freedom. With no CRS, the simulator can only restrict the influence of the adversary through the responses of the seed oracle. This creates a new challenge: because the responses of the seed oracle are revealed only at the end, we cannot use well-studied

[5] f^{-1} may be non-deterministic.

primitives, such as NIZKs. We have eliminated the most common non-interactive MPC tool for preventing malicious behavior. We must therefore develop new techniques.

Lattice-Based Lossy Trapdoor Functions. Our solution comes from lattice-based cryptography. In our protocol, each party P_i will broadcast L vectors $\boldsymbol{x_{i,1}}, \ldots, \boldsymbol{x_{i,L}} \in \mathbb{Z}_q^M$ sampled from a discrete Gaussian distribution. The seed oracle will provide matrices $A_1, \ldots, A_N \in \mathbb{Z}_q^{K \times M}$ and $B_1, \ldots, B_N \in \mathbb{Z}_q^{V \times M}$. For every index $i \in [N]$, we define the function

$$f_i : \mathbb{Z}_q^M \longrightarrow \mathbb{Z}_q^{K+V} \qquad \text{such that} \qquad f_i : \boldsymbol{x} \longmapsto (A_i \cdot \boldsymbol{x}, B_i \cdot \boldsymbol{x})$$

For the moment, assume that the parties output $\sum_{i \in [N]} f_i(\boldsymbol{x_{i,\ell}})$ for every $\ell \in [L]$, where L is a free parameter that controls the protocol's stretch.[6] We will adjust this provisional protocol several times as we explore its properties in order to achieve the properties we desire.

Each f_i can be viewed as a lossy trapdoor function [PW08] under the hardness of LWE. In particular, if M is sufficiently large compared to K, V and $\log q$, then there is a way to sample the matrices (A_i, B_i) along with a trapdoor T so that, for every $\boldsymbol{v_1} \in \mathbb{Z}_q^K$, $\boldsymbol{v_2} \in \mathbb{Z}_q^V$, the trapdoor T can be used to sample a low-norm vector $\boldsymbol{x} \in \mathbb{Z}_q^M$ such that $A_i \cdot \boldsymbol{x} = \boldsymbol{v_1}$ and $B_i \cdot \boldsymbol{x} = \boldsymbol{v_2}$. Furthermore, this sampling method yields A_i and B_i that are statistically close to uniform and $\boldsymbol{x}$ is indistinguishable from a discrete Gaussian sample [Ajt99, GPV08, AP09, MP12]. When the matrices are sampled in this way, f_i is in *injective* mode.

To use f_i in *lossy* mode, suppose that we generate B_i as $S \cdot A_i + E_i$, where $A_i \in \mathbb{Z}_q^{K \times M}$ and $S \in \mathbb{Z}_q^{V \times K}$ are sampled uniformly and $E_i \in \mathbb{Z}_q^{V \times M}$ comes from a discrete Gaussian distribution. Under LWE, the matrix B_i is indistinguishable from uniform, but every time we apply f_i on a small norm vector, we obtain a pair $(\boldsymbol{v_1}, \boldsymbol{v_2}) \in \mathbb{Z}_q^K \times \mathbb{Z}_q^V$ such that $\boldsymbol{v_2}$ is close in norm to $S \cdot \boldsymbol{v_1}$.

Limiting the Influence of the Adversary Using Lossy Trapdoor Functions. Let us consider the ideal world execution of the protocol we described above. The simulator starts by picking an arbitrary honest party P_ι. It generates the matrices $(A_i, B_i)_{i \in [N]}$ so that f_ι is in injective mode (let T_ι be the trapdoor), while all the other functions are in lossy mode. In particular, for every $i \neq \iota$, the simulator ensures that $B_i = S \cdot A_i + E_i$, where $S \xleftarrow{\$} \mathbb{Z}_q^{V \times K}$, $A_i \xleftarrow{\$} \mathbb{Z}_q^{K \times M}$ and E_i comes from a discrete Gaussian distribution over $\mathbb{Z}^{V \times M}$.

If all of the parties are honest, then the simulator has full control of the output of the protocol: if we desire the output to be the vectors $\boldsymbol{u_1}, \ldots, \boldsymbol{u_L} \in \mathbb{Z}_q^{K+V}$, the simulator must simply generate messages $\boldsymbol{x_{i,1}}, \ldots, \boldsymbol{x_{i,L}}$ following the protocol for every $i \neq \iota$. Then, using the trapdoor T_ι, it generates $\boldsymbol{x_{\iota,1}}, \ldots, \boldsymbol{x_{\iota,L}}$ such that $f_\iota(\boldsymbol{x_{\iota,\ell}}) = \boldsymbol{u_\ell} - \sum_{i \neq \iota} f_i(\boldsymbol{x_{i,\ell}})$ for every $\ell \in [L]$. That ensures that the output is exactly as desired.

If some parties are corrupted, however, the adversary has the ability to bias the output in an unpredictable but limited way by using its ability to rush. If the adversary waits until after the honest parties (including P_ι) are committed to their inputs before transmitting those of the corrupted parties, then it can contribute an additive term of $(\boldsymbol{v_{1,\ell}}, \boldsymbol{v_{2,\ell}}) := \sum_{j \in \mathcal{C}} f_j(\boldsymbol{x_{j,\ell}})$ to ℓ-th vector produced the protocol, and the simulator cannot compensate using the mechanism we have just described. Since all f_j for $j \in \mathcal{C}$ are in lossy mode with respect to the same matrix S, and they are all linear, it follows that $\boldsymbol{v_{2,\ell}} = S \cdot \boldsymbol{v_{1,\ell}} + \boldsymbol{e'_\ell}$ where $\boldsymbol{e'_\ell}$ is a vector of small norm. In other words, the entropy that the adversary can introduce is limited in this construction because the lossy trapdoor functions corresponding to the parties it corrupts are in lossy mode. We must add an additional mechanism to the protocol to correct for this adversarially-induced shift.

Adding a Correction Term. In order to allow the simulator to correct the offset induced by a rushing adversary, we augment the response of the seed oracle with two new matrices $C \in \mathbb{Z}_q^{K \times W}$ and $D \in \mathbb{Z}_q^{V \times W}$ that are indistinguishable from uniform, and a list of discrete Gaussian samples $(\boldsymbol{e_\ell})_{\ell \in [L]}$ over $\mathbb{Z}_q^W$.[7] The

[6]Note that in this provisional version, the output length is linear in L but the seed length is independent of L; the analysis of the *final* protocol's stretch will be more complex.

[7]For convenience, we say that the seed oracle outputs discrete Gaussian samples directly, but in order to meet the definition of a seed oracle it must actually output uniform coins from which such samples can be calculated. We highlight that discrete

output of our *new* protocol is the list of all vectors

$$(\boldsymbol{s_{1,\ell}}, \boldsymbol{s_{2,\ell}}) := \sum_{i\in[N]} f_i(\boldsymbol{x_{i,\ell}}) + (C\cdot \boldsymbol{e_\ell}, D\cdot \boldsymbol{e_\ell}) \quad \text{for every } \ell \in [L].$$

The simulator samples (C, D) such that they constitute a lossy-mode lossy trapdoor function. In other words, the matrix D is computed as $D \leftarrow S\cdot C + F$ where F is sampled according to a discrete Gaussian distribution over $\mathbb{Z}_q^{V\times W}$. Under the LWE assumption, D is indistinguishable from uniform, from the adversary's perspective. However, C is not sampled uniformly, as A_i for $i \neq \iota$ were. Instead, C is sampled together with a trapdoor T, much like A_ι and T_ι. The trapdoor T allows the simulator to sample preimages with respect to C that are indistinguishable from discrete Gaussian samples.

Suppose that we would like the output to be the random vectors $(\boldsymbol{u_{1,1}}, \boldsymbol{u_{2,1}}), \ldots, (\boldsymbol{u_{1,L}}, \boldsymbol{u_{2,L}}) \in \mathbb{Z}_q^K \times \mathbb{Z}_q^V$. The simulator generates the messages of all honest parties except for P_ι by following the protocol, then it samples $\boldsymbol{w_1}, \ldots, \boldsymbol{w_L} \xleftarrow{\$} \mathbb{Z}_q^K$ and, using the trapdoor T_ι, it generates the messages $\boldsymbol{x_{\iota,1}}, \ldots, \boldsymbol{x_{\iota,L}}$ of P_ι such that

$$f_\iota(\boldsymbol{x_{\iota,\ell}}) = (\boldsymbol{u_{1,\ell}} + \boldsymbol{w_\ell}, \boldsymbol{u_{2,\ell}} + S\cdot \boldsymbol{w_\ell}).$$

Now, for every $\ell \in [L]$, let

$$(\boldsymbol{v_{1,\ell}}, \boldsymbol{v_{2,\ell}}) := \sum_{i\in[N]} f_i(\boldsymbol{x_{i,\ell}})$$

where $\boldsymbol{v_{1,\ell}} \in \mathbb{Z}_q^K$ and $\boldsymbol{v_{2,\ell}} \in \mathbb{Z}_q^V$. Since all of the functions $(f_i)_{i\neq\iota}$ are in lossy mode with respect to the same matrix S, we have that for every $\ell \in [L]$,

$$\boldsymbol{v_{2,\ell}} = \boldsymbol{u_{2,\ell}} + S\cdot(\boldsymbol{v_{1,\ell}} - \boldsymbol{u_{1,\ell}}) + \boldsymbol{e''_\ell}$$

where $\boldsymbol{e''_\ell}$ is a small-norm error vector. Therefore, if the simulator uses the trapdoor T to generate the discrete Gaussian samples $(\boldsymbol{e_\ell})_{\ell\in[L]}$ such that $C\cdot \boldsymbol{e_\ell} = \boldsymbol{u_{1,\ell}} - \boldsymbol{v_{1,\ell}}$, then we have

$$(\boldsymbol{s_{1,\ell}}, \boldsymbol{s_{2,\ell}}) = (\boldsymbol{u_{1,\ell}}, \boldsymbol{u'_{2,\ell}}) \qquad \text{such that} \qquad \boldsymbol{u'_{2,\ell}} = \boldsymbol{u_{2,\ell}} + \boldsymbol{e''_\ell} + F\cdot \boldsymbol{e_\ell}$$

for every $\ell \in [L]$. The term $\boldsymbol{e''_\ell} + F\cdot \boldsymbol{e_\ell}$ has low norm, and thus $\boldsymbol{u'_{2,\ell}}$ is close in norm to $\boldsymbol{u_{2,\ell}}$. We will require one further adjustment to our protocol to make them equal. Notice first that if our output vector $\boldsymbol{u_{1,\ell}}$ is uniform, then $\boldsymbol{u_{1,\ell}} - \boldsymbol{v_{1,\ell}}$ is uniform and consequently the simulated vector $\boldsymbol{e_\ell}$ is indistinguishable from a discrete Gaussian sample. Furthermore, notice that $\boldsymbol{x_{\iota,\ell}}$ leaks nothing about $\boldsymbol{u_{1,\ell}}$, because $\boldsymbol{w_\ell}$ acts as a mask.

Final Adjustments. If two (distributions of) vectors have a low-norm difference, then their high-order bits are likely the same. Our protocol as currently written admits a simulator that can force the output to be close in norm to any desired value, so taking *only* the high-order bits to be the protocol's output will allow the very same simulation strategy to produce an exact match. Specifically, we will modify the protocol to pick a second modulus $p \ll q$ and round down each entry of $\boldsymbol{s_{2,\ell}}$ to the closest multiple of q/p.

In this next iteration of the protocol, the parties first compute

$$(\boldsymbol{s_{1,\ell}}, \boldsymbol{s_{2,\ell}}) \leftarrow \sum_{i\in[N]} f_i(\boldsymbol{x_{i,\ell}}) + (C\cdot \boldsymbol{e_\ell}, D\cdot \boldsymbol{e_\ell})$$

such that $\boldsymbol{s_{2,\ell}} \in \mathbb{Z}_q^V$ for every $\ell \in [L]$, and then compute the vector $\boldsymbol{s'_{2,\ell}} \in \mathbb{Z}_p^V$ that minimizes $\|\boldsymbol{s_{2,\ell}} - q/p\cdot \boldsymbol{s'_{2,\ell}}\|_\infty$ (we write $\boldsymbol{s'_{2,\ell}} \leftarrow \lceil \boldsymbol{s_{2,\ell}} \rfloor_p$). The output of the protocol is $(\boldsymbol{s_{1,1}}, \boldsymbol{s'_{2,1}}), \ldots, (\boldsymbol{s_{1,L}}, \boldsymbol{s'_{2,L}})$. If we would like the output to be the vectors $(\boldsymbol{u_{1,1}}, \boldsymbol{u_{2,1}}), \ldots, (\boldsymbol{u_{1,L}}, \boldsymbol{u_{2,L}}) \in \mathbb{Z}_q^K \times \mathbb{Z}_p^V$, then the simulator samples $\boldsymbol{u''_{2,\ell}} \in \mathbb{Z}_q^V$

Gaussians are explainable distributions [AWY20]. In other words, given a Gaussian sample $\boldsymbol{e}$, we are able to efficiently produce coins that produce the sample $\boldsymbol{e}$ when provided as randomness for the distribution, and the distribution of these coins is uniform, as required.

for $\ell \in [L]$ uniformly subject to $\lceil \boldsymbol{u''_{2,\ell}} + \boldsymbol{z} \rfloor_p = \boldsymbol{u_{2,\ell}}$ for every bounded-norm noise vector $\boldsymbol{z}$, uses the trapdoor T_ι to generate the messages $\boldsymbol{x_{\iota,1}}, \ldots, \boldsymbol{x_{\iota,L}}$ of P_ι such that

$$f_\iota(\boldsymbol{x_{\iota,\ell}}) = (\boldsymbol{u_{1,\ell}} + \boldsymbol{w_\ell}, \boldsymbol{u''_{2,\ell}} + S \cdot \boldsymbol{w_\ell})$$

and continues the simulation as before.

To guarantee negligible simulation error while using this technique, we must take q to be larger than p by a *superpolynomial* factor. Moreover, we must set the magnitude of the LWE noise to $\alpha \cdot q$ for a negligible function $\alpha(\lambda)$. In other words, we must assume the hardness of LWE with superpolynomial modulus-to-noise ratio. This completes the first version of our construction.

Adaptive Security. Our protocol is secure against adaptive corruption as a consequence of the non-interactive nature of the construction and the explainability of discrete Gaussian distributions [AWY20]. Given a discrete Gaussian sample $\boldsymbol{x}$, it is possible to efficiently produce random coins that, when provided as randomness for the distribution, produce the sample $\boldsymbol{x}$. We rely on this procedure every time that the adversary decides to corrupt a party after the end of the only round of interaction; recall that the messages of the parties are simply discrete Gaussian samples.

The Stretch of our First Construction. The number of random bits produced by the first version of our construction is $L \cdot (K \cdot \log q + V \cdot \log p)$, whereas the seed oracle provides

$$(K + V) \cdot (M \cdot N + W) \cdot \log q \tag{5.2}$$

bits for the matrices $(A_1, \ldots, A_N, B_1, \ldots B_N, C, D)$, and $L \cdot W \cdot \mathsf{poly}(\lambda) = L \cdot K \cdot \log q \cdot \mathsf{poly}(\lambda)$ bits for the discrete Gaussian samples $(\boldsymbol{e_\ell})_{\ell \in [L]}$. We observe that (5.2) is independent of L, so, if we pick L to be sufficiently large, then $L \cdot (K \cdot \log q + V \cdot \log p)$ becomes arbitrarily greater than (5.2). Similarly, the number of bits necessary for the discrete Gaussian samples is independent of V, so we can pick V sufficiently large to make $L \cdot (K \cdot \log q + V \cdot \log p)$ arbitrarily greater than $L \cdot W \cdot \mathsf{poly}(\lambda)$. This proves that our construction achieves an arbitrary polynomial stretch. However, notice that equation 5.2 depends linearly on N: this means that the number of bits supplied by the seed orace grows with the number of *parties*, which is undesirable. In order to fix this, we will need to make the slightly stronger assumption that LWE is hard with a *subexponential* (rather than superpolynomial) modulus-to-noise ratio. We present only this *second* construction in full in Section 5.5, but remark that the first construction that we have just sketched remains interesting, due to the slightly weaker assumption that it requires.

Improving the Complexity in the Number of Parties. Our first construction requires a number of random bits from the seed oracle that scales linearly in the number of parties. Is this necessary? We prove that it is not by devising a mechanism to deal matrices $(A_i, B_i)_{i \in [N]}$ that satisfy the properties necessary for the security of the protocol using just $O(\log N) \cdot \mathsf{poly}(\lambda)$ uniformly random bits from the seed oracle. We leverage the *fully homomorphic encryption* (FHE) scheme Gentry et al. [GSW13], hereafter called *GSW*.

In the GSW FHE scheme, the public key is a uniform-looking matrix $U \in \mathbb{Z}_q^{\Delta \times M}$ for some $\Delta \in \mathbb{N}$. An encryption of a bit b under U consists of

$$U \cdot R + b \cdot G$$

where $R \xleftarrow{\$} \mathbb{Z}_2^{M \times \Delta \cdot \log q}$ is a random binary matrix and $G \in \mathbb{Z}_q^{\Delta \times \Delta \log q}$ is the gadget matrix, i.e. a matrix for which there exists an efficient deterministic algorithm G^{-1} that produces a binary matrix $X' \in \mathbb{Z}_2^{\Delta \cdot \log q \times M'}$ such that $G \cdot X' = Y'$ for any input $Y' \in \mathbb{Z}_q^{\Delta \times M'}$ and some $M' \in \mathbb{N}$. Under the hardness of LWE, all ciphertexts look like random matrices over $\mathbb{Z}_q^{\Delta \times \Delta \cdot \log q}$. Furthermore, due to the homomorphic properties of this scheme, there exists an efficient algorithm Eval that takes as input the encryptions of bits $b_1, \ldots, b_t$ under a public key U and the description of a function $f : \{0,1\}^t \to \{0,1\}$, and produces a ciphertext $Z_f = U \cdot R_f + f(b_1, \ldots, b_t) \cdot G$ where R_f is a small norm matrix.

We modify the seed oracle so that it provides $\log N + 1$ random matrices

$$X_1, \ldots, X_{\log N} \in \mathbb{Z}_q^{(K+V) \times (K+V) \log q} \qquad \text{and} \qquad Y \in \mathbb{Z}_q^{(K+V) \times M}.$$

The parties regard $X_1, \ldots, X_{\log N}$ as GSW ciphertexts with $\Delta = K + V$,[8] and each party P_j obtains its matrices (A_j, B_j) by computing

$$(A_j^\intercal \parallel B_j^\intercal)^\intercal = Z_j \leftarrow \mathsf{Eval}(\delta_j, X_1, \ldots, X_{\log N}) \cdot G^{-1}(Y).$$

where δ_j denotes the Kronecker delta function centered on j. We assume that δ_j takes as input $\log N$ bits and regards them as the description of an integer in $[N]$. Note that A_j comprises the first K rows of Z_j and B_j the last V rows.

In the ideal world, the simulator determines the GSW public key U by sampling the first K rows uniformly over $\mathbb{Z}_q^{K\times M}$ (we denote these rows by U_1) and setting the last V rows to be $U_2 \leftarrow S \cdot U_1 + E$, where E is a discrete Gaussian sample over $\mathbb{Z}^{V\times M}$. Under LWE, U is indistinguishable from a uniformly sampled public key. Next, the simulator generates $X_1, \ldots, X_{\log N}$ by encrypting the bits of ι under U; recall that ι is the index of the honest party chosen by the simulator. It also samples the matrix Y together with a trapdoor T' that allows the simulator to compute preimages with respect to Y that are indistinguishable from discrete Gaussian samples.

The correctness of FHE evaluation implies that for any $j \neq \iota$,

$$B_j = U_2 \cdot R_{\delta_j} \cdot G^{-1}(Y) = S \cdot (U_1 \cdot R_{\delta_j} \cdot G^{-1}(Y)) + E \cdot R_{\delta_j} \cdot G^{-1}(Y) \sim S \cdot A_j.$$

In other words, the trapdoor function f_j is in lossy mode with respect to S. On the other hand, if we denote the first K rows of G and Y by G_1 and Y_1 respectively, and the last V rows by G_2 and Y_2, we have

$$\begin{aligned}A_\iota - Y_1 &= U_1 \cdot R_{\delta_\iota} \cdot G^{-1}(Y) + G_1 \cdot G^{-1}(Y) - Y_1 = U_1 \cdot R_{\delta_\iota} \cdot G^{-1}(Y),\\ B_\iota - Y_2 &= U_2 \cdot R_{\delta_\iota} \cdot G^{-1}(Y) + G_2 \cdot G^{-1}(Y) - Y_2\\ &= S \cdot (U_1 \cdot R_{\delta_\iota} \cdot G^{-1}(Y)) + E \cdot R_{\delta_\iota} \cdot G^{-1}(Y) \sim S \cdot (A_\iota - Y_1).\end{aligned}$$

In other words, f_ι is the sum of two trapdoor functions, one in injective mode (described by the matrix Y) and one in lossy mode with respect to S, and the simulator's trapdoor for f_ι is now T' rather than T_ι. The lossy-mode component of f_ι clearly introduces some error terms into the output, but we can correct them together with the bias introduced by the adversary via $(\boldsymbol{e_\ell})_{\ell\in[L]}$.

The Stretch of our Second Construction. Like our first construction, this construction produces $L \cdot (K \cdot \log q + V \cdot \log p)$ random bits, but the number of bits provided by the seed oracle has been reduced to

$$\log N \cdot (K + V)^2 \cdot \log q + (K + V) \cdot M \cdot \log q + (K + V) \cdot W \cdot \log q \tag{5.3}$$

for the matrices $(X_1, \ldots, X_{\log N}, Y, C, D)$, and $L \cdot W \cdot \mathsf{poly}(\lambda) = L \cdot K \cdot \log q \cdot \mathsf{poly}(\lambda)$ for the discrete Gaussian samples $(\boldsymbol{e_\ell})_{\ell\in[L]}$. As before, (5.3) is independent of L and the number of bits necessary for the discrete Gaussian samples is independent of V; we can again pick L and V to be sufficiently large so that $L \cdot (K \cdot \log q + V \cdot \log p)$ becomes arbitrarily greater than (5.3) and $L \cdot (K \cdot \log q + V \cdot \log p)$ becomes arbitrarily greater than $L \cdot W \cdot \mathsf{poly}(\lambda)$. This proves that our construction achieves an arbitrary polynomial stretch.

Final Remarks. Due to the noise growth induced by homomorphically evaluating the Kronecker delta function, it no longer suffices to assume the hardness of LWE with a superpolynomial modulus-to-noise ratio. The Kronecker delta function over the domain $[N]$ has a multiplicative depth of $O(\log N)$. If we assume that $N \in \mathsf{poly}(\lambda)$ then the depth is $O(\log \lambda)$ and in the simulation, the size of the noise in the GSW ciphertexts $Z_1, \ldots, Z_N$ increases by a factor $O(\log \lambda)$ relative to our first construction. Since the magnitide of the noise is now quasipolynomial, we must select a modulus-to-noise ratio that is greater than quasipolynomial in order to guarantee the correctness of the simulation with overwhelming probability. We will formalize our second construction in Section 5.5 and show in that section that the hardness of LWE with a *subexponential* modulus-to-noise ratio is sufficient.

Theorem 5.2.5 (Informal Version of Theorem 5.5.3). If the subexponential LWE assumption holds, then there exists a fully-secure N-party protocol in the $\mathcal{F}^n_{\mathsf{Coin}}$-hybrid model that UC-realizes $\mathcal{F}^m_{\mathsf{Coin}}$ in the presence of a malicious PPT adversary adaptively corrupting $N - 1$ parties.

[8]Note that in the real world, these "ciphertexts" are uniformly random and there is *no* public key corresponding to them.

5.2.4 A Lower Bound for Statistical Coin Tossing Extension

We now focus our attention on information-theoretic CTE. Hofheinz et al. [HMU06] proved that perfectly secure CTE is impossible in any model and that statistically secure CTE is impossible in the UC model, and constructed one-round statistically secure CTE with $O(\log \lambda)$ additive stretch and black-box standalone simulation. We ask whether one-round statistically secure CTE with $\omega(\log \lambda)$ additive stretch is possible, and prove that if we insist upon black-box simulation, then it is not.

Theorem 5.2.6. Every r-round CTE protocol with one call to the seed oracle and black-box standalone statistical simulation security against a rushing semi-malicious[9] adversary who corrupts a majority of parties must have additive stretch in $O(r \cdot \log \lambda)$.

Tools and Notation. Consider any r-round statistically secure CTE protocol with black-box simulation and a single call to the seed oracle. Due to Theorem 5.2.2, we can assume without loss of generality that the parties stop interacting after the call to the seed oracle. Let $\boldsymbol{s}$ denote the output of the protocol, let $\boldsymbol{u}$ be the random coins provided by the seed oracle. For every $i \in [r]$, let U_H^i and U_C^i denote all the messages sent by the honest parties and the corrupted parties respectively, up to and including the i^{th} round. We consider a very specific and rather unusual adversary (at least, in the context of lower bounds): the adversary that simply follows the protocol as if it was honest, but at the same time uses rushing; i.e., it reveals the messages of the corrupted players only after seeing the messages of the honest parties. Considering this extremely weak type of adversary makes our lower bound stronger. Our argument focuses on the *information diagram* [Yeu91] of the protocol, and it is reminiscent of a technique used by Abram et al. [AOS23].[10] We will make use of a handful of basic tools and lemmas from information theory, including Shannon entropy (denoted H), and mutual information (denoted I); these are reviewed in Section 5.3.2.

Output Entropy and Round Count. We start by observing that $\boldsymbol{s}$ is uniquely determined by U_H^r, U_C^r, and $\boldsymbol{u}$. Translating this into entropy, we have $\mathsf{H}(\boldsymbol{s}|U_H^r, U_C^r, \boldsymbol{u}) = 0$ in the real world. In the ideal-world execution, this quantity could be negligibly-far from 0 due to simulation error; we write $\mathsf{H}(\boldsymbol{s}|U_H^r, U_C^r, \boldsymbol{u}) \sim 0$ to indicate this. We can bound m as follows:

$$\begin{aligned} m \sim \mathsf{H}(\boldsymbol{s}) &= \mathtt{I}(\boldsymbol{s}; (\boldsymbol{u}, U_C^r, U_H^r)) + \mathsf{H}(\boldsymbol{s}|U_H^r, U_C^r, \boldsymbol{u}) \\ &\sim \mathtt{I}(\boldsymbol{s}; \boldsymbol{u}|U_C^r, U_H^r) + \mathtt{I}(\boldsymbol{s}; (U_C^r, U_H^r)) \\ &\leq n + \sum_{i=1}^{r} (\mathtt{I}(U_H^i; \boldsymbol{s}|U_H^{i-1}, U_C^{i-1}) + \mathtt{I}(U_C^i; \boldsymbol{s}|U_H^i, U_C^{i-1})). \end{aligned} \tag{5.4}$$

In the second and last inequality, we used the chain rule of mutual information; in the last inequality we also used the fact that $\mathtt{I}(\boldsymbol{s}; \boldsymbol{u}|U_C^r, U_H^r) \leq \mathsf{H}(\boldsymbol{u}) = n$.

Rewinding-Induced Correlation in the Ideal World. Consider a straight-line ideal-world experiment involving a statistically secure CTE protocol. In round i, the adversary (who essentially behaves honestly), receives the honest parties' messages U_H^i, and then produces the message of corrupt parties U_C^i for round i according to its view, which contains U_H^{i-1} and U_C^{i-1}. This necessarily implies that U_C^i is independent of U_H^i and $\boldsymbol{s}$, conditioned on U_H^{i-1} and U_C^{i-1}. However, the simulator might not be straight-line. We allow it the power to *rewind* the adversary, which means that it can accept or reject U_C^i based on its knowledge of $\boldsymbol{s}$ and U_H^i. This introduces some correlation between the variables.

Each time the experiment is rewound, the adversary samples a fresh U_C^i that is independent of U_H^i, $\boldsymbol{s}$, and all of the messages it produced in the previous rewindings, conditioned on U_H^{i-1} and U_C^{i-1}. Because of this independence and the fact that the number of rewindings is upper bounded by the running time Q of the

[9]A rushing semi-malicious adversary is required to follow the protocol as in the semi-honest case, however, it is allowed to maliciously choose the randomness of the corrupted players.

[10]Our setting is simpler since we consider only statistical security, but on the other hand Abram et al. focused on the one-round setting, whereas our argument applies to protocols with multiple rounds.

simulator, the simulator can induce at most $\log Q$ bits of correlation between U_C^i and $(U_H^i, \boldsymbol{s})$, conditioned on U_H^{i-1} and U_C^{i-1}. That is, when $Q = \text{poly}(\lambda)$, $\mathtt{I}(U_C^i; (U_H^i, \boldsymbol{s}) | U_H^{i-1}, U_C^{i-1}) \leq O(\log \lambda)$. By the chain rule of mutual information, this implies

$$\mathtt{I}(U_C^i; \boldsymbol{s} | U_H^i, U_C^{i-1}) \leq \mathtt{I}(U_C^i; (U_H^i, \boldsymbol{s}) | U_H^{i-1}, U_C^{i-1}) \leq O(\log \lambda).$$

Since we are considering an honestly-behaving adversary in the dishonest-majority setting, our entropy diagram must be symmetric: we can switch the roles of honest and corrupted parties. This leads us to conclude that $\mathtt{I}(U_H^i; \boldsymbol{s} | U_H^{i-1}, U_C^{i-1}) \leq \mathtt{I}(U_H^i; (U_C^i, \boldsymbol{s}) | U_H^{i-1}, U_C^{i-1}) \leq O(\log \lambda)$.

Putting It All Together. In the ideal world, we can now use the bounds $\mathtt{I}(U_C^i; \boldsymbol{s} | U_H^i, U_C^{i-1}) \leq O(\log \lambda)$ and $\mathtt{I}(U_H^i; \boldsymbol{s} | U_H^{i-1}, U_C^{i-1}) \leq O(\log \lambda)$ for all $i \in [r]$ in equation 5.4, to get the required bound $m \leq n + r \cdot O(\log(\lambda))$. This result is formalized in Section 5.8.

5.2.5 One-Round Unbiased Sampling from any Distribution

Our results so far have shown that in the dishonest majority setting, it is possible to simulatably sample m-bit random strings with guaranteed output delivery and no adversarial bias, assuming the existence of a seed oracle that produces $n \ll m$ unbiased random coins. Moreover, this can be done with a single round of interaction followed by one call to the seed oracle. We ask a final question: is there anything special about the uniform distribution, or can we actually sample values from any distribution with the same security guarantees, in the same setting? Specifically, given any efficient distribution $\mathcal{D}$, is it possible for N parties to agree on a random sample from $\mathcal{D}$ with the help of a uniform seed oracle, while leaking no additional information and allowing no bias, and denying the adversary the power to abort, even if it corrupts all of the parties but one? Can we achieve this using a single round of interaction? We prove that under strong cryptographic assumptions, this is indeed possible!

Theorem 5.2.7 (Informal Version of Theorem 5.9.5). Let $\mathcal{D}$ be an efficient distribution. Assuming the existence of indistinguishability obfuscation, injective length-doubling PRGs, and indistinguishability-preserving distributed samplers [AWZ23], there exists a one-round N-party protocol in the $\mathcal{F}_{\mathsf{Coin}}^n$-hybrid CRS model that UC-realizes the functionality $\mathcal{F}_{\mathcal{D}}$ that provides all parties with a sample from $\mathcal{D}$ in the presence of a malicious PPT adversary statically corrupting up to $N-1$ parties.

Indistinguishability-Preserving Distributed Samplers. Distributed samplers [ASY22, AOS23, AWZ23] are one-round protocols that securely sample a common output from some distribution $\mathcal{D}$. Though several security definitions have been proposed for this primitive, our final protocol relies specifically upon *indistinguishability-preserving* distributed samplers [AWZ23], which are known to exist in the CRS model under a combination of subexponentially secure indistinguishability obfuscation, multi-key FHE, extremely lossy functions [Zha16], and other, weaker tools. Unlike other flavors of distributed sampler, indistinguishability-preserving ones *do not* require idealized models such as the random oracle. Suppose that Π is an r-round protocol relying on a CRS sampled from $\mathcal{D}$, and that Π realizes some functionality $\mathcal{F}$. If Π satisfies some particular properties, indistinguishability-preserving distributed samplers permit us to compile Π into an $r+1$-round protocol realizing the same functionality $\mathcal{F}$. This new protocol will rely on a simpler CRS that is reusable, unstructured (i.e. uniformly distributed), and of length independent of $\mathcal{D}$ and Π. In our setting, we can generate this simpler CRS for the compiled protocol by a making once-and-for-all call to the seed oracle.

The Protocol We Will Compile. The protocol Π that we will compile has *zero* rounds of communication. The CRS consists of an obfuscated program that hides a puncturable PRF key. When provided with a λ-bit string $\boldsymbol{s}$ as input, this program evaluates the puncturable PRF on $\boldsymbol{s}$ and uses the result to compute a sample from $\mathcal{D}$, which is the program's output. In our zero-round protocol, the parties generate $\boldsymbol{s}$ by calling the seed oracle, feed $\boldsymbol{s}$ into the obfuscated program that is encoded in the CRS, and output the result.

It is easy to see that forgoing protocol realizes the functionality $\mathcal{F}_{\mathcal{D}}$. The simulator must simply sample a random $\hat{\boldsymbol{s}}$ and modify the obfuscated program so that it outputs the sample chosen by the functionality on input $\hat{\boldsymbol{s}}$. Then, when the parties call the seed oracle, the simulator provides $\hat{\boldsymbol{s}}$. Since our protocol is zero rounds before compilation, it will have one round after compilation, with calls to the seed oracle at the beginning and end.

In Section 5.9, we formalize the above intuition and prove that the zero-round protocol satisfies the conditions required by indistinguishability-preserving distributed samplers. This is why we construct such an unusual zero-round protocol, rather than the trivial protocol that simply provides a sample from $\mathcal{D}$ as a CRS for the parties to output: the latter trivial protocol clearly implements $\mathcal{F}_{\mathcal{D}}$, but it cannot be compiled by an indistinguishability-preserving distributed sampler, because if $\mathcal{D}$ outputs random strings of bits, then the result would be a fully-secure coin tossing protocol that contradicts Cleve's impossibility [Cle86].

5.3 Preliminaries

5.3.1 Universal Composability

We give a high-level overview of the UC model and refer the reader to Canetti [Can01] for more detail.

The *real-world* UC experiment involves N parties that execute a protocol Π, an adversary $\mathcal{A}$ that can corrupt a subset of the parties (either statically or adaptively), and an environment $\mathcal{Z}$ that is initialized with an advice-string z. All entities are initialized with the security parameter λ and with a random tape. The environment activates the parties involved in Π, chooses their inputs and receives their outputs, and communicates with the adversary $\mathcal{A}$. The experiment completes when $\mathcal{Z}$ stops activating parties and outputs a decision bit. Let $\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)$ denote the random variable representing the output of this experiment.

The *ideal-world* UC experiment involves N *dummy* parties, an ideal functionality $\mathcal{F}$, a simulator $\mathcal{S}$ (otherwise referred to as the *ideal adversary*), and an environment $\mathcal{Z}$. The dummy parties forward any message received from $\mathcal{Z}$ to $\mathcal{F}$ and vice versa. The simulator can corrupt a subset of the dummy parties and interact with $\mathcal{F}$ on their behalf, and it can communicate directly with $\mathcal{F}$ according to its specification. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z)$ denote the random variable representing the output of this experiment.

A protocol Π *UC-realizes* a functionality $\mathcal{F}$ if for every PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that for every PPT environment $\mathcal{Z}$ the following distributions are computationally indistinguishable:

$$\{\text{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N}^+,z\in\{0,1\}^{\mathsf{poly}(\lambda)}} \quad \text{and} \quad \{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N}^+,z\in\{0,1\}^{\mathsf{poly}(\lambda)}}$$

Full security cannot be expressed in the inherently asynchronous UC framework, because termination cannot be guaranteed. Katz et al. [KMTZ13] defined a synchronous variant of the UC model that captures guaranteed termination; this is the variation that we consider in this work, though we leave it implicit. Other composition frameworks that support synchrony also exist [LZM20, BDD$^+$21].

5.3.2 Information Theory

Useful information theoretic quantities. We recall some basic information-theoretic quantities (see [CT06] for a primer).

The *Shannon entropy* of a finite random variable X, denoted by $\mathsf{H}(X)$, is defined as $\sum_x \Pr[X = x] \log\left(\frac{1}{\Pr[X=x]}\right)$. The binary entropy function for parameter $\rho \in [0,1]$ is defined as $h(\rho) = -\rho\log(\rho) - (1-\rho)\log(1-\rho)$. For a pair of jointly distributed random variables (X,Y), the entropy of X conditioned on Y, denoted by $\mathsf{H}(X|Y)$, is defined as $\sum_y \Pr[Y = y]\mathsf{H}(X|Y = y)$, where $\mathsf{H}(X|Y = y)$ is the entropy of X conditioned on the event $Y = y$.

Conditional Shannon entropy preserves under statistical indistinguishability. That is, given pairs of statistically indistinguishable random variables (X_1, Y_1) and (X_2, Y_2), it holds that

$$\mathsf{H}(X_1|Y_1) = \mathsf{H}(X_2|Y_2) + \mathsf{negl}(\lambda).$$

The *mutual information* between jointly distributed X and Y, denoted by $\mathtt{I}(X;Y)$ is defined as $\mathsf{H}(X,Y) - \mathsf{H}(X|Y) - \mathsf{H}(Y|X)$, and is always non-negative. Mutual information of X and Y conditioned on a jointly distributed Z, denoted by $\mathtt{I}(X;Y|Z)$, is defined as $\mathsf{H}(X|Z) - \mathsf{H}(X|Y,Z)$. The three variate mutual information of (X,Y,Z), denoted by $\mathtt{I}(X;Y;Z)$, is defined as $\mathtt{I}(X;Y) - \mathtt{I}(X;Y|Z)$, and can be negative.

By the chain rule of mutual information, $\mathtt{I}(X;Y,Z) = \mathtt{I}(X;Y) + \mathtt{I}(X;Z|Y)$. We will use this equality in our proofs.

Definition 5.3.1 (Explainable Distribution). Let $\mathcal{D}(1^\lambda)$ and $\mathcal{D}'(\lambda)$ be distributions We say that $\mathcal{D}$ is $\mathcal{D}'$-explainable if there exists a pair of PPT algorithms $(\mathsf{Convert}, \mathsf{Explain})$ satisfying the following properties:

- $\mathsf{Convert}$ is deterministic and $\mathsf{Convert}(1^\lambda, y)$ for $y \xleftarrow{\$} \mathcal{D}'(1^\lambda)$ is statistically indistinguishable from $\mathcal{D}(1^\lambda)$.

- No PPT adversary can distinguish between

$$\left\{ x, y \,\middle|\, \begin{array}{l} y \xleftarrow{\$} \mathcal{D}'(1^\lambda) \\ x \leftarrow \mathsf{Convert}(1^\lambda, y) \end{array} \right\} \qquad \left\{ x, y \,\middle|\, \begin{array}{l} x \xleftarrow{\$} \mathcal{D}(1^\lambda) \\ y \xleftarrow{\$} \mathsf{Explain}(1^\lambda, x) \end{array} \right\}$$

We say that $\mathcal{D}$ is explainable (without specifying the distribution $\mathcal{D}'$) if it is μ_R-explainable for some polynomial function $R(\lambda)$ (μ_R denotes the uniform distribution over $\{0,1\}^{R(\lambda)}$). We call R the randomness complexity of the distribution. We say that $\mathcal{D}$ is (L,R)-convertible if $\mathcal{D}$ is μ_R-explainable and μ_L is $\mathcal{D}$-explainable.

5.3.3 Lattices

Given a lattice Λ and a positive integer s, we use $\mathcal{D}_{\Lambda,s}$ to denote the discrete Gaussian distribution over Λ with parameter s. Given positive integers p, q where $q > p$ and a vector $\boldsymbol{v} \in \mathbb{Z}_q^n$ for some $n \in \mathbb{N}$, we use $\lceil \boldsymbol{v} \rfloor_p$ to denote the n-dimensional $\mathbb{Z}_p$ vector $\boldsymbol{u}$, where, for every $i \in [n]$, the i-th entry consists of the value u_i that minimises the quantity $|v_i - u_i \cdot \lfloor q/p \rfloor|$. For any positive integers q, $K \leq M$, matrix $A \in \mathbb{Z}_q^{M \times K}$ and vector $\boldsymbol{u} \in \mathbb{Z}_q^K$, we define the following lattices

$$\Lambda(A) := \{A \cdot \boldsymbol{v} + q \cdot \boldsymbol{w} \mid \boldsymbol{v} \in \mathbb{Z}^K, \boldsymbol{w} \in \mathbb{Z}^M\}$$
$$\Lambda_{\boldsymbol{u}}^{\perp}(A) := \{\boldsymbol{v} \in \mathbb{Z}^M \mid A^{\intercal} \cdot \boldsymbol{v} \equiv \boldsymbol{u} \bmod q\}$$

We use $\Lambda^\perp(A)$ to denote $\Lambda^\perp_{\boldsymbol{0}}(A)$.

Lemma 5.3.2. For every $K, s \in \mathbb{N}$, the distributions $\mathcal{D}_{\mathbb{Z}^K,s}$ and $\mathcal{D}^K_{\mathbb{Z},s}$ are the same (we use $\mathcal{D}^K_{\mathbb{Z},s}$ to denote a K-dimensional vector of random variables where all the entries are independently distributed according to $\mathcal{D}_{\mathbb{Z},s}$).

Definition 5.3.3 (The Learning with Errors (LWE) Assumption [Reg05, Reg09]). Let $q(\lambda)$ be a positive integer and let $\alpha : \mathbb{N} \to [0,1]$ be a function of the security parameter. Define $s(\lambda) := \alpha(\lambda) \cdot q(\lambda)$. Let K, M be polynomial functions of the security parameter. We say that the (q, α, K, M)-LWE assumption holds if the following distributions are computationally indistinguishable

$$\left\{ A, \boldsymbol{v} \,\middle|\, \begin{array}{l} A \xleftarrow{\$} \mathbb{Z}_q^{M \times K} \\ \boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^K \\ \boldsymbol{e} \xleftarrow{\$} \mathcal{D}^M_{\mathbb{Z},s} \\ \boldsymbol{v} \leftarrow (A \cdot s + e) \bmod q \end{array} \right\} \qquad \left\{ A, \boldsymbol{v} \,\middle|\, \begin{array}{l} A \xleftarrow{\$} \mathbb{Z}_q^{M \times K} \\ \boldsymbol{v} \xleftarrow{\$} \mathbb{Z}_q^M \end{array} \right\}$$

We say that the assumption holds with subexponential modulo-to-noise ratio if α is $2^{-\Omega(\lambda^\epsilon)}$ for some constant $\epsilon > 0$.

Lemma 5.3.4 (Leftover Hash Lemma). Let K, M, q be positive integer and let ξ be a distribution over $\mathbb{Z}^M$ such that $\mathsf{H}_\infty(\xi) \geq K \cdot \log q + \lambda$. Then the following distribution is $2^{-\lambda}$-far from the uniform distribution.

$$\left\{ A, \boldsymbol{v} \middle| \begin{array}{l} A \xleftarrow{\$} \mathbb{Z}_q^{K \times M} \\ \boldsymbol{e} \xleftarrow{\$} \xi \\ \boldsymbol{v} \leftarrow A \cdot \boldsymbol{e} \end{array} \right\}$$

Theorem 5.3.5 ([Ajt99, GPV08, AP09, MP12]). There exists an pair of PPT algorithms $(\mathsf{TrapGen}, \mathsf{PreSample})$ with the following syntax

- $\mathsf{TrapGen}$ takes as input values 1^K, 1^M, q and outputs a pair (A, T) where $A \in \mathbb{Z}_q^{K \times M}$ and T is a trapdoor.
- $\mathsf{PreSample}$ takes as input a trapdoor T, a vector $\boldsymbol{u} \in \mathbb{Z}_q^K$ and a positive integer s. The output is a vector $\boldsymbol{e} \in \mathbb{Z}_q^M$.

and satisfying the following properties:

- There exists a constant $\beta > 1$ such that, $M \geq \beta K \cdot \log q$, the distribution of the matrix A output by $(A, T) \xleftarrow{\$} \mathsf{TrapGen}(1^K, 1^M, q)$ is $\mathsf{negl}(K)$-far from uniform.
- If $M \geq \beta K \cdot \log q$ and $s = \sqrt{K \cdot \log q} \cdot \omega(\sqrt{\log K})$, for every $\boldsymbol{u} \in \mathbb{Z}_q^K$, the distributions $\mathsf{PreSample}(T, \boldsymbol{u}, s)$ and $\mathcal{D}_{\Lambda_{\boldsymbol{u}}^\perp(A), s}$, for $(A, T) \xleftarrow{\$} \mathsf{TrapGen}(1^K, 1^M, q)$, are $\mathsf{negl}(K)$-far.

Lemma 5.3.6 ([Ban93, PR06]). Suppose that $s = \omega(\sqrt{\log K})$. Then,

$$\Pr\left[\|\boldsymbol{e}\| \geq s \cdot \sqrt{K} \middle| \boldsymbol{e} \xleftarrow{\$} \mathcal{D}_{\mathbb{Z}^K, s} \right] \leq \mathsf{negl}(K).$$

Moreover, $\mathsf{H}_\infty(\mathcal{D}_{\mathbb{Z}^K, s}) \geq K$.

Theorem 5.3.7 ([GPV08, AWY20]). Suppose that $s = \omega(\sqrt{\log K})$. Then, the distribution $\mathcal{D}_{\mathbb{Z}^K, s}$ is explainable. Furthermore, the randomness complexity is $O(K \cdot \lambda^2 + K \cdot \lambda \cdot \log s)$.

5.3.4 Preliminaries on Indistinguishability-Preserving Distributed Samplers

In this subsection, we recall the quite complex definition of indistinguishability-preserving distributed sampler [AWZ23]. For the whole subsection, let $\mathcal{D}$ denote an efficient distribution.

We start by recalling the definition of trapdoored distribution for $\mathcal{D}$. This is essentially a distribution $\mathcal{D}'$ taking as input auxiliary information aux and outputting samples R along with trapdoors T. For any choice of the auxiliary information, the trapdoored sample R looks like a sample from $\mathcal{D}$.

Definition 5.3.8 (Trapdoored Distribution [AWZ23]). A trapdoored distribution is a PPT algorithm $\mathcal{D}'$ that takes as input the security parameter 1^λ and auxiliary information $\mathsf{aux} \in \{0,1\}^{\ell(\lambda)}$, where $\ell(\lambda)$ is a polynomial, and outputs a sample R and a trapdoor T.

Let $\mathcal{D}(1^\lambda)$ be an efficient distribution. We say that $\mathcal{D}'$ is a trapdoored distribution for $\mathcal{D}$ if, for every $\mathsf{aux} \in \{0,1\}^{\ell(\lambda)}$, the following distributions are computationally indistinguishable

$$\left\{ R \middle| R \xleftarrow{\$} \mathcal{D}(1^\lambda) \right\} \qquad \left\{ R \middle| (R, T) \xleftarrow{\$} \mathcal{D}'(1^\lambda, \mathsf{aux}) \right\}$$

Next, we recall the definition of trapdoorable distributed sampler. A distributed sampler [ASY22] consists of a one-round protocol for the secure generation of samples from a distribution $\mathcal{D}$: all parties broadcast a distributed sampler message. Given the N messages, everybody can derive the output. The construction sometimes relies on a CRS. A trapdoorable distributed sampler is a distributed sampler that, by knowing a trapdoor hidden in the CRS, allows switching the distribution of the outputs from $\mathcal{D}$ to the trapdoored distribution $\mathcal{D}'(1^\lambda, \mathsf{aux})$ for any chosen aux. The trapdoor in the CRS allows also to retrieve the trapdoors relative to the outputs of the sampler (we recall that the latter are coming from $\mathcal{D}'$).

Definition 5.3.9 (Trapdoorable distributed sampler [AWZ23]). An N-party trapdoorable distributed sampler consists of a tuple of PPT algorithms $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{SimSetup}, \mathsf{SimGen}, \mathsf{Trap})$ with the following syntax:

- Setup takes as input the security parameter and outputs a CRS σ.
- Gen takes as input the security parameter, a session identity sid, the index of a party $i \in [N]$ and a CRS σ. The output is a distributed sampler message U_i.
- Sample is deterministic and takes as input N distributed sampler messages $U_1, \ldots, U_N$ a session identity sid and a CRS σ. The output is a sample R.
- $\mathsf{SimSetup}$ takes as input the security parameter and outputs a CRS σ along with the setup trapdoor information ζ.
- SimGen takes as input the security parameter, a session identity sid, an index $i \in [N]$, setup trapdoor information ζ and auxiliary information aux. The output is a distributed sampler message U_i along with message trapdoor information ξ.
- Trap is deterministic and takes as input message trapdoor information ξ and distributed sampler messages $U_1, \ldots, U_N$. The output is a pair (R, T).

We now recall all the notions necessary to define the security of indistinguishability preserving distributed samplers. Games with oracle distributions are described by an efficient challenger interacting with an adversary. The challenger impersonates a set of honest parties, while the adversary impersonates the corrupted players. Throughout the game, both the challenger and the adversary can send special messages (Sample, i) on behalf of any party they control P_i. When the challenger sent this special message on behalf of all honest player, we generate a sample from the oracle distribution and we provide it to the adversary. When also the adversary sent the special message on behalf of all corrupted players, we provide the sample also to the challenger.

The special message (Sample, i) is meant to represent the delivery of the universal sampler message of party P_i. The oracle distribution is instead meant to represent the computation of the distributed sampler output.

Definition 5.3.10 (Game with Oracle Distribution [AWZ23]). An N-party game with oracle distribution is a pair $\mathcal{G} = (\mathsf{Ch}, \mathcal{D})$, where $\mathcal{D}$ is an efficient distribution and Ch is an efficient challenger: a PPT, round-based, interactive Turing machine that, for every $i \in [N]$, sends the message (Sample, i) at most once during its execution.

Games with trapdoored oracle distribution are games with oracle distribution where the latter is trapdoored. When we generate the sample, we first ask the challenger to provide the auxiliary information aux. The trapdoor hidden in the sample is provided only to the challenger.

Definition 5.3.11 (Game with Trapdoored Oracle Distribution [AWZ23]). An N-party game with oracle distribution is a pair $\mathcal{G} = (\mathsf{Ch}, \mathcal{D}')$, where $\mathcal{D}'$ is a trapdoored distribution and Ch is an efficient challenger: a PPT, round-based, interactive Turing machine that, for every $i \in [N]$, sends the message (Sample, i) at most once during its execution.

Below we recall the notion of trapdoor security for games with trapdoor oracle distribution. The latter states that, even if unexpectedly, we do not provide the trapdoor to the challenger, the adversary does not notice it.

Definition 5.3.12 (Trapdoor Security [AWZ23]). We say that a game with a trapdoored oracle distribution $\mathcal{G} = (\mathsf{Ch}, \mathcal{D}')$ satisfies trapdoor security if no PPT adversary can win Game 5.3.13 with non-negligible advantage in λ.

Game 5.3.13. Trapdoor Security Game

1. $b \xleftarrow{\$} \{0,1\}$
2. Activate the adversary $\mathcal{A}$ with 1^λ
3. Receive ϕ and $H \subseteq [N]$ from $\mathcal{A}$
4. Activate Ch on input 1^λ, ϕ and H
5. Let Ch and $\mathcal{A}$ interact
6. When Ch has sent the message (Sample, i) for every $i \in H$, receive aux from Ch, sample $(R, T) \xleftarrow{\$} \mathcal{D}'(1^\lambda, \mathsf{aux})$ and provide R to $\mathcal{A}$
7. After the above, when $\mathcal{A}$ has sent the message (Sample, i) for every $i \notin H$, provide R to Ch. If $b = 1$, provide T too.
8. Keep letting $\mathcal{A}$ and Ch interact
9. The adversary wins if it halts outputting b

We recall the definition of chosen-sample indistinguishability. Essentially, a game with oracle distribution $\mathcal{G}_0$ is chosen-sample indistinguishable from a game with trapdoored oracle distribution $\mathcal{G}_1$ if no adversary can tell the two games apart even if it chooses the responses of the oracle distribution (no trapdoors are given to the challenger of $\mathcal{G}_1$).

Definition 5.3.14 (Chosen-Sample Indistinguishability [AWZ23]). Let $\mathcal{D}'$ be a trapdoored distribution fro $\mathcal{D}$. Let $\mathcal{G}_0 = (\mathsf{Ch}_0, \mathcal{D})$ and $\mathcal{G}_1 = (\mathsf{Ch}_1, \mathcal{D}')$ be a game with oracle distribution and a game with trapdoored oracle distribution respectively. We say that $\mathcal{G}_0$ and $\mathcal{G}_1$ are chosen-sample indistinguishable if no PPT adversary can win the Game 5.3.15 with non-negligible advantage in λ.

Game 5.3.15. Chosen-Sample Indistinguishability

1. $b \xleftarrow{\$} \{0,1\}$
2. Activate the adversary $\mathcal{A}$ with 1^λ
3. Receive ϕ and $H \subseteq [N]$ from $\mathcal{A}$
4. Activate Ch_b on input 1^λ, ϕ and H
5. Let Ch_b and $\mathcal{A}$ interact
6. When Ch_b has sent the message (Sample, i) for every $i \in H$ and $\mathcal{A}$ has sent the message (Sample, i) for every $i \notin H$, receive R from $\mathcal{A}$ and forward it to Ch_b.
7. Keep letting $\mathcal{A}$ and Ch_b interact
8. The adversary wins if it halts outputting b

We can finally recall the definition of indistinguishability-preserving distributed samplers

Definition 5.3.16 (Indistinguishability-Preserving Distributed Sampler [AWZ23]). Let $\mathcal{D}'$ be a trapdoored distribution for $\mathcal{D}$. We say that an N-party trapdoorable distributed sampler is indistinguishability-preserving for $(\mathcal{D}, \mathcal{D}')$ if, for every N-party game with oracle distribution $\mathcal{G}_0 = (\mathsf{Ch}_0, \mathcal{D})$ and N-party game with trapdoored oracle distribution $\mathcal{G}_1 = (\mathsf{Ch}_1, \mathcal{D}')$ such that

- $\mathcal{G}_0$ and $\mathcal{G}_1$ are chosen-sample indistinguishable and

- $\mathcal{G}_1$ satisfies trapdoor security

no PPT adversary can win the Game 5.3.17 with non-negligible advantage in λ.

Game 5.3.17. Indistinguishability-Preserving Distributed Sampler

Initialisation: This procedure is run only once at the beginning of the game.

1. $b \xleftarrow{\$} \{0,1\}$
2. $\sigma_0 \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$
3. $(\sigma_1, \zeta) \xleftarrow{\$} \mathsf{SimSetup}(1^\lambda)$
4. Activate the adversary $\mathcal{A}$ with 1^λ and σ_b
5. Receive a list of party identities $\mathsf{id}_1, \dots, \mathsf{id}_M$ and $H \subseteq [M]$ from $\mathcal{A}$

Session: This procedure can be queried as many times as the adversary wants and at any point in time (multiple sessions can be run simultaneously).

1. Receive $\mathsf{sid} := (\mathsf{tag}, \mathsf{id}_{i_1}, \dots, \mathsf{id}_{i_N})$ from the adversary where $\mathsf{tag} \in \{0,1\}^*$. Proceed with the following steps only if sid hasn't been queried before and if there exists no $h \neq j$ such that $i_j = i_h$.
2. Receive ψ from the adversary.
3. Activate a copy of Ch_b on input 1^λ, ϕ and $H \cap \{i_1, \dots, i_N\}$
4. Let Ch_b and $\mathcal{A}$ interact
5. When Ch_b sends (Sample, j) where $i_j \in H$ and there exists another $i_h \in H$ such that (Sample, h) hasn't yet been sent, compute $U_j \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \mathsf{sid}, j, \sigma_b)$ and send U_j to the adversary.
6. When Ch_b sends (Sample, j) where $i_j \in H$ and there exists no another $i_h \in H$ such that (Sample, h) hasn't yet been sent, perform the following
   - If $b = 0$, compute $U_j \xleftarrow{\$} \mathsf{Gen}(1^\lambda, \mathsf{sid}, j, \sigma_0)$ and send U_j to the adversary.
   - If $b = 1$, receive aux from the challenger Ch_1 and compute $(U_j, \xi) \xleftarrow{\$} \mathsf{SimGen}(1^\lambda, \mathsf{sid}, j, \zeta, \mathsf{aux})$ and send U_j to the adversary.
7. When the adversary sends a distributed sampler message U_j on behalf of a corupted party P_{i_j}, send (Sample, j) to Ch_b.
8. When Ch_b has sent the special message on behalf of all honest parties in the execution and has received the special message on behalf of all corrupted players, perform the following:
   - If $b = 0$, provide Ch_0 with $R \leftarrow \mathsf{Sample}(U_1, \dots, U_N, \mathsf{sid}, \sigma_0)$
   - If $b = 1$, provide Ch_1 with $(R, T) \leftarrow \mathsf{Trap}(\xi, U_1, \dots, U_N)$
9. Keep letting $\mathcal{A}$ and Ch_b interact
10. The adversary wins if it halts outputting b

5.3.5 Additional Cryptographic Primitives

We recall the definition of indistinguishability obfuscation and puncturable PRFs.

Definition 5.3.18 (Indistinguishability Obfuscation [BGI+01, GGH+13, JLS21]). An indistinguishability obfuscator (iO) consists of a PPT algorithm iO satisfying the following properties:

- **(Correctness).** For every circuit C

$$\Pr\left[\exists x : C'(x) \neq C(x) \middle| C' \xleftarrow{\$} \mathsf{iO}(1^\lambda, C) \quad \right] = 0.$$

- **(Security).** For every sampler $\mathcal{A}$ outputting circuits C_0, C_1 of the same size such that $C_0(x) = C_1(x)$ for every input x, along with auxiliary information aux, the following distributions are computationally indistinguishable

$$\left\{ C', \mathsf{aux} \middle| \begin{array}{l} (C_0, C_1, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}(1^\lambda) \\ C' \xleftarrow{\$} \mathsf{iO}(1^\lambda, C_0) \end{array} \right\} \qquad \left\{ C', \mathsf{aux} \middle| \begin{array}{l} (C_0, C_1, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}(1^\lambda) \\ C' \xleftarrow{\$} \mathsf{iO}(1^\lambda, C_1) \end{array} \right\}$$

Definition 5.3.19 (Puncturable PRF [KPTZ13, BW13, BGI14]). A puncturable PRF with domain $\{0,1\}^{L(\lambda)}$ and range $\{0,1\}^{M(\lambda)}$ consists of a pair of deterministic, polynomial time algorithms (F, Punct) satisfying the following properties:

- **(Correctness).** For every $K \in \{0,1\}^\lambda$ and distinct inputs $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^{L(\lambda)}$, we have

$$\Pr\left[F(K, \boldsymbol{y}) = F(\hat{K}, \boldsymbol{y}) \middle| K' \leftarrow \mathsf{Punct}(K, \boldsymbol{x}) \quad \right] = 1.$$

- **(Security).** For every input $\boldsymbol{x} \in \{0,1\}^{L(\lambda)}$, the following distributions are computationally indistinguishable

$$\left\{ \hat{K}, \boldsymbol{r} \middle| \begin{array}{l} K \xleftarrow{\$} \{0,1\}^\lambda \\ \hat{K} \leftarrow \mathsf{Punct}(K, \boldsymbol{x}) \\ \boldsymbol{r} \leftarrow F(K, \boldsymbol{x}) \end{array} \right\} \qquad \left\{ \hat{K}, \boldsymbol{r} \middle| \begin{array}{l} K \xleftarrow{\$} \{0,1\}^\lambda \\ \hat{K} \leftarrow \mathsf{Punct}(K, \boldsymbol{x}) \\ \boldsymbol{r} \xleftarrow{\$} \{0,1\}^{M(\lambda)} \end{array} \right\}$$

5.4 Coin Tossing Extension and Explainable Extractors

5.4.1 Properties of Coin Tossing Extension Protocols

In this section, we introduce and prove a few basic properties of CTE protocols, we give a formal definition for explainable extractors, and a proof that explainable extractors are implied by simulation-secure CTE protocols.

Definition 5.4.1 (Closure under Early Stopping). Let $\mathfrak{A}$ be a class of adversaries. For any $\mathcal{A} \in \mathfrak{A}$, PPT algorithm M and $r \in \mathbb{N}$, we define the adversary $\mathcal{A}_{M,r}$ as the adversary that

- if $\mathcal{A}$ uses only static corruption, $\mathcal{A}_{M,r}$ statically corrupts all parties except one chosen at random among those not corrupted by $\mathcal{A}$ (let this party be P_j). Then, it runs the parties corrupted by $\mathcal{A}$ as $\mathcal{A}$ does and the remaining parties by following the protocol. At the end of the r-th round, $\mathcal{A}_{M,r}$ receives the message from the honest party for round $r+1$, provides it to $\mathcal{A}$ and runs M on the view of $\mathcal{A}$ obtaining a bit b, messages $(\mathsf{msg}_i)_{i \neq j}$ and a value z. If $b = 0$, $\mathcal{A}_{M,r}$ immediately stops delivering messages to P_j and outputs z. Otherwise, it sends $(\mathsf{msg}_i)_{i \neq j}$ to P_j on behalf of the corrupted parties and then halts outputting z.

- if $\mathcal{A}$ uses adaptive corruption, $\mathcal{A}_{M,r}$ behaves as $\mathcal{A}$ in the first r rounds of the protocol, then corrupts all the remaining honest parties except one chosen at random (let this be P_j). The adversary $\mathcal{A}_{M,r}$ waits for the messages of P_j in the $(r+1)$-th round, provides it to $\mathcal{A}$ and runs M on the view of $\mathcal{A}$ obtaining a bit b, messages $(\mathsf{msg}_i)_{i \neq j}$ and a value z. If $b = 0$, $\mathcal{A}_{M,r}$ immediately stops delivering messages to P_j and outputs z. Otherwise, it sends $(\mathsf{msg}_i)_{i \neq j}$ to P_j on behalf of the corrupted parties and then halts outputting z.

We say that $\mathfrak{A}$ is closed under early stopping if, for every $\mathcal{A} \in \mathfrak{A}$, PPT algorithm M and $r \in \mathbb{N}$, $\mathcal{A}_i M, r$ belongs to $\mathfrak{A}$ too.

We use $\mathcal{A}_r$ to denote the adversary $\mathcal{A}_{M,r}$ where M is the algorithm that always outputs $b = 0$, $(\mathsf{msg}_i)_{i \neq j} = \perp$ and z consisting of the view of $\mathcal{A}$.

Definition 5.4.2 (Closure under Corruption-Preserving Extension). We say that $\mathfrak{A}$ is closed under corruption-preserving extension if for every adversary $\mathcal{A} \in \mathfrak{A}$ and round $r \in \mathbb{N}$, there exists an adversary $\mathcal{A}' \in \mathfrak{A}$ that behaves as $\mathcal{A}$ for the first r rounds and, after that, never corrupts any new party.

Theorem 5.4.3. Let $\mathfrak{A}$ be a class of adversaries that is closed under early stopping and corruption-preserving extension. Suppose that Π is an $\mathfrak{A}$-secure N-party coin-tossing extension protocol. Consider the following modification of Π: the parties run Π until (including) the round in which the final call to $\mathcal{F}^n_{\mathsf{Coin}}$ is made. Then, each party terminates the protocol after computing the output according to the instructions in Π for the event in which all the remaining parties remains silent in the next round. The modified protocol is an $\mathfrak{A}$-secure N-party coin-tossing extension protocol with the same stretch.

Proof. Without loss of generality, we assume that, in each round, the parties speak all simultaneously (sending the empty string corresponds to not speaking on that given channel). Consider an adversary $\mathcal{A} \in \mathfrak{A}$. And let κ be the round where the parties call $\mathcal{F}^n_{\mathsf{Coin}}$ for the last time. The simulator $\mathsf{Sim}_{\mathcal{A}}$ is pretty simple: we pick an honest party P_i and we start running the simulator for the original protocol Π against the adversary $\mathcal{A}_\kappa$ but giving instruction to the functionality to corrupt only the parties chosen by $\mathcal{A}$. Immediately after the last call to $\mathcal{F}^n_{\mathsf{Coin}}$, we stop the simulation. Clearly, if at that point, there is only one party not corrupted by $\mathcal{A}$, the value output by the functionality on behalf of the only party not corrupted by $\mathcal{A}_\kappa$ will be consistent with the execution of the protocol. Now, we need to show that if there are multiple parties not corrupted by $\mathcal{A}$, their outputs are all equal with overwhelming probability.

We follow the blueprint of the impossibility of [Cle86]. Suppose that after the last call of $\mathcal{F}^n_{\mathsf{Coin}}$, there exist ℓ rounds of interaction. Pick any $\alpha \in [m]$. For any two distinct parties $i, j \in [n]$ and $r \in [\ell]$, we define the random variable $b_{j,r}$ as the α-th bit that P_j outputs if the other parties stop speaking in the r-th round following the last call to $\mathcal{F}^n_{\mathsf{Coin}}$. Similarly, we define the random variable $b_{i,r}$ as the α-th bit that P_i outputs if the other parties stop speaking in the r-th round following the last call to $\mathcal{F}^n_{\mathsf{Coin}}$.

For every $b \in \{0, 1\}$, $r \in [\ell]$, we consider the adversary $\mathcal{A}^b_{r,0}$ that behaves as follows:

- If $\mathcal{A}$ uses only static corruption, $\mathcal{A}^b_{r,0}$ picks two random parties P_i and P_j among those not corrupted by $\mathcal{A}$ (if $\mathcal{A}$ corrupts $N-1$ parties, $\mathcal{A}^b_{r,0}$ sets a flag $e \leftarrow \perp$ and behaves as $\mathcal{A}_{\kappa+r}$), statically corrupts all the parties except P_i, models the behaviour of $\mathcal{A}$ for all the parties corrupted by $\mathcal{A}$ and makes all other parties follow the protocol (including P_j). At the r-th round after the last call to $\mathcal{F}^n_{\mathsf{Coin}}$, $\mathcal{A}^b_{r,0}$ simulates the r-th round in its head using P_i's message in the real protocol execution and simulating the execution of all other parties not corrupted by $\mathcal{A}$ by following the protocol. It then checks whether $b_{j,r+1} = b$. If that is the case it quits the execution in the r-th round without sending any message, otherwise, it quits at the following round following the execution in its head for the r-th round.

- If $\mathcal{A}$ uses adaptive corruption, let $\mathcal{A}'$ be the corruption-preserving extension of $\mathcal{A}$ with respect to round κ. The adversary $\mathcal{A}^b_{r,0}$ behaves as $\mathcal{A}'$ for the first $r-1$ rounds after the last call to $\mathcal{F}^n_{\mathsf{Coin}}$. Then, it picks a random honest party P_i and corrupts all other players. It waits for the message of P_i in the r-th round and uses it to simulate the rest of the round in its head according to $\mathcal{A}'$. Then, $\mathcal{A}^b_{r,0}$ picks a random party P_j different from P_i among those not corrupted by $\mathcal{A}$ (if this party doesn't exist $\mathcal{A}^b_{r,0}$ sets a flag $e \leftarrow \perp$ and behaves as $\mathcal{A}_{\kappa+r}$) and checks whether $b_{j,r+1} = b$. If that is the case it quits the execution in the r-th round without sending any message, otherwise, it quits at the following round following the execution in its head for the r-th round.

We also consider the adversary $\mathcal{A}^b_{r,1}$ that behaves as follows:

- If $\mathcal{A}$ uses only static corruption, $\mathcal{A}^b_{r,1}$ picks two random parties P_i and P_j among those not corrupted by $\mathcal{A}$ (if $\mathcal{A}$ corrupts $N-1$ parties, $\mathcal{A}^b_{r,1}$ sets a flag $e \leftarrow \perp$ and behaves as $\mathcal{A}_{\kappa+r}$), statically corrupts all the parties except P_j, models the behaviour of $\mathcal{A}$ for all the parties corrupted by $\mathcal{A}$ and makes all other parties follow the protocol (including P_i). It then checks whether $b_{i,r} = b$. If that is the case it quits the execution in the r-th round after the last call to $\mathcal{F}^n_{\mathsf{Coin}}$ without sending any message, otherwise, it quits at the following round.

- If $\mathcal{A}$ uses adaptive corruption, let $\mathcal{A}'$ be the corruption-preserving extension of $\mathcal{A}$ with respect to round κ. The adversary $\mathcal{A}^b_{r,1}$ behaves as $\mathcal{A}'$ for the first $r-1$ rounds after the last call to $\mathcal{F}^n_{\mathsf{Coin}}$. Then, it corrupts a random party P_i among those not corrupted by $\mathcal{A}$ (if P_i is the only remaining honest party, $\mathcal{A}^b_{r,1}$ sets a flag $e \leftarrow \bot$ and behaves as $\mathcal{A}_{\kappa+r}$). If $b_{i,r} = b$, $\mathcal{A}^b_{r,1}$ corrupts all other parties but one (let this be P_j) and halts. Otherwise, it runs for one additional round following the instructions of $\mathcal{A}$ and making P_i behave honestly. After this, it corrupts all parties but one (let this be P_j) and halts.

Let E be the event in which, at the end of round κ, the adversary $\mathcal{A}$ has not corrupted $N-1$ parties yet. The bias towards b of the α-th bit output by P_i when the adversary is $\mathcal{A}^b_{r,0}$ is

$$\begin{aligned}\mathsf{Adv}(\mathcal{A}^b_{r,0}) := &\Pr[b_{i,r} = b, b_{j,r+1} = b, E] + \Pr[b_{i,r+1} = b, b_{j,r+1} = 1-b, E]\\ &+ \Pr[b_{i,r} = b, \neg E] - 1/2\end{aligned}$$

Similarly, the bias towards b of the α-th bit output by P_j when the adversary is $\mathcal{A}^b_{r,1}$ is

$$\begin{aligned}\mathsf{Adv}(\mathcal{A}^b_{r,1}) := &\Pr[b_{j,r} = b, b_{i,r} = b, E] + \Pr[b_{j,r+1} = b, b_{i,r} = 1-b, E]\\ &- \Pr[b_{j,r} = b, \neg E] - 1/2\end{aligned}$$

Now, if we consider the following sum, we obtain

$$\begin{aligned}&\sum_{r\in[\ell]} \left(\mathsf{Adv}(\mathcal{A}^0_{r,0}) + \mathsf{Adv}(\mathcal{A}^1_{r,0}) + \mathsf{Adv}(\mathcal{A}^0_{r,1}) + \mathsf{Adv}(\mathcal{A}^1_{r,1})\right)\\ &= \sum_{r\in[\ell]} \begin{pmatrix} \Pr[b_{i,r} = 0, b_{j,r+1} = 0, E] + \Pr[b_{i,r+1} = 0, b_{j,r+1} = 1, E]\\ + \Pr[b_{i,r} = 0, \neg E] + \Pr[b_{i,r} = 1, b_{j,r+1} = 1, E]\\ + \Pr[b_{i,r+1} = 1, b_{j,r+1} = 0, E] + \Pr[b_{i,r} = 1, \neg E]\\ + \Pr[b_{j,r} = 0, b_{i,r} = 0, E] + \Pr[b_{j,r+1} = 0, b_{i,r} = 1, E]\\ + \Pr[b_{j,r} = 0, \neg E] + \Pr[b_{j,r} = 1, b_{i,r} = 1, E]\\ + \Pr[b_{j,r+1} = 1, b_{i,r} = 0, E] + \Pr[b_{j,r} = 1, \neg E] - 2 \end{pmatrix}\end{aligned}$$

We observe that, for every $r \in [\ell]$, we have $\Pr[b_{i,r} = 0, \neg E] + \Pr[b_{i,r} = 1, \neg E] = \Pr[\neg E]$ and $\Pr[b_{j,r} = 0, \neg E] + \Pr[b_{j,r} = 1, \neg E] = \Pr[\neg E]$. Furthermore,

$$\begin{aligned}\Pr[E] = &\Pr[b_{i,r} = 0, b_{j,r+1} = 0, E] + \Pr[b_{i,r} = 1, b_{j,r+1} = 0, E]\\ &+ \Pr[b_{j,r+1} = 0, b_{i,r} = 1, E] + \Pr[b_{j,r+1} = 1, b_{i,r} = 1, E]\end{aligned}$$

and

$$\begin{aligned}\Pr[E] = &\Pr[b_{i,r} = 0, b_{j,r} = 0, E] + \Pr[b_{i,r} = 1, b_{j,r} = 0, E]\\ &+ \Pr[b_{j,r} = 0, b_{i,r} = 1, E] + \Pr[b_{j,r} = 1, b_{i,r} = 1, E].\end{aligned}$$

Therefore, we obtain that

$$\begin{aligned}&\sum_{r\in[\ell]} \left(\mathsf{Adv}(\mathcal{A}^0_{r,0}) + \mathsf{Adv}(\mathcal{A}^1_{r,0}) + \mathsf{Adv}(\mathcal{A}^0_{r,1}) + \mathsf{Adv}(\mathcal{A}^1_{r,1})\right)\\ &= \Pr[b_{i,0} = 0, b_{j,0} = 0, E] + \Pr[b_{i,0} = 1, b_{j,0} = 1, E]\\ &\quad + \Pr[b_{i,\ell+1} = 0, b_{j,\ell+1} = 1, E] + \Pr[b_{i,\ell+1} = 1, b_{j,\ell+1} = 0, E] + \Pr[\neg E] - 1\\ &= \Pr[b_{i,0} = b_{j,0}, E] + \Pr[b_{i,\ell+1} \neq b_{j,\ell+1}, E] - \Pr[E].\end{aligned}$$

Due to the security of the protocol against $\mathcal{A}'$, we know that $\Pr[b_{i,\ell+1} \neq b_{j,\ell+1}, E] \leq \mathsf{negl}(\lambda)$. By the security of the protocol against $\mathcal{A}^b_{r,0}$ and $\mathcal{A}^b_{r,1}$, we also have $|\mathsf{Adv}(\mathcal{A}^b_{r,0})|, |\mathsf{Adv}(\mathcal{A}^b_{r,1})| \leq \mathsf{negl}(\lambda)$. Since r is at most polynomial in λ and all the addends are negligible functions, we have

$$\left|\sum_{r\in[\ell]} \left(\mathsf{Adv}(\mathcal{A}^0_{r,0}) + \mathsf{Adv}(\mathcal{A}^1_{r,0}) + \mathsf{Adv}(\mathcal{A}^0_{r,1}) + \mathsf{Adv}(\mathcal{A}^1_{r,1})\right)\right| \leq \mathsf{negl}(\lambda).$$

We conclude that $\Pr[b_{i,0} = b_{j,0}, E] = \Pr[E] - \mathsf{negl}(\lambda)$. Notice that this holds for any $\alpha \in [m]$. So at the time of the last call to $\mathcal{F}^n_{\mathsf{Coin}}$, all the honest parties already agree on an m-bit string. During the simulation of the protocol such string must necessarily be consistent with the output of the functionality. This ends the proof. $\square$

5.4.2 Explainable Extractors

Definition 5.4.4 (Entropy Source Class). An entropy source class $\mathcal{S}$ is a set of not-necessarily-efficient randomised algorithms that on input 1^λ, output values $\boldsymbol{x} \in \{0,1\}^{L(\lambda)}$ where $L(\lambda)$ is polynomial and auxiliary information aux.

Definition 5.4.5 (Explainable Extractor). An explainable extractor for the entropy source class $\mathcal{S}$ is a deterministic polynomial-time algorithm $\mathsf{Extract}$ taking as input the security parameter 1^λ, a strings $\boldsymbol{x} \in \{0,1\}^{L(\lambda)}$ and $\boldsymbol{u} \in \{0,1\}^{n(\lambda)}$ and outputting a string $\boldsymbol{s} \in \{0,1\}^{m(\lambda)}$. We require that, for every $S \in \mathcal{S}$, there exists a PPT algorithm $\mathsf{Explain}_S$ such that the following distributions are indistinguishable

$$\left\{ \mathsf{aux}, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{s} \;\middle|\; \begin{array}{l} (\boldsymbol{x}, \mathsf{aux}) \xleftarrow{\$} S(1^\lambda) \\ \boldsymbol{u} \xleftarrow{\$} \{0,1\}^{n(\lambda)} \\ \boldsymbol{s} \leftarrow \mathsf{Extract}(1^\lambda, \boldsymbol{x}, \boldsymbol{u}) \end{array} \right\}$$
$$\left\{ \mathsf{aux}, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{s} \;\middle|\; \begin{array}{l} \boldsymbol{s} \xleftarrow{\$} \{0,1\}^{m(\lambda)} \\ (\mathsf{aux}, \boldsymbol{x}, \boldsymbol{u}) \xleftarrow{\$} \mathsf{Explain}_S(1^\lambda, \boldsymbol{s}) \end{array} \right\}$$

We say that the explainable extractor is computational if indistinguishability holds only against PPT distinguishers. Otherwise, we say that the explainable extractor is statistically (or information-theoretically) secure.

Observe that, in every explainable extractor, for every $S \in \mathcal{S}$, the distribution of $\mathsf{Extract}(1^\lambda, \boldsymbol{x}, \boldsymbol{u})$, where $\boldsymbol{x} \xleftarrow{\$} S(1^\lambda)$ and $\boldsymbol{u} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$, is indistinguishable from the uniform distribution over $\{0,1\}^{m(\lambda)}$.

Theorem 5.4.6. If Π is an $\mathfrak{A}$-secure N-party coin tossing extension protocol ending with a call to $\mathcal{F}^n_{\mathsf{Coin}}$. Suppose that Π has public output (i.e., it is possible to compute the output given only the transcript of the protocol and the responses of $\mathcal{F}^n_{\mathsf{Coin}}$. Then, there exists an explainable extractor for the entropy source class $\mathcal{S}$ consisting of all $(S_\mathcal{A})_{\mathcal{A}\in\mathfrak{A}}$, where $S_\mathcal{A}$ simulates an execution of Π together with $\mathcal{A}$ and outputs the transcript of the protocol except for the answer to the last call to $\mathcal{F}^n_{\mathsf{Coin}}$ and (as auxiliary information) the view of $\mathcal{A}$.

Proof. We consider the explainable extractor that, given the transcript x for the protocol Π generated by $S_\mathcal{A}$, and $\boldsymbol{u} \in \{0,1\}^{n(\lambda)}$, returns the output of Π as if $\boldsymbol{u}$ was the response to the last call to $\mathcal{F}^n_{\mathsf{Coin}}$.

By the security of the coin tossing extension protocol, we know that for every adversary $\mathcal{A}$, there exists a simulator $\mathsf{Sim}_\mathcal{A}$ for which it is hard to distinguish between the interaction of $\mathcal{A}$ with Π, and the interaction of $\mathsf{Sim}_\mathcal{A}$ with $\mathcal{F}^m_{\mathsf{Coin}}$. We can therefore build $\mathsf{Explain}_{S_\mathcal{A}}$ by simply running $\mathsf{Sim}_\mathcal{A}$ on the provided sample $\boldsymbol{s} \in \{0,1\}^{m(\lambda)}$. $\square$

5.5 One-Round, One-Sample Adaptive Coin Tossing Extension from LWE

In this section, we present our construction for one-round one-query CRS-free CTE with universally composable security against adaptive adversaries, from subexponential LWE. We previously overviewed this construction in Section 5.2.3, and invite the reader to review the basic definitions and common tools we use in Section 5.3.1 and Section 5.3.3. We begin with our protocol, then prove it secure and show by corollary the class of explainable extractors that it implies.

Protocol 5.5.1. One-Round, One-Query CTE from LWE

Let $K, M, V, W, L, s_0, s_1 : \mathbb{N} \to \mathbb{N}$ be polynomial functions in the security parameter. Assume that $K = \Omega(\lambda)$. Define $p := 2$ and $q := 2^t$ where $t = \Theta(\lambda)$.[a] Let χ_0 be $\mathcal{D}_{\mathbb{Z},s_0}$, let χ_1 be $\mathcal{D}_{\mathbb{Z},s_1}$. Suppose that $W \geq \beta \cdot K \cdot \log q$ and $M \geq \beta \cdot (K+V) \cdot \log q$, where β is the constant defined in Theorem 5.3.5. Assume also that $s_0 = \sqrt{(K+V)\log q} \cdot \omega(\sqrt{\log(K+V)})$ and $s_1 = \sqrt{K \log q} \cdot \omega(\sqrt{\log K})$. Let G be a $(K+V) \times (K+V) \cdot t$ gadget matrix, and let G^{-1} be the deterministic algorithm that, on input a matrix $Y' \in \mathbb{Z}_q^{(K+V)\times M'}$ for some $M' \in \mathbb{N}$, outputs a matrix X' such that $G \cdot X' = Y'$ and $\|Q\|_\infty = 1$ (see Gentry et al. [GSW13]). For every $j \in [N]$, let δ_j denote the Kronecker delta function centered on j.

Protocol. Each party P_i performs the following operations.

1. $\forall \ell \in [L]: \quad \boldsymbol{x_{i,\ell}} \xleftarrow{\$} \chi_0^M$
2. Broadcast $(\boldsymbol{x_{i,1}}, \dots, \boldsymbol{x_{i,L}})$
3. For any $j \in [N]$ and $\ell \in [L]$ such that $\|\boldsymbol{x_{j,\ell}}\|_\infty > \sqrt{K} \cdot s_0$, set $\boldsymbol{x_{j,\ell}} \leftarrow \mathbf{0}$.
4. Call $\mathcal{F}^n_{\mathsf{Coin}}$ and interpret the response as values $(X_i)_{i\in[\lceil \log N \rceil]}, Y, C, D$ and $(\boldsymbol{e_\ell})_{\ell\in[L]}$ where
   - $\forall i \in [\lceil \log N \rceil] : X_i \in \mathbb{Z}_q^{(K+V)\times M}$
   - $Y \in \mathbb{Z}_q^{(K+V)\times M}$
   - $C \in \mathbb{Z}_q^{K\times W}$
   - $D \in \mathbb{Z}_q^{V\times W}$
   - $\boldsymbol{e_\ell}$ is a sample from ${\chi_1}^W$
5. $\forall j \in [N] : Z_j \leftarrow \mathsf{Eval}(\delta_j, X_1, \dots, X_{\lceil \log N \rceil}) \cdot G^{-1}(Y)$ (see Algorithm 5.5.2)[b]
6. For every $j \in [N]$, let $A_j \in \mathbb{Z}_q^{K\times M}$ consist of the first K rows of Z_j. Let $B_j \in \mathbb{Z}_q^{V\times M}$ consist of the last V rows of Z_j.
7. $\forall \ell \in [L] : \boldsymbol{u_\ell} \leftarrow \sum_{j\in[N]} A_j \cdot \boldsymbol{x_{j,\ell}} - C \cdot \boldsymbol{e_\ell}$
8. $\forall \ell \in [L] : \boldsymbol{v_\ell} \leftarrow \lceil \sum_{j\in[N]} B_j \cdot \boldsymbol{x_{j,\ell}} - D \cdot \boldsymbol{e_\ell}) \rfloor_p$
9. Output $(\boldsymbol{u_1}, \dots, \boldsymbol{u_L}, \boldsymbol{v_1}, \dots, \boldsymbol{v_L})$

[a] The protocol can be generalised to any p, q such that $p \cdot 2^{\log^2 \lambda} \cdot \alpha$ and p/q are negligible.

[b] We can compute $\delta_j(\boldsymbol{x})$ where $\boldsymbol{x} \in \{0,1\}^{\lceil \log N \rceil}$ by first flipping x_h for every h such that the h-th bit of j is 1. Then, we multiply all $\lceil \log N \rceil$ bits obtained in this way and we flip the result.

Algorithm 5.5.2. $\mathsf{Eval}(f, X_1, \dots, X_m)$ [GSW13]

Represent $f : Z_q^m \to \mathbb{Z}_q$ as an arithmetic circuit over $\mathbb{Z}_q$. Then, perform the following operations:

1. Associate X_i to the i-th input wire for every $i \in [m]$.
2. For every gate, perform the following operations:
   - If the gate is an addition gate and the input wires are associated with the matrices Z_1, Z_2, associate the output wire with $Z_1 + Z_2$.
   - If the gate adds a constant $k \in \mathbb{Z}_q$ to a wire associated with the matrix Z_1, associate the output wire with $Z_1 + k \cdot G$.
   - If the gate switches the sign of a wire associated with the matrix Z_1, associate the output wire with $-Z_1$.
   - If the gate is a multiplication gate and the input wires are associated with the matrices Z_1, Z_2, associate the output wire with $Z_1 \cdot G^{-1}(Z_2)$.
   - If the gate multiplies a wire associated with the matrix Z_1 by a constant $k \in \mathbb{Z}_q$, associate the output wire with $Z_1 \cdot G^{-1}(k \cdot G)$.
3. Output the matrix associated with the output wire of the circuit.

The following theorem essentially formalizes Theorem 5.2.5.

Theorem 5.5.3. Assuming the hardness of the Learning with Errors (LWE) problem with a subexponential modulus-to-noise ratio (see Definition 5.3.3), Protocol 5.5.1 UC-realizes $\mathcal{F}^m_{\mathsf{Coin}}$ among N parties in the $\mathcal{F}^n_{\mathsf{Coin}}$-hybrid model, with security against a malicious PPT adversary adaptively corrupting up to $N-1$ parties. For any function $\eta = \mathsf{poly}(\lambda)$, if we set $s_1 = \sqrt{K \cdot \log q} \cdot \log K$, $M = \beta \cdot (K+V) \cdot \log q$, $W = \beta \cdot K \cdot \log q$, $V = (\eta \cdot \lambda^2 - 1) \cdot K \cdot t$, and $L = \eta^2 \cdot t^3 \cdot \lambda^2 \cdot \log N \cdot K$, then the *multiplicative* stretch of the construction is $m/(t \cdot n) = \Omega(\eta)$.

Proof. We start by proving security. Let ι be the index of a party that is honest in the first round of the protocol. Define $\alpha := 2^{-\omega(\log^2 \lambda)}$ and set $s_2 \leftarrow \alpha \cdot q$. Let χ_2 be $\mathcal{D}_{\mathbb{Z},s_2}$. We proceed to prove security using a sequence of indistinguishable hybrids starting from the real world and arriving at the ideal world.

Hybrid $\mathcal{H}_0$. This hybrid corresponds to the real execution of the protocol.

Hybrid $\mathcal{H}_1$. In this hybrid, whenever a party P_i is corrupted after sending $(\boldsymbol{x_{i,\ell}})_{\ell \in [L]}$, instead of providing the randomness used to produce these values, we provide the adversary with $\mathsf{Explain}_{\chi_0^M}(1^\lambda, \boldsymbol{x_{i,\ell}})$ for every $\ell \in [L]$. This hybrid is indistinguishable from $\mathcal{H}_0$ due to Theorem 5.3.7.

Hybrid $\mathcal{H}_2$. In this hybrid, instead of providing the adversary with the randomness that produces $(\boldsymbol{e_\ell})_{\ell \in [L]}$, we provide it with $\mathsf{Explain}_{\chi_1^W}(1^\lambda, \boldsymbol{e_\ell})$ for every $\ell \in [L]$. This hybrid is indistinguishable from $\mathcal{H}_1$ due to Theorem 5.3.7.

Hybrid $\mathcal{H}_3$. In this hybrid, we change the distribution of $X_1, \dots, X_{\lceil \log N \rceil}$. In particular, we sample $U \xleftarrow{\$} \mathbb{Z}_q^{(K+V)\times M}$ and $R_1, \dots, R_{\lceil \log N \rceil} \xleftarrow{\$} \mathbb{Z}_2^{M \times (K+V)\cdot t}$. We then set $X_j \leftarrow U \cdot R_j + \iota_j \cdot G$ for every $j \in [\lceil \log N \rceil]$ where ι_j denote the j-th bit of ι. $\mathcal{H}_3$ is statistically indistinguishable from $\mathcal{H}_2$ thanks to the leftover hash lemma (see Lemma 5.3.4, we are using the fact that $M \geq \beta \cdot (K+V) \cdot \log q$, $\beta > 1$ and $K = \Omega(\lambda)$).

Hybrid $\mathcal{H}_4$. In this hybrid, we change the distribution of U and D. In particular, we sample $U_1 \xleftarrow{\$} \mathbb{Z}_q^{K \times M}$, $S \xleftarrow{\$} \mathbb{Z}_q^{V \times K}$, $E_1 \xleftarrow{\$} \chi_2^{V \times M}$ and $E_2 \xleftarrow{\$} {\chi_2}^{V \times W}$. We then set $U_2 \leftarrow S \cdot U_1 + E_1$, $U^\intercal \leftarrow (U_1^\intercal \parallel U_2^\intercal)$ and $D \leftarrow S \cdot C + E_2$. $\mathcal{H}_4$ is indistinguishable from $\mathcal{H}_3$ thanks to the security of LWE with subexponential modulus-to-noise ratio (see Definition 5.3.3).

Hybrid $\mathcal{H}_5$. In this hybrid, we change the distribution of C. In particular, we sample it along with a lattice trapdoor, i.e., $(C, T) \xleftarrow{\$} \mathsf{TrapGen}(1^K, 1^W, q)$. This hybrid is statistically indistinguishable from $\mathcal{H}_4$ (see Theorem 5.3.5, we are using the fact that $W \geq \beta \cdot K \cdot \log q$ and $K = \Omega(\lambda)$).

Hybrid $\mathcal{H}_6$. In this hybrid, we change the distribution of Y. In particular, we sample it along with a lattice trapdoor, i.e., $(Y, T') \xleftarrow{\$} \mathsf{TrapGen}(1^{K+V}, 1^M, q)$. This hybrid is statistically indistinguishable from $\mathcal{H}_5$ (see Theorem 5.3.5, we are using the fact that $M \geq \beta \cdot (K+V) \cdot \log q$ and $K = \Omega(\lambda)$).

Hybrid $\mathcal{H}_7$. In this hybrid, we change the distribution of $(\boldsymbol{x}_{\iota,\ell})_{\ell \in [L]}$. In particular, for every $\ell \in [L]$, we sample $\boldsymbol{u}_\ell \xleftarrow{\$} \mathbb{Z}_q^K$ $\boldsymbol{u}''_\ell \xleftarrow{\$} \mathbb{Z}_q^K$ and $\boldsymbol{v}'_\ell \xleftarrow{\$} \mathbb{Z}_q^V$. For every $\ell \in [L]$, we set $\boldsymbol{v}''_\ell \leftarrow S \cdot \boldsymbol{u}''_\ell$ and sample a preimage $\boldsymbol{x}_{\iota,\ell} \xleftarrow{\$} \mathsf{PreSample}(T', \boldsymbol{w}_\ell, s_0)$ where $\boldsymbol{w}_\ell$ is the vector obtained by concatenating $\boldsymbol{u}_\ell + \boldsymbol{u}''_\ell$ and $\boldsymbol{v}'_\ell + \boldsymbol{v}''_\ell$ (in other words, $\boldsymbol{x}_{\iota,\ell}$ looks like a discrete Gaussian sample such that $Y \cdot \boldsymbol{x}_{\iota,\ell} = \boldsymbol{w}_\ell$). This hybrid is statistically indistinguishable from $\mathcal{H}_6$ (see Theorem 5.3.5, Lemma 5.3.4 and Lemma 5.3.6, we are using the fact that $s_0 = \sqrt{(K+V)\log q} \cdot \omega(\sqrt{\log(K+V)})$, $M \geq \beta \cdot (K+V) \cdot \log q$, $\beta > 1$ and $K = \Omega(\lambda)$). We observe that the probability that $\|\boldsymbol{x}_{\iota,\ell}\|_\infty > \sqrt{K} \cdot s_0$ is negligible in λ (see Lemma 5.3.6, we are using the fact that $K = \Omega(\lambda)$).

Hybrid $\mathcal{H}_8$. In this hybrid, we change the distribution of $(\boldsymbol{e}_\ell)_{\ell \in [L]}$. In particular, for every $\ell \in [L]$, we compute $\widetilde{\boldsymbol{u}}_\ell \leftarrow \boldsymbol{u}''_\ell + \sum_{i \neq \iota} A_i \cdot \boldsymbol{x}_{i,\ell} + U_1 \cdot F_\iota \cdot G^{-1}(Y) \cdot \boldsymbol{x}_{\iota,\ell}$ where $R_\iota \leftarrow \mathsf{FullEval}(\delta_\iota, X_1, \dots, X_{\lceil \log N \rceil}, R_1, \dots, R_{\lceil \log N \rceil}, \iota)$ (see Algorithm 5.5.4). Then, we sample a preimage $\boldsymbol{e}_\ell \xleftarrow{\$} \mathsf{PreSample}(T, \widetilde{\boldsymbol{u}}_\ell, s_1)$ (in other words, $\boldsymbol{e}_\ell$ looks like a discrete Gaussian sample such that $C \cdot \boldsymbol{e}_\ell = \widetilde{\boldsymbol{u}}_\ell$). This hybrid is statistically indistinguishable from $\mathcal{H}_7$ (see Theorem 5.3.5, Lemma 5.3.4 and Lemma 5.3.6, we are using the fact that $s_1 = \sqrt{K \log q} \cdot \omega(\sqrt{\log K})$, $W \geq \beta \cdot K \cdot \log q$, $\beta > 1$ and $K = \Omega(\lambda)$). Observe that $\boldsymbol{x}_{\iota,\ell}$ leaks nothing about $\boldsymbol{u}''_\ell$ as $\boldsymbol{u}_\ell$ and $\boldsymbol{v}'_\ell$ mask all the information.

Algorithm 5.5.4. $\mathsf{FullEval}(f, X_1, \dots, X_m, R_1, \dots, R_m, \boldsymbol{x})$ **[GSW13]**

Represent $f : Z_q^m \to \mathbb{Z}_q$ as an arithmetic circuit over $\mathbb{Z}_q$. Then, perform the following operations:

1. Associate (X_i, R_i, x_i) to the i-th input wire for every $i \in [m]$.
2. For every gate, perform the following operations:
   - If the gate is an addition gate and the input wires are associated with the triples $(Z_1, S_1, z_1), (Z_2, S_2, z_2)$, associate the output wire with $(Z_1 + Z_2, S_1 + S_2, z_1 + z_2)$.
   - If the gate adds a constant $k \in \mathbb{Z}_q$ to a wire associated with the triple (Z_1, S_1, z_1), associate the output wire with $(Z_1 + k \cdot G, S_1, z_1 + k)$.
   - If the gate switches the sign of a wire associated with the triple (Z_1, S_1, z_1), associate the output wire with $(-Z_1, -S_1, -z_1)$.
   - If the gate is a multiplication gate and the input wires are associated with the triples $(Z_1, S_1, z_1), (Z_2, S_2, z_2)$, associate the output wire with $(Z_1 \cdot G^{-1}(Z_2), S_1 \cdot G^{-1}(Z_2) + z_1 \cdot S_2, z_1 \cdot z_2)$.
   - If the gate multiplies a wire associated with the triple (Z_1, S_1, z_1) by a constant $k \in \mathbb{Z}_q$, associate the output wire with $(Z_1 \cdot G^{-1}(k \cdot G), S_1 \cdot G^{-1}(k \cdot G), k \cdot z_1)$.
3. Output the second element of the triple associated with the output wire of the circuit.

Claim 5.5.5 ([GSW13]). Let $X_i = U \cdot R_i + x_i \cdot G$ where $R_i \in \mathbb{Z}_2^{M \times (K+V) \cdot t}$ for every $i \in [m]$. Let $f : Z_q^m \to \mathbb{Z}_q$ be a function. Then, $\mathsf{Eval}(f, X_1, \dots, X_m) = U \cdot R_f + f(\boldsymbol{x}) \cdot G$, where $R_f \leftarrow \mathsf{FullEval}(f, X_1, \dots, X_m, R_1, \dots, R_m, \boldsymbol{x})$.

Proof. For every wire w, let X_w be the matrix associated with w during the execution of $\mathsf{Eval}(f, X)$. Let R_w be the second element in the pair associated with w during the execution of $\mathsf{FullEval}(f, \boldsymbol{X}, \boldsymbol{R})$. Let x_w be the value associated with w during the evaluation of $f(x)$. We show that for any wire w, we have $X_w = U \cdot R_w + x_w \cdot G$. This true for the input wires, and we show that it holds for every other wire w by induction. Consider the gate that outputs wire w:

- If the gate is an addition gate with input wires u, v, then

$$\begin{aligned} X_w &= X_u + X_v = U \cdot R_u + x_u \cdot G + U \cdot R_v + x_v \cdot G \\ &= U \cdot (R_u + R_v) + (x_u + x_v) \cdot G = U \cdot R_w + x_w \cdot G. \end{aligned}$$

- If the gate adds a constant $k \in \mathbb{Z}_q$ to a wire u, then

$$\begin{aligned} X_w &= X_u + k \cdot G = U \cdot R_u + x_u \cdot G + k \cdot G \\ &= U \cdot R_u + (x_u + k) \cdot G = U \cdot R_w + x_w \cdot G. \end{aligned}$$

- If the gate switches the sign of a wire u, then

$$\begin{aligned} X_w &= -X_u = -U \cdot R_u - x_u \cdot G \\ &= U \cdot (-R_u) + (-x_u) \cdot G = U \cdot R_w + x_w \cdot G. \end{aligned}$$

- If the gate is a multiplication with input wires u, v, then

$$\begin{aligned} X_w &= X_u \cdot G^{-1}(X_v) = U \cdot R_u \cdot G^{-1}(X_v) + x_u \cdot G \cdot G^{-1}(X_v) \\ &= U \cdot R_u \cdot G^{-1}(X_v) + x_u \cdot X_v \\ &= U \cdot R_u \cdot G^{-1}(X_v) + x_u \cdot (U \cdot R_v + x_v \cdot G) \\ &= U \cdot (R_u \cdot G^{-1}(X_v) + x_u \cdot R_v) + (x_u \cdot x_v) \cdot G = U \cdot R_w + x_w \cdot G. \end{aligned}$$

- If the gate multiplies a wire u by a constant $k \in \mathbb{Z}_q$, then

$$\begin{aligned} X_w &= X_u \cdot G^{-1}(k \cdot G) = U \cdot R_u \cdot G^{-1}(k \cdot G) + x_u \cdot G \cdot G^{-1}(k \cdot G) \\ &= U \cdot (R_u \cdot G^{-1}(k \cdot G)) + (k \cdot x_u) \cdot G = U \cdot R_w + x_w \cdot G. \end{aligned}$$

This ends the proof of the claim. □

Let G_1 denote the matrix consisting of the first K rows of G. Let G_2 be the matrix consisting of the last V rows of G. Let Y_1 be the matrix consisting of the first K rows of Y. Let Y_2 be the matrix consisting of the last V rows of Y. Observe that, for every $\ell \in [L]$, we have

$$\begin{aligned} &\sum_{i \in [N]} A_i \cdot \boldsymbol{x_{i,\ell}} - C \cdot \boldsymbol{e_\ell} = A_\iota \cdot \boldsymbol{x_{\iota,\ell}} - C \cdot \boldsymbol{e_\ell} + \sum_{i \neq \iota} A_i \cdot \boldsymbol{x_{i,\ell}} \\ &\quad = U_1 \cdot F_\iota \cdot G^{-1}(Y) \cdot \boldsymbol{x_{\iota,\ell}} + \delta_\iota(\iota) \cdot G_1 \cdot G^{-1}(Y) \cdot \boldsymbol{x_{\iota,\ell}} - \widetilde{\boldsymbol{u}}_{\boldsymbol{\ell}} + \sum_{i \neq \iota} A_i \cdot \boldsymbol{x_{i,\ell}} \\ &\quad = Y \cdot \boldsymbol{x_{\iota,\ell}} - \boldsymbol{u''_\ell} = \boldsymbol{u_\ell}. \end{aligned}$$

$$\begin{aligned} &\sum_{i \in [N]} B_i \cdot \boldsymbol{x_{i,\ell}} - D \cdot \boldsymbol{e_\ell} = B_\iota \cdot \boldsymbol{x_{\iota,\ell}} - D \cdot \boldsymbol{e_\ell} + \sum_{i \neq \iota} B_i \cdot \boldsymbol{x_{i,\ell}} \\ &\quad = \sum_{i \in [N]} \big(U_2 \cdot F_i \cdot G^{-1}(Y) \cdot \boldsymbol{x_{i,\ell}} + \delta_i(\iota) \cdot G_2 \cdot G^{-1}(Y) \cdot \boldsymbol{x_{i,\ell}}\big) - (S \cdot C + E_2) \cdot \boldsymbol{e_\ell} \\ &\quad = \sum_{i \in [N]} (S \cdot U_1 + E_1) \cdot F_i \cdot G^{-1}(Y) \cdot \boldsymbol{x_{i,\ell}} + Y_2 \cdot \boldsymbol{x_{\iota,\ell}} - (S \cdot C + E_2) \cdot \boldsymbol{e_\ell} \\ &\quad = \boldsymbol{v'_\ell} + \boldsymbol{v''_\ell} + S \cdot \Big(\sum_{i \in [N]} U_1 \cdot F_i \cdot G^{-1}(Y) \cdot \boldsymbol{x_{i,\ell}} - C \cdot \boldsymbol{e_\ell} \Big) \end{aligned}$$

$$+ \sum_{i\in[N]} E_1 \cdot F_i \cdot G^{-1}(Y) \cdot \boldsymbol{x_{i,\ell}} - E_2 \cdot \boldsymbol{e_\ell}$$
$$= \boldsymbol{v'_\ell} + S \cdot \boldsymbol{u''_\ell} + S \cdot \Big(U_1 \cdot F_\iota \cdot G^{-1}(Y) \cdot \boldsymbol{x_{\iota,\ell}} + \sum_{i\neq\iota} A_i \cdot \boldsymbol{x_{i,\ell}} - \boldsymbol{\widetilde{u}_\ell}\Big)$$
$$+ \sum_{i\in[N]} E_1 \cdot F_i \cdot G^{-1}(Y) \cdot \boldsymbol{x_{i,\ell}} - E_2 \cdot \boldsymbol{e_\ell}$$
$$= \boldsymbol{v'_\ell} + \sum_{i\in[N]} E_1 \cdot F_i \cdot G^{-1}(Y) \cdot \boldsymbol{x_{i,\ell}} - E_2 \cdot \boldsymbol{e_\ell}.$$

If Γ is an upper-bound on $\|F_i\|_\infty$ for every $j \in [N]$, Lemma 5.3.6 implies a polynomial $c(\lambda)$ such that for every $\ell \in [L]$, with overwhelming probability,

$$\left\|\sum_{i\in[N]} E_1 \cdot F_i \cdot G^{-1}(Y) \cdot \boldsymbol{x_{i,\ell}} - E_2 \cdot \boldsymbol{e_\ell}\right\|_\infty$$
$$\leq M^{\frac{5}{2}}\sqrt{M+W}(K+V)t \cdot s_0 \cdot \Gamma \cdot \alpha q + W^{\frac{3}{2}}\sqrt{M+W}s_1 \cdot \alpha q$$
$$\leq c \cdot \Gamma \cdot \alpha q.$$

If we compute δ_i as in Protocol 5.5.1, then for every $i \in [N]$,

$$\|F_i\|_\infty \leq ((K+V)\cdot t)^{\lceil \log N \rceil}$$

Since N, K, V and t are polynomial quantities in λ, we have $\|F_i\|_\infty \leq 2^{O(\log^2 \lambda)}$.

Hybrid $\mathcal{H}_9$. In this hybrid, we changed the distribution of $(\boldsymbol{v'_\ell})_{\ell\in[L]}$. Specifically, for every $\ell \in [L]$, first, we sample $\boldsymbol{v_\ell} \xleftarrow{\$} \mathbb{Z}_p^V$ and then, we set $\boldsymbol{v'_\ell}$ to be a random element in $\mathbb{Z}_q^V$ such that $\lceil \boldsymbol{v'_\ell} + \boldsymbol{z} \rfloor_p = \boldsymbol{v_\ell}$ for every $\boldsymbol{z} \in \mathbb{Z}^V$ having $\|\boldsymbol{z}\|_\infty \leq c \cdot \Gamma \cdot \alpha \cdot q$. This hybrid is statistically indistinguishable from $\mathcal{H}_8$. Indeed, since $\alpha(\lambda), 1/q(\lambda) \leq \mathsf{negl}(\lambda)$, the statistical distance between the distribution of $\boldsymbol{v'_\ell}$ in this hybrid and in the previous one is upper-bounded by

$$V \cdot p \cdot \frac{2c \cdot \alpha \cdot q + 1}{q} \leq \mathsf{negl}(\lambda).$$

This is because each entry of $\boldsymbol{v'_\ell}$ is now uniformly distributed over a set with $q - p \cdot (2c \cdot \alpha \cdot q + 1)$ (i.e. all the elements in $\mathbb{Z}_q$ except those that have distance smaller than $c \cdot \alpha \cdot q$ from $q/4$ and $(3/4)q$). $\mathcal{H}_9$ corresponds to the ideal execution of the protocol. The simulation strategy is sketched in simulator 5.5.6.

Simulator 5.5.6. One-Round One-Query CTE from LWE

1. Receive the output from the functionality and interpret it as a vector $(\boldsymbol{u_1}, \dots, \boldsymbol{u_L}, \boldsymbol{v_1}, \dots, \boldsymbol{v_L})$ where, for every $\ell \in [L]$, $\boldsymbol{u_\ell} \in \mathbb{Z}_q^K$ and $\boldsymbol{v_\ell} \in \mathbb{Z}_p^V$.
2. $\forall \ell \in [L]$, sample a random $\boldsymbol{v'_\ell} \in \mathbb{Z}_q^V$ such that $\lceil \boldsymbol{v'_\ell} + \boldsymbol{z} \rfloor_p = \boldsymbol{v_\ell}$ for every $\boldsymbol{z} \in \mathbb{Z}^V$ having $\|\boldsymbol{z}\|_\infty \leq c \cdot \Gamma \cdot \alpha \cdot q$.
3. $S \xleftarrow{\$} \mathbb{Z}_q^{V\times K}$
4. $\forall \ell \in [L]$, $\boldsymbol{u''_\ell} \xleftarrow{\$} \mathbb{Z}_q^K$
5. $\forall \ell \in [L]$, $\boldsymbol{v''_\ell} \leftarrow S \cdot \boldsymbol{u''_\ell}$
6. $\forall \ell \in [L]$, $\boldsymbol{w_\ell} \leftarrow (\boldsymbol{u_\ell} + \boldsymbol{u''_\ell} \parallel \boldsymbol{v'_\ell} + \boldsymbol{v''_\ell})$
7. $(Y, T') \xleftarrow{\$} \mathsf{TrapGen}(1^{K+V}, 1^M, q)$
8. $\forall \ell \in [L]$, $\boldsymbol{x_\ell} \xleftarrow{\$} \mathsf{PreSample}(T', \boldsymbol{w_\ell}, s_0)$

```
9. Take the first honest party $P_\iota$ activated by the adversary and send $(\boldsymbol{x_\ell})_{\ell\in[L]}$ on its behalf.
10. For any other honest party $P_i$, send $\boldsymbol{x_{i,\ell}} \xleftarrow{\$} \chi_0^M$ for every $\ell \in [L]$
11. When any honest party $P_i$ is corrupted, provide the adversary with $\mathsf{Explain}_{\chi_0^M}(1^\lambda, \boldsymbol{x_{i,\ell}})$ for every $\ell \in [L]$.
12. After all parties have sent their messages, for any $i \in [N]$ and $\ell \in [L]$ such that $\|\boldsymbol{x_{i,\ell}}\|_\infty > \sqrt{K} \cdot s_0$, set $\boldsymbol{x_{i,\ell}} \leftarrow \mathbf{0}$.
13. $U_1 \xleftarrow{\$} \mathbb{Z}_q^{K\times M}$
14. $(C,T) \xleftarrow{\$} \mathsf{TrapGen}(1^K, 1^W, q)$
15. $E_1 \xleftarrow{\$} \chi_2^{V\times M}$
16. $E_2 \xleftarrow{\$} \chi_2^{V\times W}$
17. $U_2 \leftarrow S \cdot U_1 + E_1$
18. $D \leftarrow S \cdot C + E_2$
19. $U \leftarrow (U_1^\intercal \parallel U_2^\intercal)$
20. $\forall j \in [\lceil \log N \rceil] : F_j \xleftarrow{\$} \mathbb{Z}_2^{M\times(K+V)\cdot t}$
21. $\forall j \in [\lceil \log N \rceil] : X_j \leftarrow U \cdot R_j + \iota_j \cdot G$
22. $\forall i \in [N] : Z_i \leftarrow \mathsf{Eval}(\delta_i, X_1, \dots, X_{\lceil \log N \rceil}) \cdot G^{-1}(Y)$
23. For every $i \in [N]$, let $A_i \in \mathbb{Z}_q^{K\times M}$ consist of the first $K$ rows of $Z_i$. Let $B_i \in \mathbb{Z}_q^{V\times M}$ consist of the last $V$ rows of $Z_i$.
24. $F_\iota \leftarrow \mathsf{FullEval}(\delta_\iota, X_1, \dots, X_{\lceil \log N \rceil}, R_1, \dots, R_{\lceil \log N \rceil}, \iota)$
25. $\forall \ell \in [L]$, $\tilde{\boldsymbol{u}}_{\boldsymbol{\ell}} \leftarrow \boldsymbol{u}''_{\boldsymbol{\ell}} + \sum_{i\neq\iota} A_i \cdot \boldsymbol{x_{i,\ell}} + U_1 \cdot F_\iota \cdot G^{-1}(Y) \cdot \boldsymbol{x_{\iota,\ell}}$
26. $\forall \ell \in [L]$, $\boldsymbol{e_\ell} \xleftarrow{\$} \mathsf{PreSample}(T, \tilde{\boldsymbol{u}}_{\boldsymbol{\ell}}, s_1)$
27. Send $(X_j)_{j\in[\lceil \log N \rceil]}$, $Y$ $C$, $D$ and $(\mathsf{Explain}_{\chi_1^W}(1^\lambda, \boldsymbol{e_\ell}))_{\ell\in[L]}$ on behalf of $\mathcal{F}^n_{\mathsf{Coin}}$.
```

We now analyse our protocol's stretch. The number of seed bits is

$$n = O\Big(\log N \cdot (K+V) \cdot M \cdot t + (K+V) \cdot W \cdot t + L \cdot W \cdot \lambda^2 + L \cdot W \cdot \lambda \cdot \log s_1\Big).$$

If $s_1 = \sqrt{K \cdot \log q} \cdot \log K$, $M = \beta \cdot (K+V) \cdot \log q$ and $W = \beta \cdot K \cdot \log q$, then

$$n = O\Big(\log N \cdot (K+V)^2 \cdot t^2 + L \cdot K \cdot t \cdot (\lambda^2 + \lambda \cdot \log K + \lambda \cdot \log t)\Big).$$

Now, t, K are polynomial quantities in λ, so $\log t, \log K \in O(\log \lambda)$, and thus

$$n = O\Big(\log N \cdot (K+V)^2 \cdot t^2 + L \cdot K \cdot t \cdot \lambda^2\Big).$$

The number of coins produced by the protocol is $L \cdot (K \cdot t + V)$, which implies that if we pick $V = (\eta \cdot \lambda^2 - 1) \cdot K \cdot t$ and $L = \eta^2 \cdot t^3 \cdot \lambda^2 \cdot \log N \cdot K$, then the multiplicative stretch of our construction becomes $\Omega(\eta)$. □

Corollary 5.5.7. Under the hardness of LWE with a subexponential modulus-to-noise ratio, for any polynomial function $N(\lambda)$, there exists a polynomial $L(\lambda)$ and a computational explainable extractor for the class of entropy sources $\mathcal{S}$, such that for every $S \in \mathcal{S}$, there exist $i \in [N]$ and a PPT algorithm $\mathcal{M}$ such that the source S can be sampled as follows:

1. $\boldsymbol{x_i} \xleftarrow{\$} \{0,1\}^{L(\lambda)}$
2. $((\boldsymbol{x_j})_{j \neq i}, \mathsf{aux}) \xleftarrow{\$} \mathcal{M}(1^\lambda, \boldsymbol{x_i})$
3. Output $(\boldsymbol{x_1}, \ldots, \boldsymbol{x_N}), \mathsf{aux}$.

5.6 One-Round, $(1+\epsilon)$-Sample Coin Tossing Extension from Hidden Subgroups

In this section, we present our construction for one-round one-query CTE with universally composable security, from a variety of standard public-key assumptions. We previously overviewed this construction in Section 5.2.3, and invite the reader to review the basic definitions in Section 5.3.1. We begin by defining the *Hidden Subgroup Framework*, and presenting a functionality to generate the trusted setup for an instance. Next, we give our CTE protocol from hidden subgroups, after which we present a security theorem, and then prove the theorem. Finally, in Section 5.6.1 we prove that the framework can be instantiated from the DDH assumption in any group, in Section 5.6.2 from Paillier Encryption, and in Section 5.6.3 from the hidden subgroup membership assumption in class groups.

Definition 5.6.1 (Hidden Subgroup Framework for CTE). The hidden subgroup framework for coin tossing extension consists of a tuple of PPT algorithms $(m, n, \mathsf{Gen}, \mathsf{Uniform}, \mathsf{SubSample}, \mathsf{Add}, \mathsf{Convert}, \mathsf{Explain})$ with the following syntax

- m and n are polynomial functions in the security parameter.
- Gen takes as input the security parameter 1^λ. It outputs the description of a finite abelian group $(G, \cdot)$, the description of a subgroup H, a positive integer R, and auxiliary information aux.
- $\mathsf{Uniform}$ takes as input aux and outputs an element in G.
- $\mathsf{SubSample}$ is deterministic and takes as input aux along with a binary string $\boldsymbol{r} \in \{0,1\}^*$. The output is an element in the subgroup H.
- Add takes as input aux, strings $\boldsymbol{r_1}, \ldots, \boldsymbol{r_\ell} \in \{0,1\}^R$ for any $\ell \in \mathbb{N}$ and $\boldsymbol{r'} \in \{0,1\}^{n(\lambda)}$, the output is another string $\boldsymbol{s} \in \{0,1\}^{n(\lambda)}$.
- $\mathsf{Convert}$ is deterministic and takes as input a group element $g \in G$ and aux. The output is an element in $\{0,1\}^{m(\lambda)}$.
- $\mathsf{Explain}$ takes as input an element $\boldsymbol{x} \in \{0,1\}^{m(\lambda)}$ along with aux, the output is an element $g \in G$.

We require the following properties:

1. **(Samplability of Uniform Distribution).** The following distributions are statistically close

$$\left\{ u, G, H, R, \mathsf{aux} \;\middle|\; (G, H, R, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda),\ u \xleftarrow{\$} \mathsf{Uniform}(\mathsf{aux}) \right\}$$
$$\left\{ u, G, H, R, \mathsf{aux} \;\middle|\; (G, H, R, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda),\ u \xleftarrow{\$} G \right\}$$

2. **(Hidden Subgroup).** The following distributions are computationally indistinguishable

$$\left\{ (u, G, H, R, \mathsf{aux}) \middle| \begin{array}{l} (G, H, R, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \\ u \xleftarrow{\$} \mathsf{Uniform}(\mathsf{aux}) \end{array} \right\}$$

$$\left\{ (u, G, H, R, \mathsf{aux}) \middle| \begin{array}{l} (G, H, R, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \\ \boldsymbol{r} \xleftarrow{\$} \{0,1\}^R \\ u \leftarrow \mathsf{SubSample}(\mathsf{aux}, \boldsymbol{r}) \end{array} \right\}$$

3. **(Correctness of Addition).** For every PPT adversary $\mathcal{A}$, we have

$$\Pr\left[\begin{array}{c} g' \cdot \prod_{i \in [\ell]} g_i \neq h, \\ \text{and} \\ \forall j \in [\ell] : \quad \boldsymbol{r_j} \in \{0,1\}^R \end{array} \middle| \begin{array}{l} (G, H, R, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \\ (\boldsymbol{r_1}, \ldots, \boldsymbol{r_\ell}) \xleftarrow{\$} \mathcal{A}(1^\lambda, G, H, R, \mathsf{aux}) \\ \boldsymbol{r'} \xleftarrow{\$} \{0,1\}^{n(\lambda)} \\ \forall i \in [\ell] : \quad g_i \leftarrow \mathsf{SubSample}(\mathsf{aux}, \boldsymbol{r_i}) \\ g' \leftarrow \mathsf{SubSample}(\mathsf{aux}, \boldsymbol{r'}) \\ \boldsymbol{s} \xleftarrow{\$} \mathsf{Add}(\boldsymbol{r_1}, \ldots, \boldsymbol{r_\ell}, \boldsymbol{r'}) \\ h \leftarrow \mathsf{SubSample}(\mathsf{aux}, \boldsymbol{s}) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

4. **(Security of Flooding).** For every PPT adversary $\mathcal{A}$ that outputs $\boldsymbol{r_1}, \ldots, \boldsymbol{r_\ell} \in \{0,1\}^R$ for some $\ell \in \mathbb{N}$ and its internal state ϕ, the following distributions are computationally indistinguishable

$$\left\{ \boldsymbol{s}, \phi \middle| \begin{array}{l} (G, H, R, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \\ (\boldsymbol{r_1}, \ldots, \boldsymbol{r_\ell}, \phi) \xleftarrow{\$} \mathcal{A}(1^\lambda, G, H, R, \mathsf{aux}) \\ \boldsymbol{r'} \xleftarrow{\$} \{0,1\}^{n(\lambda)} \\ \boldsymbol{s} \xleftarrow{\$} \mathsf{Add}(\boldsymbol{r_1}, \ldots, \boldsymbol{r_\ell}, \boldsymbol{r'}) \end{array} \right\}$$

$$\left\{ \boldsymbol{s}, \phi \middle| \begin{array}{l} (G, H, R, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \\ (\boldsymbol{r_1}, \ldots, \boldsymbol{r_\ell}, \phi) \xleftarrow{\$} \mathcal{A}(1^\lambda, G, H, R, \mathsf{aux}) \\ \boldsymbol{s} \xleftarrow{\$} \{0,1\}^{n(\lambda)} \end{array} \right\}$$

5. **(Explainability).** The following distributions are computationally indistinguishable

$$\left\{ (\boldsymbol{x}, g, G, H, R, \mathsf{aux}) \middle| \begin{array}{l} (G, H, R, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \\ g \xleftarrow{\$} \mathsf{Uniform}(\mathsf{aux}) \\ \boldsymbol{x} \leftarrow \mathsf{Convert}(g, \mathsf{aux}) \end{array} \right\}$$

$$\left\{ (\boldsymbol{x}, g, G, H, R, \mathsf{aux}) \middle| \begin{array}{l} (G, H, R, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda) \\ \boldsymbol{x} \xleftarrow{\$} \{0,1\}^{m(\lambda)} \\ g \xleftarrow{\$} \mathsf{Explain}(\boldsymbol{x}, \mathsf{aux}) \end{array} \right\}$$

Functionality 5.6.2. Setup for the Hidden Subgroup Framework

Let NIZK be a simulation-extractable NIZK for the relation

$$\mathcal{R} := \left\{ \begin{array}{c} x := (g, R, \mathsf{aux}) \\ w := \boldsymbol{r} \end{array} \middle| \begin{array}{l} \boldsymbol{r} \in \{0,1\}^R \\ g = \mathsf{SubSample}(\mathsf{aux}, \boldsymbol{r}) \end{array} \right\}$$

Initialisation: Upon activation, perform the following operations

1. $(G, H, R, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$
2. $\sigma \xleftarrow{\$} \mathsf{NIZK.Setup}(1^\lambda)$
3. Output $(\sigma, G, H, R, \mathsf{aux})$ to all parties.

Protocol 5.6.3. One-Round CTE from Hidden Subgroups

Let NIZK be the simulation-extractable NIZK for the relation $\mathcal{R}$ used in $\mathcal{F}_{\mathsf{HS\text{-}Setup}}$ (see Functionality 5.6.2).

Initialisation: Each party P_i calls $\mathcal{F}_{\mathsf{HS\text{-}Setup}}$ and receives $(\sigma, G, H, R, \mathsf{aux})$ as a response.

Sample: Each party P_i performs the following operations.

1. $\boldsymbol{r_i} \xleftarrow{\$} \{0,1\}^R$.
2. $g_i \leftarrow \mathsf{SubSample}(\mathsf{aux}, \boldsymbol{r_i})$.
3. $\pi_i \xleftarrow{\$} \mathsf{NIZK.Prove}\big(\sigma, (g_i, R, \mathsf{aux}), \boldsymbol{r_i}\big)$.
4. Broadcast (g_i, π_i) and receive (g_j, π_j) from every other party P_j.
5. $S \leftarrow \{j \in [N] | \mathsf{NIZK.Verify}\big(\sigma, (g_j, R, \mathsf{aux}), \pi_j\big) = 1\}$.
6. Receive $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$ from $\mathcal{F}^n_{\mathsf{Coin}}$.
7. $h \leftarrow \mathsf{SubSample}(\mathsf{aux}, \boldsymbol{s})$.
8. $g \leftarrow h \cdot \prod_{i \in S} g_i$.
9. Output $\mathsf{Convert}(g, \mathsf{aux})$

The following theorem essentially formalizes Theorem 5.2.4.

Theorem 5.6.4. If $(m, n, \mathsf{Gen}, \mathsf{Uniform}, \mathsf{SubSample}, \mathsf{Add}, \mathsf{Convert}, \mathsf{Explain})$ is an instantiation of the hidden subgroup framework of Definition 5.6.1 and NIZK is a simulation-extractable NIZK for the relation $\mathcal{R}$ (see Functionality 5.6.2), then Protocol 5.6.3 UC-realizes $\mathcal{F}^m_{\mathsf{Coin}}$ among N-parties in the presence of a malicious PPT adversary statically corrupting up to $N-1$ parties, in the $(\mathcal{F}^n_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{HS\text{-}Setup}})$-hybrid model (see Functionality 5.6.2). The round complexity of the protocol is $r = 1$, the sampling complexity is $t = 1$, and the additive stretch is $m(\lambda) - n(\lambda)$.

Proof. Consider Simulator 5.6.5.

Simulator 5.6.5. Simulator for CTE from Hidden Subgroups

Initialisation:

1. $(G, H, R, \mathsf{aux}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$
2. $(\sigma, \tau) \xleftarrow{\$} \mathsf{NIZK.Sim}_1(1^\lambda)$
3. Provide the adversary with $(\sigma, G, H, R, \mathsf{aux})$

Sample: Let P_ι be a fixed honest party. The simulator performs the following operations:

1. For every honest P_i except P_ι, compute $\boldsymbol{r_i}$ and $(g_i \pi_i)$ as in Protocol 5.6.3.

2. Obtain $\boldsymbol{x} \in \{0,1\}^{m(\lambda)}$ from the functionality.
3. $u \xleftarrow{\$} \mathsf{Explain}(\boldsymbol{x}, \mathsf{aux})$
4. $\boldsymbol{r'} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
5. $g' \leftarrow \mathsf{SubSample}(\mathsf{aux}, \boldsymbol{r'})$
6. $g_\iota \leftarrow g' \cdot u$
7. $\pi_\iota \xleftarrow{\$} \mathsf{NIZK.Sim}_2\big(\tau, (g_\iota, R, \mathsf{aux})\big)$
8. Send all $(g_i, \pi_i)_{i \in H}$ to the adversary.
9. Receive $(g_j, \pi_j)_{j \notin H}$ from the adversary.
10. Let S be the subset of $j \in [N]$ such that $\mathsf{NIZK.Verify}\big(\sigma, (g_j, R, \mathsf{aux}), \pi_j\big) = 1$.
11. For every $j \in S \setminus H$ compute $\boldsymbol{r_j} \leftarrow \mathsf{NIZK.Extract}\big(\tau, (g_j, R, \mathsf{aux}), \pi_j\big)$.
12. If there exists a $j \in S$ such that $\boldsymbol{r_j} = \bot$, output $\bot$.
13. Otherwise, provide $\boldsymbol{s} \xleftarrow{\$} \mathsf{Add}(\mathsf{aux}, (\boldsymbol{r_j})_{j \in S}, \boldsymbol{r'})$ to the adversary on behalf of $\mathcal{F}^n_{\mathsf{Coin}}$.

We show that no PPT adversary can distinguish between the real protocol and the interaction between functionality and simulator. We proceed by considering the following sequence of indistinguishable hybrids.

Hybrid $\mathcal{H}_0$. This corresponds to Protocol 5.6.3.

Hybrid $\mathcal{H}_1$. In this hybrid, we change the CRS. Specifically, instead of generating σ using $\mathsf{NIZK.Setup}(1^\lambda)$, we compute $(\sigma, \tau) \xleftarrow{\$} \mathsf{NIZK.Sim}_1(1^\lambda)$. This hybrid is indistinguishable from the previous one thanks to the security of the extractable NIZK.

Hybrid $\mathcal{H}_2$. In this hybrid, we pick a honest party P_ι and we simulate its proofs π_ι using the trapdoor τ. Once again, this hybrid is indistinguishable from $\mathcal{H}_1$ thanks to zero-knowledge of NIZK.

Hybrid $\mathcal{H}_3$. In this hybrid, we try to extract the witnesses $\boldsymbol{r_j}$ from the NIZKs of all the corrupted parties. If the extraction fails, despite the NIZK verifies, we output $\bot$. This hybrid is indistinguishable from $\mathcal{H}_2$ thanks to the simulation-extractability of the NIZK.

Hybrid $\mathcal{H}_4$. In this hybrid, we sample $u \xleftarrow{\$} \mathsf{Uniform}(\mathsf{aux})$ and we modify the message of P_ι. Specifically, instead of sending $g_\iota \leftarrow \mathsf{SubSample}(\mathsf{aux}, \boldsymbol{r_\iota})$ where $\boldsymbol{r_\iota} \xleftarrow{\$} \{0,1\}^R$, we send $g_\iota \leftarrow u \cdot v$ where $v \leftarrow \mathsf{SubSample}(\mathsf{aux}, \boldsymbol{r'})$ and $\boldsymbol{r'} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$. This hybrid is indistinguishable from $\mathcal{H}_3$ due to the first and the second property of the framework.

Hybrid $\mathcal{H}_5$. In this hybrid, instead of sampling $\boldsymbol{s}$ at random in $\{0,1\}^{n(\lambda)}$, we compute it as $\boldsymbol{s} \leftarrow \mathsf{Add}(\mathsf{aux}, (\boldsymbol{r_j})_{j \in S}, \boldsymbol{r'})$. This hybrid is indistinguishable from $\mathcal{H}_4$ by the first and fourth properties of the framework. Indeed, g_ι leaks no information about $\boldsymbol{r'}$ as u is statistically close to random in G. Notice that, by the third property of the framework, the output of the protocol is now $\mathsf{Convert}(u, \mathsf{aux})$.

Hybrid $\mathcal{H}_6$. In this hybrid, instead of sampling u using $\mathsf{Uniform}$, we sample a random element $\boldsymbol{x} \xleftarrow{\$} \{0,1\}^{m(\lambda)}$ and we set $u \xleftarrow{\$} \mathsf{Explain}(\boldsymbol{x}, \mathsf{aux})$. This hybrid is indistinguishable from $\mathcal{H}_5$ thanks to the last property of the framework. This hybrid corresponds to the ideal world. □

5.6.1 The Hidden Subgroup Framework from DDH

Algorithm 5.6.6. Hidden Subgroup Framework from DDH

Let $(G_\lambda, \cdot)_{\lambda\in\mathbb{N}}$ be a sequence of cyclic groups where G_λ has prime order $p(\lambda)$. Let g_λ be a generator for G_λ. Suppose that $p(\lambda) = 2^{O(\lambda)}$. Let μ_G and μ_L be the uniform distributions over G_λ and $\{0,1\}^{L(\lambda)}$ where $L(\lambda)$ is a polynomial respectively. Suppose that μ_L is μ_G-explainable. Let $(\mathsf{Convert}_L, \mathsf{Explain}_L)$ be the conversion algorithm and explaining algorithm that guarantee this. Let $m(\lambda)$ be $2L(\lambda)$. Let $n(\lambda)$ be $\lambda + \log p(\lambda)$.

```
Gen(1^λ):
1. $r \xleftarrow{\$} [p(\lambda)]$
2. $g_1 \leftarrow g_\lambda$
3. $g_2 \leftarrow g_\lambda^r$
4. $H_\lambda \leftarrow \{(h_1,h_2) | \exists r \in [p(\lambda)] \text{ s.t. } h_1 = g_1^r, h_2 = g_2^r\}$
5. Output $\big(G_\lambda \times G_\lambda, H_\lambda, n(\lambda), \mathsf{aux} = (g_1, g_2)\big)$

Uniform($\mathsf{aux} = (g_1, g_2)$):
1. $r \xleftarrow{\$} [p(\lambda)]$
2. Output $g_\lambda^r$.

SubSample($\mathsf{aux} = (g_1, g_2), \boldsymbol{r}$):
1. Let $R$ be the length of $\boldsymbol{r}$
2. $t \leftarrow (\sum_{i=1}^{R} r[i] \cdot 2^{i-1}) \bmod p(\lambda)$
3. $h_1 \leftarrow g_1^t$
4. $h_2 \leftarrow g_2^t$
5. Output $(h_1, h_2)$

Add($\boldsymbol{r_1}, \ldots, \boldsymbol{r_\ell}, \boldsymbol{r'}$):
1. $\forall i \in [\ell] : t_i \leftarrow (\sum_{j=1}^{n(\lambda)} r_i[j] \cdot 2^{j-1}) \bmod p(\lambda)$
2. $t' \leftarrow (\sum_{j=1}^{n(\lambda)} r'[j] \cdot 2^{j-1}) \bmod p(\lambda)$
3. $z \leftarrow (t' + \sum_{i\in[\ell]} t_i) \bmod p(\lambda)$.
4. Output a random $\boldsymbol{s} \in \{0,1\}^{n(\lambda)}$ such that $\sum_{j=1}^{n(\lambda)} s[j] \cdot 2^{j-1} \equiv z \bmod p(\lambda)$.

Convert($u = (u_1, u_2), \mathsf{aux} = (g_1, g_2)$):
1. Output $\mathsf{Convert}_L(1^\lambda, u_1)$ and $\mathsf{Convert}_L(1^\lambda, u_2)$.

Explain($\boldsymbol{x}, \mathsf{aux} = (g_1, g_2)$):
1. Output $\mathsf{Explain}_L(1^\lambda, \boldsymbol{x_1})$ and $\mathsf{Explain}_L(1^\lambda, \boldsymbol{x_2})$ where $\boldsymbol{x} = (\boldsymbol{x_1} \parallel \boldsymbol{x_2})$ and $\boldsymbol{x_1}, \boldsymbol{x_2} \in \{0,1\}^{L(\lambda)}$.
```

Theorem 5.6.7. Let $(G_\lambda, \cdot)_{\lambda\in\mathbb{N}}$ be a sequence of cyclic groups where G_λ has prime order $p(\lambda)$. Let g_λ be a generator for G_λ. Let μ_G and μ_L respectively be the uniform distributions over G_λ and $\{0,1\}^{L(\lambda)}$, where $L(\lambda)$ is a polynomial. Suppose that μ_L is μ_G-explainable. If the decisional Diffie-Hellman assumption holds in $(G_\lambda, \cdot)_{\lambda \in \mathbb{N}}$, then Algorithm 5.6.6 is an instantiation of the hidden subgroup framework of Definition 5.6.1.

Proof. We immediately see that the first property holds with perfect indistinguishability. The correctness of the addition is always satisfied. The security of flooding is instead guaranteed by the fact that the uniform distribution over $[p(\lambda)]$ is statistically close to the distribution obtained by reducing a random element in $[2^{\lambda+\log p}]$ modulo $p(\lambda)$. The fifth property is guaranteed by the μ_G-explainability of μ_L. It remains to show the second property. It is easy to show that it is an immediate consequence of the DDH assumption. $\square$

5.6.2 The Hidden Subgroup Framework from Pallier Encryption.

Algorithm 5.6.8. Hidden Subgroup Framework from QR and DCR

```
Let $m(\lambda)$ be $2^{\Theta(\log \lambda)}$. Let $n(\lambda)$ be $5\lambda/2 + m(\lambda)/2$.

$\mathsf{Gen}(1^\lambda)$:
1. Sample random $(m(\lambda)+\lambda)/4$-bit safe-primes $p, q$.
2. $N \leftarrow p \cdot q$
3. $H \leftarrow \{r^{2N} | r \in \mathbb{Z}^*_{N^2}\}$
4. $h \xleftarrow{\$} H$
5. Output $\left(\mathbb{Z}^*_{N^2}, H, R(\lambda) := 3\lambda/2 + m(\lambda)/2, \mathsf{aux} := h\right)$

$\mathsf{Uniform}(\mathsf{aux} = h)$:
1. $r \xleftarrow{\$} \mathbb{Z}^*_{N^2}$
2. Output $r$.

$\mathsf{SubSample}(\mathsf{aux} = h, \boldsymbol{r})$:
1. Let $R$ be the length of $\boldsymbol{r}$
2. $t \leftarrow (\sum_{i=1}^{R} r[i] \cdot 2^{i-1})$
3. Output $h^t$

$\mathsf{Add}(\boldsymbol{r_1}, \ldots, \boldsymbol{r_\ell}, \boldsymbol{r'})$:
1. $\forall i \in [\ell] : t_i \leftarrow \sum_{j=1}^{L(\lambda)} r_i[j] \cdot 2^{j-1}$
2. $t' \leftarrow \sum_{j=1}^{n(\lambda)} r'[j] \cdot 2^{j-1}$
3. $z \leftarrow t' + \sum_{i \in [\ell]} t_i$
4. Output the bit representation of $z$.

$\mathsf{Convert}(u, \mathsf{aux} = h)$:
1. Output $u \bmod 2^{m(\lambda)}$ in binary notation.
```

```
Explain($\boldsymbol{x}$, aux $= h$):
1. $z \leftarrow \sum_{j=1}^{m(\lambda)} x[j] \cdot 2^{j-1}$
2. Output a random $u \in \mathbb{Z}_{N^2}^*$ such that $u \bmod 2^{m(\lambda)} = x$.
```

Theorem 5.6.9. If the quadratic residuosity and decisional composite residuosity assumptions hold over the Paillier group, then Algorithm 5.6.8 is an instantiation of the hidden subgroup framework of Definition 5.6.1.

Proof. We immediately see that the first property holds with perfect indistinguishability. The correctness of the addition is always satisfied. The security of flooding is instead guaranteed by the fact that the uniform distribution over $[2^{n(\lambda)}]$ is statistically close to the distribution obtained by shifting a random element in $[2^{n(\lambda)}]$ by $\sum_{i\in[\ell]} t_i$. This is because ℓ is polynomial in λ and each t_i belongs to an interval 2^λ times smaller than $2^{n(\lambda)}$. The fifth property is guaranteed by the fract that the uniform distribution over $\mathbb{Z}_{N^2}^*$ is statistically close to the uniform distribution over $\mathbb{Z}_{N^2}$. Moreover, $m(\lambda)$ is roughly $\log N^2 - \lambda$.

It remains to show the second property. Since p and q are large, random, safe-primes, there exist distinct primes p', q' such that $p = 2p' + 1$ and $q = 2q' + 1$. Therefore, it must be that $\mathbb{Z}_{N^2}^*$ is isomorphic to $\mathbb{Z}_N \times \mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_{N'}$ where $N' = p' \cdot q'$. We notice that $R(\lambda)$ is roughly $\log N' + \lambda$. Furthermore, we observe that h belongs to the subgroup of $\mathbb{Z}_{N^2}^*$ isomorphic to $\mathbb{Z}_{N'}$. We conclude that SubSample outputs values that are statistically close to uniformly random elements of H. We show that, under QR and DCR, the uniform distribution over H is indistinguishable from the uniform distribution over $\mathbb{Z}_{N^2}^*$. We do this by means of a series of hybrids.

Hybrid $\mathcal{H}_0$. In this hybrid, we provide the adversary with the RSA modulo N and $r \overset{\$}{\leftarrow} \mathbb{Z}_{N^2}^*$.

Hybrid $\mathcal{H}_1$. In this hybrid, we provide the adversary with the RSA modulo N and $r^2 \bmod N^2$ where $r \overset{\$}{\leftarrow} \mathbb{Z}_{N^2}^*$. This hybrid is indistinguishable from $\mathcal{H}_0$ due to the QR assumption.

Hybrid $\mathcal{H}_2$. In this hybrid, we provide the adversary with the RSA modulo N and $(r^N)^2 \bmod N^2$ where $r \overset{\$}{\leftarrow} \mathbb{Z}_{N^2}^*$. This hybrid is indistinguishable from $\mathcal{H}_1$ due to the DCR assumption. Observe that now we provide the adversary with a random value in H. This ends the proof. □

5.6.3 The Hidden Subgroup Framework from Class Groups.

Algorithm 5.6.10. Hidden Subgroup Framework from Class Groups

```
Let $L(\lambda)$ be $2^{\Theta(\log\lambda)}$. Let $n(\lambda)$ be $\lambda + R(\lambda)$.

Gen($1^\lambda$):
1. $(G, H, F, h, f, p, \mathsf{aux}') \overset{\$}{\leftarrow} \mathsf{CLGen}(1^\lambda, L(\lambda))$ [a]
2. Output $(G, H, L, \mathsf{aux} := (h, f, p, \mathsf{aux}'))$

Uniform(aux $= (h, f, p, \mathsf{aux}')$):
1. $r \overset{\$}{\leftarrow} [2^{L(\lambda)}]$
2. $s \overset{\$}{\leftarrow} [p]$
3. Output $f^s \cdot h^r$.

SubSample(aux $= (h, f, p, \mathsf{aux}'), \boldsymbol{r}$):
1. Let $R$ be the length of $\boldsymbol{r}$
```

2. $t \leftarrow \sum_{i=1}^{R} r[i] \cdot 2^{i-1}$

3. Output h^t

$\mathsf{Add}(\boldsymbol{r_1}, \dots, \boldsymbol{r_\ell}, \boldsymbol{r'})$:

1. $\forall i \in [\ell] : t_i \leftarrow \sum_{j=1}^{L} r_i[j] \cdot 2^{j-1}$

2. $t' \leftarrow \sum_{j=1}^{n(\lambda)} r'[j] \cdot 2^{j-1}$

3. $z \leftarrow t' + \sum_{i \in [\ell]} t_i$

4. Output the bit representation of z.

[a] $\mathsf{CLGen}(1^\lambda, L)$ generates the parameters of a class groups where the order is upper-bounded by $2^{L-\lambda}$.

Theorem 5.6.11. If the hidden subgroup membership assumption holds over class groups, then $(\mathsf{Gen}, \mathsf{SubSample}, \mathsf{Add}, \mathsf{Uniform})$ in Algorithm 5.6.10 satisfy the first four properties of the hidden subgroup framework of Definition 5.6.1.

Proof. We immediately see that the first property holds with statistical indistinguishability (we recall that $\mathsf{CLGen}(1^\lambda, L)$ outputs an element $h \in H$ of order 2^λ times smaller than 2^L, an element f of order p and $G = \langle h \rangle \times \langle f \rangle$). The correctness of the addition is always satisfied. The security of flooding is instead guaranteed by the fact that the uniform distribution over $[2^{n(\lambda)}]$ is statistically close to the distribution obtained by shifting a random element in $[2^{n(\lambda)}]$ by $\sum_{i \in [\ell]} t_i$. This is because ℓ is polynomial in λ and each t_i belongs to an interval 2^λ times smaller than $2^{n(\lambda)}$. The second property follows immediately from the hidden subgroup assumption over class groups. This ends the proof. □

5.7 $O(N)$-Round, One-Sample Coin Tossing Extension from One-Way Functions

In this section, we present our construction for constant-round one-query CTE with standalone security from one-way functions. We previously overviewed this construction in Section 5.2.3. We begin by presenting the functionality for coin tossing with identifiable abort, then describe our CTE protocol that uses the foregoing functionality to achieve full security, and finally prove security.

Functionality 5.7.1. $\mathcal{F}^m_{\mathsf{Coin+IA}}$. Coin Tossing with Identifiable Abort

Initialisation: On `init` from all parties, the functionality activates.

Sample: On receiving $(\texttt{flip}, \mathsf{sid}, T)$ where $T \subseteq [N]$ from all parties in T, the functionality samples $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{m(\lambda)}$ and sends $(\texttt{sampled}, \mathsf{sid}, \boldsymbol{s})$ to the adversary. If the adversary replies with $(\texttt{abort}, \mathsf{sid}, S)$, where $S \subseteq T$ is a non-empty subset of corrupted players, the functionality outputs $(\texttt{abort}, \mathsf{sid}, S)$ to all honest parties in T. Otherwise, it outputs $(\texttt{coins}, \mathsf{sid}, \boldsymbol{s})$ to all the honest parties in T.

Protocol 5.7.2. CTE from Coin Tossing with IA

Let $G : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$ be a PRG.

Initialisation: The parties send `init` to $\mathcal{F}^m_{\mathsf{Coin+IA}}$ and $\mathcal{F}^n_{\mathsf{Coin}}$.

Sample: Let sid be a session identity.

1. $T \leftarrow [N]$.

2. The parties send $(\mathtt{flip}, \mathsf{sid}, T)$ $\mathcal{F}^m_{\mathsf{Coin+IA}}$ obtaining a value $\boldsymbol{w}$. If $\boldsymbol{w} = (\mathtt{abort}, \mathsf{sid}, S)$ where $S \subset [N]$, the parties set $T \leftarrow T \setminus \{S\}$ and repeat the operations in this step.
3. If $|T| = 1$, the only party in T outputs $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{m(\lambda)}$.
4. Otherwise, the parties call $\mathcal{F}^n_{\mathsf{Coin}}$. Let $\boldsymbol{u}$ be the answer.
5. The parties output $\boldsymbol{s} \leftarrow \boldsymbol{w} \oplus G(\boldsymbol{u})$.

Theorem 5.7.3. Protocol 5.7.2 is a standalone fully-secure N-party coin tossing extension protocol with black-box standalone simulation against a malicious PPT adversary statically corrupting up to $N-1$ parties in the $\mathcal{F}^m_{\mathsf{Coin+IA}}$-hybrid model. The sampling complexity is one, the additive stretch of the protocol is $m(\lambda) - n(\lambda)$, and the number of sequential invocations of $\mathcal{F}^m_{\mathsf{Coin+IA}}$ is bounded by $N-1$.

Proof. We prove the theorem via simulator 5.7.4.

Simulator 5.7.4. CTE from Coin Tossing with IA

1. The simulator obtains $\boldsymbol{s}$ from the functionality.
2. Every time the parties call $\mathcal{F}^m_{\mathsf{Coin+IA}}$, the simulator samples a fresh $\boldsymbol{u} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$ and provides the adversary with $\boldsymbol{w} \leftarrow \boldsymbol{s} \oplus G(\boldsymbol{u})$.
3. When the parties call $\mathcal{F}^n_{\mathsf{Coin}}$, the simulator provides the last $\boldsymbol{u}$ it sampled.

It is easy to see that, under the security of the PRG G, no PPT adversary can distinguish between the real protocol and its simulation. Indeed, the only difference between the two worlds is that in the aborting calls to $\mathcal{F}^m_{\mathsf{Coin+IA}}$, the adversary is not provided with random values but with the actual output of the functionality masked by random PRG images. Under the security of G, the adversary cannot distinguish between the two cases. This ends the proof. $\square$

Corollary 5.2.3. If one-way functions exist, then for any constant number of parties there is a constant-round fully-secure CTE protocol in the plain model, with standalone black-box simulatability against a malicious PPT adversary statically corrupting all parties but one. This construction is black-box in the OWF.

Proof. Goyal et al. [GLOV12] proposed a constant-round protocol for black-box standalone-simulatable coin tossing in the malicious, dishonest majority setting, assuming only black-box use of one-way functions. They claim only security with (non-identifiable) abort in their own work, but a careful inspection of their protocol reveals that all aborts are unanimously traceable to a single party if all communication is performed via a broadcast channel. In other words, their protocol realizes $\mathcal{F}^m_{\mathsf{Coin+IA}}$. Combining this fact with theorem 5.7.3 and the fact that black-box constructions of PRG from OWFs are possible [HILL99] yields the corollary. $\square$

5.8 Lower Bound for Statistical, Black-Box Coin Tossing Extension

In this section, we present our lower bound against constant-round superlogarithmic-stretch black-box simulation-secure CTE. We previously overviewed this construction in Section 5.2.4, and invite the reader to review the basic definitions and common tools we use in Section 5.3.2. We begin with a lemma about the entropy difference between statistically-close distributions, after which we give our main theorem and its proof.

Lemma 5.8.1. Let $h(p)$ be $-(p \cdot \log p + (1-p) \cdot \log(1-p))$ for every $p \in]0,1[$. Let H denote Shannon's entropy. Let (X, Y) and (X', Y') be random variables over a domain of size D. Let the statistical distance between (X, Y) and (X', Y') be ϵ. Then, $\mathsf{H}(Y|X) - \mathsf{H}(Y'|X') \leq 2h(\epsilon) + 2\epsilon \cdot \log |D|$.

Proof. By definition of statistical distance, there is a coupling of (X, Y) and (X', Y') such that, defining Θ, Φ as the indicator functions for $X = X'$ and $(X, Y) = (X', Y')$, $\Pr[\Theta = 1] = \Pr[\Phi = 1] = \epsilon$. We have,

$$\mathsf{H}(Y|X) - \mathsf{H}(Y'|X') = \mathsf{H}(X, Y) - \mathsf{H}(X) - \mathsf{H}(X', Y') + \mathsf{H}(X'). \tag{5.5}$$

Since $\mathsf{H}(X, X') = \mathsf{H}(X) + \mathsf{H}(X'|X) = \mathsf{H}(X') + \mathsf{H}(X|X')$, $\mathsf{H}(X) - \mathsf{H}(X') = \mathsf{H}(X|X') - \mathsf{H}(X'|X)$. We use the argument used to show Fano's inequality to bound $\mathsf{H}(X|X')$. Since Θ is a function of X, X',

$$\begin{aligned}
0 &\leq \mathsf{H}(X|X') \\
&= \mathsf{H}(\Theta, X|X') \\
&\leq \mathsf{H}(\Theta) + \mathsf{H}(X|X', \Theta) \\
&\leq h(\epsilon) + \Pr[\Theta = 0] \cdot \mathsf{H}(X|X', \Theta = 0) + \Pr[\Theta = 1] \cdot \mathsf{H}(X|X', \Theta = 1) \\
&= h(\epsilon) + \Pr[\Theta = 0] \cdot \mathsf{H}(X|X', \Theta = 0) \\
&\leq h(\epsilon) + \epsilon \cdot \log |D|.
\end{aligned}$$

Similarly, bound exists for $\mathsf{H}(X'|X)$; hence $|\mathsf{H}(X) - \mathsf{H}(X')| \leq h(\epsilon) + \epsilon \cdot \log |D|$. Further, using the same argument, $|\mathsf{H}(X, Y) - \mathsf{H}(X', Y')| \leq h(\epsilon) + \epsilon \cdot \log |D|$. Thus, by (5.5), $\mathsf{H}(Y|X) - \mathsf{H}(Y'|X') \leq 2h(\epsilon) + 2\epsilon \cdot \log |D|$. □

Theorem 5.2.6. Every r-round CTE protocol with one call to the seed oracle and black-box standalone statistical simulation security against a rushing semi-malicious adversary who corrupts a majority of parties must have additive stretch in $O(r \cdot \log \lambda)$.

Proof. Due to Theorem 5.4.3, we can assume that the protocol terminates with the call to $\mathcal{F}^n_{\mathsf{Coin}}$. We consider the adversary that, in each round, after receiving the messages of the honest parties, generates the messages of the corrupted players following the protocol. Let U^i_H be the random variable that denotes the messages of the honest parties up to the i-th round (included). Similarly, let U^i_C denote the messages of the corrupted players up to the i-th round (included). We set U^0_H and U^0_C to the empty string. Let $\boldsymbol{u} \in \{0, 1\}^{n(\lambda)}$ be the random coins produced by $\mathcal{F}^n_{\mathsf{Coin}}$. Finally, let $\boldsymbol{s} \in \{0, 1\}^{m(\lambda)}$ be the output of the protocol.

We consider the Shannon entropy diagram of the protocol, a technique previously used by Abram, Obremski, and Scholl [AOS23]. For simplicity, for any random variables (X_0, Y_0) and (X_1, Y_1) parametrised by the security parameter λ, we write $\mathsf{H}(X_0|Y_0) \sim \mathsf{H}(X_1|Y_1)$ if $\mathsf{H}(X_0|Y_0) = \mathsf{H}(X_1|Y_1) + \mathsf{negl}(\lambda)$. In a similar way, for functions $f(\lambda)$ and $g(\lambda)$, we write $f \lesssim g$ if there exists a negligible function $\eta = \mathsf{negl}(\lambda)$ such that $f \leq g + \eta$.

We recall that the statistical distance of $(U^r_H, U^r_C, \boldsymbol{u}, \boldsymbol{s})$ in the ideal and in the real world is smaller than ϵ, where $\epsilon = \mathsf{negl}(\lambda)$ (this is implied by the security of the protocol). We also notice that $h(\epsilon) := -\epsilon \cdot \log \epsilon - (1 - \epsilon) \cdot \log(1 - \epsilon)$ is negligible when ϵ is negligible. Finally, we notice that the probability space of our protocol has size $2^{\lambda^{O(1)}}$. Due to these facts, and Lemma 5.8.1, we conclude that conditional entropies in the ideal world and the real world differ only by a negligible amount.

We start by observing that $\mathsf{H}(\boldsymbol{s}|U^r_H, U^r_C, \boldsymbol{u}) \sim 0$. Indeed, the output of the protocol is, with overwhelming probability, uniquely determined by U^r_H, U^r_C and $\boldsymbol{u}$.

In the real world, $\boldsymbol{u}$ is independent of both U^r_H and U^r_C. We conclude that the mutual information $\mathtt{I}(\boldsymbol{u}; (U^r_H, U^r_C)) \sim 0$. In the real world, since we are considering a honest adversary, we also have that, for every $i \in [r]$, U^i_C is independent of U^i_H conditioned on (U^{i-1}_H, U^{i-1}_C). Moreover, U^1_H and U^1_C are independent. Therefore, $\mathtt{I}(U^i_H; U^i_C|U^{i-1}_H, U^{i-1}_C) \sim 0$ for every $i \in [r]$.

Now, consider the ideal world and let Q be a polynomial upper bound on the running time of the simulator. If we were in the UC model, it is easy to notice that U^i_C is independent of $(U^i_H, \boldsymbol{s})$ conditioned on U^{i-1}_H, U^{i-1}_C. This of course, it is no longer true when the simulator is allowed to rewind the adversary. Let T_i be the random variable denoting the number of times the simulator rewinds the adversary in the i-th round. Then, $T_i \leq Q$, and U^i_C can be thought of as is the T_i'th value in the sequence $\{U^i_C[j]\}_{j \in [Q]}$ where $U^i_C[j]$ is the response of the adversary in j'th rewind. Each $U^i_C[j]$ is independently sampled by the 'honest' adversary according to the protocol instruction, conditioned on the messages up to round $i-1$ being (U_H^{i-1}, U_C^{i-1}). Hence, for each $j \in [Q]$, $\mathtt{I}(U_C^i[j]; (U_H^i, \boldsymbol{s}) | U_H^{i-1}, U_C^{i-1}, \{U_C^i[k]\}_{k\in[j-1]}) = 0$. Then,

$$\begin{aligned}
\mathtt{I}(U_C^i; (U_H^i, \boldsymbol{s}) | U_C^{i-1}, U_H^{i-1}) &= \mathtt{I}(U_C^i[T_i]; (U_H^i, \boldsymbol{s}) | U_C^{i-1}, U_H^{i-1}) \\
&\leq \mathtt{I}((U_C^i[T_i], \{U_C^i[k]\}_{k\in[Q]}, T_i); (U_H^i, \boldsymbol{s}) | U_C^{i-1}, U_H^{i-1}) \\
&\overset{(a)}{=} \mathtt{I}((U_C^i[1], \ldots, U_C^i[Q], T_i); (U_H^i, \boldsymbol{s}) | U_C^{i-1}, U_H^{i-1}) \\
&\overset{(b)}{=} \sum_{j=1}^{Q} \mathtt{I}(U_C^i[j]; (U_H^i, \boldsymbol{s}) | U_C^{i-1}, U_H^{i-1}, \{U_C^i[k]\}_{k\in[j-1]}) \\
&\quad\quad + \mathtt{I}(T_i; (U_H^i, \boldsymbol{s}) | U_C^{i-1}, U_H^{i-1}, \{U_C^i[k]\}_{k\in[Q]}) \\
&\overset{(c)}{\leq} H(T_i) \leq \log Q.
\end{aligned}$$

Here, (a) follows since $U_C^i[T_i]$ is determined by $\{U_C^i[k]\}_{k\in[Q]}$ and T_i; (b) follows by chain rule; and (c) follows from the above observation and since $T_i \in [Q]$. Thus, $\mathtt{I}(U_C^i; \boldsymbol{s} | U_H^{i-1}, U_C^{i-1}) \leq \mathtt{I}(U_C^i; (U_H^i, \boldsymbol{s}) | U_H^{i-1}, U_C^{i-1}) \leq \log Q$. Moreover, since in the real world, the distribution of U_H^r coincides with the distribution of U_C^r with switched roles, we have that

$$\mathtt{I}(U_H^i; (U_C^i, \boldsymbol{s}) | U_H^{i-1}, U_C^{i-1}) \lesssim \log Q.$$

Therefore, by the chain rule,

$$\mathtt{I}(U_H^i; \boldsymbol{s} | U_C^i, U_H^{i-1}) \leq \mathtt{I}(U_H^i; (U_C^i, \boldsymbol{s}) | U_H^{i-1}, U_C^{i-1}) \lesssim \log Q.$$

Putting everything together, we obtain

$$\begin{aligned}
m \sim \mathsf{H}(\boldsymbol{s}) &= \mathsf{H}(\boldsymbol{s} | \boldsymbol{u}, U_H^r, U_C^r) + \mathtt{I}(\boldsymbol{u}; \boldsymbol{s} | U_C^r, U_H^r) \\
&\quad + \sum_{i\in[r]} \left(\mathtt{I}(U_H^i; \boldsymbol{s} | U_C^i, U_H^{i-1}) + \mathtt{I}(U_C^i; \boldsymbol{s} | U_H^{i-1}, U_C^{i-1}) \right) \\
&= \mathtt{I}(\boldsymbol{u}; \boldsymbol{s} | U_C^r, U_H^r) + O(r \cdot \log Q) \\
&\leq \mathsf{H}(\boldsymbol{u}) + O(r \cdot \log Q) \\
&= n + O(r \cdot \log \lambda).
\end{aligned}$$

□

5.9 One-Round Unbiased Sampling for any Distribution

In this section, we present our construction for one-round fully-secure sampling from an arbitrary (efficient) distribution. We previously overviewed this construction in Section 5.2.3, and invite the reader to review the basic definitions in Section 5.3.1, Section 5.3.4, and Section 5.3.5. We begin by precisely specifying the sampling functionality, after which we give our protocol, and finally prove a security theorem.

Functionality 5.9.1. $\mathcal{F}_{\mathcal{D}}$. Distributed Sampling

Initialisation: On receiving `init` from all parties, the functionality activates.

Sample: On receiving $(\mathtt{sample}, \mathsf{sid})$ from all parties, the functionality samples $R \overset{\$}{\leftarrow} \mathcal{D}(1^\lambda)$ and outputs $(\mathtt{sampled}, \mathsf{sid}, R)$ to all parties.

Protocol 5.9.2. One-round unbiased sampling protocol

Let $\mathcal{D}(1^\lambda)$ be an efficient distribution. We consider the following algorithms.

- $\widetilde{\mathcal{D}}(1^\lambda)$
  1. $\boldsymbol{K} \overset{\$}{\leftarrow} \{0,1\}^\lambda$

2. $S \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}[\boldsymbol{K}])$ (see Program 5.9.3)
3. Output S

- $\widetilde{\mathcal{D}}'(1^\lambda, \hat{R})$

1. $\hat{\boldsymbol{s}} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
2. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^\lambda$
3. $S \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}'[\boldsymbol{K}, \hat{\boldsymbol{s}}, \hat{R}])$ (see Program 5.9.4)
4. Output S and $\hat{\boldsymbol{s}}$

Let $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{SimSetup}, \mathsf{SimGen}, \mathsf{Trap})$ be an indistinguishability-preserving distributed sampler with unstructured CRS for the distributions $(\widetilde{\mathcal{D}}, \widetilde{\mathcal{D}}')$ [AWZ23, Section 5.2].

Initialisation: The parties call $\mathcal{F}^n_{\mathsf{Coin}}$ and receive the distributed sampler CRS crs.

Sample: On input $(\texttt{sample}, \mathsf{sid})$ from the environment, each party P_i performs the following operations:

1. $U_i \xleftarrow{\$} \mathsf{DS.Gen}(1^\lambda, \mathsf{sid}, i, \mathsf{crs})$.
2. Broadcast U_i and receive $(U_j)_{j \neq i}$ from the other parties.
3. $S \leftarrow \mathsf{DS.Sample}(U_1, \dots, U_n, \mathsf{crs})$.
4. Call $\mathcal{F}^n_{\mathsf{Coin}}$ to receive $\boldsymbol{s}$.
5. Output $(\texttt{sampled}, \mathsf{sid}, S(s))$.

Program 5.9.3. $\mathcal{P}[\boldsymbol{K}]$

Hard-Coded: Puncturable PRF key $\boldsymbol{K}$

Input: A random string $\boldsymbol{s} \in \{0,1\}^{n(\lambda)}$

1. $\boldsymbol{r} \leftarrow F(\boldsymbol{K}, \boldsymbol{s})$
2. $R \leftarrow \mathcal{D}(1^\lambda; \boldsymbol{r})$
3. Output R

Program 5.9.4. $\mathcal{P}'[\boldsymbol{K}, \widehat{\boldsymbol{s}}, \widehat{R}]$

Hard-Coded: Puncturable PRF key $\boldsymbol{K}$

Input: A random string $\boldsymbol{s} \in \{0,1\}^{n(\lambda)}$

1. $\boldsymbol{r} \leftarrow F(\boldsymbol{K}, \boldsymbol{s})$
2. $R \leftarrow \mathcal{D}(1^\lambda; \boldsymbol{r})$
3. If $\boldsymbol{s} = \hat{\boldsymbol{s}}$, output $\hat{R}$, otherwise, output R.

The following theorem essentially formalizes Theorem 5.2.7.

Theorem 5.9.5. Let $\mathcal{D}$ be an efficient distribution and let $n = \Theta(\lambda)$. Assume the existence of indistinguishability-preserving distributed samplers with unstructured CRS for the distributions $(\widetilde{\mathcal{D}}, \widetilde{\mathcal{D}}')$ [AWZ23, Section 5.2], indistinguishability obfuscation, and injective length-doubling PRGs. It follows that protocol 5.9.2 UC-realizes the functionality $\mathcal{F}_{\mathcal{D}}$ against any malicious PPT adversary that statically corrupts up to $N-1$ parties, in the $\mathcal{F}^n_{\mathsf{Coin}}$-hybrid model. The round complexity of the protocol is one and the amortised sampling complexity is one.

Proof. In this proof, we rely on the notation, definitions and theorems of [AWZ23, Section 5.2].

We consider simulator 5.9.6. We prove that no adversary can distinguish between the real protocol and the interaction between this simulator and $\mathcal{F}_{\mathcal{D}}$.

Simulator 5.9.6. $\mathcal{S}_{\mathcal{D}}$. Distributed Sampling

Initialisation: When all the corrupted parties have sent `init` to $\mathcal{F}^n_{\mathsf{Coin}}$, the simulator initialises $\mathcal{F}_{\mathcal{D}}$ on behalf of the corrupted parties and computes $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{DS.SimSetup}(1^\lambda)$. When the adversary calls $\mathcal{F}^n_{\mathsf{Coin}}$, the simulator replies with crs.

Sample: Let sid be the session identity and let ι be the index of an honest party. Let H be the set of honest players. The simulator performs the following operations:

1. Receive $\hat{R}$ from the functionality.
2. $\forall i \in H \setminus \{\iota\} : U_i \xleftarrow{\$} \mathsf{DS.Gen}(1^\lambda, \mathsf{sid}, i, \mathsf{crs})$
3. $(U_\iota, \xi) \xleftarrow{\$} \mathsf{DS.SimGen}(1^\lambda, \mathsf{sid}, \iota, \zeta, \hat{R})$
4. Send $(U_i)_{i\in H}$ to the adversary and receive $(U_i)_{i\notin H}$ as a reply.
5. $(S, \hat{\boldsymbol{s}}) \leftarrow \mathsf{DS.Trap}(\xi, (U_i)_{i\in[N]})$.
6. When the corrupted parties call $\mathcal{F}^n_{\mathsf{Coin}}$, provide $\hat{\boldsymbol{s}}$.

We consider games 5.9.7 (see [AWZ23, Definition 6]) and 5.9.8 (see [AWZ23, Definition 9]).

Game 5.9.7. Game with Oracle Distribution $(\mathsf{Ch}_0, \widetilde{\mathcal{D}})$

The challenger Ch_0 interacts with the adversary and the oracle $\widetilde{\mathcal{D}}$. It performs the following operations:

1. send $(\texttt{sample}, i)$ for every honest party P_i
2. wait for $(\texttt{sample}, i)$ from every corrupted party P_i
3. provide the adversary with $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

Game 5.9.8. Game with Trapdoored Oracle Distribution $(\mathsf{Ch}_1, \widetilde{\mathcal{D}}')$

The challenger Ch_1 interacts with the adversary and the oracle $\widetilde{\mathcal{D}}'$. It performs the following operations:

1. $\hat{R} \xleftarrow{\$} \mathcal{D}(1^\lambda)$
2. send $(\texttt{sample}, i)$ for every honest party P_i
3. provide $\mathsf{aux}' := \hat{R}$ to the oracle
4. wait for $(\texttt{sample}, i)$ from every corrupted party P_i
5. If Ch_1 receives a trapdoor $\hat{\boldsymbol{s}}$ from the oracle, it forwards it to the adversary. Otherwise, Ch_1 provides the adversary with $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$.

We observe that if we compile $(\mathsf{Ch}_0, \widetilde{\mathcal{D}})$ using the indistinguishability preserving distributed sampler, the view of the adversary becomes as in protocol 5.9.2 (see [AWZ23, Definition 7]). If instead we compile the trapdoored game $(\mathsf{Ch}_1, \widetilde{\mathcal{D}}')$ (see [AWZ23, Definition 12]), the view of the adversary becomes as in the interaction between $\mathcal{F}_{\mathcal{D}}$ and $\mathcal{S}_{\mathcal{D}}$ (see Functionality 5.9.1 and Simulator 5.9.6). Our goal is to show that such views are indistinguishable. Observe that if $(\mathsf{Ch}_0, \widetilde{\mathcal{D}})$ and $(\mathsf{Ch}_1, \widetilde{\mathcal{D}}')$ satisfy the conditions necessary to apply indistinguishability-preserving distributed samplers, we are done [AWZ23, Definition 14]. We therefore have to verify the following properties:

- $\widetilde{\mathcal{D}}'$ is a trapdoored distribution for $\widetilde{\mathcal{D}}$ (see [AWZ23, Definition 8]).
- $(\mathsf{Ch}_1, \widetilde{\mathcal{D}}')$ satisfies trapdoor security (see [AWZ23, Definition 10]).
- $(\mathsf{Ch}_0, \widetilde{\mathcal{D}})$ and $(\mathsf{Ch}_1, \widetilde{\mathcal{D}}')$ are chosen-sample indistinguishable (see [AWZ23, Definition 13]).

First property. We start by showing that $\widetilde{\mathcal{D}}'$ is a trapdoored distribution for $\widetilde{\mathcal{D}}$ (see [AWZ23, Definition 8]). We do it by means of a series of indistinguishable hybrids. Let $\hat{R}$ be the auxiliary information given to $\widetilde{\mathcal{D}}'$.

Hybrid $\mathcal{H}_0$. This hybrid corresponds to the distribution $\widetilde{\mathcal{D}}$

1. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^\lambda$
2. $S \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}[\boldsymbol{K}])$ (see Program 5.9.3)
3. Output S

Hybrid $\mathcal{H}_1$. Let $G : \{0,1\}^{n(\lambda)} \to \{0,1\}^{2n(\lambda)}$ be an injective PRG. In this hybrid, we modify the obfuscated program S as follows.

1. $\hat{\boldsymbol{s}} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
2. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^\lambda$
3. $\hat{\boldsymbol{r}} \leftarrow F(\boldsymbol{K}, \hat{\boldsymbol{s}})$
4. $\boldsymbol{K}^* \leftarrow \mathsf{Punct}(\boldsymbol{K}, \hat{\boldsymbol{s}})$
5. $R' \leftarrow \mathcal{D}(1^\lambda; \hat{\boldsymbol{r}})$
6. $\hat{\boldsymbol{w}} \leftarrow G(\hat{\boldsymbol{s}})$
7. $S \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}_1[\boldsymbol{K}^*, \hat{\boldsymbol{w}}, R'])$ (see Program 5.9.9)
8. Output S

Thanks to the injectivity of G and the correctness of the puncturable PRF, the input-output behaviour of S remains the same as in $\mathcal{H}_0$. Therefore, the two hybrids are indistinguishable due to the security of iO.

Program 5.9.9. $\mathcal{P}_1[\boldsymbol{K}^*, \hat{\boldsymbol{w}}, R']$

Hard-Coded: Puncturable PRF key $\boldsymbol{K}^*$, the string $\hat{\boldsymbol{w}}$ and sample R'. **Input:** A random string $\boldsymbol{s} \in \{0,1\}^{n(\lambda)}$

```
1. $\boldsymbol{w} \leftarrow G(\boldsymbol{s})$
2. If $\boldsymbol{w} = \hat{\boldsymbol{w}}$, output $R'$
3. $\boldsymbol{r} \leftarrow F(\boldsymbol{K}^*, \boldsymbol{s})$
4. $R \leftarrow \mathcal{D}(1^\lambda; \boldsymbol{r})$
```

5. Output R

Hybrid $\mathcal{H}_2$. In this hybrid, we modify the obfuscated program S as follows.

1. $\hat{\boldsymbol{s}} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
2. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^{\lambda}$
3. $\boldsymbol{K}^* \leftarrow \mathsf{Punct}(\boldsymbol{K}, \hat{\boldsymbol{s}})$
4. $R' \xleftarrow{\$} \mathcal{D}(1^{\lambda})$
5. $\hat{\boldsymbol{w}} \leftarrow G(\hat{\boldsymbol{s}})$
6. $S \xleftarrow{\$} \mathsf{iO}(1^{\lambda}, \mathcal{P}_1[\boldsymbol{K}^*, \hat{\boldsymbol{w}}, R'])$ (see Program 5.9.9)
7. Output S

$\mathcal{H}_1$ and $\mathcal{H}_2$ are indistinguishable thanks to the security of the puncturable PRF F.

Hybrid $\mathcal{H}_3$. In this hybrid, we modify the program $\mathcal{P}$ as follows.

1. $\hat{\boldsymbol{s}} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
2. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^{\lambda}$
3. $R' \xleftarrow{\$} \mathcal{D}(1^{\lambda})$
4. $\hat{\boldsymbol{w}} \leftarrow G(\hat{\boldsymbol{s}})$
5. $S \xleftarrow{\$} \mathsf{iO}(1^{\lambda}, \mathcal{P}_1[\boldsymbol{K}, \hat{\boldsymbol{w}}, R'])$ (see Program 5.9.9)
6. Output S

Notice that the key hardcoded in S is not punctured anymore. The input-output behaviour of S has not changed due to the correctness of the puncturable PRF, so, $\mathcal{H}_2$ and $\mathcal{H}_3$ are indistinguishable thanks to the security of iO.

Hybrid $\mathcal{H}_4$. In this hybrid, we modify the obfuscated program S as follows.

1. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^{\lambda}$
2. $R' \xleftarrow{\$} \mathcal{D}(1^{\lambda})$
3. $\hat{\boldsymbol{w}} \xleftarrow{\$} \{0,1\}^{2n(\lambda)}$
4. $S \xleftarrow{\$} \mathsf{iO}(1^{\lambda}, \mathcal{P}_1[\boldsymbol{K}, \hat{\boldsymbol{w}}, R'])$ (see Program 5.9.9)
5. Output S

$\mathcal{H}_4$ is indistinguishable from $\mathcal{H}_3$ thanks to the security of the PRG G.

Hybrid $\mathcal{H}_5$. In this hybrid, we modify the obfuscated program S as follows.

1. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^{\lambda}$
2. $\hat{\boldsymbol{w}} \xleftarrow{\$} \{0,1\}^{2n(\lambda)}$
3. $S \xleftarrow{\$} \mathsf{iO}(1^{\lambda}, \mathcal{P}_1[\boldsymbol{K}, \hat{\boldsymbol{w}}, \hat{R}])$ (see Program 5.9.9)

4. Output S

With overwhelming probability, $\hat{\boldsymbol{w}}$ does not lie in the image of the PRG G, therefore, the input-output behaviour of S in $\mathcal{H}_5$ and $\mathcal{H}_4$ remains the same. The two hybrids are therefore indistinguishable due to the security of iO.

Hybrid $\mathcal{H}_6$. In this hybrid, we modify the obfuscated program S as follows.

1. $\hat{\boldsymbol{s}} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
2. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^{\lambda}$
3. $\hat{\boldsymbol{w}} \leftarrow G(\hat{\boldsymbol{s}})$
4. $S \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}_1[\boldsymbol{K}, \hat{\boldsymbol{w}}, \hat{R}])$ (see Program 5.9.9)
5. Output S

$\mathcal{H}_6$ is indistinguishable from the previous one by the security of the PRG G.

Hybrid $\mathcal{H}_7$. In this hybrid, we modify the obfuscated program S as follows.

1. $\hat{\boldsymbol{s}} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
2. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^{\lambda}$
3. $S \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}'[\boldsymbol{K}, \hat{\boldsymbol{s}}, \hat{R}])$ (see Program 5.9.4)
4. Output S

Thanks to the injectivity of G, the input-output behaviour of S remains the same as in $\mathcal{H}_6$. Therefore, the two hybrids are indistinguishable due to the security of iO. Notice that the distribution of S in $\mathcal{H}_7$ is the same if it was produced by $\widetilde{\mathcal{D}}'(1^\lambda, \hat{R})$.

Second property. We show that $(\mathsf{Ch}_1, \widetilde{\mathcal{D}}')$ satisfies trapdoor security (see [AWZ23, Definition 10]). We proceed again by means of series of indistinguishable hybrids.

Hybrid $\mathcal{H}_0$. This hybrid corresponds to the trapdoor game in which the oracle does not provide the trapdoor $\hat{\boldsymbol{s}}$ to Ch_1. In particular, the view of the adversary consists of pair $(S, \boldsymbol{s})$ generated as follows:

1. $\hat{R} \xleftarrow{\$} \mathcal{D}(1^\lambda)$
2. $\hat{\boldsymbol{s}} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
3. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^{\lambda}$
4. $S \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}'[\boldsymbol{K}, \hat{\boldsymbol{s}}, \hat{R}])$ (see Program 5.9.4)
5. $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$
6. Provide the adversary with S and $\boldsymbol{s}$

Hybrid $\mathcal{H}_1$. In this hybrid, we modify the distribution of the obfuscated program S given to the adversary. In particular, we puncture the key $\boldsymbol{K}$ in position $\hat{\boldsymbol{s}}$. The input-output behaviour of S remains the same, so Hybrid 0 and Hybrid 1 are indistinguishable thanks to iO. Formally, we generate the pair $(S, \boldsymbol{s})$ as follows.

1. $\hat{R} \xleftarrow{\$} \mathcal{D}(1^\lambda)$

2. $\hat{\boldsymbol{s}} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

3. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^{\lambda}$

4. $\boldsymbol{K}^* \leftarrow \mathsf{Punct}(\boldsymbol{K}, \hat{\boldsymbol{s}})$

5. $S \xleftarrow{\$} \mathsf{iO}(1^{\lambda}, \mathcal{P}'[\boldsymbol{K}^*, \hat{\boldsymbol{s}}, \hat{R}])$ (see Program 5.9.4)

6. $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

7. Provide the adversary with S and $\boldsymbol{s}$

Hybrid $\mathcal{H}_2$. In this hybrid, we modify again the distribution of the obfuscated program S given to the adversary. In particular, we sample $\hat{R}$ using the randomness produced by $F(\boldsymbol{K}, \hat{\boldsymbol{s}})$. $\mathcal{H}_2$ and $\mathcal{H}_1$ are indistinguishable thanks to the security of the puncturable PRF. Formally, we generate the pair $(S, \boldsymbol{s})$ as follows.

1. $\hat{\boldsymbol{s}} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

2. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^{\lambda}$

3. $\boldsymbol{K}^* \leftarrow \mathsf{Punct}(\boldsymbol{K}, \hat{\boldsymbol{s}})$

4. $\hat{\boldsymbol{r}} \leftarrow F(\boldsymbol{K}, \hat{\boldsymbol{s}})$

5. $\hat{R} \leftarrow \mathcal{D}(1^{\lambda}; \hat{\boldsymbol{r}})$

6. $S \xleftarrow{\$} \mathsf{iO}(1^{\lambda}, \mathcal{P}'[\boldsymbol{K}^*, \hat{\boldsymbol{s}}, \hat{R}])$ (see Program 5.9.4)

7. $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

8. Provide the adversary with S and $\boldsymbol{s}$

Hybrid $\mathcal{H}_3$. In this hybrid, we modify the program: we puncture the key is position $\boldsymbol{s}$ and we hardcode the relative output. The input-output behaviour of S remains unvaried, so, $\mathcal{H}_3$ and $\mathcal{H}_2$ are indistinguishable thanks to the security of iO. Formally, we generate the pair $(S, \boldsymbol{s})$ as follows.

1. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^{\lambda}$

2. $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

3. $\boldsymbol{K}^* \leftarrow \mathsf{Punct}(\boldsymbol{K}, \boldsymbol{s})$

4. $\boldsymbol{r} \leftarrow F(\boldsymbol{K}, \boldsymbol{s})$

5. $R \leftarrow \mathcal{D}(1^{\lambda}; \boldsymbol{r})$

6. $S \xleftarrow{\$} \mathsf{iO}(1^{\lambda}, \mathcal{P}'[\boldsymbol{K}^*, \boldsymbol{s}, R])$ (see Program 5.9.4)

7. Provide the adversary with S and $\boldsymbol{s}$

Hybrid $\mathcal{H}_4$. In this hybrid, we sample R using true randomness instead of $F(\boldsymbol{K}, \boldsymbol{s})$. $\mathcal{H}_4$ and $\mathcal{H}_3$ are indistinguishable thanks to the security of the puncturable PRF. Formally, we generate the pair $(S, \boldsymbol{s})$ as follows.

1. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^{\lambda}$

2. $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

3. $\boldsymbol{K}^* \leftarrow \mathsf{Punct}(\boldsymbol{K}, \boldsymbol{s})$

4. $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$

5. $S \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}'[\boldsymbol{K}^*, \boldsymbol{s}, R])$ (see Program 5.9.4)

6. Provide the adversary with S and $\boldsymbol{s}$

Hybrid $\mathcal{H}_5$. In this hybrid, we modify the obfuscated program S. In particular, we hardcode $\boldsymbol{K}$ instead of $\boldsymbol{K}^*$. The input-output behaviour of S remains unvaried, so, $\mathcal{H}_5$ and $\mathcal{H}_4$ 5 are indistinguishable thanks to the security of iO. Formally, we generate the pair $(S, \boldsymbol{s})$ as follows.

1. $\boldsymbol{K} \xleftarrow{\$} \{0,1\}^\lambda$

2. $\boldsymbol{s} \xleftarrow{\$} \{0,1\}^{n(\lambda)}$

3. $R \xleftarrow{\$} \mathcal{D}(1^\lambda)$

4. $S \xleftarrow{\$} \mathsf{iO}(1^\lambda, \mathcal{P}'[\boldsymbol{K}, \boldsymbol{s}, R])$ (see Program 5.9.4)

5. Provide the adversary with S and $\boldsymbol{s}$

Observe that the view of the adversary in $\mathcal{H}_5$ is identical to the trapdoored game in the case in which the trapdoor $\hat{\boldsymbol{s}}$ is provided to the challenger Ch_1.

Third property. It is straightforward to see that $(\mathsf{Ch}_0, \widetilde{\mathcal{D}})$ and $(\mathsf{Ch}_1, \widetilde{\mathcal{D}}')$ are chosen-sample indistinguishable (see [AWZ23, Definition 13]). Indeed, the view of te adversary in the two worlds are identical except for the fact that, in one case, the oracle provides the adversary with an obfuscated program S generated using $\widetilde{\mathcal{D}}(1^\lambda)$. In the other case, S is generated using $\widetilde{\mathcal{D}}'(1^\lambda, \hat{R})$ for a random $\hat{R} \xleftarrow{\$} \mathcal{D}(1^\lambda)$. No information about the trapdoor generated by $\widetilde{\mathcal{D}}'(1^\lambda, \hat{R})$ is provided. Since $\widetilde{\mathcal{D}}'$ is a trapdoored distribution for $\widetilde{\mathcal{D}}$, the two worlds are computationally indistinguishable. This ends the proof.

□

Bibliography

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452. Springer, Heidelberg, August 2022.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP 99*, volume 1644 of *LNCS*, pages 1–9. Springer, Heidelberg, July 1999.

[AOS23] Damiano Abram, Maciej Obremski, and Peter Scholl. On the (Im)possibility of Distributed Samplers: Lower Bounds and Party-Dynamic Constructions. Cryptology ePrint Archive, Report 2023/863, 2023. https://eprint.iacr.org/2023/863.

[AP09] Joel Alwen and Chris Peikert. Generating shorter bases for hard random lattices. In *STACS*, 2009.

[ASY22] Damiano Abram, Peter Scholl, and Sophia Yakoubov. Distributed (correlation) samplers: How to remove a trusted dealer in one round. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 790–820. Springer, Heidelberg, May / June 2022.

[AWY20] Shweta Agrawal, Daniel Wichs, and Shota Yamada. Optimal broadcast encryption from LWE and pairings in the standard model. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 149–178. Springer, Heidelberg, November 2020.

[AWZ23] Damiano Abram, Brent Waters, and Mark Zhandry. Security-Preserving Distributed Samplers: How to Generate Any CRS in One Round Without Random Oracles. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 489–514, Cham, 2023. Springer Nature Switzerland.

[Ban93] Wojciech Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 1993.

[BDD$^+$21] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. TARDIS: A foundation of time-lock puzzles in UC. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 429–459. Springer, Heidelberg, October 2021.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

[BGR96] Mihir Bellare, Juan A. Garay, and Tal Rabin. Distributed pseudo-random bit generators - a new way to speed-up shared coin tossing. In James E. Burns and Yoram Moses, editors, *15th ACM PODC*, pages 191–200. ACM, August 1996.

[BHLT17] Niv Buchbinder, Iftach Haitner, Nissan Levi, and Eliad Tsfadia. Fair coin flipping: Tighter analysis and the many-party case. In Philip N. Klein, editor, *28th SODA*, pages 2580–2600. ACM-SIAM, January 2017.

[Blu82] Manuel Blum. Coin flipping by telephone. In *Proc. IEEE Spring COMPCOM*, pages 133–137, 1982.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CL15] Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In Kaisa Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 487–505. Springer, Heidelberg, April 2015.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory 2nd Edition (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, July 2006.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GLOV12] Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *53rd FOCS*, pages 51–60. IEEE Computer Society Press, October 2012.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HMU06] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. On the (im-)possibility of extending coin toss. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 504–521. Springer, Heidelberg, May / June 2006.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st ACM STOC*, pages 12–24. ACM Press, May 1989.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, Heidelberg, August 2014.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

[Lin03] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology*, 16(3):143–184, June 2003.

[LZM20] Chen-Da Liu-Zhang and Ueli Maurer. Synchronous constructive cryptography. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 439–472. Springer, Heidelberg, November 2020.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, 1996.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

[PR06] Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 145–166. Springer, Heidelberg, March 2006.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 187–196. ACM Press, May 2008.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 2009.

[Yeu91] R.W. Yeung. A new outlook on shannon's information measures. *IEEE Transactions on Information Theory*, 37(3):466–474, 1991.

[Zha16] Mark Zhandry. The magic of ELFs. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 479–508. Springer, Heidelberg, August 2016.