# Secure Multi-Party Protocols Under a Modern Lens

by

ELETTE CHANTAE BOYLE

B.S., CALIFORNIA INSTITUTE OF TECHNOLOGY (2008)

Submitted to the Department of Mathematics
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

Signature of Author: [illegible]
Department of Mathematics
May 13, 2013

Certified by: [illegible] 5/13/13
Shafi Goldwasser
RSA Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: [illegible]
Michel Goemans
Chairman, Graduate Applied Mathematics Committee

SECURE MULTI-PARTY PROTOCOLS UNDER A MODERN LENS

by

ELETTE CHANTAE BOYLE

Submitted to the Department of Mathematics on June 10, 2013
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Mathematics

## Abstract

A secure multi-party computation (MPC) protocol for computing a function $f$ allows a group of parties to jointly evaluate $f$ over their private inputs, such that a computationally bounded adversary who corrupts a subset of the parties can not learn anything beyond the inputs of the corrupted parties and the output of the function $f$.

General MPC completeness theorems in the 1980s showed that every efficiently computable function can be evaluated securely in this fashion [Yao86, GMW87, CCD87, BGW88] using the existence of cryptography. In the following decades, progress has been made toward making MPC protocols efficient enough to be deployed in real-world applications. However, recent technological developments have brought with them a slew of new challenges, from new security threats to a question of whether protocols can scale up with the demand of distributed computations on massive data. Before one can make effective use of MPC, these challenges must be addressed.

In this thesis, we focus on two lines of research toward this goal:

- Protocols resilient to side-channel attacks.
  We consider a strengthened adversarial model where, in addition to corrupting a subset of parties, the adversary may *leak* partial information on the secret states of honest parties during the protocol. In presence of such adversary, we first focus on preserving the *correctness* guarantees of MPC computations. We then proceed to address security guarantees, using cryptography. We provide two results: an MPC protocol whose security provably "degrades gracefully" with the amount of leakage information obtained by the adversary, and a second protocol which provides *complete* security assuming a (necessary) one-time preprocessing phase during which leakage cannot occur.

- Protocols with scalable communication requirements.
  We devise MPC protocols with *communication locality*: namely, each party only needs to communicate with a small (polylog) number of dynamically chosen parties. Our techniques use digital signatures and extend particularly well to the case when the function $f$ is a *sublinear algorithm* whose execution depends on $o(n)$ of the $n$ parties' inputs.

THESIS SUPERVISOR: Shafi Goldwasser
TITLE: RSA Professor of Electrical Engineering and Computer Science

# Acknowledgments

I am so fortunate to have been surrounded by such an assortment of brilliant and fun people during my graduate career.

Thank you foremost to Shafi Goldwasser and to Yael Tauman Kalai. Your drive, enthusiasm, guidance, brilliance, humor, and sometimes-brutal honesty have made me the researcher I am today. For your immeasurable support I will be forever thankful.

A special call-out to Silvio Micali, whose irresistible teaching personality drew me to cryptography at the start. To my mathematics department advisor, Peter Shor. And, to my thesis committee, Shafi, Peter, Yael, and Jonathan Kelner.

I want to thank my mentor and friend Gil Segev, the incredible cryptography group at MIT, and my other talented coauthors: Sanjam Garg, Abhishek Jain, Amit Sahai, Stefano Tessaro, and Daniel Wichs.

Deep thanks to my family and friends, without whom I would have given up long ago.

And, to Zvika. When the curtain closes, a bright light remains.

# Contents

# Chapter 1

# Introduction

The notion of *secure multi-party computation* (MPC), introduced in the seminal works of Yao [Yao82] and Goldreich, Micali and Wigderson [GMW87], is one of the cornerstones in cryptography. An MPC protocol for computing a function $f$ allows a group of parties to jointly evaluate $f$ over their private inputs, with the property that an adversary who corrupts a subset of the parties does not learn anything beyond the inputs of the corrupted parties and the output of the function $f$.

To demonstrate the power and utility of MPC protocols, consider the following online auction scenario. A collection of parties wish to bid for certain goods; an auctioning party receives bidding information from these potential buyers and allocates goods in accordance with a prespecified mechanism. Questions of how to design good allocation mechanisms have received much attention in recent years, aiming (for example) to guarantee maximum revenue or social welfare. However, in these works it is assumed that the auctioneer is an *honest* agent. If in reality this is not the case, then questions of mechanism design are moot—a malicious auctioneer can simply allocate the goods however he likes without verification, and (just as dangerously) he will learn the *entirety* of the bidders' sensitive bidding information.

Instead, an MPC protocol allows the involved parties to *jointly* evaluate the allocation functionality, with the strong guarantees that (as long as sufficiently many parties are acting honestly) the computation will result in the correct allocation, and no information will be revealed about the parties' bidding information beyond this output allocation.

General MPC completeness theorems in the 1980s showed that every efficiently computable function can be evaluated securely in this fashion [Yao86, GMW87, CCD87, BGW88]. Over the years since then, much progress has been made toward making MPC protocols secure and efficient enough to be deployed in real-world applications. Indeed, MPC protocols have found a wide range of applications, such as in protocols for auctions, electronic voting, private information retrieval, and threshold and proactive cryptography.

However, new waves of recent technological development have brought with them a slew of new security challenges. Important computations are now performed on small, portable devices such as smart cards and cell phones, opening them to new realms of adversarial attacks. Massive data analyses are executed distributively in the cloud, placing extreme demands on MPC protocol efficiency. Before one can effectively make use of MPC within modern applications, these challenges must be addressed.

In this thesis, we focus on two lines of research toward this goal: designing secure protocols resilient to newly prominent and dangerous *physical* attacks, and constructing protocols adhering to stringent communication requirements in order to extend feasibly to large-scale settings.

**Protecting against physical attacks.** In cryptography, a rigorous claim that a system is "secure" consists of: (1) an assumed class of adversarial behaviors, (2) a notion of security capturing what it means to break the system, and (3) a proof that no adversary within this class is able to break the system.

Since the mid 1980s, a strong collection of formal adversarial models and security definitions have emerged. Within MPC, researchers have considered *fail-stop* adversaries, modeling accidental failure of a computing party; *passive* adversaries, who attempt to learn additional information while properly following the protocol; and, in the strongest form, *malicious* adversaries, who corrupt and arbitrarily control a subset of the parties in attempt to break security. The malicious corruption model is very strong and expressive, capturing (for example) collusion among deviating parties. However, underlying it and each of the aforementioned models is the implicit assumption that an adversary maintains *black-box* access to the honest parties within the protocol.

In recent years, many cryptographic schemes proven secure under standard adversarial models have been broken in practice (e.g., Diffie-Hellman based schemes [Koc96] and RSA [Koc96, HSH$^+$09, HS09]). Disturbingly, they were not broken by breaking the underlying assumption (say, that factoring is hard), but rather by executing a class of *physical* attacks not taken into account in the classical adversarial models. In these so-called *side-channel* attacks, an adversary may learn additional information about the internal secret state of a system by measuring properties resulting from specific physical implementations: e.g., timing information, detection of internal faults, electromagnetic radiation, and power consumption [BS97, BDL97, Koc96, KJJ99], which is not captured by black-box interaction.

To help bridge this gap between theory and practice, the *memory leakage model* [AGV09] was defined, capturing additional information gained by the adversary through side-channel attacks. This model allows the adversary to specify *leakage* functions $f$, and receive back the evaluation of $f$ on the entire internal secret state of the system. It has become an important research agenda to design *leakage-resilient* cryptographic systems whose security holds within these more demanding settings.

In this thesis, we present three results within a line of research designing MPC protocols secure in the presence of side-channel attacks. In the first, we focus on preserving *correctness* guarantees of the computation. Namely, we describe how to collectively sample an unbiased random string, which suffices for correctly evaluating any randomized functionality on public inputs. We then proceed to study the effects of side-channel leakage on the *input privacy* of MPC protocols. We provide two results: an MPC protocol whose security provably "degrades gracefully" with the amount of leakage information obtained by the adversary, and a second protocol which provides *complete* security assuming a (necessary) one-time preprocessing phase during which leakage cannot occur.

**Protocols with scalable communication requirements.** Since the original constructions in the 1980s, tremendous progress has been made in designing MPC protocols with improved communication requirements. Focus has traditionally been placed on optimizing parameters such as the round complexity and total communication complexity of the protocol. However, despite decades of extensive study, the question of *locality* of communication (i.e., how many parties one needs to contact in the course of the protocol) has received little to no attention.

Optimizing communication locality is a crucial step toward providing feasible MPC solutions for the current generation of large-scale data computations. Large data sets, such as medical data, transaction data, the web and web access logs, or network traffic data, are now in abundance. Much of the data is stored or made accessible in a distributed fashion. This necessitated the development of efficient distributed protocols that compute over such data. To address the privacy concerns associated with such protocols, cryptographic techniques such as MPC for secure function evaluation where *data items are equated with servers* can be utilized to prevent unnecessary leakage of information. However, this demands MPC protocols whose communication requirements scale reasonably for large numbers of participating parties.

Essentially all works in MPC yield protocols where each party sends and receives messages from all $n$ parties (e.g., [GMW87, BGW88, CCD88, DI06, DN07, DIK$^+$08, DIK10, LATV12, AJLA$^+$12]). Yet in a setting where potentially hundreds of thousands, or even millions of parties are participating in a large-scale data analysis or computation over the internet, requiring coordination between each pair of parties is unrealistic. A small collection of prior works provide protocols with communication locality in mind; however, these works either do *not* guarantee that all parties receive the correct output [KSSV06, CGO10, CGO12], or (in a recent and independent work), require each party to communicate to $\Omega(\sqrt{n})$ other parties [DKMS12].

In this thesis, we describe a recent work achieving secure MPC for general randomized functionalities with communication locality that is *polylogarithmic* in the number of parties.

We now expand on the thesis works mentioned above.

## 1.1 Leakage-Resilient Coin Tossing

Randomization, and the ability to keep your local randomness and local state private, are fundamental ingredients at the disposal of fault-tolerant distributed algorithms. This was realized originating with the work of Rabin [Rab83], introducing the power of a shared global common coin to obtain a dramatic reduction in round complexity with respect to Ben-Or's asynchronous randomized consensus algorithm [Ben83][1]; and continued to be utilized in many beautiful distributed algorithms to this day in various network models: synchronous and asynchronous, faulty majority and faulty minority, private channels and full information, and networks with and without broadcast channels.

In this work, we address the question of how leakage from the local state of non-faulty parties affects the correctness of fault-tolerant distributed protocols. Here, in addition to the fact that some of the parties are faulty and fully compromised, the adversary who is coordinating the action of the faulty parties can obtain partial information in the form of leakage on the local state of all honest parties. This may potentially enable the adversary to alter the course of the protocol's

[1]Ben-Or's ingenious protocol does not require the local coin outcomes to remain ever private. All that is required of the coin is to be random. Alas, the number of rounds is exponential.

execution. We note that in this context, the coordinating adversary can adaptively choose the leakage functions, depending on the history of communication it sees thus far.

Randomized distributed algorithms address many different tasks. In some tasks, such as Byzantine agreement, leader election, and collective coin tossing, the parties have no secret inputs and the emphasis is on getting the *correct distribution over the outputs* rather than on input privacy. In other tasks such as secure distributed function evaluation, both perfect input privacy and the correctness of output distribution are required. In this work, we focus on the output correctness aspect of distributed protocols in the presence of leakage attacks. In particular, we provide a fault-tolerant and leakage-resilient protocol for collective unbiased coin tossing among $n$ parties, which is complete for *correctly* evaluating any (randomized) functionality on public inputs.

The problem of collective coin tossing in a distributed setting has received a great deal of prior attention, starting with the work of Rabin [Rab83] on distributed consensus. When there is no honest majority of parties, results from the two-party setting by [Cle86] showed that a bias of $\Omega(\frac{1}{r})$ must be incurred by any $r$-round protocol (this was recently shown optimal in a work of Moran et al. [MNS09]). Loosely speaking, the problem is that a dishonest party can bias the output by doing the following: At the last round, before sending his final message, he can compute the outcome, and abort if he does not favor this outcome, thus biasing the output. When there is an honest majority of parties, this attack can be prevented using *verifiable secret sharing* (VSS), a notion defined by Chor et al. [CGMA85]. Verifiable secret sharing allows each of the $n$ parties to toss a coin locally and share it among the $n$ parties. After all the local coins have been shared via a VSS, the parties reconstruct the different values. The output coin is set to be the result of simply taking the exclusive or of the local coins. The works of ([BGW88, CCD88]) on secure multi-party computation show how to achieve VSS in expected $O(1)$ rounds, and thus how to construct an unbiased coin tossing protocol that runs in $O(1)$ rounds. These results ([BGW88, CCD88]) hold unconditionally in the synchronous network model with less than a third Byzantine faults, assuming perfectly secure channels of communication between pairs of users and the use of a broadcast channel.

However, each of these protocols require the parties to generate and hold secret values, and security is guaranteed only under the assumption that these secrets are *completely* hidden from the adversarial view. It is easy to check that correctness breaks down if the adversary obtains some partial information about these secrets. This is the starting point of our work.

We remark that this work focuses on the problem of guaranteeing *correctness* of a multi-party computation. The more general problem of multi-party secure function evaluation (SFE), which addresses both correctness and privacy of parties' inputs, runs into immediate definitional problems in the presence of leakage attacks. Since leakage on the private inputs is available to the adversary, it is impossible to meet the SFE problem specification as is, since they require the inputs of honest parties to remain private beyond what is revealed by the SFE output value. Possible ways out of this conundrum are to relax the attack model to allow some form of a leak-free pre-processing phase, or to relax the security guarantees of an SFE. These two relaxations will be considered in the following two chapters.

### 1.1.1 Our Contributions

#### Leakage-Resilient Coin Tossing

We construct a leakage-resilient collective coin-tossing protocol in synchronous point-to-point networks with a broadcast channel and secure communication channels between pairs of parties. Our result is unconditional, and does not rely on computational assumptions.

We allow up to one third colluding statically corrupted malicious parties. Namely, a computationally unbounded rushing adversary can a priori choose parties to corrupt; during the protocol, he sees the internal state of all corrupted parties and can select the messages of these parties at any round, as a function of his entire view up to this point, including all honest parties' messages up to (and including) this round. In addition, the adversary can make *leakage queries* at every round, in the form of specifying a party and a leakage function, and obtain the result of the leakage functions applied to the internal state of the corresponding parties.

We allow the adversary to leak *arbitrary* functions of parties' secret states, as long as the total number of bits leaked from each party is at most some (pre-specified) $\lambda$ fraction of its entire secret state.[2] Each leakage query is applied to the secret state of a single party. Since participants of a distributed protocol typically run on different physical hardware (and reside in different locations), we believe that it is reasonable to assume each leakage query modeling a physical measurement reveals information about each party's execution separately. To maximize generality within this setting, we allow the leakage queries on different parties' secret states to be interleaved (i.e., leak some from party $i$ then some from party $j$, and then some more from party $i$), and the choice of leakage queries to be adaptively selected as a function of prior leakage. We remark that this distributed leakage model is similar to a model proposed by Akavia et al. [AGH12] in their work on public-key encryption in which the secret key of the decryption algorithm is distributed among two parties.

We call a $n$-party distributed protocol $(t, \lambda)$-leakage-resilient if it achieves its desired functionality in presence of an adversary who can control up to $t$ parties and can leak up to a $\lambda$ fraction of the internal secret state of each honest party (as above). We can now state our main theorem, to be formally stated within Chapter 2.

**Informal Theorem (LR Coin Toss).** For any constants $\delta, \epsilon > 0$, any $\lambda \leq \frac{\delta(1-\epsilon)}{10+6\delta}$, any sufficiently large $n \geq (3+\delta)t$, and any $m$, there exists a $(t, \lambda)$-leakage-resilient $n$-party distributed protocol that outputs a value $v \in \{0,1\}^m$, and terminates in $O(1)$ rounds, satisfying:

- **Agreement:** At the conclusion of the protocol, each party outputs a value $v_i \in \{0,1\}^m$. For all honest parties $P_i, P_j$, it holds that $v_i = v_j$.
- **Randomness:** The distribution of the honest output value $v$ is close to uniform in $\{0,1\}^m$; namely, the statistical distance between the two distributions is negligible in $n$.

A few remarks are in order.

[2]Our methods extend to also tolerate the Naor and Segev [NS09] leakage model which allows leakage functions which are not necessarily shrinking but leave the internal local state with enough min-entropy.

**Fairness:** We emphasize that our protocol achieves fairness, in the sense that even if the dishonest parties abort prematurely, the honest parties will output a random string.

**Strings versus Bits:** The output of our coin tossing protocol can be a long random string, as opposed to just a single bit. In the leak-free setting, this point is not worth emphasizing, since the coin-tossing protocol can be run in parallel to output as many bits as desired. However, in the leaky setting, if we run many protocols in parallel, leakage bounds may deteriorate quickly: if we run $k$ protocols, where each protocol tolerates leakage rate $\lambda$, then in the resulting parallel execution, the leakage rate becomes only $\frac{\lambda}{k}$. Thus, to maintain leakage bounds we would need to run the protocol sequentially, resulting in many rounds of communication. Our protocol has the property that it can output as many bits as desired in a constant number of rounds with constant leakage rate.

**Number of Parties:** Note that the guarantees of our protocol improve with the number of participating parties. However, the tools we develop likely will also have useful implications for the setting with few parties.

**Weakening the Secure Channels Assumption:** We assumed physically secure channels; however, our leakage model immediately implies we can tolerate leakage of information from these channels. This is because parties' messages are computed as a function of public information and their personal secret state, which we allow leakage on. To remove the secure channels assumption altogether, we would need to send the messages between honest parties using encryption, which would necessitate a computational assumption supporting the strength of the encryption algorithm. Furthermore, one would have to consider whether leakage from the secret keys of the decryption algorithm and the randomness used by the encryption algorithm can be tolerated. A recent work of Bitansky et al. [BCH11] suggests that by sending messages encrypted with non-committing encryption (introduced by Canetti et al. [CFGN96]), protocols in the secure channels model can be transformed into leakage-resilient secure protocols that do not assume secure channels.

**Relation to Using Imperfect Random Sources in Distributed Computing:** The question of achieving $O(1)$-round Byzantine Agreement and multi-party computation when parties do not have access to perfect local randomness, but rather to independent imperfect random sources such as min-entropy sources [GSV05, KLRZ08, KLR09], seems strongly related to our work here. Indeed, one may naturally view a random secret with leakage as a secret a-priori drawn from a min-entropy source. The crucial difference between these works and our own is that our leakage model allows the adversary to leak adaptively *during* the protocol, as opposed to non-adaptively before the protocol begins. More specifically, the approach taken in [GSV05, KLRZ08, KLR09] is to first generate *truly* random strings from the weak random sources, and then to use these random strings in the underlying protocol execution. This approach will not work in our setting, since the adversary can simply choose to leak on the newly generated random strings. On the other hand, we note that the works of [GSV05, KLRZ08, KLR09] consider randomness coming from an *arbitrary* min-entropy distribution, whereas our model considers perfect randomness that is being leaked so as to leave min-entropy in the distribution.

**Coin Flipping versus Byzantine Agreement:** Achieving a weak form of collective coin tossing was an important building block to construct Byzantine agreement protocols in many works, most notably in the work of Dwork et al. [DSS90], and of Feldman and Micali [FM85]. Our collective coin tossing protocol utilizes a broadcast channel as a primitive (which is equivalent to Byzantine agreement), and thus obviously cannot be used to construct Byzantine agreement. It is an interesting question for future research how to achieve coin tossing in the presence of leakage without assuming broadcast channels.

**Using Coin Tossing to Force Honest Behavior:** An important technique in multi-party protocols, initially proposed by Goldwasser and Micali [GM82] in their work on mental poker protocols, is to force parties to use the result of a common coin toss as their local randomness, to ensure parties do not rig their coins. In this case, the result of the coin toss will be known only to one party, Alice, and yet all other parties will be able to verify (via, say, zero-knowledge protocols) that Alice is using the result of the collective coins in her computations. This idea was later used in the compiler of [GMW87] from the $n$-party secure function evaluation protocol against an honest-but-curious adversary to one against a malicious adversary. Our coin tossing protocol can similarly be turned into one where only one party Alice knows the result but all other parties can verify (via, say, a leakage-resilient zero knowledge protocol [GJS11]) that Alice is using the result of the collective coins in her computations. Note, however, that this yields security only assuming an honest majority.

### 1.1.2 Leakage-Resilient Verifiable Secret Sharing

One of the tools in our construction, which is of independent interest, is a new *leakage-resilient verifiable secret sharing scheme.* Verifiable secret sharing (VSS) extends Shamir's [Sha79] secret sharing to ensure not only secrecy (i.e., corrupted parties do not gain information about the dealer's secret), but also unique reconstruction of a secret $s'$ even if the dealer and/or a subset of parties are dishonest, where for an honest dealer, $s'$ will be his original secret. *Weakly leakage-resilient (WLR)* VSS is a VSS scheme with the additional guarantee that given the view of any $(t, \lambda)$ adversary who corrupts up to $t$ parties *and leaks* a constant $\lambda$-fraction of each honest party's secret state (including the dealer's), the secret still retains a constant fraction of its original entropy. We refer to this property as weak leakage resilience. We now state our second main theorem.

**Informal Theorem (WLR-VSS):** Let $n = (3 + \delta)t$ for some constant $\delta > 0$. Then for any constants $\epsilon < 1$ and $\lambda \leq \frac{\delta(1-\epsilon)}{10+6\delta}$, there exists a $(t, \lambda)$-leakage resilient VSS protocol that runs in $O(1)$ rounds, with the following modified secrecy guarantee: If the dealer is honest during the sharing phase, with input distribution $S$, then for any $(t, \lambda)$ adversary $\mathcal{A}$, with high probability, given the view of $\mathcal{A}$ at the conclusion of the sharing phase, the distribution $S$ retains $\epsilon$ fraction of its original entropy.

WLR-VSS is sufficient for our coin tossing construction; however, we also define and obtain a stronger version of leakage-resilient VSS, in which given the view of a leaking adversary, the secret $s$ retains its *full* entropy. This stringent secrecy property rules out the possibility of standard VSS, since leakage from the dealer directly reveals information on $s$ once it has been sampled. We thus put forth a new notion and a construction of *oblivious* secret sharing, where a dealer shares a uniformly distributed secret whose value *he does not know.* We believe that this primitive can serve as a useful building block for constructing future leakage-resilient protocols, which anyway make use of VSS in this fashion (e.g., in [FM85] to achieve Byzantine Agreement). We refer the reader to Section 2.4.3 for details.

#### Disjoint Committee Election

As a second tool in our construction, we present a 1-round public-coin protocol for electing $\log^2 n$ *disjoint* "good" committees of size approximately $n^{1/2}$ from among $n$ parties. This is achieved using a modified version of the Feige committee election protocol [Fei99], in which several of the "lightest bins" are elected (see Section 2.5).

## 1.2 Secure Computation Against Adaptive Auxiliary Information

Historically, when formalizing security definitions for cryptographic protocols, it was noted that adversarial parties may enter a protocol with relevant knowledge *from the past.* Meaningful security definitions should thus embed the important requirement that, even armed with such side information, an adversary still must not be able to break the security of the protocol. For example, in a zero-knowledge proof system, it should not be the case that an adversarial verifier with partial information about a witness, can now suddenly recover the entire witness from the protocol. This natural property was formalized by requiring that, for any adversary with any potential auxiliary input $z$ on the inputs of the honest parties *prior* to the execution of the protocol, this adversary still learns *nothing* beyond the inputs and prescribed outputs of corrupted parties (and, of course, the auxiliary input $z$ it learned prior).

In the last two decades, as cryptographic protocols have become increasingly prevalent (often within everyday online activities, run in parallel), and as new classes of strong attacks have emerged, it has become increasingly evident that adversaries may also acquire auxiliary leakage information on the internal state of honest parties *during the protocol execution.* This may take place, e.g., by performing *physical attacks* on an implementation of a processor (say, a smart card or a hardware token) [Koc96, AK96, QS01, GMO01, OST06, HSH$^+$08], or when the randomness used by an honest party in a protocol is correlated with randomness used in other applications. Unfortunately, this case is no longer covered under historical definitions: the moment an adversary is able to learn any side information, say, about the randomness of an honest party in the protocol, the security guarantees break down.

In this work, we seek to extend the standard notion of security with (static) auxiliary inputs to the setting of general *adaptive* auxiliary information. We study secure general two-party and multi-party computation in the setting where an adversary, who corrupts an arbitrary subset of parties in the protocol, is able to learn arbitrary (efficiently computable) auxiliary information on the *entire states* of all the honest parties (including their inputs and random coins), in an *adaptive* manner, *throughout the protocol execution.* We formalize a meaningful definition of security within this setting, and construct two-party and multi-party computation protocols satisfying our definition.

We note that our work is greatly inspired by the recent work of Bitansky, Canetti, and Halevi [BCH11], who proposed the study of secure computation in the "leakage" setting, with the guarantee that all a "leaky" adversary can learn while running the protocol is the leakage itself. We refer the reader to Section 1.2.2 for details.

**How to Define MPC Secure Against Adaptive Auxiliary Information?** Security of MPC protocols is formalized by comparing the real-world protocol execution to an ideal-world experiment where the parties interact directly with a trusted party who receives all parties' inputs and responds only with the correct function output. Formally, an MPC protocol is said to be secure under the classical definition if for every real-world adversary with some auxiliary input (say) $z$, there exists an ideal-world adversary (a.k.a. the simulator) with the same auxiliary input $z$, who simulates the output of the real-world experiment.

Our goal is to generalize this definition to the setting where side information can be learned during the protocol. We model *adaptive* auxiliary information by allowing the adversary to specify (efficiently computable) functions $f_i$, adaptively throughout the protocol. For each such query, the selected function is evaluated on the entire secret states of the honest parties, and the result is given to the adversary as auxiliary information. Intuitively, we wish to guarantee that an adversary who participates in the protocol and receives adaptive auxiliary information throughout the protocol's lifetime still learns nothing beyond the inputs and outputs of the corrupted parties, and the auxiliary information.

Note that it is not immediately apparent how to formalize this notion. Whereas in the static setting both the adversary and the simulator receive the exact same auxiliary input $z$, in the adaptive setting, it doesn't make sense even syntactically to say that the *same* auxiliary input functions are applied both in the real world and in the ideal world: this is because the real-world auxiliary input function will expect to take as input the secret states of parties executing a protocol, whereas in the ideal world no protocol is ongoing.

A natural attempt to formalize security in this setting may be to require that if the real adversary learns $\ell$ bits of auxiliary information, then the simulator can also learn at most $\ell$ bits of auxiliary information. While this is a natural requirement (and our definition will achieve at least this requirement), unfortunately, it may be too weak. For example, the auxiliary information learnt by a real-world adversary (say, via physical processes) may be large but "noisy," giving very little information about the honest parties' inputs. Or, the honest parties' inputs may be information-theoretically determined but *computationally unpredictable*[3] given the real-world auxiliary information. In these cases, the above definition may not provide a meaningful security guarantee since the simulator may be able to simulate the honest parties "trivially" by first learning a large portion (or all!) of their inputs as auxiliary information. Ideally, we would like to formalize the intuitive requirement that the auxiliary information in the ideal world is "no more" than that in the real world.

**Our Security Definition.** We capture the desired security notion by placing additional restrictions on the ideal-world simulator. In particular, for each auxiliary input function $f$ that the real-world adversary generates, we require that the simulator generates a "translation function" $T$ that takes as input only the secret inputs of the honest parties, and generates "simulated states" for the honest parties at each point in the protocol. These simulated states should be computationally indistinguishable from the real states, and should be consistent with the simulated transcript.

[3]In other words, the honest parties' inputs have high unpredictability entropy.

Then, for each auxiliary input function $f$ that is requested in the real world, the *same* auxiliary information function is applied in the ideal world, but it is applied to the simulated states. In other words, the ideal world auxiliary function will be the composed function $f \circ T$. This prevents the ideal-world adversary from "learning too much" via the ideal-world auxiliary information: For example, if the requested function $f$ has short output, reveals only useless unused information, or leaves its inputs unpredictable, then the same restrictions will also hold for the ideal-world auxiliary information.

We say that an MPC protocol is *secure against adaptive auxiliary information* if for every PPT real-world adversary who makes arbitrary adaptive (efficiently computable) auxiliary information queries, there exists a PPT ideal-world simulator who, given the corresponding auxiliary information (as described above), is able to simulate the output of the real-world experiment. Intuitively, this definition guarantees that *the security of the honest parties "gracefully degrades" with the amount of auxiliary information that the real adversary is able to obtain.* We refer the reader to Section 3.3 for more details. We also remark that our definition is similar to that of [BCH11, GJS11], occurring within the setting of zero-knowledge and protocols against passive adversaries.

### 1.2.1 Our Results

We construct *two-party* and *multi-party* computation protocols for any efficiently computable function secure against adaptive auxiliary information in the universal composability (UC) framework [Can05] in the common reference string (CRS) model. Namely, we prove the following theorem:

**Informal Theorem (MPC with Adaptive Auxiliary Information).** *For any $n \geq 2$, there exists a UC-secure $n$-party protocol in the CRS model for evaluating any efficiently computable function, such that for any malicious PPT adversary who (statically) corrupts any subset of parties and learns any amount of (efficiently computable) adaptive auxiliary information $Z$, this adversary learns nothing beyond the inputs and outputs of corrupted parties, and the same auxiliary information $Z$ (formalized as discussed above). This holds based on the linear assumption over bilinear groups and the $n$-th residuosity assumption.*[4]

**No bound on the auxiliary information.** We emphasize that, as in the classical (static auxiliary input) setting, our result does *not* require any a priori bound on the amount of the auxiliary information that the adversary may be able to learn. Instead, our protocol guarantees that for *any* amount of information the real-world adversary is able to (adaptively) acquire throughout the protocol, this "same amount" of auxiliary information is given to the ideal-world simulator, thus providing *graceful degradation* of security. This advantageous property is in contrast with nearly all existing results in leakage-resilient cryptography, which require the user to specify *at design time* an amount of information leakage he wishes to protect against (growing the system parameters accordingly); if an adversary is able to garner more leakage at runtime than the preset bound, then security of these schemes no longer hold.

**New Techniques.** The technical heart of our result is a new construction of an oblivious transfer (OT) protocol that is secure against adversaries that may use "bad" randomness during the

[4]The $n$-th residuosity assumption can be replaced with any lossy trapdoor function (LTDF) with some specific properties. Roughly speaking, we require LTDFs that are bijective and "sufficiently lossy".

protocol, and in addition receive adaptive auxiliary information on the secret state of the honest party. Note that in the plain model (where adaptive auxiliary information is not allowed), achieving security against adversaries who may use bad randomness is straightforward, by simply using a standard coin-tossing protocol. In contrast, as we discuss in Section 3.1, constructing a coin-tossing protocol that is secure against adaptive auxiliary information is, in fact, impossible [CLL+13].

We introduce a new approach for generating private randomness for parties *non-interactively* (and reusably) via a CRS by using lossy trapdoor functions to prevent the adversary from choosing randomness in a small "bad" set. The non-interactive nature of our randomness generation procedure allows us to tolerate an *unbounded* amount of adaptive auxiliary information (as stated above, this is contrast with nearly all existing works in leakage-resilient cryptography). We additionally demonstrate a method for effectively embedding this randomness generation procedure within a cryptographic application, using lossy keys to guarantee information theoretic security in the underlying application as long as the adversary's randomness does not fall within a very small "bad" set. These techniques may prove useful outside the present work.

**Application to Leakage-Resilient Protocols.** There has been an extensive amount of work on leakage-resilient cryptography in recent years, primarily focused on the setting of *non-interactive* primitives (e.g., [ISW03, DP08, AGV09, DKL09, NS09, KV09, BKKV10, DHLW10b]). Our result can be used to achieve leakage-resilient *interactive protocols* with input privacy guarantees, an area that has received comparatively little attention (see Section 1.2.2 for details).

As alluded to above, our MPC protocol directly provides meaningful security guarantees in the setting of leakage: where such a "leaky" adversary learns no more than the inputs and outputs of the corrupted parties, and the leakage information. This can be seen by viewing the adaptive auxiliary information as joint leakage on the secret states of the honest parties during the protocol execution. We note, however, that this differs from the security model considered in [BCH11], where leakage on the state of each party is "disjoint" in both the real- and ideal-world experiments. Indeed, achieving security in such a model is an interesting open problem.

Further, when combined with previous work on leakage-resilient cryptography, our result yields applications where *"standard"* security is guaranteed in the face of *bounded* leakage. Below, we discuss two such applications:

- *Leakage-Resilient MPC in the leak-free preprocessing model.* A recent work of Boyle et al. [BGJK12] (described in Chapter 4) builds upon our results to construct multi-party secure computation protocols to achieve *standard ideal-world security* against real-world adversaries that may leak continuously from the secret state of each honest player *separately*, assuming a one-time leak-free preprocessing phase and a large number of parties. At a very high level, they achieve their result by applying our *multi-party* secure computation protocol to the leakage-resilient computation compiler of Goldwasser and Rothblum [GR12]. We stress that they are able to achieve *standard ideal-world security* (where no leakage is allowed in the ideal world), despite the fact that our result only achieves "leaky" ideal-world security (where the simulator is given leakage in the ideal world). Thus, even if end-user applications only consider *standard security* as a realistic model, our results can still be an important tool.
- *Leakage-resilient threshold cryptosystems.* In a threshold cryptosystem [DF89], multiple parties hold shares of a secret key, and only a quorum of parties can jointly execute the corre-

sponding secret functionality (e.g., decryption). Our MPC protocol, when combined with an underlying cryptographic primitive that is resilient to leakage on the secret key (e.g., [AGV09] for public-key encryption), yields a corresponding leakage-resilient threshold cryptosystem. Specifically, the security guarantee of our protocol implies that any information garnered by an adversary who controls an arbitrary strict subset of a quorum, and obtains arbitrary leakage on the joint secret states of all honest parties during the collective decryption protocol, reduces to simply the output value and corresponding leakage on the underlying secret key.

### 1.2.2 Related Work

The notion of security considered in this paper is somewhat analogous to that considered by Bitansky, Canetti, and Halevi [BCH11] and Garg, Jain, and Sahai [GJS11], in the context of leakage.

Garg *et. al.* [GJS11] consider zero-knowledge proof systems where a malicious verifier can adaptively leak arbitrary information on the state (i.e., witness and randomness) of the honest prover during the execution of the protocol. They consider a definition that intuitively guarantees that such an adversarial verifier does not learn anymore than what it may learn by leaking directly on the witness. They present protocols satisfying this definition both in the interactive and non-interactive setting, based on standard assumptions.

Bitansky *et. al.* [BCH11] put forth a general definition of leakage tolerance in the setting of two-party interactive protocols, extending the notion of universally composable security. As in the model of [GJS11], the simulator is allowed access to leakage on the input of the honest party. Bitansky *et. al.* present protocols achieving this notion against a *semi-honest* adversary, for a collection of specific tasks (secure message transmission, oblivious transfer, and commitment); and also construct a zero knowledge proof system with such guarantees in the CRS model. In addition, they prove a composition theorem for leakage-tolerant protocols. We use this composition theorem in our work.

A concurrent and independent work of Damgård, Hazay, and Patra [DHP11] constructs a general two-party secure function evaluation protocol in the leakage setting for semi-honest adversaries. Damgård *et. al.* formalize a security definition along the lines of entropic leakage as in [NS09, HL11], overcoming the limitations of a bounded leakage definition, and provide constructions realizing their definition in the semi-honest setting. We note that our results, cast in the leakage context, also satisfy their definition (for the case of 2-party protocols). A notable difference with our work is that Damgård *et. al.* deal with semi-honest adversaries, whereas malicious security is the focus of this work, and we address the multi-party case.

## 1.3 Multi-Party Computation Secure Against Continual Memory Leakage

In this work, we study MPC in the setting where an adversary, who corrupts an arbitrary subset of parties in the protocol, can also leak information about the entire secret state of each honest party throughout the protocol execution (except during a designated leak-free preprocessing stage). As has become customary in the field, leakage is modeled by allowing the adversary to query leakage functions, as follows. Each leakage function is computed by an arbitrary poly-size circuit, with bounded output-length, which is applied to the secret state of an honest processor. The adversary may choose the leakage functions adaptively, based on the entire history of communication, previous leakage, and internal state of corrupted processors.

The security guarantee we aim for and will achieve, is that any adversary in the above leakage model, *does not learn anything beyond the inputs of the corrupted parties and output values of the functions* computed by the MPC protocol. This is formalized via the standard real/ideal world paradigm. In the ideal world, parties do not interact directly, but rather send their inputs to an "ideal functionality", who computes the function for them, and sends them the output. There is no leakage in the ideal world. An MPC protocol is said to be secure, if for every "real world" leakage adversary $\mathcal{A}$ (as above) there exists an "ideal world" simulator $S$, such that the output of all the parties (including the adversary) in the real world, is computationally indistinguishable from the output of all the parties (including the simulator) in the ideal world.

***Weakly* Leakage-Resilient MPC.** We note that recently there have been several results that consider the problem of constructing leakage-resilient MPC protocols [GJS11, BCH11, DHP11, BGJ$^+$13]. However, in contrast to the security guarantee we consider here, all these results give a *weakened* security guarantee: an adversary that runs the protocol and leaks $\ell$ bits about the honest parties' secret state, does not learn more than the output of the function being computed, *and an additional $\ell$ bits* about the private inputs of the honest parties. We note that in some applications, leakage of $\ell$ bits on the private inputs of the honest parties could be detrimental to the security of the entire MPC protocol. For example, say the function to be computed by the MPC protocol is to tally up the binary votes of the parties. Then, the $\ell$ bits can be exactly the complete $\ell$ votes of any $\ell$ honest parties, rendering the protocol useless.

Interestingly, we use the result in [BGJ$^+$13], which constructs an MPC protocol with the weak security guarantee, as a building block to construct a leakage-resilient MPC protocol with the classical (strong) security guarantee.

**Security Against Continual Leakage.** We further remark that the weaker security notion previously achieved cannot be extended meaningfully to continual leakage in the MPC setting. That is, it cannot address the setting where the $n$ users do not just perform a one-shot MPC protocol, but rather engage in an unbounded number of MPC protocols for many functions, and during each MPC invocation the adversary leaks $\ell$ bits from each of the honest party's internal state. This is obvious, as allowing the repeated leakage of new $\ell$ bits of information on the honest parties' inputs would eventually leak the honest parties' inputs in their entirety. For example, in the setting where a set of parties jointly compute a threshold decryption function (as described above), they may want to carry out many decryption computations, where leakage happens repeatedly. Since each $\ell$ bits of leakage corresponds to $\ell$ bits of leakage on the decryption key, the decryption key may eventually be completely leaked! Nonetheless, we use the result of [BGJ$^+$13] as a building block to achieve our stronger *continual leakage* security guarantee.

### 1.3.1 Our Result: Continual Leakage-Resilient MPC

In this work, we construct a leakage-resilient MPC protocol for any function $f$, *without weakening the security guarantee.* We consider a continual setting, where parties over time compute many functions on their inputs. Our security guarantee is that an adversary does not learn *anything* beyond the inputs of the corrupted parties and the output of the functions computed, even if he continually leaks information about the honest parties' secret states throughout the protocol executions. Parties' secret states are periodically updated via an update procedure, during which the adversary can continue to leak information. We allow each of the adversary's leakage functions to be an *arbitrary* (shrinking) polynomial time computable function of the *entire* secret state of each honest party (separately), and these leakage functions can be chosen adaptively on all information the adversary has seen thus far.

**Informal Theorem (LR-MPC):** Under (standard) intractability assumptions, for every constant $\epsilon > 0$ there exists an MPC protocol for computing an unbounded number of functions among $n$ parties of which at least $\epsilon$ fraction are honest. The protocol is secure against *continual* leakage, assuming a *one-time* leak-free pre-processing stage, and where the security parameter is polynomially related to the number of parties $n$.

A few remarks about our result statement are in order.

**Standard "Ideal World" Security.** We formalize security via the real/ideal paradigm, and as stated above, we guarantee the *standard* ideal world security. More specifically, we consider a real world execution where a ppt adversary $\mathcal{A}$ that controls a subset of the parties, can additionally leak in an adaptive manner on the secret states of each honest party, throughout the protocol execution (except the "leak-free" pre-processing stage; see below). We require that for any such adversary there exists a probabilistic polynomial-time adversary $\mathcal{S}$ (referred to as the simulator) in the ideal world who corrupts the same set of parties, and interacts with the trusted party who computes the desired functionality, and simulates the view of the adversary $\mathcal{A}$ in the real world in an indistinguishable manner. Note that unlike the previous works, the simulator in our definition is *not* allowed to leak on the honest parties' inputs.

A formal description of our security model is given in Section 4.3.

**"Leak-free" Pre-Processing.** We assume the existence of a leak-free pre-processing stage. We stress that this is a *necessary* assumption to obtain our strong guarantee, since otherwise an adversary can simply leak $\ell$ bits about an honest party's secret input, before the MPC even commences. More generally, we note that such a leak-free pre-processing stage is a necessary step in the construction of any leakage-resilient cryptographic primitive which receives a secret input, and where the security guarantee is that the secret input does not leak. This is the case, for example, in the compilers of [ISW03, FRR$^+$10, JV10, GR10, GR12], which transform algorithms with a secret state into a functionally equivalent leakage-resilient variant of the same algorithm.

We remark that our pre-processing stage in fact has the nice property that it can be decomposed into two parts, namely, (a) an interactive pre-processing phase that is *independent* of the parties' inputs and the functions to be computed, and (b) a *non-interactive* input dispersal phase. We stress that the first phase is run only *once* in the beginning of time, before the parties know what their inputs are or what functions they wish to compute. The second (non-interactive) phase is run whenever the parties choose a set of inputs.

While both of these parts are assumed to be "leak-free", we do allow leakage between them. We refer the reader to Section 4.3 for a formal description of our model.

**Multi-function MPC and Continual Leakage.** We note that in the standard (leak-free) MPC literature, one typically considers a one-shot MPC protocol, as opposed to considering the setting where the parties compute an unbounded (polynomial) number of functions. The reason we focus on the latter setting, is to emphasize that we need to run the leak-free preprocessing stage only *once*, and then the parties can compute any unbounded number of functions $f_1, \ldots, f_\ell$ in a leaky environment.

We further emphasize that we allow the adversary to leak *continuously* on the secret states of the parties during the (unbounded) computations; the only (necessary) requirement is that the secret states of the parties are periodically updated (since otherwise they will eventually be completely leaked). However, the adversary is allowed to leak even *during* each update procedure. We do not bound the total number of bits that the adversary leaks, but rather only bound the leakage rate: i.e., the number of bits leaked *between updates.*

**Extending to Multi-input MPC** We stated our theorem for the case of computing many functions on a single set of inputs. However, our construction is easily extended to the many-input case. Whenever a party chooses a new input, the (leak-free) non-interactive input phase described above can be repeated. Namely, party $P_i$ on new input $x_i$ performs a local computation on $x_i$, sends a message to the other parties, and erases $x_i$. One may think of this model as a "hot potato" model, where the parties never store their inputs for very long (since they are concerned with leakage), but rather immediately share their input (as if it were a "hot potato").

**Number of Parties vs Security Parameter.** Notice that in our theorem, the security parameter is polynomially related to the number of parties. Namely, the security increases with the number of parties. Therefore, this theorem is meaningful only when the number of parties in the MPC protocol is large. One may ask whether this restriction on the number of parties being large, or the restriction that an $\epsilon$-fraction is honest, is inherent, or whether it is simply an artifact of our techniques. Unfortunately, it turns out that this restriction cannot be removed altogether. In particular, one can prove that there does not exist a secure leakage-resilient *two-party* computation protocol in our model.[5] Similarly, one can show that there does not exist a secure leakage-resilient MPC protocol if all the parties except one are malicious. Moreover, it turns out that proving this

[5]The reason is the following: Assume the adversary controls party $P_1$. In this case, he knows the entire secret state $s_1$ of $P_1$, and can choose his leakage function $L$ to depend on $s_1$; i.e., $L = L_{s_1}$. Note that $L$ is a function that takes as input the secret state $s_2$ of $P_2$. Thus the adversary can leak any (shrinking) function $g(s_1, s_2)$ by setting $L_{s_1}(s_2) \triangleq g(s_1, s_2)$. Recall that from the shares $(s_1, s_2)$ the parties can compute any function of the original inputs $(x_1, x_2)$. Therefore, the function leaked can be an arbitrary function of the original inputs. Clearly, such leakage cannot be simulated in the ideal world.

theorem for constant number of parties implies an "OCL compiler" (without leak-free hardware) that has only a constant number of sub-computations (or "modules"), which is an interesting open problem on its own.

**Assumptions.** In our construction, we rely on several underlying cryptographic primitives, including a fully homomorphic encryption (FHE) scheme [Gen09, BGV11], a non-interactive zero-knowledge (NIZK) proof-of-knowledge system [FLS90], a standard MPC protocol [GMW87], an equivocal commitment scheme [FS89], a weakly leakage-resilient MPC protocol [BGJ$^+$13], and an LDS compiler [BCG$^+$11] (which can be thought of as a stronger version of an OCL compiler as in [JV10, GR10, GR12]). These primitives have been shown to exist under various (standard) computational intractability assumptions. The FHE scheme and the NIZK scheme that we use are not required to be leakage resilient, nor do we know of such schemes.

We note that the use of FHE in our construction is solely to achieve independence between the complexity of the functions computed by the MPC and the number of parties required, as well as to reduce the interaction between parties.[6]

We refer the reader to Section 4.2 for details on these primitives, and the corresponding assumptions.

**Applications.** We demonstrate the application of our result to the problem of delegating multiparty computation to outside servers. Generally, the setting is of a large set of parties who need to perform a joint computation, and they would like a service (such as Amazon) to do the computation for them. However, they do not trust any one server, and further believe that any server can be leaked upon.

Usually, MPC provides a solution around the trust problem by using several servers, as follows: Each party secret shares her input, and gives one share to each server; then the servers carry out the desired computation by running an MPC protocol; finally, one argues that if there are sufficiently many honest parties, then security is guaranteed. However, if an adversary can obtain leakage information from the honest servers, then this is no longer true. To argue security in the leaky setting, the servers will need to run a *leakage-resilient* MPC protocol. Moreover, if the servers compute many functions on the secret inputs, then they will need to run an MPC protocol that is secure against *continual leakage*. Let us demonstrate three examples of this setting.

- *Electronic election*: Say an electronic election among many voters is to be held. Clearly running an MPC protocol among all voters is prohibitive, since it requires interaction between every two voters. Instead, the MPC protocol is run by a proxy of $n$ servers. Since these

[6]We also emphasize that, while FHE immediately solves the related problem of computing on encrypted data, FHE does *not* suffice for our purposes. To illustrate, suppose the parties collectively generate a public key $\mathsf{pk}$ for the FHE scheme, so that they each hold a secret share of the corresponding secret key, and then each goes to publish an encryption of their input $x_i$. Then for any efficiently computable function $f$, they can easily produce *an encryption* of the desired output, $\mathsf{Enc}_{\mathsf{pk}}(f(\vec{x}))$. However, the challenge is (even for a one-shot function computation) how to enable the parties to collectively *decrypt* this ciphertext and reveal $f(\vec{x})$ itself, while simultaneously ensuring that the adversary (who can corrupt nearly all of the parties, and leak on all the rest) is not able to learn any information on the $x_i$'s.

servers compute on very sensitive information, attackers may try to employ various side-channel attacks to learn this information. Thus, to ensure the secrecy of the individual votes, the servers should run a *leakage-resilient* MPC protocol.

- *Medical Data*: One may envision a huge database which contains the medical data of every patient in the US. To compute any global statistic on this data, one would not want to put complete trust in any single database. Instead, it is distributed to $n$ different databases. Each time they need to compute statistics on this data, they engage in an MPC protocol. As in the voting example, since these databases contain very sensitive information, an adversary may try to obtain this information via a leakage attack. Thus, to ensure security, the databases must run an MPC protocol that is secure against *continual leakage.*

- *Differential Privacy*: In the area of differential privacy, great care is taken to ensure that the data of individuals is protected. However, usually it is assumed that there is an *honest* curator, and that the people in the database hand their secret data to this curator. However, it seems likely that people may not trust any single curator with highly sensitive information (such as whether they do or do not have a disease which may scare off life insurance providers). Thus, as in the previous examples, this trusted curator can be replaced by a multitude of parties of which only a small fraction is assumed to be honest. Moreover, if these parties compute on the database using a leakage-resilient MPC protocol, then security is guaranteed even if all the honest parties are leaked upon (as long as some $\epsilon$-fraction of the honest parties are not *fully* leaked upon).

### 1.3.2 Related Work

**Leakage-Resilient Non-Interactive Primitives.** There has been an extensive amount of research on leakage-resilient cryptography in the past few years. Most prior works construct specific leakage-resilient *non-interactive* primitives, such as leakage-resilient encryption schemes and leakage-resilient signature schemes [DP08, AGV09, Pie09, DKL09, ADW09, NS09, KV09, DGK+10, FKPR10, ADN+10, KP10, GR10, JV10, BG10, BKKV10, DP10, DHLW10a, DHLW10b, LRW11, MTVY11, BSW11, LLW11, DLWW11, BCG+11].

***Weakly* Leakage-Resilient Interactive Protocols.** There has also been prior work on the problem of constructing leakage-resilient interactive protocols [GJS11, BCH11, BGK11, DHP11, BGJ+13]. Garg *et. al.* [GJS11] present a leakage-resilient zero-knowledge proof system. Bitansky *et. al.* [BCH11] present leakage-resilient protocols for various functionalities (such as secure message transmission, oblivious transfer, and commitments) which are secure against semi-honest adversaries, and also zero knowledge, in the UC framework. Boyle *et. al.* [BGK11] present a leakage-resilient multi-party coin tossing protocol. Dåmgard, Hazay, and Patra [DHP11] present a leakage-resilient two-party secure function evaluation protocol for functions in $NC^1$ in the *semi-honest* setting. Finally, very recently Boyle *et. al.* [BGJ+13] constructed a general leakage-resilient MPC protocol that is secure in the UC setting against malicious adversaries (as described in Chapter 3).

However, all the results in the interactive setting mentioned above offer a *weakened* security guarantee, that an adversary that leaks $\ell$ bits in the real world, gains at most $\ell$ bits of secret information about the secret inputs of the parties.[7] Moreover, the $\ell$ bits of secret information gained is an arbitrary (poly-size) function of the *joint* inputs $x_1, \dots, x_n$.

**Only Computation Leaks Model.** Finally, we mention that various leakage models have been considered in the literature, which restrict the leakage functions in different ways. Most notable is the *only computation leaks* (OCL) model of Micali and Reyzin [MR04]. The axiom of this model is that secret information that is merely stored in memory does not leak, but any information that is used during a computation may leak.

Several results prove security for specific cryptographic primitives in the OCL leakage model [DP08, Pie09, FKPR10]. More generally, it is known how to convert *any* circuit into one that is secure in the OCL model [GR10, JV10, GR12]. In particular, a recent work of Goldwasser and Rothblum [GR12] shows how to do this unconditionally, making no intractability assumptions, and without resorting to secure leak-free hardware, unlike the previous works. Specifically, Goldwasser and Rothblum construct an efficient compiler that takes any circuit (with some secret values hard-wired) and converts it into a leakage-resilient one, consisting of several *modules*, each of which performs a specific sub-computation. The security guarantee is that an adversary, who at any point of time throughout the computation obtains bounded leakage from the "currently active" module, does not learn any more information than having black-box access to the circuit. We will use a variant of this result (namely, an LDS compiler) to construct our leakage-resilient MPC scheme. In particular, we use [GR12] as a building block in our construction. See Section 4.1 for details.

We stress that our result does *not* use the OCL assumption, and we allow the adversary to compute leakage functions on *everything* held in the memory of each party (except during the pre-processing phase and during the input phase).

## 1.4 Communication Locality in Secure Multi-party Computation

These days, an emerging area of potential applications for secure MPC is to address privacy concerns in data aggregation and analysis to match the explosive current growth of the amount of available data. Large data sets, such as medical data, transaction data, the web and web access logs, or network traffic data, are now in abundance. Much of the data is stored or made accessible in a distributed fashion. This necessitated the development of efficient distributed protocols that compute over such data. In order to address the privacy concerns associated with such protocols, cryptographic techniques such as MPC for SFE where *data items are equated with servers* can be utilized to prevent unnecessary leakage of information.

However, before MPC can be effectively used to address today's challenges, we need protocols whose efficiency and communication requirements scale practically to the modern regime of massive data. An important metric that has great effect on feasibility but has attracted surprisingly little attention thus far is the *number of other parties* that each party must communicate with during

[7]An exception is the work of [BGK11] that considered the specific coin-tossing functionality, where the parties do not have any secret inputs.

the course of the protocol. We refer to this as the communication *locality*. Indeed, if we consider a setting where potentially hundreds of thousands, or even millions of parties are participating in a computation over the internet, requiring coordination between each pair of parties will be unrealistic.

In this work, we work to optimize the communication locality for general secure function evaluation on data which is held distributively among $n$ players connected via a complete synchronous communication network, of whom $(\frac{1}{3} - \epsilon)n$ may be computationally bounded Byzantine faults. We do *not* assume the existence of broadcast channels.

We also focus on a particularly interesting setting in which the randomized function $f$ to be computed is a *sublinear algorithm*: namely, a random execution of $f(x_1, ..., x_n)$ depends on at most $q = o(n)$ of the inputs $x_i$. We consider both non-adaptive and adaptive sublinear algorithms, in which the identities of the selected inputs may depend on the randomness $r$ of execution, or on both $r$ and the values of $x_i$ queried thus far. Sublinear algorithms play an important role in efficiently testing properties and trends and computing on large data sets. Here, the sublinear query complexity makes it possible in principle to dramatically reduce the *amount* of information that needs to be communicated within the protocol. However, the challenge is to achieve this while maintaining security—in particular, keeping the identities of the selected inputs completely hidden.

Straightforward application of known general MPC techniques results in protocols where each party sends and receives messages from all $n$ parties, and where the overall communication complexity is $O(n^2)$, regardless of the complexity of the function to be computed. We remark that this is obviously the case for the classical general SFE protocols (beginning with [GMW87, CCD87, BGW88]) in which every party first secret shares its input among all other players, and exchanges messages between all $n$ parties at the evaluation of every gate of the circuit of the function being computed. Furthermore, although much progress was made in the MPC literature of the last two decades to make MPC protocols more efficient and suitable for practice, it is still the case both in works on scalable MPC [DI06, DN07, DIK$^+$08, DIK10] and more recent works utilizing the existence of fully homomorphic encryption schemes [LATV12, AJLA$^+$12] for MPC. The latter achieve communication complexity that is independent of the circuit size, but not of the number of players when broadcast channels are not available.

A recent notable exception to the need of each player to communicate with all other players is the beautiful work of King, Saia, Sanwalani and Vee [KSSV06] on what they call scalable protocols for the Byzantine agreement and leader election problem, requiring each honest party to send and process a $\mathsf{polylog}(n)$ number of bits. On the downside, the protocols of [KSSV06] do not guarantee that all honest parties will achieve agreement but only guarantee that $(1 - o(1))$ fraction of the good processors reach agreement—achieving only the so-called *almost everywhere agreement*. In another work of King et al [KLST11], it is shown how using $\tilde{O}(\sqrt{n})$ communication, full Byzantine agreement can be achieved. The technique of almost everywhere leader election of [KSSV06] will be the technical starting point of our work.

### 1.4.1 Our Results

We provide multi-party computation protocols for general secure function evaluation with communication locality that is *polylogarithmic* in the number of parties. That is, starting with $n$ parties connected via a complete and synchronous network, nearly one third of which are Byzantine faults, the protocol requires each party to send messages to (and process messages from) at most $\mathsf{polylog}(n)$ other parties in the network, using $\mathsf{polylog}(n)$ rounds. Our main theorem is as follows:

**Theorem 1 (MPC with Communication Locality).** Let $f$ be any polynomial-time randomized functionality on $n$ inputs. Then, for every constant $\epsilon > 0$, there exists an $n$-party protocol $\Pi_f$ that securely computes a random evaluation of $f$, tolerating $t < (1/3 - \epsilon)n$ statically scheduled active corruptions, with the following complexities:

(1) Communication locality: $\mathsf{polylog}(n)$.

(2) Round complexity: $\mathsf{polylog}(n)$.

(3) Message sizes: $O(n \cdot l \cdot \mathsf{polylog}(n))$, where $l = |x_i|$ is the individual input size.

(4) The protocol uses a setup consisting of $n \cdot \mathsf{polylog}(n)$ signing keys of size $\mathsf{polylog}(n)$, as well as a $\mathsf{polylog}(n)$-long additional common random string (CRS).[8]

The protocol assumes a secure multisignature scheme, a fully homomorphic encryption (FHE) scheme, simulation-sound NIZK arguments, as well as pseudorandom generators.

Assuming *only* a standard signature scheme and semantically secure public-key encryption, and setup as in (4), there exists a protocol for securely computing $f$ with $\mathsf{polylog}(n)$ communication locality.

Multi-signatures [MOR01, LOS+06] are digital signatures which enable the verification that a large number of signers have signed a given message where the number of signers is not fixed in advance. The size of a multisignature is independent of the number of signers, but in order to determine their identities one needs to attach identifying information to the signatures. Standard instantiations of such schemes exist under the bilinear computational Diffie-Hellman assumption [Wat05, LOS+06].

The use of multi-signatures rather than standard digital signatures enables us to bound the *size* of the messages sent in the protocol. Further, the use of FHE enables us to bound the *number* of messages sent as in the theorem, rather than depend on the time complexity of the function $f$ to be computed and polynomially on the input size. However, we can obtain the most important feature of our complexity, the need of every player to send messages to (and process messages from) only $\mathsf{polylog}(n)$ parties in the network, solely under the assumption that digital signatures and public-key encryption exist.

In addition, we show how to convert an arbitrary sublinear algorithm with query complexity $q = \mathsf{polylog}(n)$ into a multi-party protocol to evaluate a randomized run of the algorithm $S$ with $O(\mathsf{polylog}(n))$ communication locality and rounds, and where the *total communication complexity* sent by each party is only $O(\mathsf{polylog}(n) \cdot (l + n))$ for $l = |x|$ an individual input size. We prove that participating in the MPC reveals no information beyond the output of the sublinear algorithm execution using a standard Ideal/Real simulation-based security definition. In particular, corrupted parties do not learn any information on he randomness used during the execution orwhich inputs $x_i$ were inspected by the algorithm.

For underlying query complexity $q$, our second main theorem is as follows:

[8]Adversarial corruptions may be made as a function of this setup information.

**Theorem 2 (MPC for Sublinear Algorithms).** Let $\mathsf{SLA}$ be a sublinear algorithm which retrieves $q = q(n) = o(n)$ different inputs. Then, for all constant $\epsilon > 0$, there exists an $n$-party protocol $\Pi_{\mathsf{SLA}}$ that securely computes an execution of the sublinear algorithm $\mathsf{SLA}$ tolerating $t < (1/3 - \epsilon)n$ statically scheduled active corruptions, with the following complexities, where $l$ is the size of the individual inputs held by the parties:

(1) Communication locality: $q \cdot \mathsf{polylog}(n)$.

(2) Round complexity: $O(q) + \mathsf{polylog}(n)$.

(3) Message sizes: $O((l + n) \cdot \mathsf{polylog}(n))$.

(4) The protocol uses a setup consisting of $n \cdot \mathsf{polylog}(n)$ signing keys of size $\mathsf{polylog}(n)$, as well as a $\mathsf{polylog}(n)$-long additional CRS.

The protocol assumes a secure multisignature scheme, an FHE scheme, simulation-sound NIZK arguments, and pseudorandom generators.

**Remarks.** A few remarks are in order.

*Flooding by faulty parties.* There is no limit (nor can there be) on how many messages are sent by faulty players to honest players. To address this issue in [KSSV06, KLST11, KS11, DKMS12], for example, it is (implicitly) assumed that the authenticated channels between players can "recognize" messages from unwarranted senders which should not be processed and automatically drop them, whereas we will use a digital signature verification procedure to recognize and drop these messages which should not be processed.

*Security definition for sublinear algorithms.* The security definition we achieve is the standard definition of secure multi-party computation (MPC). Informally, the parties will receive the output corresponding to a random execution of the sublinear algorithm but nothing else. Formally, we use the ideal/real simulation based type definition. We note that in works of [HKKN01, FIM$^+$06, IW06] on MPC for approximation algorithms for functions $f$, privacy is defined as to mean not to reveal information beyond the exact value of $f$ rather than beyond the approximate value of $f$ computed by the protocol. One may ask for a similar privacy definition for sublinear algorithms which are an approximation algorithm of sorts. However this is an orthogonal concern to the one we address in this work.

### 1.4.2 Further Related Work

Work on MPC in partially connected networks such as the recent work of Chandran, Garay and Ostrovsky [CGO10, CGO12] shows MPC protocols for network graphs of degree $O(\mathsf{polylog}(n))$ (thus each player is connected to no more than $\mathsf{polylog}(n)$ players). They can only show how to achieve MPC amongst all but $o(n)$ honest players. Indeed, in the case of a (static) partially connected network, it is unavoidable for some of the honest players to be cut out from every other honest player. In contrast, in the present work, we assume that although the $n$ players are connected via a complete network and potentially any player can communicate with any other player, our protocols require each honest party to communicate with only at most polylog($n$) players whose identity is only determined during the course of the protocol execution.

The problem of sublinear communication in MPC has also been considered in the realm of *two-party* protocols, e.g. by [NN01] who provide communication-preserving protocols for secure function evaluation (but which require superpolynomial computational effort), and in a recent collection of works including [GKK$^+$11] which achieve amortized sublinear time protocols, and the work of [IW06] which show polylogarithmic communication for specific functions.

An interesting point of comparison to our result is the work of Halevi, Lindell and Pinkas [HLP11]. They design computationally secure MPC protocols for $n$ parties in which one party is singled out as a server and all other parties communicate directly with the server in sequence (in one round of communication each). However, it is easy to see that protocols in this model can only provide a limited privacy guarantee: for example, as pointed out by the authors, if the last $i$ parties collude with the server then they can always evaluate the function on as many input settings as they wish for variable positions $(n-i), (n-i+1), \ldots, n$. No such limitations exist in our model.

In a recent and independent work to the current paper, King et al [DKMS12] extends [KLST11] to show a protocol for unconditionally secure SFE for general $f$ that requires every party to send at most $O(\frac{m}{n}+\sqrt{n})$ messages, where $m$ is the size of a circuit representation of $f$. A cursory comparison to our work shows that in [DKMS12] each party sends messages to $\Omega(\sqrt{n})$ other parties.

Finally, let us point out that our approach to anonymize access patterns to parties is similar in spirit to problems arising in the context of Oblivious RAM [GO96], and uses similar ideas to the obfuscated secret shuffling protocols of Adida and Wilkström [AW07].

# Chapter 2

# Leakage-Resilient Coin Tossing

In this work, we present an $O(1)$-round protocol for collectively generating an unbiased common coin, in the presence of leakage on the local state of the honest parties. We tolerate $t \leq (\frac{1}{3} - \epsilon)n$ computationally unbounded statically scheduled Byzantine faults and in addition a $\Theta(1)$-fraction leakage on each (honest) party's secret state. Our results hold in the memory leakage model of [AGV09] adapted to the distributed setting.

An additional contribution of this work is a tool we develop in order to achieve collective coin tossing—*leakage-resilient verifiable secret sharing (VSS)*. Informally, this is a variant of ordinary VSS in which secrecy guarantees are maintained even if information is leaked on individual shares of the secret.

We begin in Section 2.1 by giving a high-level overview of our solution. In Section 2.2, we introduce some preliminaries and notation. Section 2.3 contains a discussion on the leakage model considered. In Section 2.4, we define and construct a leakage-resilient verifiable secret sharing scheme, a tool used in our construction. In Section 2.5, we present a protocol for electing several disjoint committees. Section 2.6 contains the construction and proof of our leakage-resilient coin tossing protocol.

## 2.1 Overview of Our Solution

Let us first see why simple and known coin tossing protocols are not resilient to leakage. Consider the following well-known coin tossing protocol paradigm: First, each party $P_i$ chooses a random value $r_i$ and secret shares it to all other parties using a *verifiable secret sharing* (VSS) protocol. Then, all the parties reveal their shares and reconstruct $r_1, \ldots, r_n$. Finally, the parties output $\oplus r_i$. This protocol is not resilient to leakage for several reasons.

First, the reduction from coin tossing to VSS fails. For example, a malicious party $P_j$ can simply leak from each party $P_i$ the least significant bit of $r_i$, and then choose $r_j$ such that the xor of these least significant bits is zero. Thus, the problem is that in the leaky setting, we cannot claim that the $r_i$'s look random to the adversary. Instead, all we can claim is that they have high min-entropy. To address this problem, the first idea is to use a *multi-source extractor* instead of the xor function. Namely, output $\mathsf{Ext}(r_1, \ldots, r_n)$, where $\mathsf{Ext}$ is an extractor that takes $n$ independent sources and outputs a string that is statistically close to uniform. Note however, that we cannot use just any such multi-source extractor, since some of the sources (i.e., some of the $r_j$'s) may be chosen maliciously. Thus, what we need is a multi-source extractor that outputs a (statistically close to) uniform string, even if some of the sources are arbitrary, but independent of the "honest" sources. Indeed, such an extractor was constructed by Kamp, Rao, Vadhan and Zuckerman [KRVZ06].

Secondly, VSS protocols by and large are not resilient to leakage. Consider a single VSS protocol execution in the above paradigm. If the adversary leaks $\lambda$-fraction from each share, the total number of bits leaked is too large (indeed, potentially larger than the size of the secret being shared), and we cannot even guarantee that the secret $r_i$ has any min entropy. Thus, we cannot use just any VSS scheme, but rather we need to use a *leakage-resilient* one, with the guarantee that even if $\lambda$-fraction of each share is leaked, the secret still has high min-entropy. Indeed, we construct such a weakly leakage-resilient (WLR) VSS in Section 2.4.2. We note that many distributed protocols use VSS schemes, which immediately make them susceptible to leakage. Thus, our leakage-resilient VSS scheme may be useful for other protocols as well.

Finally, two technical difficulties remain. In the above coin-tossing paradigm utilizing WLR-VSS, each party shares his random value with all other $n$ parties, and thus each honest party holds information on *all* secret values $r_i$. Since the leakage is computed on a party's entire secret state, the adversary may learn information on the joint distribution of the $r_i$'s. This creates a dependency issue: recall that the output of the multi-source extractor is only guaranteed to be random if the sources $r_i$ are independent. Further, in this paradigm the secret state of each party will be quite large, consisting of $n$ secret shares (one for each secret value $r_i$). This will yield poor leakage bounds, with leakage rate less than $\frac{1}{n}$, if we want to ensure no share of one particular secret can be entirely leaked.

We avoid these problems by ensuring that each of the $n$ parties will never hold more than one secret share of the $r_i$'s. To this end, we follow a two-step approach. The first step is a universe reduction idea similar to the one going back to Bracha [Bra84]. Instead of having *all* parties generate and secret share random strings $r_i$, we elect a small committee $\mathcal{E}$ (of size approximately $\log^2 n$), and only the members of $\mathcal{E}$ choose a random string $r_i$ which will be shared via WLR-VSS (and later used in the construction of the collective coin). We utilize Feige's protocol [Fei99] to elect this committee, which guarantees with high probability that the fraction of faulty parties in $\mathcal{E}$ is essentially the same as in the global network (see Section 2.2.4). The second idea is that members of this committee do not WLR-VSS the $r_i$ they chose to all $n$ parties, but rather to small secondary committees. Namely, for every party $i \in \mathcal{E}$, all parties elect a secondary subcommittee $\mathcal{E}_i$, and party $P_i$ will WLR-VSS her random string $r_i$ only to parties in $\mathcal{E}_i$. We need to ensure that all the secondary committees $\mathcal{E}_i$ are disjoint, to avoid the case where one party has many shares. For this purpose, we consider a modified version of Feige's lightest bin committee election protocol [Fei99], in which we elect *multiple* (disjoint) committees by taking the *several* lightest bins. Care must be taken when electing several bins, since the adversary can "pool" his forces to target a single committee, and a failure in any one committee is a failure overall. However, in Proposition 2.5.1, we prove that for appropriately chosen parameters, (with overwhelming probability) the adversary still cannot force too many malicious parties to be elected in any committee.

## 2.2 Preliminaries

### 2.2.1 Distributions of Random Variables

**Definition 2.2.1.** A function $\nu(k)$ is *negligible* in $k$ if $\nu(k) < k^{-c}$ for sufficiently large $k$, for all constant $c > 0$. We say an event occurs with *overwhelming probability in* $k$ if it takes place with probability $1 - \nu(k)$ for some negligible function $\nu(k)$.

**Definition 2.2.2.** The *statistical distance* between two distributions $D_1$ and $D_2$ on a space $\Omega$ is defined to be

$$\Delta(D_1, D_2) = \frac{1}{2} \sum_{\omega \in \Omega} |\Pr[D_1 = \omega] - \Pr[D_2 = \omega]| .$$

We say that two distributions $D_1, D_2$ are *statistically close* (in $k$) if $\Delta(D_1, D_2) = \nu(k)$ for some negligible function $\nu(k)$.

We now define the *min-entropy* and *average* (conditional) min-entropy of a random variable.

**Definition 2.2.3.** A random variable $X \subseteq \{0,1\}^n$ is said to have min entropy $k$, denoted by $H_\infty(X) = k$, if for every $x \in \{0,1\}^n$, $\Pr[X = x] \leq \frac{1}{2^k}$. It is said to have min-entropy rate $\alpha$ if $H_\infty(X) \geq \alpha n$.

**Definition 2.2.4.** Let $(X, Z)$ be a pair of random variables. The *average min-entropy* of $X$ conditioned on $Z$ is

$$\tilde{H}_\infty(X|Z) = -\log \mathop{\mathbb{E}}_{z \leftarrow Z} \max_x \Pr[X = x|Z = z] = -\log \left[ \mathop{\mathbb{E}}_{z \leftarrow Z} (2^{-H_\infty(X|Z=z)}) \right] .$$

We make use of the following lemma from [DORS08]:

**Lemma 2.2.5** (Lemma 2.2 in [DORS08]). *Let $A, B, C$ be random variables. Then the following hold.*

*(a). For any $\delta > 0$, the conditional entropy $H_\infty(A|B = b)$ is at least $\tilde{H}_\infty(A|B) - \log(1/\delta)$ with probability at least $1 - \delta$ over the choice of $b$.*

*(b). If $B$ has at most $2^\ell$ possible values, then $\tilde{H}_\infty(A|(B, C)) \geq \tilde{H}_\infty((A, B)|C) - \ell \geq \tilde{H}(A|C) - \ell$.*

### 2.2.2 Verifiable Secret Sharing

A *secret sharing* scheme, a notion introduced by Shamir [Sha79], is a protocol that allows a dealer who holds a secret input $s$, to share his secret among $n$ parties. The guarantee is that even if $t$ of the parties are malicious, they gain no information about the secret $s$. A *verifiable secret sharing* (VSS) scheme, introduced by Chor et al. [CGMA85], is a secret sharing scheme with the additional guarantee that after the sharing phase, a dishonest dealer is either rejected, or is committed to a single secret $s$, that the honest parties can later reconstruct. Further, if the dealer is honest, then the original secret will be reconstructed, even if dishonest parties do not provide their correct shares.

**Definition 2.2.6** (Verifiable Secret Sharing). A *VSS protocol tolerating $t$ malicious parties* for parties $\mathcal{P} = \{P_1, ..., P_n\}$ is a two-phase protocol (Share, Rec), where a distinguished dealer $P^* \in \mathcal{P}$ holds an initial input $s$, such that the following conditions hold for any adversary controlling at most $t$ parties:

- **Reconstruction:** After the sharing phase, there exists a value $s'$ such that all honest parties output $s'$ in the reconstruction phase.
- **Validity:** If the dealer is honest, then $s' = s$.
- **Secrecy:** If the dealer is honest, then at the end of the sharing phase the joint view of the malicious parties is independent of the dealer's input $s$.

### 2.2.3 Multi-Source Randomness Extractors

A multi-source randomness extractor is a deterministic function which takes as input independent sources, each with sufficient amount of min-entropy, and outputs a string that is statistically close to uniform.

#### Two-Source Extractors

Constructions of two-source extractors are given, for example, by Raz [Raz05] and Bourgain [Bou05]. We use the following simplified version of the Bourgain result in the construction of leakage-resilient oblivious VSS in Section 2.4.3.

**Theorem 2.2.7** ([Bou05]). *There exists a polynomial-time computable two-source extractor* $\mathsf{Ext}_2 : (\{0,1\}^d)^2 \to \{0,1\}^m$ *that takes as input two independent sources $X$ and $Y$ and outputs an $m$-bit string that is $\epsilon$-close to uniform, as long as $H_\infty(X), H_\infty(Y) \geq \frac{1}{2}d$, and where $m = \Omega(d)$ and $\epsilon = 2^{-\Omega(d)}$.*

#### Robust Multi-Source Extractors

In this work, we need a (stronger) multi-source extractor that extracts randomness even if some of the sources are "malicious," but independent of the "honest" ones.[1] Such an extractor, which we refer to as a robust multi-source extractor, was constructed by Kamp, Rao, Vadhan and Zuckerman [KRVZ06].

**Theorem 2.2.8** ([KRVZ06]). *For any constant $\delta > 0$ there exists a constant $n_0 = n_0(\delta)$ such that for any $n > n_0$ there is a polynomial-time computable robust multi-source extractor* $\mathsf{Ext} : \left(\{0,1\}^d\right)^n \to \{0,1\}^m$ *that takes as input $n$ independent sources, each in $\{0,1\}^d$, and produces an $m$-bit string that is $\epsilon$-close to uniform, as long as at least $n_0$ of the sources have nonzero entropy, the min-entropy rate of the combined sources is $\delta$, and where $m = 0.99\delta nd$ and $\epsilon = 2^{-\Omega((nd)/\log^3(nd))}$.*

[1]Note that if the malicious sources may depend on the honest ones, then such (deterministic) extractors do not exist.

### 2.2.4 Feige Committee Election Protocol

Our leakage-resilient coin flipping protocol uses Feige's lightest bin committee election protocol as a subroutine [Fei99]. Feige's protocol gives a method for selecting a committee of approximately $k$ parties for a given parameter $k$.[2] It consists of one round, in which each party $P_i$ chooses and broadcasts a random integer "bin" $b_i \in \left[\frac{n}{k}\right]$. The committee consists of the parties in the "lightest bin": that is, those parties $P_j$ whose value $b_j$ was selected by the smallest number of total parties. See Figure 2.1 for an explicit description of the protocol.

We remark that the Feige election protocol is immediately leakage resilient, as parties do not maintain any secret state. (Indeed, all locally generated values are immediately broadcast to all parties).

**Feige Committee Election Protocol** $(n, k)$
Each party $P_i$ follows the following procedure:

1. Sample a random integer $b_i \leftarrow \left[\frac{n}{k}\right]$. Broadcast $b_i$ to all parties.
2. Let $b_j$ be the value received from each other party $P_j$, and denote by $b_{\mathsf{lightest}}$ the value in $\left[\frac{n}{k}\right]$ occurring least frequently (or the smallest such value, in case of a tie). Output as the elected committee the set of parties $\{P_j : b_j = b_{\mathsf{lightest}}\}$.

**Figure 2.1:** Feige committee election protocol for $n$ parties to elect a committee of approximate size $k$.

**Lemma 2.2.9** (Feige). *For any constant $\beta > 0$ and any $k < n$, Feige's lightest bin protocol is a 1-round public-coin protocol for electing a committee $\mathcal{E}$ such that for any set of corrupted parties $\mathcal{C} \subset [n]$ of size $t = \beta n$,*

1. $|\mathcal{E}| \leq k$,
2. $\Pr[|\mathcal{E} \setminus \mathcal{C}| \leq (1 - \beta - \epsilon)k] < \frac{n}{k} e^{-\frac{\epsilon^2 k}{2(1-\beta)}}$ $\forall$ *constant* $\epsilon > 0$,
3. $\Pr\left[\frac{|\mathcal{E} \cap \mathcal{C}|}{|\mathcal{E}|} \geq \beta + \epsilon\right] < \frac{n}{k} e^{-\frac{\epsilon^2 k}{2(1-\beta)}}$ $\forall$ *constant* $\epsilon > 0$.

*Proof.* We first note that the lightest bin necessarily contains no more than $n/(\frac{n}{k}) = k$ parties, implying property (1).

For each bin $b$ and honest party $i$, we define the indicator variable $X_{i,b}$ to be 1 if and only if party $i$ selects bin $b$. Since we consider only honest parties, this is a Bernoulli random variable with $p = \frac{k}{n}$. For a particular bin $b$, we can now bound the probability that few honest parties selected

[2]In Feige's original work [Fei99], he considered the specific case of $k = \log n$. For our purpose, we need to elect larger committees (to achieve negligible error), and thus we consider general $k$.

this bin as compared to the expected value $(1-\beta)k$.

$$\begin{aligned}\Pr\left[\sum_{i\notin C} X_{i,b} < (1-\beta-\epsilon)k\right] &= \Pr\left[\sum_{i\notin C} X_{i,b} < \left(1-\frac{\epsilon}{1-\beta}\right)(1-\beta)\,k\right] \\ &< e^{-(1-\beta)k(\frac{\epsilon}{1-\beta})^2/2} \\ &= e^{-\frac{\epsilon^2 k}{2(1-\beta)}},\end{aligned}$$

where the second inequality holds by a Chernoff bound.[3] Now, taking a union bound, the probability that *any* bin $b$ has fewer than $(1-\beta-\epsilon)k$ honest parties will be

$$\Pr[\exists \text{ Bin } b : \sum_{i\notin C} X_{i,b} < (1-\beta-\epsilon)k] < \frac{n}{k}e^{-\frac{\epsilon^2 k}{2(1-\beta)}},$$

proving property (2). Finally, combining properties (1) and (2), we have that with probability $1-\frac{n}{k}e^{-\frac{\epsilon^2 k}{2(1-\beta)}}$,

$$\frac{|\mathcal{E}\cap C|}{|\mathcal{E}|} = 1 - \frac{|\mathcal{E}\setminus C|}{|\mathcal{E}|} \leq 1 - \frac{(1-\beta-\epsilon)k}{k} = \beta+\epsilon,$$

implying property (3). □

**Remark 2.2.10.** *We further note that a stronger statement of Property (2) of Lemma 2.2.9 follows from the proof above: namely, with high probability,* each *bin will have several honest parties in it (not just the elected bin $\mathcal{E}$). That is,*

$$\Pr\left[\exists \textit{ bin } b \in \left[\frac{n}{k}\right] \textit{ s.t. } |b\setminus C| \leq (1-\beta-\epsilon)k\right] < \frac{n}{k}e^{-\frac{\epsilon^2 k}{2(1-\beta)}} \quad \forall \textit{ constant } \epsilon > 0.$$

**Corollary 2.2.11.** *Feige's lightest bin protocol for $k = \log^2 n$ is a 1-round public-coin protocol such that for any set of corrupted parties $C$ of size $\beta n$, for any constant $\epsilon > 0$, with* overwhelming probability *in $n$, a committee $\mathcal{E}$ will be elected such that $(1-\beta-\epsilon)\log^2 n \leq |\mathcal{E}| \leq \log^2 n$ and $|\mathcal{E}\setminus C| \geq (1-\beta-\epsilon)\log^2 n$.*

## 2.3 Modeling Leakage in Distributed Protocols

We consider synchronous point-to-point networks with a broadcast channel. Point-to-point channels are assumed to be authenticated and to provide partial privacy guarantees (see discussion below). We consider $n$-party protocols where up to $t$ statically corrupted parties perform arbitrary malicious faults. More precisely, we consider a computationally unbounded adversary who sees the internal state of all corrupted parties and controls their actions. We also assume the adversary is *rushing*, i.e. in any round he can wait until all honest parties send their messages before selecting the

[3]Exact Chernoff bound used: For $X_1, ..., X_n$ independent Bernoulli random variables and $\mu = \mathbb{E}[\sum_i X_i]$, then for $0 < \delta < 1$, it holds that $\Pr[\sum_i X_i < (1-\delta)\mu] < e^{-\mu\delta^2/2}$.

**Adversarial Interaction** $(t, \lambda)$

1. The adversary $\mathcal{A}$ selects a subset of parties $M \subset [n]$ of size $|M| \leq t$ to corrupt.
2. Protocol execution begins. For each party $P_i$, initialize $\mathsf{state}_i \leftarrow \emptyset$ and $\mathsf{leaked}_i = 0$. For each honest party $P_i$, denote by $\mathsf{size}_i$ the maximum value of $|\mathsf{state}_i|$ as dictated by the protocol. Each round of the protocol proceeds as follows:
   (a) Each honest party $P_i$ samples randomness to use in this round. Any randomness $R_{\mathsf{sec}}$ which must remain secret is appended to $P_i$'s secret state: $\mathsf{state}_i \leftarrow \mathsf{state}_i \cup R_{\mathsf{sec}}$. Any remaining randomness $R_{\mathsf{pub}}$ is provided to the adversary.
   (b) Honest parties send messages as dictated by the protocol.
   (c) The adversary $\mathcal{A}$ receives all messages sent by honest parties to corrupt parties.
   (d) The adversary $\mathcal{A}$ is allowed to make any collection of adaptive leakage queries of the form $\mathsf{Leak}(i, f)$, where $i \in [n]$ and $f : \{0,1\}^* \to \{0,1\}$ is any function. For each such query, if $\mathsf{leaked}_i < \lambda \cdot \mathsf{size}_i$, then $\mathcal{A}$ receives the evaluation $f(\mathsf{state}_i)$ of $f$ on the secret state of party $P_i$, and $\mathsf{leaked}_i \leftarrow \mathsf{leaked}_i + 1$. (Otherwise, $\mathcal{A}$'s query is ignored).
   (e) The adversary $\mathcal{A}$ selects outgoing messages on behalf of all corrupted parties.
   (f) Each honest party $P_i$ receives his incoming messages (from all parties). Any incoming messages $\mathsf{msg}_{\mathsf{sec}}$ that must remain secret are appended to $P_i$'s secret state: $\mathsf{state}_i \leftarrow \mathsf{state}_i \cup \mathsf{msg}_{\mathsf{sec}}$. Any remaining messages $\mathsf{msg}_{\mathsf{pub}}$ are provided to the adversary.

**Figure 2.2:** Model of $(t, \lambda)$ adversarial interaction.

messages of corrupted parties. Our results hold information theoretically, with no computational assumptions.

In this work we propose a strengthening of the standard model, where in addition the adversary is able to *leak* a constant fraction of information on the secret state of each (honest) party. We model this by allowing the adversary to adaptively make leakage queries $(i, f)$ throughout the protocol, where $i \in [n]$ and $f : \{0,1\}^* \to \{0,1\}$, and giving him the evaluation of $f$ on the secret state of party $i$. Note that this also captures leakage on communication channels, as parties' messages are computed as a function of public information and their personal secret state; thus, we do not need to assume fully private channels, but rather channels that achieve privacy with bounded information leakage.

For simplicity, we consider length-bounded leakage. Namely, we require that no more than $\lambda|\mathsf{state}_i|$ leakage queries can be made on any single party $i$'s secret state for some leakage rate $\lambda$, where $|\mathsf{state}_i|$ denotes the maximal size of the secret state of party $i$ at any given time during the protocol. But, our constructions work equally well in the more general model of [NS09] where the output length of the leakage on $\mathsf{state}_i$ is not restricted, as long as the entropy of $\mathsf{state}_i$ is decreased by no more than the fraction $\lambda$.

Note that in this model, each leakage query is applied to the secret state of a single party. Since participants of a distributed protocol typically run on different physical hardware (and in fact in many cases in different locations across the world), it is reasonable to assume each physical attack reveals information about one party's execution. To maximize generality within this setting, we allow leakage queries on different parties' secret states to be interleaved (i.e., leak from party $i$, then from party $j$, and then again from party $i$), and to be adaptively selected as a function of prior leakage.

We assume honest parties have the ability to generate randomness on the fly. This assumption is crucial for achieving leakage resilience, and is a natural requirement in a leaky setting, where any values that are held unnecessarily long over time suffer from additional leakage attacks.

We refer to such an adversary who can corrupt $t$ parties and leak $\lambda$ fraction from the secret state of each honest party as a $(t, \lambda)$ *adversary*, and say that a distributed protocol is $(t, \lambda)$ *leakage resilient* if its original properties are satisfied against such an adversary. See Figure 2.2 for a full description of the $(t, \lambda)$ adversarial model.

In this paper, we will focus on constructing a leakage-resilient unbiased coin tossing protocol.

**Definition 2.3.1** (Leakage-Resilient Distributed Coin Tossing). A protocol for parties $\mathcal{P} = \{P_1, ..., P_n\}$ is a $(t, \lambda)$ *leakage-resilient $m$-bit distributed coin tossing protocol* if the following conditions hold for any $(t, \lambda)$ adversary $\mathcal{A}$ with overwhelming probability in $n$:

- **Agreement:** At the conclusion of the protocol, each party outputs a value $v_i \in \{0,1\}^m$. For all honest parties $P_i, P_j$, it holds that $v_i = v_j$.
- **Randomness:** (Even if malicious parties abort prematurely), it holds that

$$\Delta\big((V|\mathsf{view}_{\mathcal{A}}), U_m\big) < \nu(n),$$

where $V|\mathsf{view}_{\mathcal{A}}$ is the distribution of the honest output value $v$ conditioned on the view of $\mathcal{A}$ (over the randomness of honest parties), $U_m$ is the uniform distribution over $\{0,1\}^m$, and $\nu(n)$ is a negligible function in $n$.

## 2.4 Verifiable Secret Sharing with Leakage

One of the subroutines in our leakage-resilient coin tossing protocol is a protocol achieving verifiable secret sharing (VSS) in the presence of leakage. Recall the standard VSS guarantee is that for any adversary $\mathcal{A}$ who corrupts up to $t$ parties, a dishonest dealer is committed to a single secret which will be reconstructed by honest parties, and the secret input $s$ of an honest dealer retains full entropy given the view of $\mathcal{A}$ at the conclusion of the sharing phase. For our purposes, we will need stronger guarantees, where for any adversary $\mathcal{A}$ who corrupts up to $t$ parties *and leaks* $\lambda$-fraction of each honest party's secret state (including the dealer's), the VSS reconstruction property still holds, and the secret input $s$ of an honest dealer retains a constant fraction of its original entropy given the entire view of $\mathcal{A}$ (including leakage). We refer to this property as weak leakage resilience.

In Section 2.4.1, we show that a modified version of the Shamir secret sharing scheme [Sha79] satisfies a notion of weak leakage resilience. In Section 2.4.2, we use this underlying secret sharing scheme to construct a weakly leakage-resilient VSS protocol by incorporating a method of verifying that the dealer has distributed good shares.

We note that one can define a stronger version of leakage-resilient VSS, with the requirement that the secret looks *uniform* even if all the shares are partially leaked. Although this stronger version is not required for our coin-tossing construction, we believe that it is of independent interest. We formally define and construct such a protocol in Section 2.4.3.

Throughout this section, we denote by $s$ the secret value being shared, and we denote by $S$ the distribution of $s$. For our applications, $S$ is always uniform; however, we state our results for general distributions $S$.

### 2.4.1 Weakly Leakage-Resilient Secret Sharing

Recall in the standard Shamir secret sharing scheme, to secret share an input $s$ the dealer samples a random degree $d$ polynomial $a_0 + a_1x + \cdots + a_dx^d \in \mathbb{F}[x]$ such that $a_0 = s$, and generates the shares by evaluating the polynomial at different values of $x$. The degree $d$ of the polynomial is typically chosen to be equal to the number of assumed corrupted parties, $t$.

We modify the standard Shamir secret sharing scheme in two ways. First, we take $d$ to be *strictly greater* than the number of corrupted parties. Second, we take the relative size of the secret to be larger: instead of $s$ being a single element $s \in \mathbb{F}$ embedded as the single coefficient $a_0$, we consider secrets $s \in \mathbb{F}^{d-t+1}$ consisting of $d - t + 1$ elements, embedded as the first several coefficients $s = a_0||a_1|| \cdots ||a_{d-t}$ (where $||$ denotes concatenation). The reason we embed $s$ only as $a_0, ..., a_{d-t}$ as opposed to all $a_0, ..., a_d$ is to avoid the situation where shares of corrupted parties give information about the secret $s$. By increasing the degree and making our secrets larger while maintaining the size of each secret share, we can allow a higher fraction of leakage from honest shares without reducing the entropy of $s$ by too much.

This leaves us with the question of how to set $d$, given values for $n$ and $t$. The larger the $d$ we choose, the more leakage we will be able to tolerate while maintaining entropy in the secret $s$. However, we also require the original secret to be recoverable even if $t$ parties reveal incorrect shares. To achieve this we rely on the decoding properties of the Shamir secret sharing scheme when viewed as a Reed-Solomon error-correcting code [MS81]. Namely, we can uniquely (and efficiently) decode any vector of secret shares with up to $\left\lfloor \frac{n-(d+1)}{2} \right\rfloor$ errors. To guarantee decoding of up to $t$ errors, we must have $d \leq n - 2t - 1$. To maximize leakage resilience while maintaining unique decoding, we will thus take $d = n - 2t - 1$.

We now formalize the scheme described above, which we denote by $(\mathsf{SS}, \mathsf{RecSS})$.

- $\mathsf{SS}(n, t, s)$. Let $d = n - 2t - 1$. Split the secret $s$ into $(d - t + 1)$ pieces: $s = a_0|| \cdots ||a_{d-t}$. Choose $t$ random values $a_{d-t+1}, ..., a_d \leftarrow \mathbb{F}$. The secret share for party $P_i$ is the evaluation $f(i)$, where $f(x) = a_0 + a_1x + \cdots + a_dx^d$.

- $\mathsf{RecSS}(m_1, ..., m_n)$. Use Reed-Solomon decoding on the vector of shares $(m_1, ..., m_n)$ to yield $(a_0, ..., a_d)$. Output $s = a_0|| \cdots ||a_{d-t}$.

**Proposition 2.4.1.** *For $n = (3 + \delta)t$, the following properties of* $(\mathsf{SS}, \mathsf{RecSS})$ *hold.*

1. *Unique Decoding: Given any vector $(s_1, ..., s_n)$ of distance at most $t$ from a valid codeword corresponding to a secret $s$, the output of* $\mathsf{RecSS}(s_1, ..., s_n)$ *will be $s$.*

2. *Leakage Resilience: For any distribution $S$ of the secret $s$, and any adversary corrupting up to $t$ parties and leaking a total of $\ell$ bits on the secret state of honest parties, with overwhelming probability in the security parameter $k$, the secret $s$ retains at least $H_\infty(S) - \ell - \log^2 k = \delta t \log|\mathbb{F}| - \ell - \log^2 k$ bits of min-entropy.*

*Proof.* Property 1 holds immediately from the decoding property of Reed-Solomon codes [MS81]. Property 2 holds since $H_\infty(S) = (d-t+1)\log|\mathbb{F}| = ((n-2t-1)+1)\log|\mathbb{F}| = \delta t \log|\mathbb{F}|$, together with a standard entropy argument (e.g., Lemma 2.2 of [DORS08]). □

### 2.4.2 Weakly Leakage-Resilient VSS

In our coin tossing protocol, we make use of a VSS protocol satisfying the following notion of weak leakage resilience.

**Definition 2.4.2** (WLR-VSS). A $(\lambda, \epsilon)$-*weakly leakage-resilient VSS protocol tolerating $t$ malicious parties* for parties $\mathcal{P} = \{P_1, ..., P_n\}$ is a VSS protocol such that for any $(t, \lambda)$ adversary $\mathcal{A}$, with overwhelming probability in the security parameter $k$, the following are satisfied:

- **Reconstruction, Validity:** The standard VSS reconstruction and validity properties hold.
- **Secrecy:** For any distribution $S$, if the dealer is honest during the sharing phase with secret input distribution $S$, then with overwhelming probability in $k$ over the distribution of the view of $\mathcal{A}$ at the conclusion of the sharing phase of the protocol $\mathsf{view}_\mathcal{A} \leftarrow VIEW_\mathcal{A}(S)$, it holds that $H_\infty(S|\mathsf{view}_\mathcal{A}) \geq \epsilon H_\infty(S)$.

We emphasize that the security level of the WLR-VSS depends on the security parameter $k$, which may be selected independent of the number of parties.

We now construct a $(\lambda, \epsilon)$-weakly leakage-resilient VSS protocol $(\mathsf{Share}_{\mathsf{WLR}}, \mathsf{Rec}_{\mathsf{WLR}})$, taking inspiration from the VSS construction of [BGW88]. We use as a black box the secret sharing scheme $(\mathsf{SS}, \mathsf{RecSS})$ with polynomial degree $d = n - 2t - 1$ (see Section 2.4.1 above).

At a high level, the WLR-VSS protocol proceeds as follows. First, the dealer secret shares his input $s$ via SS, along with two additional random values $r, r'$. Using the additive homomorphic property of SS, the parties check the dealer by broadcasting a (randomly selected) linear combination of their given shares, and verifying that together they form a valid codeword. To protect an honest dealer from being disqualified due to malicious parties giving bad values, the dealer will broadcast the true shares of complaining parties, and these values will be verified in a second check of the same form.

Loosely, since a dishonest dealer does not know what linear combination will be chosen, it is unlikely that he can distribute bad shares that pass these tests. Leakage information will not help, as the only secret values in the protocol are the distributed shares, which the dealer already knows (in fact, chooses) himself. On the other hand, no information on an *honest* dealer's secret $s$ is revealed from the linear combinations, since shares of $s$ are masked by shares of the random $r, r'$. So the only information learned about $s$ comes from leakage, which leaves sufficient entropy remaining by the properties of SS.

Let $\mathbb{F}$ be a field with $\log|\mathbb{F}| = kn$, where $k$ is the security parameter. We define $(\mathsf{Share}_{\mathsf{WLR}}, \mathsf{Rec}_{\mathsf{WLR}})$ in Figure 2.3.

$\mathsf{Share}_{\mathsf{WLR}}(1^k, s)$:

**Round 1:** The dealer $P^*$ selects two values $r, r' \leftarrow \mathbb{F}^{\delta t}$ uniformly at random, and runs three independent executions of the secret sharing algorithm $(\mathsf{SS}, \mathsf{RecSS})$, as defined in Section 2.4.1: $(s_1, ..., s_n) \leftarrow \mathsf{SS}(n, t, s)$, $(r_1, ..., r_n) \leftarrow \mathsf{SS}(n, t, r)$, $(r'_1, ..., r'_n) \leftarrow \mathsf{SS}(n, t, r')$. To each party $i$, the dealer sends the corresponding three shares $s_i, r_i$, and $r'_i$.

**Rounds 2-4:** Each party $P_i$ samples and broadcasts a random $k$-tuple of bits $\alpha_i \in \{0,1\}^k$. This is repeated (in two additional rounds) for random $\beta_i, \gamma_i \in \{0,1\}^k$. Take $\alpha, \beta, \gamma$ to be the corresponding elements in $\mathbb{F}$ with bit descriptions $(\alpha_1, ..., \alpha_n)$, $(\beta_1, ..., \beta_n)$, $(\gamma_1, ..., \gamma_n)$. (Recall $\log |\mathbb{F}| = kn$).

**Round 5:** Each party $P_i$ broadcasts the linear combination of his shares

$$\alpha s_i + \beta r_i + \gamma r'_i \in \mathbb{F}.$$

**Round 6:** Consider the received vector $v = (v_1, ..., v_n)$, where supposedly $v_i = \alpha s_i + \beta r_i + \gamma r'_i \ \forall i$.

- If $v$ is a valid codeword (i.e., all points lie on a degree-$d$ polynomial), the dealer is accepted, and the sharing phase concludes.
- If $v$ is distance $> t$ away from a valid codeword, the dealer is rejected, and the sharing phase concludes.
- Otherwise, let $D \subset [n]$ be the components $i$ in disagreement with those of the nearest codeword. For each $i \in D$, the dealer $P^*$ broadcasts all three shares $s_i, r_i, r'_i$. If any linear combination $\alpha s_i + \beta r_i + \gamma r'_i$ with $i \in D$ is inconsistent with the nearest codeword, the dealer is rejected. Otherwise, all parties continue to the next step.

**Rounds 7-10:** Repeat Rounds 2-5. That is, each party samples and broadcasts new random bit $k$-tuples $\tilde{\alpha}_i, \tilde{\beta}_i, \tilde{\gamma}_i$ (in three separate rounds), and then broadcasts the linear combination $\tilde{v}_i = \tilde{\alpha} s_i + \tilde{\beta} r_i + \tilde{\gamma} r'_i$ of his shares, where $\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma} \in \mathbb{F}$ are defined by $(\tilde{\alpha}_1, ..., \tilde{\alpha}_n)$, $(\tilde{\beta}_1, ..., \tilde{\beta}_n)$, $(\tilde{\gamma}_1, ..., \tilde{\gamma}_n)$ as above.

**Local Computation:** Consider the new vector $\tilde{v}$ of values received in Round 10, where $\forall i \in D$ we use the values $(s, r_i, r'_i)$ broadcast by the dealer in Round 6.

- If $\tilde{v}$ is distance $> t$ away from a valid codeword, the dealer is rejected.
- Otherwise, let $\tilde{D}$ be the set of parties whose values differ from the codeword closest to $\tilde{v}$.
  - If $D \cap \tilde{D} \neq \emptyset$ or $|D \cup \tilde{D}| > t$, then the dealer is rejected.
  - Otherwise, the dealer is accepted.

$\mathsf{Rec}_{\mathsf{WLR}}()$:

**Round 1:** Each party $P_i$ broadcasts his share $s_i$.

**Local Computation:** Locally, $P_i$ runs the modified Shamir secret sharing reconstruction algorithm $s' \leftarrow \mathsf{RecSS}(s'_1, ..., s'_n)$, where $s'_i$ is the value broadcast by party $P_i$, and outputs this value $s'$.

**Figure 2.3:** Weakly leakage-resilient VSS protocol, $(\mathsf{Share}_{\mathsf{WLR}}, \mathsf{Rec}_{\mathsf{WLR}})$.

**Theorem 2.4.3.** *Let $n = (3+\delta)t$ for some constant $\delta > 0$. Then for any constants $\epsilon < 1$ and $\lambda \leq \frac{\delta(1-\epsilon)}{10+6\delta}$, the protocol $(\mathsf{Share}_{\mathsf{WLR}}, \mathsf{Rec}_{\mathsf{WLR}})$ is a $(\lambda, \epsilon)$-weakly leakage-resilient VSS protocol tolerating $t$ malicious parties that runs in $O(1)$ rounds.*

*Proof.* We show that $(\mathsf{Share}_{\mathsf{WLR}}, \mathsf{Rec}_{\mathsf{WLR}})$ satisfies the validity, reconstruction, and secrecy properties described in Definition 2.4.2.

**Validity.** If the dealer is honest, then only malicious parties can complain of bad shares; the dealer will broadcast honest shares to the complaining parties and thus will not be rejected. Further, by the unique decoding property of the underlying secret sharing scheme $(\mathsf{SS}, \mathsf{RecSS})$ (i.e., Property 2.4.1.1), any secret sharing of $s$ with up to $t$ corrupted shares will be uniquely decoded to yield the original secret $s$.

**Reconstruction.** Consider the case of a dishonest dealer: we show that if the dealer is not rejected, then at the end of the sharing phase all honest parties hold consistent shares of some value $s$. This will be sufficient to argue that all honest parties will output $s$ at the conclusion of the reconstruction phase, again by Property 2.4.1.1.

**Lemma 2.4.4.** *If the dealer is accepted in the sharing phase of the protocol, then at the conclusion of the sharing phase all honest parties hold shares consistent with a single degree-$d$ polynomial.*

Before we prove the lemma, we introduce some notation and prove a claim that we will invoke later. Let $H \subseteq [n]$ be the set of honest parties. For any vector $v \in \mathbb{F}^n$ and subset $W \subseteq [n]$, we denote by $v_W$ the vector in $\mathbb{F}^{|W|}$ formed by taking the components $v_i$ with $i \in W$. We say that a set of shares $v_W$ is "$d$-consistent" if the interpolation of the corresponding points yields a polynomial of degree no greater than $d$. Note that any set of at most $d+1$ shares is trivially $d$-consistent.

**Claim 2.4.5.** *Let $p_1, p_2, p_3 \in \mathbb{F}[x]$ be polynomials with $\deg p_1 > d$. Then for any distributions $A, B, C$ over $\mathbb{F}$ such that $H_\infty(A), H_\infty(B), H_\infty(C) \geq m$, it holds that*

$$\Pr_{\substack{\alpha \leftarrow A, \beta \leftarrow B, \\ \gamma \leftarrow C}} [\deg(\alpha p_1 + \beta p_2 + \gamma p_3) \leq d] \leq \frac{1}{2^{m-1}}.$$

*Proof.* Consider the term of highest degree $ax^{d'}$ in $p_1$, and let $bx^{d'}$ and $cx^{d'}$ be the terms of corresponding degree in $p_2$ and $p_3$.

$$\begin{aligned}
\Pr[\deg(\alpha p_1 + \beta p_2 + \gamma p_3) \leq d] &\leq \Pr[\alpha a + \beta b + \gamma c = 0] \\
&\leq \Pr[\alpha = 0] + \Pr[\alpha a + \beta b + \gamma c = 0 | \alpha \neq 0] \\
&\leq \frac{1}{2^m} + \Pr[a = -\alpha^{-1}(\beta b + \gamma c) | \alpha \neq 0] \\
&\leq \frac{1}{2^m} + \frac{1}{2^m} = \frac{1}{2^{m-1}}.
\end{aligned}$$

□

We will use this claim to argue that if any collection of shares $s_W$ (or $r_W$ or $r'_W$) of size $|W| > d+1$ is not $d$-consistent (ie, they interpolate to a polynomial of degree strictly greater than $d$), then with high probability over the choice of $\alpha, \beta, \gamma$, the linear combination of these shares $\alpha s_W + \beta r_W + \gamma r'_W$ will not be $d$-consistent.

*Proof of Lemma 2.4.4.* Consider the vector $v = (v_1, ..., v_n)$ received in Round 6 of the protocol, where allegedly $v_i = \alpha s_i + \beta r_i + \gamma r'_i \ \forall i$. Recall if $v$ is distance greater than $t$ from any codeword, then the dealer is rejected at this stage, and the lemma holds. Otherwise, there exists a unique codeword closest to $v$, corresponding to some polynomial $p$ of degree $d$, and we define $D \subseteq H$ be the set of honest parties $i$ for which $v_i \neq p(i)$.[4] For each such party $i \in D$, update their shares $s_i, r_i, r'_i$ with the corresponding values broadcast by the dealer.

Consider the collection of shares $v_{H \setminus D}$. By definition of $D$, this collection of remaining shares is $d$-consistent. Since these are honest parties, we know that $v_i = \alpha s_i + \beta r_i + \gamma r'_i$ for each $i \in H \setminus D$. Claim 2.4.5 tells us that for any set $W \subset [n]$ chosen before $\alpha, \beta, \gamma$,

$$\Pr_{\alpha,\beta,\gamma}[(v_W \ d\text{-consistent}) \wedge (s_W \text{ not } d\text{-consistent})] \leq 2^{-(m-1)},$$

where $m = H_\infty(\alpha) = H_\infty(\beta) = H_\infty(\gamma)$. Note since $|\alpha_i| = |\beta_i| = |\gamma_i| = k$ and the number of honest parties is $n - t$, then $m \geq k(n-t)$. In our case, the set $H \setminus D$ is not defined a priori, but rather is determined as a function of the random variables $\alpha, \beta, \gamma$ themselves, so we cannot apply this claim for $W = H \setminus D$ outright. But,

$$\begin{aligned}
&\Pr_{\alpha,\beta,\gamma}[(v_{H\setminus D} \ d\text{-consistent}) \wedge (s_{H\setminus D} \text{ not } d\text{-consistent})] \\
&\qquad \leq \Pr_{\alpha,\beta,\gamma}[\exists W \subseteq [n] \text{ s.t. } (v_W \ d\text{-consistent}) \wedge (s_W \text{ not } d\text{-consistent})] \\
&\qquad \leq (2^n)(2^{-(m-1)}) \\
&\qquad = (2^n)2^{-k(n-t)+1} \\
&\qquad = 2^{-\Omega(kn)},
\end{aligned}$$

where the second inequality follows from the union bound and Claim 4.1, and the last equality follows from the fact that $n = (3+\delta)t$. The same probability bound holds for $r_{H\setminus D}$ and $r'_{H\setminus D}$ in the place of $s_{H\setminus D}$. Thus, by a simple union bound, the probability that any one of $s_{H\setminus D}, r_{H\setminus D}$, or $r'_{H\setminus D}$ is not $d$-consistent given that $v$ is $d$-consistent is negligible, and thus we will assume it is not the case.

In particular, this implies the shares $\tilde{v}_{H\setminus D}$, defined by $\tilde{v}_i = \tilde{\alpha} s_i + \tilde{\beta} r_i + \tilde{\gamma} r'_i \ \forall i \in H \setminus D$, are $d$-consistent. Consider the rest of this vector $\tilde{v} = (\tilde{v}_1, ..., \tilde{v}_n)$ received in Round 10 of the protocol. We wish to show that either the updated shares of $s$ held by honest parties ($s_H$) are all $d$-consistent, or the vector $\tilde{v}$ will lead us to reject the dealer. We consider two cases:

**Case 1:** $\tilde{v}_H$ is $d$-consistent. In this case, by Claim 2.4.5, with probability at least $1 - 2^{-(m-1)} = 1 - \mathsf{negl}(k)$ we have that $s_H$ is $d$-consistent, and we are done. (Note that here Claim 2.4.5 *can* be applied directly, since the set $H$ does not depend on $\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}$).

[4]Note the minor inconsistency in notation, where before we defined $D$ to be the set of *all* parties (not just honest parties) whose component is in disagreement.

**Case 2:** $\tilde{v}_H$ is not $d$-consistent. We show that in this case the dealer is rejected. If $\tilde{v}$ is distance greater than $t$ from any codeword, then the dealer is rejected immediately, and we are done. Otherwise, there is a unique closest codeword to $\tilde{v}$, corresponding to some polynomial $\tilde{p}$ of degree $d$. Let $\tilde{D} \subseteq H$ be the set of honest parties $i$ for which $\tilde{v}_i \neq \tilde{p}(i)$. If $D \cap \tilde{D} \neq \emptyset$, then the dealer will be rejected, and again we are done. So assume $\tilde{v}_i = \tilde{p}(i)$ for all $i \in D$. We know that $\tilde{V}_{H\setminus D}$ is $d$-consistent, corresponding to the evaluations of some degree $d$ polynomial $p'$. But, since we are in the case that $\tilde{v}_H$ is not $d$-consistent, it must be that $p' \neq \tilde{p}$. Since $p'$ and $\tilde{p}$ are polynomials of degree $d$, this means $p'(i)$ can equal $\tilde{p}(i)$ for at most $d$ values of $i$. Thus, $|\tilde{D}| \geq |H \setminus D| - d$. But, this means $|D \cup \tilde{D}| \geq |D| + (|H \setminus D| - d) = |H| - d = (2+\delta)t - ((1+\delta)t - 1) = t + 1$, and therefore the dealer will be rejected.

□

**Secrecy.** We now consider the case of an honest dealer, and show that even an adversary who corrupts $t$ parties and leaks cannot learn too much about the secret value $s$.

**Lemma 2.4.6.** *Let $S$ be any distribution over $\mathbb{F}^{\delta t}$ of secret inputs. Let $\mathcal{A}$ be a computationally unbounded adversary for the VSS protocol who adaptively leaks a total of $\ell$ bits from shares of honest parties during the execution of the protocol. Let $\mathsf{view}_{\mathcal{A}}(S)$ denote the distribution of the view of $\mathcal{A}$ at the conclusion of the sharing phase. Then with overwhelming probability in $k$ over $\mathsf{view}_{\mathcal{A}} \leftarrow VIEW_{\mathcal{A}}(S)$, it holds that*

$$H_\infty(S|\mathsf{view}_{\mathcal{A}}) \geq H_\infty(S) - \ell - \log^2 k.$$

*Proof.* At the conclusion of the sharing phase, the view of the adversary consists of five pieces of information: (1) his internal randomness $\mathsf{rand}_{\mathcal{A}}$ (which, without loss of generality, is sampled before the protocol execution, independent of all learned information), (2) the secret shares of corrupted parties $\{s_i, r_i, r'_i\}_{i\in C}$, (3) the honest parties' contributions $(\alpha_j, \beta_j, \gamma_j)_{j\notin C}, (\tilde{\alpha}_j, \tilde{\beta}_j, \tilde{\gamma}_j)_{j\notin C}$ to the coefficients, (4) the corresponding linear combinations of honest parties' shares $\{\alpha s_j + \beta r_j + \gamma r'_j\}_{j\notin C}, \{\tilde{\alpha} s_j + \tilde{\beta} r_j + \tilde{\gamma} r'_j\}_{j\notin C}$, and (5) the answers to his leakage queries $\mathsf{leakage}$. (Recall that the adversary's contribution to the coefficients $(\alpha_i, \beta_i, \gamma_i)_{i\in C}, (\tilde{\alpha}_i, \tilde{\beta}_i, \tilde{\gamma}_i)_{i\in C}$ on behalf of corrupted parties is determined by the remaining portions of his view). For each variable in the view of $\mathcal{A}$, we denote its corresponding distribution with capital letters.

Note that by Lemma 2.2(a) of [DORS08] (see Lemma 2.2.5 of the present), it holds with overwhelming probability in $k$ over $\mathsf{view}_{\mathcal{A}} \leftarrow VIEW_{\mathcal{A}}(S)$ that

$$H_\infty(S|\mathsf{view}_{\mathcal{A}}) \geq \tilde{H}_\infty(S|VIEW_{\mathcal{A}}(S)) - \log^2 k.$$

Thus, it suffices to analyze the average conditional min entropy of $S$ given the distribution

$$VIEW_{\mathcal{A}}(S) = \Big(\mathsf{RAND}_{\mathcal{A}}, \{S_i, R_i, R'_i\}_{i\in C}, (\alpha_j, \beta_j, \gamma_j)_{j\notin C}, (\tilde{\alpha}_j, \tilde{\beta}_j, \tilde{\gamma}_j)_{j\notin C},$$
$$\{\alpha S_j + \beta R_j + \gamma R'_j\}_{j\notin C}, \{\tilde{\alpha} S_j + \tilde{\beta} R_j + \tilde{\gamma} R'_j\}_{j\notin C}, \mathsf{LEAKAGE}\Big).$$

We accomplish this via a sequence of intermediate claims.

We begin by proving that the secret shares of corrupted parties $\{S_i, R_i, R'_i\}_{i\in C}$ are independent of the secrets $S, R, R'$, and that the linear combinations of the honest parties' secret shares $\{\alpha S_j +$

$\beta R_j + \gamma R'_j\}_{j \notin \mathcal{C}}, \{\tilde{\alpha} S_j + \tilde{\beta} R_j + \tilde{\gamma} R'_j\}_{j \notin \mathcal{C}}$ do not reveal additional information beyond the underlying linear combinations of secrets, $(\alpha S + \beta R + \gamma R')$ and $(\tilde{\alpha} S + \tilde{\beta} R + \tilde{\gamma} R')$.[5]

**Claim 2.4.7.** *There exists a distribution* $\mathsf{LEAKAGE}'$ *with* $|\mathsf{LEAKAGE}'| = |\mathsf{LEAKAGE}|$ *for which*

$$\tilde{H}_\infty(S \mid VIEW_{\mathcal{A}}(S)) \geq \tilde{H}_\infty\Big(S \;\Big|\; \mathsf{RAND}_{\mathcal{A}}, (\alpha_j, \beta_j, \gamma_j)_{j \notin \mathcal{C}}, (\tilde{\alpha}_j, \tilde{\beta}_j, \tilde{\gamma}_j)_{j \notin \mathcal{C}}, U^S_{d'}, U^R_{d'}, U^{R'}_{d'}, (\alpha S + \beta R + \gamma R'), (\tilde{\alpha} S + \tilde{\beta} R + \tilde{\gamma} R'), \mathsf{LEAKAGE}'\Big),$$

*where* $U^S_{d'}, U^R_{d'}, U^{R'}_{d'}$ *are uniformly random given* $S, R, R'$, *and* $\mathsf{RAND}_{\mathcal{A}}$.

*Proof.* We will show the distribution of the entire view of the adversary $VIEW_{\mathcal{A}}(S)$ can be simulated given only $\mathsf{RAND}_{\mathcal{A}}, (\alpha_j, \beta_j, \gamma_j)_{j \notin \mathcal{C}}, (\tilde{\alpha}_j, \tilde{\beta}_j, \tilde{\gamma}_j)_{j \notin \mathcal{C}}, U^S_{d'}, U^R_{d'}, U^{R'}_{d'}, (\alpha S + \beta R + \gamma R'), (\tilde{\alpha} S + \tilde{\beta} R + \tilde{\gamma} R')$, and $\mathsf{LEAKAGE}'$.

Recall that in the secret sharing scheme (Share, Reconstruct), shares of a secret value $s \in \mathbb{F}^{\delta t}$ are evaluations of a random polynomial over $\mathbb{F}$ of fixed degree $d$ whose $\delta t$ lowest coefficients correspond to $s$. Given $s$, this distribution of secret shares can equivalently be generated by selecting at random the first $d + 1 - \delta t$ polynomial evaluations, and then solving (deterministically) for all remaining evaluations to be consistent with $s$. In particular, taking the set of corrupt parties' shares to be among the first $d + 1 - \delta t$, and defining $d' := d + 1 - \delta t$, we have the following equivalence of distributions:

$$\Big(\{S_i\}_{i \in \mathcal{C}}, \{S_j\}_{j \notin \mathcal{C}, \leq d'}, \{S_j\}_{j \notin \mathcal{C}, > d'}\Big) \equiv \Big(U_{|\mathcal{C}|}, U_{d' - |\mathcal{C}|}, \mathsf{Share}'(S, U_{|\mathcal{C}|}, U_{d' - |\mathcal{C}|})\Big),$$

where $U_m$ denotes the uniform distribution over $\mathbb{F}^m$, and $\mathsf{Share}'$ is a *deterministic* function. The same holds for shares of $R$ and $R'$. Now, denote by $U^S_{d'}, U^R_{d'}, U^{R'}_{d'}$ the randomness sampled for the sharing of $S, R, R'$, respectively. By the homomorphic properties of the polynomial secret sharing scheme (i.e., adding two polynomials corresponds to adding evaluations point-wise, and also to adding coefficients term-wise), it holds that the shares of the $(\alpha, \beta, \gamma)$-linear combination of these secret shares of $S, R, R'$ are given by

$$\Big(\{\alpha S_i + \beta R_i + \gamma R'_i\}_{i \leq d'}, \{\alpha S_j + \beta R_j + \gamma R'_j\}_{j > d'}\Big)$$
$$\equiv \Big((\alpha U^S_{d'} + \beta U^R_{d'} + \gamma U^{R'}_{d'}), \mathsf{Share}'\big((\alpha S + \beta R + \gamma R'), (\alpha U^S_{d'} + \beta U^R_{d'} + \gamma U^{R'}_{d'})\big)\Big).$$

The same holds for the $(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$-linear combinations of shares. Namely,

$$\Big([\{\alpha S_i + \beta R_i + \gamma R'_i\}_{i \in [n]}], [\{\tilde{\alpha} S_i + \tilde{\beta} R_i + \tilde{\gamma} R'_i\}_{i \in [n]}]\Big)$$
$$\equiv \Big(\Big[(\alpha U^S_{d'} + \beta U^R_{d'} + \gamma U^{R'}_{d'}), \mathsf{Share}'\big((\alpha S + \beta R + \gamma R'), (\alpha U^S_{d'} + \beta U^R_{d'} + \gamma U^{R'}_{d'})\big)\Big],$$
$$\Big[(\tilde{\alpha} U^S_{d'} + \tilde{\beta} U^R_{d'} + \tilde{\gamma} U^{R'}_{d'}), \mathsf{Share}'\big((\tilde{\alpha} S + \tilde{\beta} R + \tilde{\gamma} R'), (\tilde{\alpha} U^S_{d'} + \tilde{\beta} U^R_{d'} + \tilde{\gamma} U^{R'}_{d'})\big)\Big]\Big).$$

[5]Where $\alpha S$ (resp., $\beta R, \gamma R'$) denotes scalar multiplication of $\alpha$ (resp., $\beta, \gamma$) with the $\mathbb{F}^{\delta t}$-vector $S$ (resp., $R, R'$) over $\mathbb{F}$.

Therefore, given $U^S_{d'}, U^R_{d'}, U^{R'}_{d'}$ and the linear combinations of secrets $(\alpha S+\beta R+\gamma R'), (\tilde{\alpha} S+\tilde{\beta} R+\tilde{\gamma} R')$, one can exactly simulate the following portions of the adversary's view:

1. $\{S_i, R_i, R'_i\}_{i\in\mathcal{C}} = \{U^S_j, U^R_j, U^{R'}_j\}_{j\le|\mathcal{C}|}$ are simply the first $|\mathcal{C}|$ random elements of $U^S_{d'}, U^R_{d'}, U^{R'}_{d'}$.
2. $\{\alpha S_j+\beta R_j+\gamma R'_j\}_{j\notin\mathcal{C},\le d'} = \{\alpha U^S_j+\beta U^R_j+\gamma U^{R'}_j\}_{|\mathcal{C}|<j\le d'}$, since we have $S_j=U^S_j$, $R_j=U^R_j$, and $R'_j=U^{R'}_j$ for the first $d'-|\mathcal{C}|$ honest parties' shares $j$.
3. $\{\alpha S_j+\beta R_j+\gamma R'_j\}_{j\notin\mathcal{C},>d'} = \mathsf{Share}'\big((\alpha S+\beta R+\gamma R'), (\alpha U^S_{d'}+\beta U^R_{d'}+\gamma U^{R'}_{d'})\big)$.
4. $\{\tilde{\alpha} S_j+\tilde{\beta} R_j+\tilde{\gamma} R'_j\}_{j\notin\mathcal{C},\le d'}$ and $\{\tilde{\alpha} S_j+\tilde{\beta} R_j+\tilde{\gamma} R'_j\}_{j\notin\mathcal{C},>d'}$ are simulated analogous to items 2 and 3 above, using the coefficients $\tilde{\alpha},\tilde{\beta},\tilde{\gamma}$ in the place of $\alpha,\beta,\gamma$.
5. Leakage on any secret share $S_j, R_j, R'_j$ can be simulated given the *same amount* of leakage directly on $S$, $R$, and $R'$, since all shares can be reconstructed given these secrets, together with the information $\mathsf{RAND}_{\mathcal{A}}, (\alpha_j,\beta_j,\gamma_j)_{j\notin\mathcal{C}}, (\tilde{\alpha}_j,\tilde{\beta}_j,\tilde{\gamma}_j)_{j\notin\mathcal{C}}, U^S_{d'}, U^R_{d'}, U^{R'}_{d'}, (\alpha S+\beta R+\gamma R'), (\tilde{\alpha} S+\tilde{\beta} R+\tilde{\gamma} R')$.

□

We next argue that revealing the sums $(\alpha S+\beta R+\gamma R')$ and $(\tilde{\alpha} S+\tilde{\beta} R+\tilde{\gamma} R')$ does not decrease the entropy of $S$ by too much. This is done via the following two claims.

**Claim 2.4.8.** *For any distribution $Y$, it holds that*

$$\begin{aligned}&\tilde{H}_\infty\Big(S \mid Y, (\alpha S+\beta R+\gamma R'), (\tilde{\alpha} S+\tilde{\beta} R+\tilde{\gamma} R')\Big)\\ &\ge \tilde{H}_\infty\Big(S, (\alpha S+\beta R+\gamma R'), (\tilde{\alpha} S+\tilde{\beta} R+\tilde{\gamma} R') \mid Y\Big) - \big|(\alpha S+\beta R+\gamma R')\big| - \big|(\tilde{\alpha} S+\tilde{\beta} R+\tilde{\gamma} R')\big|.\end{aligned}$$

*Proof.* Follows directly by the chain rule for min entropy (see Lemma 2.2.5). □

**Claim 2.4.9.** *With overwhelming probability in $k$ over the randomness of honest parties in the protocol, it holds that*

$$\begin{aligned}\tilde{H}_\infty(S, (\alpha S+\beta R+\gamma R'), (\tilde{\alpha} S+\tilde{\beta} R+\tilde{\gamma} R') \mid \mathsf{RAND}_{\mathcal{A}}, (\alpha_j,\beta_j,\gamma_j)_{j\notin\mathcal{C}}, (\tilde{\alpha}_j,\tilde{\beta}_j,\tilde{\gamma}_j)_{j\notin\mathcal{C}}, U^S_{d'}, U^R_{d'}, U^{R'}_{d'}, \mathsf{LEAKAGE}')\\ = \tilde{H}_\infty((S,R,R') \mid \mathsf{RAND}_{\mathcal{A}}, (\alpha_j,\beta_j,\gamma_j)_{j\notin\mathcal{C}}, (\tilde{\alpha}_j,\tilde{\beta}_j,\tilde{\gamma}_j)_{j\notin\mathcal{C}}, U^S_{d'}, U^R_{d'}, U^{R'}_{d'}, \mathsf{LEAKAGE}').\end{aligned}$$

*Proof.* If it is the case that $\beta\tilde{\gamma}\neq\tilde{\beta}\gamma$, then the coefficient matrix

$$\begin{pmatrix} 1 & 0 & 0\\ \alpha & \beta & \gamma\\ \tilde{\alpha} & \tilde{\beta} & \tilde{\gamma}\end{pmatrix}$$

is nonsingular, and hence the distributions $\big(S, (\alpha S+\beta R+\gamma R'), (\tilde{\alpha} S+\tilde{\beta} R+\tilde{\gamma} R)\big)$ and $(S,R,R')$ are in bijection. In this case, the claim follows. It thus remains to show that $\beta\tilde{\gamma}\neq\tilde{\beta}\gamma$ will hold with overwhelming probability in $k$ within the protocol.

Recall that each coefficient is selected one round at a time, in which every honest party samples and broadcasts $k$ random bits, and the adversary arbitrarily selects $k$ bits on behalf of each corrupted party. First, note that with overwhelming probability in $k$, no coefficient $\alpha, \beta, \gamma, \tilde{\alpha}, \tilde{\beta}, \tilde{\gamma} \in \mathbb{F}$ will be the element 0. This is because this would require every honest party to have randomly sampled $k$ bits of 0 for his contribution to the coefficient, which will only occur with probability $2^{-k(n-|\mathcal{C}|)} \in 2^{-\Omega(kn)}$.

Now, consider the selection round for the final coefficient $\tilde{\gamma}$. At this point, $\beta, \gamma$, and $\tilde{\beta}$ are each completely determined. Then, even if the adversary could completely choose their values maliciously (subject to them being nonzero), there will exist a *single* "bad" value of $\tilde{\gamma} \in \mathbb{F}$ (namely, $\beta^{-1}\tilde{\beta}\gamma \in \mathbb{F}$). But, as above, because the honest parties are sampling their bits at random, the probability that this exact element will be selected is bounded by $2^{-\Omega(kn)}$.

□

We now address the leakage information.

**Claim 2.4.10.** *It holds that*

$$\begin{aligned}\tilde{H}_\infty\big((S,R,R') \mid \mathsf{RAND}_\mathcal{A}, (\alpha_j,\beta_j,\gamma_j)_{j\notin C}, (\tilde{\alpha}_j,\tilde{\beta}_j,\tilde{\gamma}_j)_{j\notin C}, U^S_{d'}, U^R_{d'}, U^{R'}_{d'}, \mathsf{LEAKAGE'}\big)\\ \geq \tilde{H}_\infty\big((S,R,R') \mid \mathsf{RAND}_\mathcal{A}, (\alpha_j,\beta_j,\gamma_j)_{j\notin C}, (\tilde{\alpha}_j,\tilde{\beta}_j,\tilde{\gamma}_j)_{j\notin C}, U^S_{d'}, U^R_{d'}, U^{R'}_{d'}\big) - \ell.\end{aligned}$$

*Proof.* Follows by Lemma 2.2(b) of [DORS08] (see Lemma 2.2.5 in the present), since $|\mathsf{LEAKAGE'}| = \ell$. □

We are ready for our final claim.

**Claim 2.4.11.** *It holds that*

$$\tilde{H}_\infty\big((S,R,R') \mid \mathsf{RAND}_\mathcal{A}, (\alpha_j,\beta_j,\gamma_j)_{j\notin C}, (\tilde{\alpha}_j,\tilde{\beta}_j,\tilde{\gamma}_j)_{j\notin C}, U^S_{d'}, U^R_{d'}, U^{R'}_{d'}\big) = H_\infty(S) + 2\delta t \log|\mathbb{F}|.$$

*Proof.* The values $\mathsf{RAND}_\mathcal{A}, (\alpha_j,\beta_j,\gamma_j)_{j\notin C}, (\tilde{\alpha}_j,\tilde{\beta}_j,\tilde{\gamma}_j)_{j\notin C}, U^S_{d'}, U^R_{d'}, U^{R'}_{d'}$, corresponding to the adversary's internal randomness (chosen without loss of generality at random, before the protocol begins), the honest parties' random contributions to the coefficients, and the randomness used for secret sharing, are each sampled independent of $(S,R,R')$. Thus,

$$\tilde{H}_\infty\big((S,R,R') \mid \mathsf{RAND}_\mathcal{A}, (\alpha_j,\beta_j,\gamma_j)_{j\notin C}, (\tilde{\alpha}_j,\tilde{\beta}_j,\tilde{\gamma}_j)_{j\notin C}, U^S_{d'}, U^R_{d'}, U^{R'}_{d'}\big) = H_\infty\big((S,R,R')\big).$$

Further, $R$ and $R'$ are also sampled at random from $\mathbb{F}^{\delta t}$ by the honest dealer, independent of each other and $S$. Hence, $H_\infty\big((S,R,R')\big) = H_\infty(S) + \delta t \log|\mathbb{F}| + \delta t \log|\mathbb{F}|$, as required. □

Combining Claims 2.4.7 - 2.4.11, we have that

$$\begin{aligned}\tilde{H}_\infty(S|VIEW_\mathcal{A}(S)) &\geq ((H_\infty(S) + 2\delta t \log|\mathbb{F}|) - \ell) - \delta t \log|\mathbb{F}| - \delta t \log|\mathbb{F}|\\ &= H_\infty(S) - \ell,\end{aligned}$$

where the first line holds since $|(\alpha S + \beta R + \gamma R')| = |(\tilde{\alpha} S + \tilde{\beta} R + \tilde{\gamma} R')| = \delta t \log|\mathbb{F}|$. Thus, as discussed above, it holds by Lemma 2.2(a) of [DORS08] (Lemma 2.2.5 of the present) that with overwhelming probability in $k$ over $\mathsf{view}_\mathcal{A} \leftarrow VIEW_\mathcal{A}(S)$ that

$$H_\infty(S|\mathsf{view}_\mathcal{A}) \geq H_\infty(S) - \ell - \log^2 k,$$

as desired.

□

**Remark 2.4.12.** Note that the proof of Lemma 2.4.6 holds also for a stronger adversarial leakage model, in which the adversary may leak on the *joint* secret state of all honest parties. That is, the VSS protocol of Figure 2.3 satisfies the property that leaking $\ell$ bits on *any* secret information during the protocol cannot reveal significantly more than $\ell$ bits of information on the secret $s$, without requiring the leakage to take place independently on each party. However, independent leakage will be required for later parts of the overall coin tossing protocol.

What remains is to prove that $H_\infty(S) - \ell - \log^2 n \geq \epsilon H_\infty(S)$, where $\ell$ is the total amount of leakage. Recall that the adversary can leak $\lambda = \frac{\delta(1-\epsilon)}{10+6\delta}$ fraction of each honest party's secret state. Further, recall that the secret state contains only those values that must remain secret (whereas other values generated by honest parties are assumed to be given to the adversary in their entirety; see Figure 2.2 and discussion in Section 2.3). The secret state of each non-dealer party consists of precisely three elements of $\mathbb{F}$, corresponding to his shares $s_i, r_i$, and $r'_i$. (Note that $\alpha_i, \beta_i, \gamma_i, \tilde{\alpha}_i, \tilde{\beta}_i, \tilde{\gamma}_i$ are not part of the secret state, since they are broadcast immediately after being generated.) The dealer must hold additional secret information, since he must be able to produce a valid secret share for any complaining party in Round 6. Thus, he must store $3(d+1) = 3(1+\delta)t$ secret elements of $\mathbb{F}$, corresponding to the coefficients of the secret sharing polynomials for $s, r$, and $r'$. Thus,

$$\begin{aligned}
\ell = \lambda \sum_{i \notin C} |\mathsf{state}_i| &= \lambda\Big((n-t)(3\log|\mathbb{F}|) + 3(d+1)\log|\mathbb{F}|\Big) \\
&= \lambda\Big((2+\delta)t(3\log|\mathbb{F}|) + 3(1+\delta)t\log|\mathbb{F}|\Big) \\
&= 3t\log|\mathbb{F}|\lambda(2+\delta+1+\delta) \\
&= 3t\log|\mathbb{F}|\lambda(3+2\delta)
\end{aligned}$$

Combining this with Lemma 2.4.6, we have that with overwhelming probability:

$$\begin{aligned}
H_\infty(S|\mathsf{view}_{\mathcal{A}}(s)) &\geq H_\infty(S) - \ell - \log^2 n \\
&= \delta t\log|\mathbb{F}| - 3t\log|\mathbb{F}|\lambda(3+2\delta) - \log^2 n \\
&= t\log|\mathbb{F}|\left(\delta - \lambda 3(3+2\delta) - \frac{\log^2 n}{t\log|\mathbb{F}|}\right) \\
&\geq t\log|\mathbb{F}|\left(\delta - \lambda\Big((9+6\delta) + \frac{1}{n}\Big)\right) \\
&\geq t\log|\mathbb{F}|\Big(\delta - \lambda(10+6\delta)\Big) \\
&= t\log|\mathbb{F}|\Big(\delta - \frac{\delta(1-\epsilon)}{10+6\delta}(10+6\delta)\Big) \\
&= \epsilon\delta t\log|\mathbb{F}| = \epsilon H_\infty(S)
\end{aligned}$$

Thus, the protocol $(\mathsf{Share}_{\mathsf{WLR}}, \mathsf{Rec}_{\mathsf{WLR}})$ satisfies the properties of a $(\lambda, \epsilon)$-weakly leakage-resilient VSS protocol.

□

### 2.4.3 Leakage-Resilient Oblivious VSS

For our coin tossing protocol, we only need VSS achieving a weak notion of leakage resilience, where for any adversary who corrupts $t$ parties and leaks a constant fraction of the secret state of the remaining parties, the secret still retains a constant fraction of its original entropy. However, one can also consider a stronger version of leakage resilience, where the secret retains its *full* entropy.

Naturally, this notion cannot be achieved if any party knows the secret in its entirety, since the adversary can simply leak on this value outright. In particular, this immediately rules out the possibility of standard VSS, since the dealer himself cannot know the secret! We thus put forth the notion of ***oblivious*** secret sharing, where the dealer secret shares a uniformly distributed secret, ***whose value he does not know.*** We also show that this is, in fact, achievable (see below). We believe that leakage-resilient oblivious VSS primitives can serve as a useful building block for constructing future leakage-resilient protocols, which anyway make use of VSS in this fashion (e.g., in [FM85] to achieve Byzantine Agreement).

**Definition 2.4.13** (Leakage-Resilient Oblivious VSS). A *$\lambda$-leakage-resilient oblivious VSS protocol* for parties $\mathcal{P} = \{P_1, ..., P_n\}$ *tolerating $t$ malicious parties* is a VSS protocol satisfying the following for any $(t, \lambda)$ adversary $\mathcal{A}$, with overwhelming probability in $n$:

- **Reconstruction:** The standard VSS reconstruction property holds.
- **Validity:** If the dealer is honest during the sharing phase, then the distribution $S'$ of the value reconstructed in the second phase (over the randomness of the dealer in the sharing phase) is uniform.
- **Secrecy:** If the dealer is honest during the sharing phase, then the distribution $VIEW_{\mathcal{A}}$ of the view of $\mathcal{A}$ at the conclusion of this phase is *independent* of $S'$. In other words, $S'|VIEW_{\mathcal{A}}$ is uniform.

Even without leakage considerations, it is not immediately clear whether one can hope to achieve oblivious secret sharing robust to malicious parties. Consider, as an example, the Shamir secret sharing scheme. The dealer can sample random values for individual shares; but in order to ultimately make the shares consistent, he must somehow sample from a polynomial—without knowing the polynomial!

We show that this *can* be done. We present a $\lambda$ leakage-resilient (oblivious) VSS protocol for $\lambda = \Omega(1)$, tolerating $t \leq \frac{n}{3+\delta}$ malicious parties (for any constant $\delta > 0$).[6] Our construction uses the tool of a *weakly* leakage-resilient VSS protocol as a black box (see Definition 2.4.2). At a high level, the protocol works by having the dealer sample and share two random values $x$ and $y$ using the weakly leakage-resilient protocol; the final output will be $\mathsf{Ext}_2(x, y)$, where $\mathsf{Ext}_2$ is a two-source extractor. To ensure that information is never leaked on $x$ and $y$ together, the dealer first samples and verifiably secret shares $x$, erases it, then samples and verifiably secret shares $y$. (Note that we do not need to assume complete erasures, but rather can allow some fraction of information to remain, which is simply treated as leakage). As before, to ensure independence, instead of sharing $x$ and $y$ to all parties, he will share $x$ and $y$ among two disjoint committees, which are selected by all parties using a version of the Feige committee election protocol.

[6]However, note that the guarantees of our protocol require a large number of parties.

**Theorem 2.4.14.** *Let $n = (3+\delta)t$ for any constant $\delta > 0$. Then for any constant $\lambda \leq \frac{\delta}{4(5+3\delta)}$, there exists a leakage-resilient oblivious VSS protocol tolerating $t$ malicious parties that runs in $O(1)$ rounds.*

*Proof.* Let $\delta'$ be any constant such that $\delta' < \delta$. Fix any constant $0 < \epsilon < 1$. We construct the desired protocol $(\mathsf{Share}_{\mathsf{LR}}, \mathsf{Rec}_{\mathsf{LR}})$, making use of the following tools:

1. $\mathsf{Elect}$: Feige's 1-round public-coin protocol to elect a primary committee of size approximately $n' = n^\epsilon$, as in Lemma 2.2.9.
2. $(\mathsf{Share}_{\mathsf{WLR}}, \mathsf{Rec}_{\mathsf{WLR}})$: a $\left(\lambda, \frac{1}{2}\right)$-*weakly* leakage-resilient VSS protocol for $n'$ parties tolerating $t' = \frac{n'}{3+\delta'}$ corrupted parties, terminating in $O(1)$ rounds, as in Theorem 2.4.3. (Recall $\frac{1}{2}$ refers to the fraction of entropy guaranteed to remain in the secret.)
3. $\mathsf{Ext}_2 : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^m$: a two-source extractor, where $k = \delta' t' \log |\mathbb{F}|$ and $m = \Omega(k)$, as in Theorem 2.2.7.

A description of the protocol $(\mathsf{Share}_{\mathsf{LR}}, \mathsf{Rec}_{\mathsf{LR}})$ is given in Figure 2.4.

By Lemma 2.2.9, with overwhelming probability in $n$, both committees $\mathcal{E}_1, \mathcal{E}_2$ will be "good," in that they each have size $n^{\epsilon/2} \leq |\mathcal{E}_i| \leq n^\epsilon$ and it holds that $n'_i \geq (3+\delta')t'_i$, where $n'_i = |\mathcal{E}_i|$ and $t'_i = |\mathcal{E}_i \cap \mathcal{C}|$ for $i \in \{1,2\}$. We will thus assume this is the case. Since $n'_i \geq (3+\delta')t'_i$, the validity, reconstruction, and secrecy properties of the $(\lambda, \frac{1}{2})$-weakly leakage-resilient VSS protocol (see Definition 2.4.2) will hold for the $i$th execution of $(\mathsf{Share}_{\mathsf{WLR}}, \mathsf{Rec}_{\mathsf{WLR}})$ with overwhelming probability in $n'_i$ (and thus in $n$).

We now show that $(\mathsf{Share}_{\mathsf{LR}}, \mathsf{Rec}_{\mathsf{LR}})$ satisfies the reconstruction and secrecy properties given in Definition 2.4.13.

**Reconstruction** By the reconstruction property of the underlying $\lambda$-weakly leakage-resilient VSS protocol, the honest parties in $\mathcal{E}_1$ (respectively, in $\mathcal{E}_2$) will agree on the reconstructed value $x' \leftarrow \mathsf{Rec}_{\mathsf{WLR}}()$ (resp, $y' \leftarrow \mathsf{Rec}_{\mathsf{WLR}}$), it will hold that $x' = x$ (resp, $y' = y$), and the honest parties will broadcast this value to all parties in Step 1 of the reconstruction phase. Since a majority of the parties in $\mathcal{E}_i$ are honest, all honest parties in $[n]$ will agree on the values of $x^* = x, y^* = y$, and thus will output the same value $\mathsf{Ext}_2(x^*, y^*)$.

**Secrecy** Assume the dealer is honest. Note that since the dealer erases $x$ (and all values related to $x$) before generating $y$, any leakage function will be a function of purely $x$ or $y$, when conditioned on prior leakage. Thus, conditioned on the leakage, the distribution of $x$ and $y$ will be independent. By the secrecy property of the underlying $\lambda$-weakly leakage-resilient VSS protocol, given the view of the adversary, both $x$ and $y$ retain at least $\frac{1}{2}$ of their original entropy. Therefore, by Theorem 2.2.7, the final output $\mathsf{Ext}_2(x, y)$ will be statistically close to uniform over $\{0,1\}^m$ for $m = \Omega(k)$. □

## 2.5 Disjoint Committee Election

We now exhibit a 1-round public-coin protocol for electing $m = \log^2 n$ *disjoint* "good" committees $\mathcal{E}_1, ..., \mathcal{E}_m$ of size approximately $n^{1/2}$.

$\mathsf{Share}_{\mathsf{LR}}()$:

**Step 1:** Run Elect to elect a committee $\mathcal{E}_1$ of approximate size $n' = n^{\epsilon}$ from the set of non-dealer parties $[n] \setminus \{P^*\}$ (see Lemma 2.2.9).

**Step 2:** The dealer $P^*$ samples a random value $x \leftarrow \mathbb{F}^{\delta' t'}$ and verifiably secret shares it among the parties in the committee $\mathcal{E}_1$, using the WLR-VSS. That is, he acts as a dealer in an execution of $\mathsf{Share}_{\mathsf{WLR}}(x)$. He then erases $x$ (and all values related to $x$).

**Step 3:** Run Elect to elect a second committee $\mathcal{E}_2$ of approximate size $n'$ from $[n] \setminus (\{P^*\} \cup \mathcal{E}_1)$ (see Lemma 2.2.9).

**Step 4:** The dealer $P^*$ samples a random value $y \leftarrow \mathbb{F}^{\delta' t'}$ and verifiably secret shares it among the parties in the committee $\mathcal{E}_2$ in the same fashion. That is, he acts as a dealer in an execution of $\mathsf{Share}_{\mathsf{WLR}}(y)$. He then erases $y$ (and all values related to $y$).

**Step 5:** Each party in $\mathcal{E}_1$ and $\mathcal{E}_2$ broadcasts Accept or Reject, corresponding to whether the dealer was accepted or rejected in $\mathsf{Share}_{\mathsf{WLR}}(x)$ during Step 2 or 4, respectively.

**Local Computation:** The dealer is accepted if a majority of parties in both $\mathcal{E}_1$ and $\mathcal{E}_2$ broadcast Accept. Otherwise, the dealer is rejected.

$\mathsf{Rec}_{\mathsf{LR}}()$:

**Step 1:** All parties in $\mathcal{E}_1$ (respectively, $\mathcal{E}_2$) execute the reconstruction phase $x \leftarrow \mathsf{Rec}_{\mathsf{WLR}}()$ (resp, $y \leftarrow \mathsf{Rec}_{\mathsf{WLR}}()$), on the shares dealt in Step 2 (resp, Step 4). Each committee member broadcasts his reconstructed value of $x$ (resp, $y$).

**Local computation:** Let $x^*, y^*$ be the most common value received from the parties in $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively, in the previous step. Output $s \leftarrow \mathsf{Ext}_2(x^*, y^*)$.

**Figure 2.4:** Leakage-resilient oblivious VSS protocol, $(\mathsf{Share}_{\mathsf{LR}}, \mathsf{Rec}_{\mathsf{LR}})$.

Let $m = \log^2 n$ and $k = n^{1/2}$. We consider the Feige lightest bin protocol with $\frac{n}{k}$ bins (See Section 2.2.4), where instead of taking just the lightest bin, we elect the *$m$ lightest bins.* More explicitly, we define the protocol ElectDisj as follows. In a single round, each party $i \in [n]$ broadcasts a random value $r_i \leftarrow \left[\frac{n}{k}\right]$. For $j = 1, ..., m$, the $j$th committee $\mathcal{E}_j$ is defined to be the parties in the $j$th lightest bin (where ties are further ordered by the existing bin numbering).

**Proposition 2.5.1.** *The protocol* ElectDisj *is a 1-round public-coin protocol for electing $m = \log^2 n$ committees $\mathcal{E}_i$ such that for any constants $\beta, \epsilon > 0$, and any set of corrupted parties $\mathcal{C} \subset [n]$ of size $\beta n$, the following events simultaneously occur with overwhelming probability in $n$:*

1. $\forall i \neq j,\ \mathcal{E}_i \cap \mathcal{E}_j = \emptyset,$
2. $\forall i,\ (1 - \beta - \epsilon)n^{1/2} \leq |\mathcal{E}_i| \leq (1 + o(1))n^{1/2},$
3. $\forall i,\ \frac{|\mathcal{E}_i \cap \mathcal{C}|}{|\mathcal{E}_i|} < \beta + \epsilon.$

*Proof.* Property (1) holds immediately by construction. From the proof of Lemma 2.2.9 (see Remark 2.2.10), with overwhelming probability each bin $b \in [\frac{n}{k}]$ – and in particular, each elected $\mathcal{E}_i$ – has at least $(1 - \beta - \frac{\epsilon}{2})n^{1/2}$ *honest* parties. It thus remains to show that $|\mathcal{E}_i| \leq (1 + o(1))n^{1/2}$ for each $i$. Indeed, if this holds, then Property (2) holds, and further, with overwhelming probability we will have

$$\frac{|\mathcal{E}_i \cap \mathcal{C}|}{|\mathcal{E}_i|} = 1 - \frac{|\mathcal{E}_i \setminus \mathcal{C}|}{|\mathcal{E}_i|} \leq 1 - \frac{(1 - \beta - \frac{\epsilon}{2})n^{1/2}}{(1 + o(1))n^{1/2}} \leq \beta + \epsilon,$$

implying property (3).

Suppose there exists an elected committee $\mathcal{E}_i$ for which $|\mathcal{E}_i| = n^{1/2} + \ell$. We will argue that with overwhelming probability, $\ell \in o(n^{1/2})$. Since $\mathcal{E}_i$ is one of the $\log^2 n$ lightest bins, it must be that each of the remaining $n^{1/2} - \log^2 n$ bins has size at least $n^{1/2} + \ell$. Now, we know that with overwhelming probability, each of the $\log^2 n$ elected bins $\mathcal{E}_i$ has at least $(1 - \beta - \frac{\epsilon}{2})n^{1/2}$ honest parties. This means that (with overwhelming probability), the total number of parties in all *remaining* (non-elected) bins can be no greater than

$$n - (\log^2 n)\left(1 - \beta - \frac{\epsilon}{2}\right)n^{1/2}.$$

Thus, with overwhelming probability, we must have

$$\begin{aligned}
&(n^{1/2} - \log^2 n)(n^{1/2} + \ell) \leq n - (\log^2 n)\left(1 - \beta - \frac{\epsilon}{2}\right)n^{1/2}. \\
&\implies (n^{1/2} - \log^2 n)\ell \leq n - (\log^2 n)\left(1 - \beta - \frac{\epsilon}{2}\right)n^{1/2} - (n - n^{1/2}\log^2 n). \\
&\implies \ell \leq \frac{\left(\beta + \frac{\epsilon}{2}\right)n^{1/2}\log^2 n}{n^{1/2} - \log^2 n} \\
&\qquad \leq \left(\beta + \frac{\epsilon}{2}\right)\log^2 n + 1.
\end{aligned}$$

In particular, this implies that each elected committee $\mathcal{E}_i$ must satisfy

$$\begin{aligned}|\mathcal{E}_i| &\leq n^{1/2} + \ell \\ &\leq n^{1/2} + \left(\beta + \frac{\epsilon}{2}\right)\log^2 n + 1 \\ &\in (1+o(1))n^{1/2},\end{aligned}$$

as desired.

□

## 2.6 Unbiased Coin Tossing with Leakage

In this section, we construct our final leakage-resilient coin tossing protocol, as characterized by Definition 2.3.1. Our construction makes black-box use of the tools developed in the previous sections: in particular, a weakly leakage-resilient verifiable secret sharing (WLR-VSS) protocol (from Section 2.4.2), and a disjoint committee election protocol (from Section 2.5).

Recall we are within the model of a synchronous point-to-point network with broadcast, and that channels are assumed to be authenticated and private (with leakage). Our results are information theoretic, without cryptographic assumptions.

**Theorem 2.6.1.** *For any constants $\delta, \epsilon > 0$, any $\lambda \leq \frac{\delta(1-\epsilon)}{10+6\delta}$, any sufficiently large $n \geq (3+\delta)t$, and any $m$, there exists a $\lambda$-leakage-resilient $n$-party distributed coin tossing protocol tolerating $t$ malicious parties that generates an $m$-bit string that is statistically close to uniform (w.r.t. $n$), and terminates in $O(1)$ rounds.*

*Proof.* Let $\delta'$ be any constant such that $\delta' < \delta$, and take $\mathbb{F}$ to be a field for which $\log|\mathbb{F}| \geq \max\{2n,\ m(.99\epsilon(\frac{2}{3}\log^2 n))^{-1}(\delta' \frac{n^{1/2}}{3+\delta'})^{-1}\}$ (where the second requirement comes from the output length properties of the extractor). In Figure 2.5, we construct the desired coin tossing protocol CoinToss using the following tools:

1. Elect: Feige's 1-round public-coin protocol to elect a primary committee of size approximately $\log^2 n$, as in Corollary 2.2.11.

2. ElectDisj: a 1-round public-coin protocol for electing $\log^2 n$ *disjoint* secondary committees of size $n' \approx n^{1/2}$,[7] as in Proposition 2.5.1.

3. $(\mathsf{Share}_{\mathsf{WLR}}, \mathsf{Rec}_{\mathsf{WLR}})$: a $(\lambda, \epsilon)$ WLR-VSS protocol for $n'$ parties, tolerating $t' \leq \frac{n'}{3+\delta'}$ malicious parties, terminating in $O(1)$ rounds, as in Theorem 2.4.3.

4. $\mathsf{Ext} : (\{0,1\}^d)^{\log^2 n} \to \{0,1\}^r$: a robust multi-source extractor, where $r = .99(\frac{2}{3}\log^2 n)(\epsilon d)$, as in Theorem 4.4.11. The inputs to Ext will be elements of $\mathbb{F}^{\delta' t'}$, which we interpret as elements of $\{0,1\}^d$ for $d = \delta' t' \log|\mathbb{F}|$. By the choice of $|\mathbb{F}|$ above, this gives $r \geq m$.

[7]Note that we will use prime notation (e.g., $n', t', \delta'$) to denote parameters pertaining to the secondary committees.

CoinToss:

**Step 1:** Run Elect to elect a primary committee of approximate size $\log^2 n$ (see Lemma 2.2.9). Denote the set of indices of elected parties by $\mathcal{E} \subset [n]$.

**Step 2:** Run the ElectDisj protocol on the remaining parties $[n] \setminus \mathcal{E}$ to elect $|\mathcal{E}|$ disjoint secondary committees $\mathcal{E}'_1, ..., \mathcal{E}'_{|\mathcal{E}|}$, each of size approximately $n^{1/2}$ (see Prop. 2.5.1).

**Step 3:** $\forall i \in \mathcal{E}$, $P_i$ samples a random value $r_i \leftarrow \mathbb{F}^{\delta' t'}$ and verifiably secret shares it among the parties in his corresponding secondary committee, $\mathcal{E}'_i$. That is, he acts as a dealer in an execution of $\mathsf{Share}_{\mathsf{WLR}}(r_i)$.

**Step 4:** For each $i \in \mathcal{E}$, all parties in the secondary committee $\mathcal{E}'_i$ execute the reconstruction phase $r_i \leftarrow \mathsf{Rec}_{\mathsf{WLR}}()$ on the shares dealt by $P_i$. For any party $i \in \mathcal{E}$ who was rejected as a dealer in the previous step, set $r_i = 0$. Each secondary committee member broadcasts his reconstructed value for $r_i$.

**Local Computation:** Let $r_i^*$ be the most common value received from the parties in secondary committee $\mathcal{E}'_i$ in the previous step. Output $r \leftarrow \mathsf{Ext}(\{r_i^*\}_{i \in \mathcal{E}})$.

**Figure 2.5:** Leakage-resilient coin tossing protocol.

By Proposition 2.5.1, with overwhelming probability in $n$, the disjoint secondary committees $\mathcal{E}'_i$ will be "good," in that they each have size $n^{1/2-\zeta} \le |\mathcal{E}'_i| \le n^{1/2+\zeta}$ for any constant $\zeta > 0$ and it holds that $n'_i \ge (3+\delta')t'_i$, where $n'_i = |\mathcal{E}'_i|$ and $t'_i = |\mathcal{E}'_i \cap \mathcal{C}|$ (where $\mathcal{C}$ is the set of corrupted parties). We will thus assume this is the case. Since $n'_i \ge (3+\delta')t'_i$, the validity, reconstruction, and secrecy properties of the $(\lambda, \epsilon)$ WLR-VSS protocol (see Definition 2.4.2) will hold for the $i$th execution of $(\mathsf{Share}_{\mathsf{WLR}}, \mathsf{Rec}_{\mathsf{WLR}})$ with overwhelming probability in $n'_i$ (and thus in $n$). We now show that the protocol CoinToss satisfies the desired agreement and randomness properties (see Definition 2.3.1).

**Agreement.** By the reconstruction property of the WLR-VSS protocol, for each $P_i \in \mathcal{E}$, the honest parties in $\mathcal{E}'_i$ will agree on the reconstructed value $r_i \leftarrow \mathsf{Rec}_{\mathsf{WLR}}()$ and will broadcast this value to all parties in Step 4 (where $r_i = 0$ if $P_i$ was rejected as a dealer in the sharing phase of the VSS). Since a majority of the parties in $\mathcal{E}'_i$ are honest, all honest parties in $[n]$ will agree on $r_i^* = r_i$ for each $i$, and so will agree on the final output $r$.

**Randomness.** Consider the values $r_i$ reconstructed by each secondary committee $\mathcal{E}'_i$. By the reconstruction property of the WLR-VSS, each $r_i$ is fully determined by the conclusion of the sharing phase (Step 3 of the CoinToss protocol). The secrecy property of the WLR-VSS implies that with overwhelming probability at the end of the sharing phase, even given the view of the adversary up to this point ($\mathsf{view}_{\mathcal{A}}$), each *honest* party's random variable $R_i$ retains at least $\epsilon \cdot (\delta' t' \log |\mathbb{F}|)$ bits of entropy. We now argue that, after conditioning on $\mathsf{view}_{\mathcal{A}}$ (which includes leakage), the resulting distributions $(R_1^* | \mathsf{view}_{\mathcal{A}}), ..., (R_{|\mathcal{E}|}^* | \mathsf{view}_{\mathcal{A}})$ over $\mathbb{F}^{\delta' t'}$ remain *independent*, where for all $j \in \mathcal{E} \cap \mathcal{C}$ we think of $R_j^*$ as fixed.

This is proved via induction on the number of leakage queries made by the adversary. In the base case, before any leakage queries are made, the adversary's view is independent of $R_i^*$ for all honest parties $P_i$ (by the secrecy properties of the WLR-VSS and underlying secret sharing scheme), and the claim holds directly. Now, suppose that $(R_1^*|\mathsf{view}_{\mathcal{A}}), ..., (R_{|\mathcal{E}|}^*|\mathsf{view}_{\mathcal{A}})$ are independent after the first $j$ leakage queries. Recall that a leakage query is made on the secret state of a single party. Further, in the protocol, any given party $P_i$ is contained in at most one secondary committee $\mathcal{E}_{i'}$, and thus only sends, receives, and holds information on at most one random variable $R_{i'}^*$. Thus, a leakage query response on the secret state of party $P_i$ can reveal information on a single variable $R_{i'}^*$ but is *independent* of all the other variables. (Note that this holds even when the choice of queried leakage function depends on prior leakage information on other variables, as this does not reveal *new* information on these other variables).

Therefore, we know that $(R_1^*|\mathsf{view}_{\mathcal{A}}), ..., (R_{|\mathcal{E}|}^*|\mathsf{view}_{\mathcal{A}})$ are independent random variables, and together they have total min-entropy at least $(|\mathcal{E} \setminus \mathcal{C}|)(\epsilon\delta' t' \log|\mathbb{F}|)$. By Lemma 2.2.9, $|\mathcal{E} \setminus \mathcal{C}| \geq (1 - \frac{1}{3+\delta} - \zeta)\log^2 n$ for any constant $\zeta > 0$, with overwhelming probability in $n$. Since the robust multi-source extractor we use can extract even when many of the sources $R_j^*$ are fixed, we can simply take the loose bound $|\mathcal{E}\setminus\mathcal{C}| \geq \frac{2}{3}\log^2 n$. Hence, by Theorem 4.4.11, the final output $r = \mathsf{Ext}(\{r_i^*\}_{i\in\mathcal{C}})$ will be statistically close (w.r.t. $n$) to uniform over $\{0,1\}^m$ with $m = .99(\frac{2}{3}\log^2 n)(\epsilon\delta' t' \log|\mathbb{F}|)$.

□

# Chapter 3

# Secure Computation Against Adaptive Auxiliary Information

In this work, we study the problem of secure two-party and multi-party computation (MPC) in a setting where a cheating polynomial-time adversary can corrupt an arbitrary subset of parties and, in addition, learn arbitrary auxiliary information on the *entire states* of all honest parties (including their inputs and random coins), in an adaptive manner, *throughout the protocol execution.* We formalize a definition of multiparty computation secure against adaptive auxiliary information (AI-MPC), that intuitively guarantees that such an adversary *learns no more than the function output and the adaptive auxiliary information.* In particular, if the auxiliary information contains only partial, "noisy," or computationally invertible information on secret inputs, then *only* such information should be revealed. Our definition is a natural generalization of the standard security notion for MPC, where the adversary is restricted to (static) auxiliary information on the inputs of the honest parties *prior* to the protocol execution.

We construct a universally composable AI-MPC protocol that realizes any (efficiently computable) functionality against malicious adversaries in the common reference string (CRS) model, based on the linear assumption over bilinear groups and the $n$-th residuosity assumption. Our protocol tolerates an arbitrary number of corruptions, and applies to both the two-party setting as well as the multi-party setting. Apart from theoretical interest, our result has interesting applications to the regime of leakage-resilient cryptography. Indeed, our result is already used as an essential tool for constructing leakage-resilient MPC protocols in the leak-free preprocessing model [BGJK12] (as described in Chapter 4).

At the heart of our construction is a new two-round oblivious transfer protocol secure against *malicious* adversaries who may receive adaptive auxiliary information, in the CRS model. We believe that this may be of independent interest.

We begin in Section refsec:WLR-techniques by giving an overview of our solution. In Section 3.2, we present some definitions and tools that are used in our construction. In Section 3.3, we formally describe our model and security definition. In Section 3.4, we present the technical core of our work: the construction of an oblivious transfer protocol secure against adaptive auxiliary information in the *semi-malicious* model. In Section 3.5, we present our complete MPC protocol in the *semi-malicious model*, together with a proof of security. In Section 3.6, we provide a compiler taking any protocol secure against adaptive auxiliary information in the semi-malicious model to one secure against *malicious* adversaries in the standalone model. Finally, in Section 3.7, we describe an analogous compiler within the (stronger) universal composability framework.

## 3.1 Technical Overview

Our starting point is the GMW paradigm for building MPC protocols [GMW87].

**The First Approach.** Recall the GMW paradigm begins by designing an MPC protocol secure against semi-honest (i.e., passive) adversaries, and then compiles the protocol into one secure against malicious adversaries by "enforcing" semi-honest behavior via use of zero-knowledge proofs, a commitment scheme, and a coin-tossing protocol.

We begin by mirroring this approach in the setting of adaptive auxiliary input. We directly achieve an MPC protocol that is secure against adaptive auxiliary information in the semi-honest setting by instantiating the basic GMW protocol with the oblivious transfer protocol of [BCH11] (that has analogous semi-honest security properties). More generally, building on the techniques of [GJS11, BCH11], one can show that *any* adaptively secure MPC protocol can be easily converted into one achieving security against adaptive auxiliary information within the semi-honest model.

Continuing onto the GMW compiler, we see that zero-knowledge proofs secure against adaptive auxiliary information were already constructed in [GJS11, BCH11], and *equivocal* commitment schemes [FS89] were also shown to have the required security properties [GJS11, BCH11]. We are thus almost within reach of our final goal. Unfortunately, in the remaining step, one runs into serious problems. It is not clear how to construct a *coin-tossing* protocol secure against adaptive auxiliary information.[1] The problem is that in order to reduce the malicious security of the compiled protocol to the semi-honest security of the original protocol, the coin-tossing protocol to be used in the compiler must be *fully simulatable*, in that the simulator must be able to choose the output of the coin toss, and simulate the protocol to force this output, for both honest and corrupted parties. This is not only troublesome, but even recently shown to be impossible to achieve if the adversary may attain joint leakage on the secret states of all honest parties [CLL$^+$13].

We now briefly illustrate the problem with constructing such a coin-tossing protocol. Let us consider a simple template for an $n$-party coin-tossing protocol to generate (private) randomness for a party $P_i$. First, each party $P_j$ (s.t. $j \neq i$) commits to a random string $r_j$, then $P_i$ commits to $r_i$, and finally, each $P_j$ decommits. The output is $\oplus_{\ell=1}^{n} r_\ell$. Now, note that if $P_i$ is honest, then the simulator can easily obtain the desired output by first extracting the values $r_j$, and then choosing an "appropriate" $r_i$ (that is consistent with the desired output). On the other hand, if $P_i$ is corrupted, then in the plain setting (without adaptive auxiliary information) the simulator can cheat in the decommitment step in order to achieve the desired output. In our setting, however, the adversary can learn auxiliary information on the decommitment value *before* the simulator is able to extract $r_i$; as a result, the simulator cannot "change its mind" about the decommitment later on (and thus cannot force the desired output).[2]

[1]We remark that the leakage-resilient coin tossing result of [BGK11] is not relevant to this setting. Their construction requires an honest majority of parties (in order to attain stronger, information theoretic guarantees), whereas our model allows an arbitrary number of corruptions.

[2]We stress that this attack succeeds even if one uses equivocal commitments [FS89].

One may consider alternative templates for coin-tossing in an attempt to bypass the above problem, for example, by changing the order of commitments by the parties. Unfortunately, in each such attempt, it seems that the simulator fails to force the coins for *either* the honest party, *or* the adversary (while, as discussed above, we need the simulator to be able to force the coins in *both* of these cases).

**A New Stepping Stone: "Semi-Malicious" Adversaries.** We thus abandon the approach of mimicking the GMW paradigm "out of the box." We instead consider a different intermediate step, lying closer to security against malicious adversaries, with the goal of eliminating the necessity for fully simulatable coin-tossing in the final compiler. This amounts to constructing protocols that remain secure even if an adversary potentially uses "bad" randomness in the protocol execution. To formalize this requirement, we consider the notion of a *semi-malicious* adversary that follows the protocol execution (similar to a semi-honest adversary), but can choose its random coins (and inputs) in any arbitrary manner.[3]

Once we construct a protocol for semi-malicious adversaries (that can learn arbitrary auxiliary information), we can easily compile it into a secure protocol for *malicious* adversaries by standard techniques. We do so using a modified version of the GMW compiler adapted to our setting, implemented with equivocal commitments [FS89, CLOS02] and the UC-NIZKs of [GOS06a] that were shown to be secure against adaptive auxiliary information by Garg et al. [GJS11]. (We refer the reader to the technical sections for more details.) The task then remains to construct an MPC protocol that is secure against adaptive auxiliary information in the presence of semi-malicious adversaries.

A close look at the basic GMW construction reveals that constructing semi-malicious MPC reduces to constructing a semi-malicious oblivious transfer (OT) protocol. (We note that this observation is also implicit in [IPS08].) Since our goal is to protect against adversaries who may learn adaptive auxiliary information, we aim to construct OT protocols with similar security guarantees against semi-malicious adversaries. We discuss this next.

**Semi-Malicious OT.** Our starting point is the adaptively secure semi-honest OT protocol of Canetti et al. [CLOS02]. The [CLOS02] construction follows the "standard template" of [EGL85] for semi-honest OT, but replaces the underlying encryption scheme with a non-committing encryption (NCE) scheme [CFGN96]. Namely, (1) The receiver $R$ generates and sends two public keys $pk_0, pk_1$ for the (non-committing) encryption scheme—one for which he knows the secret key, and one "obliviously" sampled; and (2) the sender $S$ sends an encryption of each of his messages $m_i$, under the corresponding public key $pk_i$. The [CLOS02] scheme was shown to be secure against adaptive auxiliary information in the semi-honest model (using our terminology) by [BCH11]. However, the protocol fails in the *semi-malicious* model. Indeed, a semi-malicious receiver can simply choose bad randomness to "obliviously" sample public keys for which he can decrypt (and thus learn both messages of the sender). Further, a semi-malicious sender may be able to create "malformed" ciphertexts that cause the honest receiver to abort depending on his secret input. Circumventing these fatal roadblocks demands a new set of techniques. We solve these problems as follows:

- First, we construct an underlying NCE scheme with strong security properties, which will

[3]The notion of semi-malicious adversaries is somewhat similar in spirit to the notion of defensible adversaries considered by [HIK$^+$11]. We refer the reader to Section 3.3 for a comparison of the two notions.

guarantee security in the OT protocol *as long as the adversary's randomness does not fall within a very small "bad" set.* We achieve this by building an NCE scheme where the public keys generated via the oblivious key generation algorithm are almost always *lossy* (except if they belong to some exponentially small set, such as the set of DDH tuples). Now, unless the adversary's randomness falls within this very small set, the encryption of non-requested messages under his obliviously sampled public keys will *information theoretically* hide the messages.

- Second, we develop a new methodology for generating private randomness that *prevents* a malicious party from choosing randomness within this small bad set. The challenge is doing so *in the presence of adaptive auxiliary information*, and while *simultaneously* providing the simulator the necessary flexibility to "force" any randomness of his choice for honest parties.

  A potential idea is to design a modified coin tossing protocol to ensure a malicious party's output randomness still maintains sufficient entropy in the auxiliary information setting. However, approaches of this kind seem to inherently necessitate an *a priori bound* on how much auxiliary information can be handled: if honest parties cannot hold onto *any* secret entropy during the protocol, this path appears hopeless.

  We provide a different approach. We construct a *non-interactive* randomness generation procedure that achieves the desired properties by use of *lossy trapdoor functions* (LTDF), together with a CRS. Namely, each party $P_i$ is assigned an LTDF seed $\sigma_i$ in the CRS; each time the party must sample randomness in the protocol, he first chooses a random value $r$ in the LTDF domain, and then uses the LTDF evaluation $F(\sigma_i, r)$ as his protocol randomness. Loosely speaking, in the simulation, honest parties will be assigned seeds for *injective* functions, whereas corrupted parties will be assigned (computationally indistinguishable) seeds for *lossy* functions. This allows the simulator to efficiently "explain" any possible output for honest parties, while simultaneously restricting malicious parties to a small set of attainable output values that does not "hit" the small set of *bad* values. We refer the reader to Section 3.4.3 for more details.

We prove that the resulting OT protocol is secure against adaptive auxiliary information in the semi-malicious model. This constitutes the technical heart of our construction.

**Final Touches.** While the above ideas essentially handle the issue of "bad" randomness, we still need to find a way to answer the auxiliary information queries of the adversary correctly. Our starting point for this is the observation of [GJS11, BCH11] that adaptive security (without erasures) provides simulators that can generate random tapes for parties, which can be used to answer auxiliary information queries. However, this is possible if the simulator is able to decide its random tape *after* viewing the (possibly bad) random tape of the adversary. Unfortunately, depending upon the "structure" of the protocol (e.g., if the honest party is required to proceed before a corrupted party), this may not always be possible (since we do not know how to construct a fully simulatable coin-tossing protocol). We address this problem as follows: to somewhat "soften" this asymmetry, instead of generating the entire random tapes of each party a priori, we generate them in an "online" fashion. In part because the GMW semi-honest protocol provides perfect security in the OT-hybrid model, it turns out that this essentially suffices for simulation. Combining all the above ideas, we are finally able to construct a simulator (for the GMW protocol instantiated with our OT protocol) that has the ability to perform "honest party state reconstruction" at every step of the protocol using the auxiliary information and the state of the adversary.

Some important issues still must be addressed, and are discussed within the paper in detail. For example, in the GMW protocol (on which we base our construction), it is not clear how to send the output shares in a manner consistent with prior auxiliary information received by the adversary, in the multi-party setting for more than two parties. Recall that in the GMW protocol, each party broadcasts its output share to everyone. However, the simulator is given access only to the *combined* value of all honest party shares (corresponding to the function output), and it may not be computationally feasible to directly simulate the individual output shares consistent with prior auxiliary information. To solve this issue, we use a specific re-randomization technique that reduces the number of "unknown" output shares to a single "unknown" output share (that can be easily determined from the protocol output and the adversary's state). We defer more details to the technical sections of the paper.

## 3.2 Preliminaries

In this section, we present some basic notions and definitions that are used in our construction.

### 3.2.1 Non-committing Encryption

The notion of *non-committing encryption* was introduced by Canetti *et. al.* [CFGN96]. Informally, non-committing (bit) encryption schemes are semantically secure, possibly interactive encryption schemes, with the additional property that a simulator can generate special ciphertexts that can be "opened" to (i.e. demonstrated to be the encryption of) both 0 and 1.

**Definition 3.2.1.** [CFGN96, CDMW09] A non-committing (bit) encryption scheme consists of a tuple
$(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{NCSim})$, where $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a semantically secure encryption scheme, and $\mathsf{NCSim}$ is a PPT simulation algorithm that on input $1^k$ outputs a tuple $(e, c, r_G^0, r_E^0, r_G^1, r_E^1)$ such that for every $b \in \{0,1\}$ the following distributions are computationally indistinguishable:

1. The joint view of an honest sender and an honest receiver in a normal encryption of $b$:

$$\{(e, c, r_G, r_E) : (e,d) \leftarrow \mathsf{Gen}(1^k; r_G), c \leftarrow \mathsf{Enc}_e(b; r_E)\} .$$

2. A simulated view of an encryption of $b$:

$$\{(e, c, r_G^b, r_E^b) : (e, c, r_G^0, r_E^0, r_G^1, r_E^1) \leftarrow \mathsf{NCSim}(1^k)\} .$$

**Augmented NCE.** In our MPC protocol, we will actually use an "augmented" NCE scheme that has the following three additional properties:

**Oblivious key generation:** It should be possible to sample an encryption key without "knowing" a corresponding secret key. More precisely, there exists an *oblivious key generation* algorithm, denoted by $\mathsf{OGen}$, that takes as input security parameter $1^k$, and outputs a key $e \leftarrow \mathsf{OGen}(1^k)$, where $e$ is indistinguishable from the encryption key chosen by the normal key generation algorithm $\mathsf{Gen}$. Furthermore, $\{\mathsf{Enc}_e(0)\}$ is indistinguishable from $\{\mathsf{Enc}_e(1)\}$ even given the randomness input to $\mathsf{OGen}$.

**Invertible samplability:** The key generation and the oblivious key generation algorithms $\mathsf{Gen}$ and $\mathsf{OGen}$ should be "invertible." That is, given an output that lies in the range of $\mathsf{Gen}$ (resp., $\mathsf{OGen}$) that was potentially generated via a *different* algorithm (e.g., $\mathsf{NCSim}$), we can efficiently generate randomness that "explains" the output as being generated via $\mathsf{Gen}$ (resp., $\mathsf{OGen}$).

Explicitly, there exist two inverting algorithms, as follows. The first inverting algorithm takes any (simulated) key pair $(e, d^b)$ from the output of $\mathsf{NCSim}$ (for either $b \in \{0,1\}$) and outputs $r$ such that $\mathsf{Gen}(r) = (e, d^b)$. The second inverting algorithm receives any $e$ output by $\mathsf{Gen}$ or $\mathsf{NCSim}$ and and outputs $r$ such that $\mathsf{OGen}(r) = e$.

To simplify notation, throughout the paper, we will make the simplifying assumptions that: (a) the $\mathsf{Gen}$ algorithm simply outputs the input randomness $r$ as the decryption key $d$, and (b) the $\mathsf{OGen}$ algorithm simply outputs the input randomness $r$ as the public key $e$. Thus, given these assumptions, we note that both the above described inverting algorithms are "trivial." Indeed, the specific NCE scheme that we use (described in Section 3.4.2) satisfies both these assumptions.

**Alternative simulation:** Note that in the standard NCE definition, $\mathsf{NCSim}$ generates a simulated ciphertext (and randomness values) *together* with an encryption key $e$. For our purposes, we want a slightly stronger property, where we can generate a simulated ciphertext for a *fixed* encryption key – namely, one that is obliviously sampled by another party.[4]

Explicitly, there exists an *alternative simulation* algorithm, denoted by $\mathsf{NCSim}'$, that takes as input an (obliviously generated) public key $e$, and outputs a tuple $(c, r_0, r_1)$, such that for every $b \in \{0,1\}$, the distribution

$$\{(e, c, r_E) : e \leftarrow \mathsf{OGen}(1^k), c \leftarrow \mathsf{Enc}_e(b; r_E)\}$$

is computationally indistinguishable from the distribution

$$\{(e, c, r_E^b) : e \leftarrow \mathsf{OGen}(1^k), (c, r_E^0, r_E^1) \leftarrow \mathsf{NCSim}'(e)\}.$$

Here, we remark that the first two (additional) properties of augmented NCE were also used by [CLOS02] in their construction of adaptively secure OT protocols.

**NCE Construction.** In this work, we will use a variant of the NCE construction due to Choi *et. al* [CDMW09], which can be based on any public-key encryption (PKE) scheme that is *simulatable* [DN00]. Loosely speaking, a simulatable PKE scheme is a semantically secure encryption

[4]Jumping ahead: this property will be required to prove security of our OT protocol against a *semi-malicious* receiver. Here, unlike the semi-honest setting, the simulator (simulating an honest sender) cannot choose the NCE encryption keys that are used, and must be able to generate simulated NCE ciphertexts for keys that are given to him during the OT protocol. We refer the reader to Section 3.4 for details.

scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with oblivious sampling algorithms $(\mathsf{OGen}, \mathsf{OEnc})$ for both public keys and ciphertexts, and additionally with respective inverting algorithms $(\mathsf{IGen}, \mathsf{IEnc})$ that take as input any public key (resp., ciphertext), and output randomness that is consistent with *obliviously* sampling the given public key (resp., ciphertext).

**Definition 3.2.2.** A *simulatable PKE scheme* consists of a tuple $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{OGen}, \mathsf{OEnc}, \mathsf{IGen}, \mathsf{IEnc})$, such that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a semantically secure encryption scheme and, for all messages $m \in \{0,1\}^\ell$, the following distributions are computationally indistinguishable:

1. The joint view of a sender and receiver who *obliviously* sampled their public key and ciphertext:
$$\{(\mathsf{pk}, c, r_G, r_E) : \mathsf{pk} \leftarrow \mathsf{OGen}(1^k; r_G), c \leftarrow \mathsf{OEnc}_{\mathsf{pk}}(1^k; r_E)\}.$$

2. A simulated view of the same, where the public key and ciphertext are *not* generated obliviously, and the randomness is given by the corresponding inverting algorithms:
$$\{(\mathsf{pk}, c, \hat{r}_G, \hat{r}_E) : \mathsf{pk} \leftarrow \mathsf{Gen}(1^k; r_G), c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m; r_E), \hat{r}_G \leftarrow \mathsf{IGen}(\mathsf{pk}), \hat{r}_E \leftarrow \mathsf{IEnc}(c)\}.$$

**Theorem 3.2.3.** *There exists an augmented NCE scheme based on any simulatable public-key encryption scheme.*

*Sketch of proof.* The construction and proof are nearly identical to that of Choi *et. al.* [CDMW09]. We note that the precise result of Choi *et. al.* assumes a slightly weaker primitive (namely, a *trapdoor* simulatable PKE scheme, a notion they introduce), and constructs a *standard* (not augmented) NCE scheme. We now briefly address the three additional algorithms required to achieve augmented NCE.

The NCE construction of Choi *et. al.* possesses straightforward algorithms for oblivious key generation and invertible samplability, both of which are inherited from the underlying simulatable PKE scheme. Finally, by assuming the underlying PKE scheme is simulatable (instead of only trapdoor simulatable), the simulation of the public key and the ciphertext in their algorithm $\mathsf{NCSim}$ can easily be decoupled. In particular, it becomes possible to generate a simulated ciphertext without requiring any trapdoor information on the generation of the public key. This yields the desired alternative simulator algorithm, $\mathsf{NCSim}'$. □

We refer the reader to Section 3.4.2 for more information on the resulting augmented NCE scheme.

We remark that constructions of simulatable encryption schemes exist under an assortment of computational assumptions (e.g., DDH, RSA [DN00]). In Section 3.4.2, we provide a construction of a specific simulatable encryption scheme based on DDH, with an *additional* desired property (namely, that public keys generated using the oblivious sampling algorithm $\mathsf{OGen}$ are almost always lossy), and show that this property is inherited by the resulting augmented NCE scheme.

### 3.2.2 Equivocal Commitments

Informally speaking, a bit-commitment scheme is *equivocal* if it satisfies the following additional requirement. There exists an efficient simulator that outputs a fake commitment such that: (a) the commitment can be decommitted to both 0 and 1, and (b) the simulated commitment and decommitment pair is indistinguishable from a real pair. We now formally define the equivocability property for bit-commitment schemes in the CRS model.

The following definition is adapted from [FS89, CIO98].

**Definition 3.2.4.** A non-interactive bit-commitment scheme $(\mathsf{crsGen}, \mathsf{Com}, \mathsf{Rec})$ in the CRS model is said to be an *equivocal bit-commitment scheme in the CRS model* if there exists a PPT simulator algorithm
$\mathsf{Sim}_{\mathsf{eq}} = (\mathsf{crsSim}_{\mathsf{eq}}, \mathsf{comSim}_{\mathsf{eq}})$ such that $\mathsf{crsSim}_{\mathsf{eq}}$ takes as input the security parameter $1^k$ and outputs a CRS and trapdoor pair, $(\mathsf{crs}, \mathsf{trap})$; and $\mathsf{comSim}_{\mathsf{eq}}$ takes as input such a pair $(\mathsf{crs}, \mathsf{trap})$ and generates a tuple $(\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1)$ of a commitment string $\mathsf{eqcom}$ and two decommitments $\mathsf{eqdec}^0$ and $\mathsf{eqdec}^1$ (for 0 and 1), such that the following holds.

1. For every $b \in \{0,1\}$ and every $(\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1) \leftarrow \mathsf{comSim}_{\mathsf{eq}}(\mathsf{crs}, \mathsf{trap})$, it holds that
$$\mathsf{Rec}(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}^b) = b.$$
2. For every $b \in \{0,1\}$, the random variables
$$\{(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}) : \mathsf{crs} \leftarrow \mathsf{crsGen}(1^k), (\mathsf{eqcom}, \mathsf{eqdec}) \leftarrow \mathsf{Com}(\mathsf{crs}, b)\}$$
and
$$\{(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}^b) : (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{crsSim}_{\mathsf{eq}}(1^k), (\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1) \leftarrow \mathsf{comSim}_{\mathsf{eq}}(\mathsf{crs}, \mathsf{trap})\}$$
are computationally indistinguishable.

**Reusable CRS.** Note that the simulator algorithms $\mathsf{crsSim}_{\mathsf{eq}}$ and $\mathsf{comSim}_{\mathsf{eq}}$ are described as separate algorithms in the Definition 4.4.4 to highlight that it is not necessary to create a separate CRS for every equivocal commitment, i.e., the CRS is *reusable*. In this case, Definition 4.4.4 can be extended in a straightforward manner to consider indistinguishability of an honestly generated tuple consisting of CRS and polynomially many commitment, decommitment pairs, from a simulated tuple. Explicitly, we will use the following property.

**Corollary 3.2.5.** *For any polynomial $p(k)$, and any collection of bits $\{b_1, ..., b_{p(k)}\}$, the following distributions are indistinguishable:*
$$\left\{\left(\mathsf{crs}, \{\mathsf{eqcom}_i, \mathsf{eqdec}_i\}_{i=1}^{p(k)}\right) : \mathsf{crs} \leftarrow \mathsf{crsGen}(1^k), (\mathsf{eqcom}_i, \mathsf{eqdec}_i) \leftarrow \mathsf{Com}(\mathsf{crs}, b_i)\right\}$$
*and*
$$\left\{\left(\mathsf{crs}, \{\mathsf{eqcom}_i, \mathsf{eqdec}_i^{b_i}\}_{i=1}^{p(k)}\right) : \begin{array}{l} (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{crsSim}_{eq}(1^k) \\ (\mathsf{eqcom}_i, \mathsf{eqdec}_i^0, \mathsf{eqdec}_i^1) \leftarrow \mathsf{comSim}_{eq}(\mathsf{crs}, \mathsf{trap}) \end{array}\right\}.$$

*Proof.* Follows from Definition 4.4.4 by a standard hybrid argument. □

**Theorem 3.2.6** ([FS89, CLOS02]). *Assuming the existence of one-way functions, there exists an equivocal bit commitment scheme in the (reusable) CRS model.*

**Remark 3.2.7.** Throughout this paper, we assume that our equivocal commitment scheme has two additional properties: namely, that the decommitment information $\mathsf{eqdec}$ contains *all randomness* used to generate the corresponding commitment, and that the commitment scheme is *statistically binding*. Indeed, the scheme of Feige and Shamir [FS89] satisfies these additional properties.

**String Equivocal Commitments.** For our purposes, we actually use *string* equivocal commitment schemes. Note that such a scheme can be easily constructed by simply repeating the above bit commitment scheme in *parallel.*

**Definition 3.2.8.** An *extractable* equivocal commitment scheme in the CRS model is an equivocal commitment scheme $(\mathsf{crsGen}, \mathsf{Com}, \mathsf{Rec}, \mathsf{Sim}_{\mathsf{eq}})$ (as in Definition 4.4.4) with additional algorithms $(\mathsf{crsGen}_E, E)$ such that

$$\{\mathsf{crs} \leftarrow \mathsf{crsGen}_E(1^k)\} \overset{c}{\cong} \{\mathsf{crs} \leftarrow \mathsf{crsGen}(1^k)\},$$

and for all PPT adversaries $\mathcal{A}$,

$$\Pr[(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{crsGen}_E(1^k); \mathsf{com} \leftarrow \mathcal{A}(\mathsf{crs}); y \leftarrow E(\mathsf{crs}, \mathsf{com}, \mathsf{trap}) : \exists\ \mathsf{dec}, x \neq y \\ \text{s.t. } x = \mathsf{Rec}(\mathsf{crs}, \mathsf{dec}, \mathsf{com})] \leq \mathsf{negl}(k).$$

**Remark 3.2.9.** For our purposes, we want an extractable equivocal commitment scheme for *strings.* Note that such a scheme can be easily constructed by simply repeating the above bit commitment scheme in *parallel* (as was done explicitly in the previous sections). In this section, we abbreviate this notation, and denote a commitment to a string of length $m$ as a *vector* $\mathsf{eqcom} = (\mathsf{eqcom}_1, ..., \mathsf{eqcom}_m)$, with corresponding decommitment vector $\mathsf{eqdec} = (\mathsf{eqdec}_1, ..., \mathsf{eqdec}_m)$. The simulator algorithm $\mathcal{S}^{\mathsf{com}}$ produces a commitment vector and a *pair* of decommitment vectors $d^0 = (\mathsf{eqdec}_1^0, ..., \mathsf{eqdec}_m^0)$, $d^1 = (\mathsf{eqdec}_1^1, ..., \mathsf{eqdec}_m^1)$. A decommitment to any particular bit string $a = (a, ..., a_m)$ can be formed by selecting the appropriate decommitment values $(\mathsf{eqdec}_1^{a_1}, \ldots, \mathsf{eqdec}_n^{a_m})$, and will be denoted by $d^a$.

In [GOS06a], it is shown how to construct an extractable equivocal commitment scheme from any equivocal commitment scheme, given any encryption scheme with pseudorandom ciphertexts. In particular, the following statement holds.

**Theorem 3.2.10** ([GS08, GOS06a, FS89]). *There exists an extractable equivocal commitment scheme in the CRS model, based on any trapdoor permutation.*

### 3.2.3 Leakage-Resilient Non-Interactive Zero Knowledge Proof System

**Definition 3.2.11** (Non-interactive proof system). A tuple of PPT algorithms $(K, P, V)$ is called a *non-interactive proof system* for a language $\mathcal{L}$ with an NP relation $\mathcal{R}$ if the following two conditions hold:

- Completeness: For all $x, w$ such that $\mathcal{R}(x, w) = 1$, and for CRS values $\sigma \leftarrow K(1^k)$,

$$\Pr[V(\sigma, x, \pi) = 1 : \pi \leftarrow P(\sigma, x, w)] \geq 1 - \mathsf{negl}(k),$$

  where the probability is taken over $\sigma \leftarrow K(1^k)$ and the random coins of $P$.

- Soundness: For all adversaries $\mathcal{A}$,

  $$\Pr[(x,\pi) \leftarrow \mathcal{A}(\sigma) : V(\sigma,x,\pi) = 1 \text{ if } x \notin \mathcal{L}] \leq \mathsf{negl}(k),$$

  where the probability is taken over $\sigma \leftarrow K(1^k)$ and the random coins of $\mathcal{A}$.

If the soundness condition holds only against ppt adversaries, then we say that $(K, P, V)$ is a non-interactive *argument* system.

**Definition 3.2.12** (LR-NIZK). [GJS11, BCH11] A non-interactive argument system $(K, P, V)$ for an NP relation $\mathcal{R}$ is said to be a *leakage-resilient NIZK* if there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ such that for all PPT adversaries $\mathcal{A}$,

$$\left|\Pr[\mathcal{A}^{PR(\sigma,\cdot,\cdot,\cdot)}(\sigma) = 1 : \sigma \leftarrow K(1^k)] - \Pr[\mathcal{A}^{SR^{\mathcal{O}_w(\cdot)}(\sigma,\tau,\cdot,\cdot,\cdot)}(\sigma) = 1 : (\sigma,\tau) \leftarrow \mathcal{S}_1(1^k)]\right| = \mathsf{negl}(k),$$

where $PR$ and $SR$ are oracles that do the following:

- $PR(\sigma, x, w, L)$ samples $r \leftarrow \{0,1\}^*$, runs the honest prover algorithm $\pi \leftarrow P(\sigma, x, w; r)$ using randomness $r$, computes the leakage $y = L(w||r)$, and returns $(\pi, y)$.
- $SR^{\mathcal{O}_w(\cdot)}(\sigma, \tau, x, w, L)$ samples $r \leftarrow \{0,1\}^*$, runs the simulator algorithm $\pi \leftarrow \mathcal{S}_2(\sigma, \tau, x; r)$ to generate a proof, generates a state translation function $T \leftarrow \mathcal{S}_3(\sigma, \tau, x, r, L)$, makes the request $y \leftarrow \mathcal{O}_w(L \circ T)$ for $L \circ T$ from his own leakage oracle $\mathcal{O}_w$, and returns $(\pi, y)$.

Here, $L$ and $T$ are expressed as polynomial-size circuits. Both the oracles $PR$ and $SR$ output fail if $(x, w) \notin \mathcal{R}$.

**Theorem 3.2.13** ([GOS06a, GJS11]). *There exists a leakage-resilient NIZK argument system for any NP language, based on the Decisional Linear assumption.*

**Remark 3.2.14.** We note that the above theorem is also relevant to our setting of adaptive auxiliary information since leakage can simply be viewed as auxiliary information. When referring to the above theorem, we do not differentiate between leakage and auxiliary information.

### 3.2.4 Lossy Trapdoor Functions (LTDF)

A collection of lossy trapdoor functions (LTDF) [PW11] consists of two families of functions. Functions in one family are injective and can be efficiently inverted using a trapdoor. Functions in the other family are "lossy," in that the size of their image is significantly smaller than the size of their domain. The only computational requirement is that a description of a randomly chosen function from the family of injective functions is computationally indistinguishable from the description of a randomly chosen function from the family of lossy functions.

We consider a slight generalization of a LTDF family, allowing for *seed-dependent domains*, as introduced by Freeman *et. al.* [FGK$^+$10]. Here, the domain size of each function in the family may depend on both the security parameter *and* the corresponding seed. For simplicity of notation, we will simply refer to this as an LTDF family (although we emphasize that this is not the standard notion of an LTDF family as in [PW11]).

**Definition 3.2.15.** [Lossy Trapdoor Functions (with Seed-Dependent Domains)] [PW11, FGK$^+$10] Let $m : \mathbb{N} \rightarrow \mathbb{N}$ and $\ell : \mathbb{N} \rightarrow \mathbb{R}$ be two non-negative functions, and for any $k \in \mathbb{N}$, let $m = m(k)$ and $\ell = \ell(k)$. A *collection of $(m, \ell)$-lossy trapdoor functions with seed-dependent domains* is a 5-tuple of PPT algorithms $(G_{\mathsf{Loss}}, G_{\mathsf{Inj}}, S, F, F^{-1})$ such that:

1. **Sampling a lossy function:** $G_{\mathsf{Loss}}(1^k)$ outputs a seed $\sigma \in \{0,1\}^*$.

2. **Sampling an injective function:** $G_{\mathsf{Inj}}(1^k)$ outputs a pair $(\sigma, \tau) \in \{0,1\}^* \times \{0,1\}^*$. (Here $\sigma$ is a seed and $\tau$ is a trapdoor.)

3. **Sampling an input:** For every value $\sigma$ produced by either $G_{\mathsf{Loss}}$ or $G_{\mathsf{Inj}}$, the algorithm $S(\sigma)$ outputs an element sampled uniformly from the domain $D_\sigma$ of the function $f_\sigma$. We require that $|D_\sigma| \geq 2^{m-1}$.

4. **Evaluation of lossy functions:** For every seed $\sigma$ produced by $G_{\mathsf{Loss}}$, the algorithm $F(\sigma, \cdot)$ computes a function $f_\sigma : D_\sigma \rightarrow \{0,1\}^*$, whose image is of size at most $|D_\sigma| \cdot 2^{-\ell}$.

5. **Evaluation of injective functions:** For every pair $(\sigma, \tau)$ produced by $G_{\mathsf{Inj}}$, the algorithm $F(\sigma, \cdot)$ computes an injective function $f_\sigma : D_\sigma \rightarrow \{0,1\}^*$.

6. **Inversion of injective functions:** For every pair $(\sigma, \tau)$ produced by $G_{\mathsf{Inj}}$ and every $x \in D_\sigma$, we have $F^{-1}(\tau, F(\sigma, x)) = x$.

7. **Security:** The two ensembles $\{\sigma : \sigma \leftarrow G_{\mathsf{Loss}}(1^k)\}_{k \in \mathbb{N}}$ and $\{\sigma : (\sigma, \tau) \leftarrow G_{\mathsf{Inj}}(1^k)\}_{k \in \mathbb{N}}$ are computationally indistinguishable.

For our purposes, we will need a slight strengthening of a LTDF family. Primarily, we require that the injective functions in the family in fact be *bijective*: i.e., they are also surjective onto their target space. In addition, we will require certain conditions to hold with respect to the relation between parameters. We formalize these properties below.

**Definition 3.2.16.** We say that a collection of $(m, \ell)$-lossy trapdoor functions $(G_{\mathsf{Loss}}, G_{\mathsf{Inj}}, S, F, F^{-1})$ is *bijective* and *$(D, \alpha)$-admissible* if it satisfies the following additional properties:

- **Bijective:** For every pair $(\sigma, \tau)$ produced by $G_{\mathsf{Inj}}$, the algorithm $F(\sigma, \tau)$ computes a *bijective* function $f_\sigma : D_\sigma \rightarrow R_\sigma$ (where $R_\sigma$ is the target output space of $f_\sigma$).

- **$(D, \alpha)$-Admissible:** The following hold, corresponding to target output size $D$, where $\alpha$ fraction of the $D$ values are "bad":

  1. *Efficiently invertible domain sampling:* There exists a PPT algorithm $S^{-1}$ such that for each $x \in D_\sigma$, $\{S^{-1}(x)\} \overset{s}{\approx} \{r : S(r) = x\}$.
  2. *Sufficiently large output space:* For each seed $\sigma$ produced by either $G_{\mathsf{Loss}}(1^k)$ or $G_{\mathsf{Inj}}(1^k)$, the size of the output space $R_\sigma$ satisfies

$$|R_\sigma| \geq D.$$

3. *Sufficiently small image of lossy functions:* There exists a negligible function $\mu(k)$ such that for every seed $\sigma$ produced by $G_{\mathsf{Loss}}$,
$$\alpha \cdot \left(|D_\sigma| \cdot 2^{-\ell(k)}\right) < \mu(k).$$
Recall that $|D_\sigma| \cdot 2^{-\ell(k)}$ is an upper bound for the image size of a lossy function with seed $\sigma$.

Note that for any pair $(D', \alpha')$ for which $D' \leq D$ and $\alpha' \leq \alpha$, the property of being $(D, \alpha)$-admissible immediately implies $(D', \alpha')$-admissibility. We now show that the composite residuosity-based LTDF family construction (with seed-dependent domains) of Freeman et. al. [FGK+10] satisfies our required properties.

**Theorem 3.2.17.** *[FGK+10] Under the decisional composite residuosity assumption, there exists a* bijective, $(D, \alpha)$-admissible *collection of lossy trapdoor functions* $(G_{\mathsf{Loss}}, G_{\mathsf{Inj}}, S, F, F^{-1})$ *with seed-dependent domains, for the following two choices of* $(D, \alpha)$*:*

*1.* $D = \binom{4k}{k}$. $\alpha = \binom{3k}{k} / \binom{4k}{k}$.

*2.* $D = 2^k$. $\alpha = 2^{-k/3}$.

We remark that the parameters for these two cases are highly tailored to our application: namely, two different types of randomness that must be sampled when generating an encryption for our non-committing encryption (NCE) scheme (see Sections 3.4.2 and 3.4.3 for details on the NCE scheme and the use of LTDFs for generating randomness).

*Proof of Theorem 3.2.17.* We begin by recalling the composite residuosity-based LTDF family construction of [FGK+10], which is based on the Damgård-Jurik encryption scheme. The Damgård-Jurik encryption scheme makes use of the fact that, for a modulus $N = PQ$ with $\gcd(N, \phi(N)) = 1$, the map
$$\begin{aligned} \psi_s : \mathbb{Z}_{N^s} \times \mathbb{Z}_N^* &\to \mathbb{Z}_{N^{s+1}}^* \\ (x,\ r) &\mapsto (1+N)^x r^{N^s} \mod N^{s+1} \end{aligned}$$
is a group isomorphism, and is efficiently invertible given trapdoor information $\lambda = \mathsf{lcm}(P-1, Q-1)$. (In the Damgård-Jurik encryption scheme, $x, r, \lambda$ correspond to the message, encryption randomness, and secret key for the encryption scheme, respectively). In the LTDF construction of [FGK+10], each function description consists of a freshly sampled modulus $N$ together with a Damgård-Jurik ciphertext $c$ that is either an encryption of 0 (in the case of a lossy branch) or an encryption of 1 (in the case of an injective branch). The value of $s = s(k')$ below can be chosen as a parameter of the scheme.

Sampling a lossy function $G_{\mathsf{Loss}}(1^{k'})$: Sample a $k'$-bit modulus $N = PQ$ such that $\gcd(N, \phi(N)) = 1$, a random element $r \leftarrow \mathbb{Z}_N^*$, and let $c = r^{N^s} \mod N^{s+1}$. The function description is $\sigma = (N, c)$, and the corresponding domain is $D_\sigma = \mathbb{Z}_{N^s} \times \mathbb{Z}_N^*$.

Sampling an injective function $G_{\mathsf{Inj}}(1^{k'})$: Sample an admissible $k'$-bit modulus $N = PQ$ such that $\gcd(N, \phi(N)) = 1$, a random element $r \leftarrow \mathbb{Z}_N^*$, and let $c = (1+N)r^{N^s} \mod N^{s+1}$. The function description is $\sigma = (N, c)$, the domain is $D_\sigma = \mathbb{Z}_{N^s} \times \mathbb{Z}_N^*$, and the trapdoor is $\tau = (\lambda, r)$, where $\lambda = \mathsf{lcm}(P-1, Q-1)$.

Sampling an input $S(N, c)$: Output a uniform value $(x, y) \leftarrow \mathbb{Z}_{N^s} \times \mathbb{Z}_N^*$.

Evaluation $F((N, c), (x, y))$: Output $c^x y^{N^s} \mod N^{s+1}$.

Inversion $F^{-1}((N, c), z, (\lambda, r))$: Invoke the ciphertext decryption algorithm $\psi_s^{-1}(z) = (x, r^x y)$ using the secret key $\lambda$. Then, using the value of $r$, recover $x$ and $y$.

As proved in [FGK+10], under the decisional composite residuosity assumption, for any polynomial $s = s(k')$, the tuple $(G_{\mathsf{Loss}}, G_{\mathsf{Inj}}, S, F, F^{-1})$ described above is a collection of $(m, \ell)$-lossy trapdoor functions with seed-dependent domains for $m(k') = (k'-1)(s+1)$ and $\ell(k') = (k'-1)s-1$. We now address our additional required properties.

*Bijective.* Consider any seed $\sigma = (N, c)$, $c = (1+N)r^{N^s}$, corresponding to an *injective* branch (i.e., $\sigma$ produced by $G_{\mathsf{Inj}}$). Then the corresponding function $f_\sigma$ computed by $F$ is:

$$f_\sigma : (x, y) \mapsto (1+N)^x (r^x y)^{N^s}.$$

Notice that $f_\sigma(x, y) = \psi_s(x, r^x y)$ is precisely the map given by evaluating the isomorphism $\psi_s$ (defined above) at the pair $(x, r^x y)$. That is, $f_\sigma = \psi_s \circ (1 \times \rho_r)$, where $(1 \times \rho_r)$ is the map on $\mathbb{Z}_{N^s} \times \mathbb{Z}_N^*$ that takes elements $(x, y) \mapsto (x, r^x y)$. We know that $\psi_s$ is bijective, as it is an isomorphism. Further, $(1 \times \rho_r)$ is a bijection, with inverse $(z, w) \mapsto (z, w \cdot (r^z)^{-1})$. Thus, the composition $f_\sigma$ is also bijective, as desired.

*Admissible* Note that it suffices to exhibit $(D, \alpha)$-admissibility for the single pair of values $D = \binom{4k}{k} > 2^k$ and $\alpha = 2^{-k/3} > \binom{3k}{k}/\binom{4k}{k}$, and then $(D, \alpha)$-admissibility for both parameter choices (1) and (2) in the theorem statement will follow.

*Efficient inversion of domain sampling algorithm.* For each seed $\sigma = (N, c)$, the domain sampling algorithm $S$ simply selects a uniform element of $\mathbb{Z}_{N^s} \times \mathbb{Z}_N^*$; thus, efficient inversion is immediate.

*Sufficiently large output space.* To achieve the desired parameters with respect to our security parameter $k$, we will invoke the [FGK+10] construction with $k' = k/4$ and $s(k') = 15$. In this case, we have

$$\begin{aligned} 2^{m(k')-1} &= 2^{(k'-1)(s+1)-1} \\ &= 2^{(k/4-1)(15+1)-1} = 2^{4k-17} \\ &> \binom{4k}{k} \quad \text{for } k \geq 19. \end{aligned}$$

*Sufficiently small image of lossy functions.* For each lossy seed $\sigma = (N, c)$ produced by $G_{\mathsf{Loss}}$, the corresponding domain is $D_\sigma = \mathbb{Z}_{N^s} \times \mathbb{Z}_N^*$. So $|D_\sigma| = N^s \cdot \phi(N) < N^{s+1} < 2^{k'(s+1)}$, since $N$ is a $k'$-bit integer. Thus, for $k' = k/4$ and $s(k') = 15$, we have

$$\begin{aligned}
|D_\sigma| \cdot 2^{-\ell(k)} \cdot \left(\frac{1}{2^{k/3}}\right) &\leq 2^{k'(s+1)} \cdot 2^{-(k'-1)s+1} \cdot 2^{-k/3} \\
&= 2^{k'+s+1} \cdot 2^{-k/3} \\
&= 2^{k/4+15+1} \cdot 2^{-k/3} \\
&\leq 2^{16} \cdot 2^{-k/12},
\end{aligned}$$

which is a negligible function in $k$, as desired.

□

**A note on function descriptions.** For our MPC application, a collection of randomly selected LTDF seeds will be included as part of the CRS. In the LTDF construction of Freeman *et. al.* [FGK+10] described above, the function indices $\sigma$ cannot take arbitrary uniform values, but rather have a specific structure: in particular, the seed contains a modulus of the form $N = PQ$. Thus, with this LTDF instantiation, the CRS of our scheme will in fact be a common *reference* string, instead of a common random string, and there is an inherent assumption that the CRS setup is performed without leaking. However, any new construction of an LTDF family satisfying the properties of Definition 3.2.16 and for which the seeds are uniformly distributed can be immediately plugged in to our construction, alleviating this point. We leave this as an open problem for future research.

## 3.3 Our Model

To define multiparty computation secure against adaptive auxiliary information, we turn to the real/ideal paradigm. Very briefly, we consider a real-world execution where an adversary, in addition to corrupting a number of parties, can adaptively learn arbitrary auxiliary information on the *joint* secret states of the honest parties, throughout the protocol execution. Following the works on leakage-resilient cryptography, we model the above scenario by allowing the adversary to make auxiliary information queries of the form $L$, where $L$ is the circuit representation of an efficiently computable function. On making such a query, the adversary learns $L(\mathsf{state})$, where $\mathsf{state}$ represents the joint secret state of the honest parties. In the ideal world experiment, the ideal world adversary, i.e., the simulator is allowed to request auxiliary information on the inputs of all the parties from the trusted party. We remark that our security model is similar to those considered in some recent works [GJS11, BCH11], although we note that prior works focused only on the two-party case, whereas we deal with both the two-party case and the multi-party case.

Below, we describe a standalone security definition. Later in the paper, we will consider the generalization to the UC setting. We now describe the ideal and real world experiments and then give the formal security definition.

**Ideal World.** We first define the ideal world experiment, where $n$ parties $P_1, \ldots, P_n$ interact with a trusted party for computing a function $f$. As in the standard MPC ideal world experiment, the parties send their inputs to the trusted party and receive the output of $f$ evaluated on all inputs. The main difference from the standard ideal world experiment is that the adversary is allowed to make auxiliary information queries on the (joint) inputs of the honest parties. The ideal world execution proceeds as follows.

- **Inputs:** Each party $P_i$ obtains an input $x_i$. The adversary is given (initial) auxiliary input $z$, selects a subset of parties $M \subset \mathcal{P}$ to corrupt, and receives $x_i$ for every $P_i \in M$.
- **Sending inputs to trusted party:** Each honest party $P_i$ sends its input $x_i$ to the trusted party. For each corrupted party $P_i \in M$, the adversary may select any value $x'_i$ and send it to the trusted party.
- **Trusted party computes output:** Let $x'_1, \ldots, x'_n$ be the inputs that were sent to the trusted party. The trusted party computes $f(x'_1, \ldots, x'_n)$.
- **Adversary learns output:** The trusted party first sends the evaluation $f(x'_1, ..., x'_n)$ to the adversary. The adversary replies with either continue or abort.
- **Honest parties learn output:** If the message is abort, the trusted party sends $\perp$ to all honest parties. If the adversary's message was continue, then the trusted party sends the function evaluation $f(x'_1, \ldots, x'_n)$ to all honest parties.[5]
- **Auxiliary information queries on inputs:** The adversary may send (adaptively chosen) auxiliary information queries in the form of efficiently computable functions $L_j$ (described as a circuit). On receiving such a query, the trusted party computes $L_j(x'_1, \ldots, x'_n)$ and returns the output to the adversary. (We in fact, place further restriction on the communication between the adversary and the trusted party w.r.t. the auxiliary information queries; we discuss this in more detail below.)
- **Outputs:** Honest parties each output the message they obtained from the trusted party. Malicious parties may output an arbitrary PPT function of their initial inputs, auxiliary input, and the messages they obtained from the trusted party.

**Real World.** The real-world execution begins by an adversary $\mathcal{A}$ selecting any arbitrary subset $M \subset \mathcal{P}$ of the parties to corrupt. On being corrupted, each party $P_i \in M$ hands over its input to $\mathcal{A}$. The parties $P_1, \ldots, P_n$ now engage in an execution of a real $n$-party protocol $\Pi$ (without any trusted third party). The adversary $\mathcal{A}$ sends all messages on behalf of the corrupted parties, and may follow an arbitrary PPT strategy. In contrast, the honest parties follow the instructions of $\Pi$. Furthermore, at any point during the protocol execution, the adversary may make auxiliary information queries of the form $L$ and learn $L(\mathsf{state}_{\mathcal{P} \setminus M})$, where $\mathsf{state}_{\mathcal{P} \setminus M}$ denotes the concatenation of the protocol states $\mathsf{state}_i$ of each honest party $P_i$. We allow the adversary to choose the auxiliary information queries adaptively based on all the information that $\mathcal{A}$ received up to that point (including responses to previous such queries). Honest parties have the ability to toss fresh coins at any point in the protocol; these coins are added to the state of that party at the time they are

[5]We can also define a more general case, where $f$ may output a *different* value $f_i(x'_1, ..., x'_n)$ to each party $P_i$. In this setting, the adversary first learns the set of outputs $\{f_i(x'_1, ..., x'_n)\}_{i \in M}$ corresponding to corrupted parties, and then decides whether to abort or to allow the honest parties to receive their respective outputs. We do not dwell on this detail for simplicity of exposition; however, our construction also handles this more general case.

generated. At the conclusion of the protocol execution, each honest party $P_i$ generates its output according to $\Pi$. Malicious parties may output an arbitrary PPT function of the view of $\mathcal{A}$.

**Security Definition.** Having defined the ideal and real world experiments, we now give our formal definition of MPC secure against adaptive auxiliary information. Our definition crucially relies on the notion of *leakage-oblivious simulation* as defined in [GJS11, BCH11]. We recall it below.

*Leakage-Oblivious Simulation.* Loosely speaking, an ideal world adversary, i.e., a simulator $\mathcal{S}$, is said to be leakage-oblivious if the auxiliary information obtained by the simulator is used *only* for the purposes of simulating answers to the auxiliary information queries of the real adversary. More formally, we require that the simulator $\mathcal{S}$ has a special subroutine $\widetilde{\mathcal{S}}$ for handling auxiliary information queries. Whenever $\mathcal{S}$ receives an auxiliary information query $L$ from the real world adversary, $\widetilde{\mathcal{S}}$ is invoked to produce a "state translation circuit" $T$ that takes as input the inputs of the honest parties and produces their joint states. Once $T$ is produced, the ideal functionality is queried on the composed circuit $L \circ T$. When the auxiliary information is returned, it is forwarded directly to the real adversary and $\mathcal{S}$ returns to its state prior to the event. Such a simulator is referred to as a leakage-oblivious simulator.

We now define security w.r.t. the real and ideal world experiments as discussed above, except that we consider leakage-oblivious simulators in the ideal world experiment. The output of the ideal-world experiment consists of the inputs and outputs of all parties, and the answers of all the auxiliary information queries. It is denoted by $\mathsf{IDEAL}^f_{\mathcal{S},M}(1^k, \vec{x}, z)$. The overall output of the real-world experiment consists of the inputs and outputs of all parties at the conclusion of the protocol, and all the auxiliary information learnt by the adversary (including the protocol transcript). It is denoted by $\mathsf{REAL}^{\Pi}_{\mathcal{A},M}(1^k, \vec{x}, z)$. Below, we give our formal security definition.

**Definition 3.3.1** (Multiparty Computation Secure Against Adaptive Auxiliary Information). A protocol $\Pi$ evaluating a functionality $f$ is said to be *secure against adaptive auxiliary information* if for every ppt real adversary $\mathcal{A}$, there exists a ppt leakage-oblivious simulator $\mathcal{S}$ such that for every input vector $\vec{x}$, $z \in \{0,1\}^*$, and $M \subset \mathcal{P}$, it holds that,

$$\left\{\mathsf{IDEAL}^f_{\mathcal{S},M}(1^k, \vec{x}, z)\right\}_{k \in N} \approx_c \left\{\mathsf{REAL}^{\Pi}_{\mathcal{A},M}(1^k, \vec{x}, z)\right\}_{k \in N}.$$

**Remark 3.3.2.** Our security definition guarantees that if the auxiliary information queries of the real adversary have short output, or "noisy" long outputs, or leave the honest parties' inputs computationally unpredictable, then the same restrictions will also hold for the ideal-world auxiliary information. (As a special case of the above, our definition satisfies the basic requirement that if the real adversary learns $\ell$ bits of auxiliary information, then the simulator can also learn at most $\ell$ bits of auxiliary information.)

## 3.3.1 Security Against Semi-Malicious Adversaries

As a stepping stone toward realizing our definition of MPC secure against adaptive auxiliary information in the presence of malicious adversaries, we define the notion of a *semi-malicious* adversary, whose power lies in between the standard notions of malicious and semi-honest adversaries.

**Semi-malicious adversary.** Intuitively, a semi-malicious adversary is similar to a "standard" (real-world) semi-honest adversary, in that it follows the protocol specification. However, it differs from semi-honest adversaries in that it may choose its input and its "random" coins for any protocol step in an online fashion, adaptively, following any arbitrary PPT strategy. Once it has chosen these values, however, it must follow the protocol as specified, given the chosen input, and using the chosen coins in place of the random coins. Furthermore, in our setting, a semi-malicious adversary is allowed to learn arbitrary auxiliary information on the (joint) secret states of the honest parties.

More formally, a *semi-malicious adversary* $\mathcal{A}$ is modeled as an interactive Turing machine (ITM) which, in addition to the standard tapes, has a special auxiliary tape. At the start of the protocol, $\mathcal{A}$ selects for each corrupted party $P_k$ an input $x_k$ (which may depend on the original inputs of corrupted parties), and writes $x_k$ to its special input auxiliary tape. Then in each round of the protocol, whenever $\mathcal{A}$ produces a new protocol message $m$ on behalf of some party $P_k$, it must also append to its special auxiliary tape *some* randomness that *explains its behavior.* More specifically, all of the protocol messages sent by the adversary on behalf of $P_k$ up to that point, including the new message $m$, must exactly match the honest protocol specification for $P_k$ when executed with input $x_k$ and randomness $r_k$ written in the special auxiliary tape. We allow $\mathcal{A}$ to make auxiliary information queries on the joint states of the honest parties in the same manner as discussed earlier. We further assume that the adversary is rushing and hence may choose the randomness $r$ in each round adaptively, after seeing the protocol messages of the honest parties in that round (and all prior rounds), as well as all the auxiliary information that it may have obtained so far. Lastly, the adversary may choose to abort the execution on behalf of $P_k$ in any step of the interaction.

**Definition 3.3.3** (MPC Secure Against Adaptive Auxiliary Information in the Semi-Malicious Model). We say that a protocol $\Pi$ evaluating a function $f$ is *secure against adaptive auxiliary information in the semi-malicious model* if it satisfies Definition 3.3.1 when we only quantify over all semi-malicious adversaries $\mathcal{A}$.

**Remark.** The definition of a semi-malicious adversary is reminiscent of the notion of a *defensible adversary*, introduced by Haitner *et. al.* [HIK+11] within a different setting. One can think of a semi-malicious adversary as a defensible adversary with the additional requirement that he must generate each piece of his defense *at the time it would be used* during the protocol, instead of altogether at the end of the protocol execution. The notion of a semi-malicious adversary was also used in the work of Asharov *et. al.* [AJL+12]. However, in line with the focus of this work, our semi-malicious adversary is also allowed to adaptively learn auxiliary information on the secret states of honest parties.

## 3.4 Semi-Malicious Oblivious Transfer

In this section, we construct an oblivious transfer (OT) protocol that is secure against adaptive auxiliary information for semi-malicious adversaries. This protocol is the technical heart of our result.

We begin in Section 3.4.1 by giving some intuition behind our OT protocol construction. In Section 3.4.2, we work to obtain one of the primary tools that will be used: a non-committing encryption scheme with certain desired properties. In Section 3.4.3 we describe our randomness generation procedure for our OT protocol. Then, in Section 3.4.4, we present our OT protocol construction itself.

### 3.4.1 Overview

We first recall the basic construction of an honest-but-curious OT protocol.

**The "standard" honest-but-curious OT protocol.** This protocol uses as a building block any semantic secure encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with the property that the public keys can be obliviously sampled without knowing a corresponding secret key. We denote this oblivious key generation algorithm by $\mathsf{OGen}$.

1. The receiver, who holds a bit $b \in \{0,1\}$, samples a key pair $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{Gen}(1^k)$, and *obliviously* samples a public key $\mathsf{pk}_{1-b} \leftarrow \mathsf{OGen}(1^k)$. It sends the pair $(\mathsf{pk}_0, \mathsf{pk}_1)$ to the sender.
2. The sender, who holds two strings $M_0, M_1$, sends to the receiver the corresponding ciphertexts $(c_0, c_1)$, where $c_0 \leftarrow \mathsf{Enc}(M_0; \mathsf{pk}_0)$ and $c_1 \leftarrow \mathsf{Enc}(M_1; \mathsf{pk}_1)$.
3. The receiver decrypts $c_b$ using the corresponding secret key $\mathsf{sk}_b$, to retrieve $M_b = \mathsf{Dec}(c_b; \mathsf{sk}_b)$.

This protocol is known to be secure against a "standard" *honest-but-curious* adversary. However, for our purposes, we wish to construct an OT protocol that is secure against *semi-malicious* adversaries, where the adversary is able to (a) learn adaptive auxiliary information from honest parties, and (b) use *bad randomness.* Below, we first give a high-level intuition summarizing our main ideas towards constructing such an OT protocol.

**Towards OT secure against adaptive auxiliary information in semi-malicious model.** In an attempt to construct an OT protocol secure against adaptive auxiliary information, the first idea that comes to mind is to use an NCE scheme, instead of a standard encryption scheme. It was observed in recent works [BCH11, BCG+11] that non-committing encryption is resilient to auxiliary information on both the secret key and the randomness used to encrypt. Indeed, if we use an "augmented" NCE scheme (a notion similar to that used in [CLOS02], then the resulting OT protocol is secure against an honest-but-curious adversary that may receive auxiliary information from the honest party [BCH11].

Unfortunately, one can easily see that the above protocol (even when implemented with an NCE scheme) is still not secure against a *semi-malicious* adversary, who in addition to learning auxiliary information information can also choose his randomness maliciously. The reason is that a semi-malicious receiver can use bad randomness $r^*$ to obliviously generate $\mathsf{pk}_{1-b} = \mathsf{ONCGen}(1^k; r^*)$ such that he can decrypt ciphertexts encrypted with this specific public key. Furthermore, a semi-malicious sender may be able to create "malformed" ciphertexts that cause the honest receiver to abort depending on his secret input.

We fix these problems via two key new insights. First, we choose an underlying non-committing encryption scheme that guarantees security *as long as the adversary's randomness does not fall within a very small set.* We achieve this by using an underlying non-committing encryption scheme with strong correctness guarantees (such that a very small subset of encryption randomness yields malformed ciphertexts), and with the additional property that public keys that are generated using the oblivious key generation algorithm are almost always *lossy* (except if they belong to some exponentially small set, such as the set consisting of DDH tuples). Now, unless the adversary's randomness falls within these one of these two very small sets, all his encryptions will decrypt correctly, and the encryption of non-requested messages under his obliviously sampled public keys will *information theoretically* hide the messages.

Second, we devise a method for generating private randomness that *prevents* the adversary from choosing randomness within these small bad sets, while still providing the simulator full flexibility to "force" randomness of his choice for honest parties. This is achieved via use of *lossy trapdoor functions* (LTDF). Namely, each party $P_i$ is assigned an LTDF seed $\sigma_i$ in the CRS; each time the party must sample randomness in the protocol, he first chooses a random value $r$ in the LTDF domain, and then uses the LTDF evaluation $F(\sigma_i, r)$ as his protocol randomness. Loosely speaking, in the simulation honest parties will be assigned seeds for injective functions, whereas corrupted parties will be assigned (computationally indistinguishable) seeds for *lossy* functions, thus restricting their attainable output randomness values to a very small set that does not "hit" the small set of *bad* randomness values. We now give more details.

### 3.4.2 Augmented NCE with *Lossy* Encryptions

Recall that our goal is to obtain an augmented NCE scheme with the property that public keys generated using the oblivious key generation algorithm are almost always lossy. The specific NCE scheme we use is a slight variant of the one due to Choi *et. al.* [CDMW09], which requires an underlying encryption scheme that has oblivious sampling and inverting algorithms (i.e., that is *simulatable*, as in Definition 3.2.2). For our construction, we will use a simulatable encryption scheme that is also *lossy*. We first describe this underlying lossy encryption scheme, and then show how it fits in to yield an NCE scheme with the desired properties.

**Our Lossy Simulatable Encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{OGen}, \mathsf{OEnc}, \mathsf{IGen}, \mathsf{IEnc})$.**

- **Key generation** Gen. Let $\mathbb{G}$ be a group of prime order $p$. On input security parameter $1^k$, output a pair (pk, sk) where $\mathsf{pk} = (g_1, g_2, g_1^u, g_2^u)$ and $\mathsf{sk} = u$, for $u \leftarrow \mathbb{Z}_p$ and $g_1, g_2 \leftarrow \mathbb{G}$.

  For simplicity of notation, we will sometimes assume the decryption key sk contains *all* randomness used during key generation. Namely, $\mathsf{sk} = \{u, g_1, g_2\}$.
- **Encryption algorithm** Enc. On input a message $m \in \mathbb{G}$ and a public key $(g_1, g_2, g_3, g_4)$, output $(g_1^{\beta_1} g_2^{\beta_2}, g_3^{\beta_1} g_4^{\beta_2} \cdot m)$, where $\beta_1, \beta_2 \leftarrow \mathbb{Z}_p$.
- **Decryption algorithm** Dec. On input a ciphertext $(c_1, c_2)$ and a secret key $\mathsf{sk} = u$, output $m = \frac{c_2}{c_1^u}$.
- **Oblivious sampling algorithms** OGen **and** OEnc. Both these algorithms take as input a security parameter $1^k$ and obliviously generate a public key or a ciphertext, respectively. The algorithm $\mathsf{OGen}(1^k)$ generates a public key by simply choosing at random $g_1, g_2, g_3, g_4 \leftarrow \mathbb{G}$, and outputting $\mathsf{pk} = (g_1, \dots, g_4)$. The algorithm $\mathsf{OEnc}(1^k)$ generates a ciphertext by simply choosing at random $c_1, c_2 \leftarrow \mathbb{G}$, and outputting $c = (c_1, c_2)$. Note that assuming hardness of DDH in $\mathbb{G}$, the resulting public key and ciphertext are computationally indistinguishable from ones generated according to Gen and Enc.
- **Inverting algorithms** IGen **and** IEnc. The algorithm IGen takes as input a public key pk computed by Gen and outputs randomness $r$ such that $\mathsf{OGen}(r) = \mathsf{pk}$. Note that since the algorithm OGen simply outputs the input randomness $r = (g_1, g_2, g_3, g_4)$ as the public key

$\mathsf{pk} = (g_1, g_2, g_3, g_4)$, the algorithm $\mathsf{IGen}$ is trivial. Further, algorithm $\mathsf{IEnc}$ takes as input a ciphertext $c$ and outputs randomness $r$ such that $\mathsf{OEnc}(r) = c$. Again, note that since the algorithm $\mathsf{OEnc}$ simply outputs the input randomness $r = (c_1, c_2)$ as the obliviously sampled ciphertext $c = (c_1, c_2)$, the algorithm $\mathsf{IEnc}$ is trivial.

**Lemma 3.4.1.** *Assuming the hardness of DDH in $\mathbb{G}$, the scheme $\mathcal{E}$ described above is a simulatable encryption scheme, as per Definition 3.2.2.*

*Proof.* We first note that $\mathcal{E}$ is a semantically secure encryption scheme. The security proof follows in the same manner as the El-Gamal encryption scheme, and is therefore omitted. The simulatability property holds trivially for $\mathcal{E}$ due to the specific nature of the oblivious sampling algorithms (see discussion above). $\square$

Next, we claim that when a public key $\mathsf{pk}$ in $\mathcal{E}$ is a non-DDH tuple, then any ciphertexts computed using $\mathsf{pk}$ are "lossy."

**Claim 3.4.2.** *The encryption scheme $\mathcal{E}$ has the property that if a public key $\mathsf{pk}$ is not a DDH tuple then the encryption scheme is lossy: in particular, for any message $m \in \mathbb{G}$, the distribution $\mathsf{Enc}_{\mathsf{pk}}(m)$ is statistically close to uniform.*

The proof follows from Lemma 4 of [PVW08], and is omitted.

**Remark 3.4.3.** Finally, we remark that the scheme $\mathcal{E}$ has perfect completeness for keys generated via the standard algorithm $\mathsf{Gen}$. That is, for any randomness $r_G$, any message $m \in \mathbb{G}$, and any randomness $r_E$, it holds that $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m; r_E)) = m$, where $(\mathsf{pk}, \mathsf{sk}) = \mathsf{Gen}(1^k; r_G)$.

**The Augmented NCE scheme $\mathcal{E}_{\mathsf{NCE}} = (\mathsf{NCGen}, \mathsf{NCEnc}, \mathsf{NCDec}, \mathsf{NCSim}, \mathsf{ONCGen}, \mathsf{NCSim}')$.** We now describe our augmented NCE scheme, which is essentially the NCE scheme of Choi *et. al.* [CDMW09], instantiated with the lossy encryption scheme $\mathcal{E}$ described above. Below, we first recall the basic "honest party" algorithms $(\mathsf{NCGen}, \mathsf{NCEnc}, \mathsf{NCDec})$ of the NCE scheme of [CDMW09]. Here, $K$ is a parameter of the scheme.[6]

- $\mathsf{NCGen}(1^k)$: Generate $4K$ public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_{4K}$ for the underlying encryption scheme, where $K$ of them are generated according to the key generation algorithm $\mathsf{Gen}$, and $3K$ of them are generated according to the oblivious sampling algorithm $\mathsf{OGen}$. More specifically, choose $K$ random coordinates $i_1, \ldots, i_K \leftarrow [4K]$; denote the set of these coordinates by $I$. For every $i \in I$, sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^k)$, and for every $i \in [4K] \setminus I$, obliviously sample $\mathsf{pk}_i \leftarrow \mathsf{OGen}(1^k)$. In addition, choose two random messages $M_0, M_1 \leftarrow \mathbb{G}$ from the message space of the underlying encryption scheme. Output $e = (\mathsf{pk}_1, \ldots, \mathsf{pk}_{4K}, M_0, M_1)$ as the public key, and $d = (I, \{\mathsf{sk}_i\}_{i \in I})$ as the secret key.
- $\mathsf{NCEnc}_e(b)$: Compute $4K$ ciphertexts $c_1, \ldots, c_{4K}$ (one for each of the $\mathsf{pk}_1, \ldots, \mathsf{pk}_{4K}$), of which $K$ are encryptions of $M_b$ and the remaining ones are obliviously sampled. More specifically, choose a random set of coordinates $J \subset [4K]$ of size $K$. For every $j \in [J]$ compute $c_j \leftarrow \mathsf{Enc}_{\mathsf{pk}_j}(M_b)$, and for every $j \in [4K] \setminus [J]$ compute $c_j \leftarrow \mathsf{OEnc}(1^K)$. Output $c = (c_1, \ldots, c_{4K})$.

[6]In [CDMW09], $K$ is simply equal to $k$. For our purposes, $K$ will be set to $K = 3(k + L)$, where $L$ is the leakage bound.

- $\mathsf{NCDec}_d(c_1, \dots, c_{4K})$: Decrypt the $K$ ciphertexts for which it knows the secret key. Namely, decrypt $\{c_i\}_{i \in I}$. If there exists a ciphertext that decrypts to $M_b$, and no another ciphertext decrypts to $M_{1-b}$ then output $M_b$. Otherwise, output $\perp$.

In the following subsection, we describe a simulator algorithm NCSim for the above described NCE scheme, which is a slight modification of that in [CDMW09].

**Simulator NCSim.** Choose $M_0, M_1 \leftarrow \mathbb{G}$ at random. Choose random sets of coordinates $I_0, J_0 \subset [4K]$ each of size $K$, which will correspond to an encryption of 0. From the *remaining* coordinates, choose additional random sets $I_1, J_1 \subset [4K] \setminus (I_0 \cup J_0)$ each of size $K$, with the additional constraint that $|I_1 \cap J_1| = |I_0 \cap J_0|$. Perform the following steps:

1. Generating keys: For $i = 1, \dots, 4K$, let

$$(\mathsf{pk}_i, \mathsf{sk}_i) = \begin{cases} \mathsf{Gen}(1^k; r_G^i) & \text{if } i \in I_0 \cup I_1 \\ \mathsf{OGen}(1^k; \hat{r}_G^i) & \text{otherwise.} \end{cases}$$

2. Generating ciphertext: For $i = 1, \dots, 4K$, let

$$c_i = \begin{cases} \mathsf{Enc}_{\mathsf{pk}_i}(M_0; r_E^i) & \text{if } i \in J_0 \\ \mathsf{Enc}_{\mathsf{pk}_i}(M_1; r_E^i) & \text{if } i \in J_1 \\ \mathsf{OEnc}_{\mathsf{pk}_i}(\hat{r}_E^i) & \text{otherwise.} \end{cases}$$

3. Simulating an opening to $b$: Open the values in $I_b$ (correspondingly, $J_b$) honestly, and use the inverting algorithms to claim the values generated for $I_{1-b}$ (corresp, $J_{1-b}$) were obliviously generated. Namely, set

$$\sigma_G^b = \{I_b, u_G^{b,1}, \dots, u_G^{b,4K}\},$$
$$\sigma_E^b = \{J_b, u_E^{b,1}, \dots, u_E^{b,4K}\}, \text{ where}$$

$$u_G^{b,i} = \begin{cases} r_G^i & \text{if } i \in I_b \\ \mathsf{IGen}(\mathsf{pk}_i) & \text{if } i \in I_{1-b} \\ \hat{r}_G^i & \text{otherwise} \end{cases} \qquad u_E^{b,i} = \begin{cases} r_E^i & \text{if } i \in J_b \\ \mathsf{IEnc}(c_i) & \text{if } i \in J_{1-b} \\ \hat{r}_E^i & \text{otherwise.} \end{cases}$$

Set $e = (M_0, M_1, (\mathsf{pk}_1, \dots, \mathsf{pk}_{4K})), c = (c_1, \dots, c_{4K})$. Also output $\sigma_G^0, \sigma_G^1, \sigma_E^0, \sigma_E^1$. This completes the description of NCSim.

**Properties of Augmented NCE Scheme.** We now prove two properties of the above augmented NCE scheme: first, correctness of decryption holds for all but a tiny fraction of encryption randomness; and second, the NCE scheme inherits certain lossy properties of the underlying encryption scheme.

**Claim 3.4.4** (Correctness of NCE). *Let $\mathbb{G}$ be the group in which the underlying encryption scheme $\mathcal{E}$ is implemented, and let* $(\mathsf{pk}, \mathsf{sk})$ *be any key pair in the output of* $\mathsf{NCGen}$. *Then for any $b \in \{0,1\}$,*

$$\Pr_{r_E}[\mathsf{NCDec}_{\mathsf{sk}}(\mathsf{NCEnc}_{\mathsf{pk}}(b; r_E)) \neq b] \leq \frac{\binom{3K}{K}}{\binom{4K}{K}} + \frac{K}{|\mathbb{G}|}.$$

*Proof.* Let $\mathsf{sk} = (I, \{\mathsf{sk}_i\}_{i \in I})$, and consider any ciphertext $c = (c_1, ..., c_{4K})$ in the image of $\mathsf{NCEnc}_{\mathsf{pk}}(b)$. Recall that $c$ is decrypted using sk by decrypting the underlying ciphertexts $c_i$ for which $i \in I$. There are two events in which $c$ fails to decrypt correctly to $b$: (1) none of the $c_i$ with $i \in I$ satisfies $\mathsf{Dec}_{\mathsf{sk}_i}(c_i) = M_b$, or (2) there exists $c_i$ with $i \in I$ such that $\mathsf{Dec}_{\mathsf{sk}_i}(c_i) = M_{1-b}$. We bound the probability of each of these two events occurring.

By the perfect correctness of the underlying scheme $\mathcal{E}$, any of the $K$ ciphertexts $c_i$ that were generated using the standard encryption algorithm Enc (and for which $i \in I$) will necessarily decrypt to $M_b$. Thus, the only way that event (1) can occur is if the subsets $I$ and $J$ (corresponding to sk and $c$) are disjoint. Since $J$ is a random subset of $[4K]$ of size $K$, the probability that the chosen $J$ does not intersect the fixed $I$ is $\binom{3K}{K}/\binom{4K}{K}$.

Consider event (2). Again by the perfect correctness of the underlying encryption scheme, the encryption of $M_{1-b}$ must come from one of the oblivious ciphertext generations. However, this requires that the randomness used to obliviously generate some $c_i$ (with $i \in I$) lies in the set

$$A_i = \left\{(a,b) : \frac{b}{a^{\mathsf{sk}_i}} = M_{1-b}\right\} \subset \mathbb{G} \times \mathbb{G},$$

where $\mathsf{sk}_i$ is the corresponding $i$th decryption key. For any one value of $i \in I$, the probability of landing in $A_i$ when randomly sampling an element from $\mathbb{G} \times \mathbb{G}$ is $\frac{1}{|\mathbb{G}|}$. Thus, taking a union bound over the $K$ elements $i \in I$, the probability of event (2) taking place is $\frac{K}{|\mathbb{G}|}$.

The claim follows by a union bound over the bad events (1) and (2). □

We now show that public keys sampled via ONCGen are frequently lossy. In particular, we show with high probability, the output of the alternative simulator is *statistically* close to honestly generated values.

**Lemma 3.4.5.** *The following holds with probability $1 - \frac{4K}{|\mathbb{G}|}$ over obliviously sampled public keys* $\mathsf{pk} \leftarrow \mathsf{ONCGen}(1^k)$*: for each $b \in \{0,1\}$, the distribution*

$$\{(c, r) : r \leftarrow \{0,1\}^*,\ c = \mathsf{NCEnc}_{\mathsf{pk}}(b; r)\}$$

*is statistically indistinguishable from*

$$\{(c, r_b) : (c, r_0, r_1) \leftarrow \mathsf{NCSim}'(\mathsf{pk})\}.$$

*Proof.* Recall that ONCGen simply runs the oblivious key generation algorithm OGen (of the underlying scheme $\mathcal{E}$) $4K$ times to generate $\mathsf{pk} = (\mathsf{pk}_1, ..., \mathsf{pk}_{4K})$. By Claim 3.4.2, for each $i \in [4K]$, $\mathsf{pk}_i$ is *lossy* unless $\mathsf{pk}_i$ is a DDH tuple, which occurs with probability $\frac{1}{|\mathbb{G}|}$ over $\mathsf{pk}_i \leftarrow \mathsf{OGen}(1^k)$. Taking a union bound, with probability at least $1 - 4K\frac{1}{|\mathbb{G}|}$ over $\mathsf{pk} \leftarrow \mathsf{ONCGen}(1^k)$, *all* $4K$ of the values $\mathsf{pk}_i$ will be lossy.

If all $4K$ of the underlying public keys are lossy, then for each component $i$, and any message $m \in \mathbb{G}$ (in particular, for $m = M_0, M_1$), the distribution of encryptions of $m$ is statistically close to uniform:

$$\{c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}_i}(m)\} \approx_s \{c_i \leftarrow \mathbb{G} \times \mathbb{G}\}.$$

Now, consider a particular value of $b \in \{0,1\}$. The only difference between an honestly generated NCE ciphertext $c = (c_1, ..., c_{4K})$ of $b$ and a ciphertext $c' = (c'_1, ..., c'_{4K})$ generated by the alternative simulator algorithm $\mathsf{NCSim}'$ is that $c'$ has $K$ components $c'_i$ that were actually generated as encryptions of $M_{1-b}$, and then "explained" as being obliviously sampled (i.e., as if the resulting ciphertext $c'_i$ was the randomness used in $\mathsf{OEnc}$). But, when all the keys $\mathsf{pk}_i$ are lossy, the distribution $\{c'_i \leftarrow \mathsf{Enc}_{\mathsf{pk}_i}(M_{1-b})\}$ of these encryptions of $M_{1-b}$ is statistically close to uniform. Thus, since each component is generated independently, the entire ciphertext $c'$ is statistically close to an honestly generated ciphertext $c$ of $b$, and the entire corresponding "explained" randomness $r_b$ is statistically close to uniform, as desired. □

We now describe the additional algorithms – oblivious key generator $\mathsf{ONCGen}$, inverting algorithms, and alternative simulator $\mathsf{NCSim}'$ – that are necessary for an augmented NCE scheme (see Section 3.2.1).

- **Oblivious key generator** $\mathsf{ONCGen}$**:** Run the oblivious key generator for the underlying encryption scheme $\mathsf{OGen}$ $4K$ times to generate $\mathsf{pk}_1, \ldots, \mathsf{pk}_{4K}$. In addition, choose two random messages $M_0, M_1 \leftarrow \mathbb{G}$. Output $e = (\mathsf{pk}_1, \ldots, \mathsf{pk}_{4K}, M_0, M_1)$.
- **Invertible samplability:** Note that $\mathsf{NCGen}$ simply outputs the input randomness as the secret key, and $\mathsf{ONCGen}$ simply outputs the input randomness as the (obliviously sampled) public key. (Indeed, $\mathsf{NCGen}$ and $\mathsf{ONCGen}$ inherit these properties from the corresponding algorithms $\mathsf{Gen}$, $\mathsf{OGen}$ of the underlying lossy encryption scheme). Thus, $\mathsf{NCGen}$ and $\mathsf{ONCGen}$ have trivial inverting algorithms.
- **Alternative simulator** $\mathsf{NCSim}'$**:** This simulator algorithm is essentially the same as $\mathsf{NCSim}$, except that it does not simulate the key generation process.

We argue that these algorithms together yield an augmented NCE scheme with additional lossy properties.

**Lemma 3.4.6.** *The scheme $\mathcal{E}_{\mathsf{NCE}}$ described above is an augmented NCE scheme.*

*Proof.* Recall that the underlying encryption scheme $\mathcal{E}$ is simulatable by Lemma 3.4.1. It thus follows from Theorem 3.2.3 that $\mathcal{E}_{\mathsf{NCE}}$ is an augmented NCE scheme (note that this is precisely the construction discussed in the proof sketch for Theorem 3.2.3). □

### 3.4.3 Randomness Generation Procedure

In this section, we present and analyze a non-interactive procedure for generating private randomness that is used within the oblivious transfer protocol from Section 3.4 in order to achieve security in the semi-malicious model, *even when there is no bound on the amount of information revealed to the adversary.* Intuitively, we would like the procedure to have the following two properties:

**Semi-malicious $P_j$ cannot force "bad" output:** If party $P_j$ is semi-malicious, then he cannot force the output randomness to lie inside a "special" exponentially small subset of the total space $\{0,1\}^t$.

**Simulator can retroactively force any output for honest $P_j$:** On the other hand, if party $P_j$ is honest, then given a trapdoor to the CRS, the simulator can retroactively "explain" any desired outcome within the "special" subset of $\{0,1\}^t$ on behalf of $P_j$.

Recall that private randomness is required in the following steps of the OT protocol:

**Receiver:** The receiver samples two public keys for the NCE scheme: one using the standard $\mathsf{Gen}_{\mathsf{NCE}}$ algorithm, and one sampled obliviously.

**Sender:** The sender generates an encryption of each of his two message bits, with respect to the public keys sent by the receiver.

Depending on whether the receiver and/or the sender is corrupted, we must ensure the following properties hold in order to guarantee security of the OT.

- **Honest Receiver, Honest Sender.** In this case, both parties sample randomness honestly, and thus security will automatically follow from the semi-honest security of the OT protocol.
- **Honest Receiver, Corrupted Sender.** We must guarantee that a corrupted sender cannot generate malformed ciphertexts, which may cause the honest receiver to abort depending on his secret input bit. Recall that an NCE ciphertext encrypting message $m \in \{0,1\}$ is generated by choosing a set $J \subset [4k]$ of size $k$, generating $k$ ciphertexts of the special message $M_m$ in the underlying encryption scheme (one for each position in $J$), and obliviously sampling $3k$ additional ciphertexts in the underlying scheme. Such an NCE ciphertext $(c_1, \ldots, c_{4k})$ decrypts correctly with respect to secret key $sk = (I, \{u_i\}_{i \in I})$ as long as the two following properties hold:
  - $I \cap J \neq \emptyset$. (So that at least one of the $c_i$ encrypting $M_m$ is contained within the set $I$ of positions whose secret keys are known).
  - For every *obliviously generated* ciphertext $c_i = (x_i, y_i)$, $i \notin J$, we have $\frac{y_i}{x_i^u} \neq M_{1-m}$. That is, the obliviously generated ciphertexts did not "accidentally" encrypt the second special message $M_{1-m}$ corresponding to an overall encryption of $(1-m)$.
- **Corrupted Receiver, Honest Sender.** To ensure that the corrupted receiver does not learn information about an honest sender's second message, it suffices to guarantee that all the underlying public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_{4k}$ that the receiver obliviously samples (when obliviously sampling an NCE public key) are *lossy*. By Lemma 3.4.5, this is the case as long as each of the obliviously sampled $\mathsf{pk}_i = (g_1, g_2, g_3, g_4)$ is *not* a DDH tuple.
- **Corrupted Receiver, Corrupted Sender.** Here, it does not matter if the parties generate a malformed ciphertext (decrypting to $\perp$), since this will cause an abort in the real-world execution, and can be identified and simulated appropriately. However, we must guarantee

that the parties cannot choose randomness that yields a *wrong* decryption: i.e., they cannot choose randomness for sampling an NCE (pk, sk) key pair and for encrypting the correct message $m$ that decrypts under sk to the wrong message $1-m$. Indeed, in this case the simulator will abort, but the honest parties in the real world will not.

To guarantee this does not happen, it suffices to show that the adversary *cannot* choose any secret key $u_i$ in the underlying encryption scheme together with an obliviously generated ciphertext $(x_i, y_i)$ such that $\frac{y_i}{x_i^{u_i}} = M_{1-m}$. (Recall that by the perfect completeness of the underlying encryption scheme, all ciphertexts that were not obliviously sampled will necessarily decrypt to the correct value $M_m$).

Combining all of these cases, it suffices to provide ways to generate randomness for the following:

1. Obliviously sampling an underlying public key $(g_1, g_2, g_3, g_4) \in \mathbb{G}^4$ (performed by the receiver). A corrupted receiver must be restricted to generate only *non-DDH* tuples (yielding lossy public keys), whereas the simulator should be able to choose DDH tuples on behalf of honest parties (allowing him to "obliviously sample" public keys for which he can decrypt).

2. Sampling a random set $J \subset [4k]$ of size $k$ (used as part of the encryption randomness by the sender). A corrupted sender must be restricted so that with overwhelming probability over (honestly) chosen $I \subset [4k]$ with $|I| = k$, it holds that $I \cap J \neq \emptyset$, even if he knows $I$ completely. However, on behalf of an honest sender, the simulator should be able to retroactively "explain" arbitrary $J$ values (which will be required in order to generate simulated ciphertexts that open to both 0 and 1).

3. Sampling an underlying secret key $u \in \mathbb{Z}_p$, and obliviously sampling an underlying ciphertext $(x, y) \in \mathbb{G}^2$, so that $\frac{y}{x^u} \neq M_{1-m}$, where the message $M_{1-m}$ is randomly chosen and fixed as part of the CRS. (This case is relevant when both sender and receiver are controlled by the adversary). However, the simulator should be able to retroactively explain arbitrary values for $u$ and $(x, y)$ (and, in particular, will need to choose values for which $\frac{y}{x^u} = M_{1-m}$).

All remaining randomness used in the OT protocol can be selected arbitrarily (for example, the selection of $I \subset [4k]$ in the generation of the NCE public keys). We now construct and analyze randomness generation procedures for each of the three above applications.

## Forcing Obliviously Sampled Public Keys to be Lossy

We wish to ensure that semi-malicious parties *cannot* generate randomness that falls within the set of DDH tuples, while still ensuring that in the simulation, the simulator can retroactively explain any arbitrary DDH tuple output (with known discrete logs) on behalf of honest parties. This can be achieved as follows.

We append an additional 4-tuple of random elements $\vec{g} = (g_1, g_2, g_3, g_4) \in \mathbb{G}^4$ to the CRS of each party. Each time a party $P_j$ must generate randomness for a Case 1 application, he does so by rerandomizing his tuple $\vec{g}_j$: namely, he executes the following procedure DDH-Rerand.

Note that if the original tuple $(g_1, g_2, g_3, g_4)$ is a DDH tuple, then the output will be a random DDH tuple; however, if the original tuple is *not* a DDH tuple, then for *no value* of $\alpha, \beta, \gamma \neq 0$ will

DDH-Rerand$(g_1, g_2, g_3, g_4)$:

1. Sample random exponents $\alpha, \beta, \gamma \leftarrow \mathbb{Z}_p^*$ (where $|\mathbb{G}| = p$ prime).
2. Output the tuple $(g_1^\alpha, g_2^\beta, g_3^{\alpha\cdot\gamma}, g_4^{\beta\cdot\gamma})$ as the desired randomness.

the resulting tuple become DDH. We will take advantage of this during the simulation, by replacing the CRS of each honest party with a randomly selected DDH tuple, and generating the CRS of each corrupted party to be a *non*-DDH tuple. We now formalize this intuition.

*Simulator can retroactively force any DDH output for honest $P_j$.* Define $\mathbb{G}_{DDH}^4 := \{(g, g^a, g^b, g^{ab}) : g \in G \setminus \{1\},\ a, b \neq 0\}$ In the simulation, the CRS of honest parties will be generated as a random DDH tuple $(g, g^a, g^b, g^{ab}) \in \mathbb{G}_{DDH}^4$, and the simulator will know the corresponding discrete logs $a, b, ab$. The following claim tells us that the simulator can retroactively "explain" *any* desired DDH output tuple as being generated via the rerandomization procedure from this original tuple $(g, g^a, g^b, g^{ab})$, as long as the discrete logs of the target tuple with respect to $g$ are known. Note that this procedure will be used within the MPC construction for obliviously generating public keys for the non-committing encryption scheme. For the case of honest parties, the simulator will sample a *standard* public key (whose discrete logs are known), and will force this public key tuple to be his alleged rerandomized output.

**Claim 3.4.7.** *Let $(g, g^a, g^b, g^{ab}) \in \mathbb{G}_{DDH}^4$ be an arbitrary DDH tuple. Then for any target tuple $(g^x, g^y, g^z, g^w) \in \mathbb{G}_{DDH}^4$, there exists $\alpha, \beta, \gamma \in \mathbb{Z}_p^*$ such that $g^x = (g)^\alpha$, $g^y = (g^a)^\beta$, $g^z = (g^b)^{\alpha\gamma}$, and $g^w = (g^{ab})^{\beta\gamma}$.*

*Proof.* Take $\alpha = x$, $\beta = (a^{-1}y) \mod p$, and $\gamma = (b^{-1}x^{-1}z) \mod p$. First, note that the inverses $a^{-1}, b^{-1}, x^{-1}$ each exist modulo $p$ since $a, b, x \neq 0 \mod p$, and the given expressions for $\alpha, \beta, \gamma$ can be computed efficiently, since we are given all exponent information $a, b, x, y, z, w$. Further, every DDH tuple contained in $\mathbb{G}_{DDH}^4$ can be expressed in the form $(g^x, g^y, g^z, g^w)$, since $g \neq 1$ is necessarily a generator of $\mathbb{G}$. We see that the first three required conditions hold by construction. Finally, for these values of $\alpha, \beta, \gamma$, it also holds that $(g^b)^{\alpha\gamma} = (g^b)^{x\cdot b^{-1}x^{-1}z \mod p} = g^z$, as desired. □

*Semi-malicious $P_j$ cannot force "bad" (DDH) output.* In the simulation, the CRS of each corrupted party will be sampled as a random *non*-DDH tuple. To ensure that corrupted parties cannot learn information about messages encrypted under public keys they obliviously sample, we guarantee their rerandomization procedure *cannot* yield a DDH tuple output as an obliviously sampled public key, even if the rerandomization values $\alpha, \beta, \gamma$ are chosen maliciously:

**Claim 3.4.8.** *Let $(g_1, g_2, g_3, g_4) \in \mathbb{G}^4$ be an arbitrary non-DDH tuple. Then for* every *possible choice of $\alpha, \beta, \gamma \in \mathbb{Z}_p^*$, the tuple $(g_1^\alpha, g_2^\beta, g_3^{\alpha\cdot\gamma}, g_4^{\beta\cdot\gamma})$ is also* not *a DDH tuple.*

*Proof.* Let $g$ be an arbitrary generator of $\mathbb{G}$, and define $a, b, c, d$ to be such that $g_1 = g^a$, $g_2 = g^b$, $g_3 = g^c$, $g_4 = g^d$. Since we began with a non-DDH tuple, it holds that $ad \neq bc$. Thus, for any collection of invertible elements $\alpha, \beta, \gamma \in \mathbb{Z}_p^*$, it follows that $ad \cdot \alpha\beta\gamma \neq bc \cdot \alpha\beta\gamma$: that is, $(g_1^\alpha, g_2^\beta, g_3^{\alpha\gamma}, g_4^{\beta\gamma}) = (g^{a\alpha}, g^{b\beta}, g^{c\alpha\gamma}, g^{d\beta\gamma})$ is *not* a DDH tuple. □

**Forcing Subset $J \subset [4k]$ to Intersect an *Honestly* Chosen $I \subset [4k]$**

Recall that a corrupted sender must be prevented from generating malformed ciphertexts (which may cause an honest receiver to abort conditionally on his secret selector bit). In particular, when generating a ciphertext, the corrupted sender must not be able to choose the subset $J \subset [4k]$ to be disjoint from the randomly chosen subset $I \subset [4k]$ given by the honest receiver's secret key, otherwise the receiver will only be able to decrypt obliviously sampled ciphertexts, which decrypt to garbage (and neither special message $M_m$ or $M_{1-m}$).

To achieve this guarantee, our second randomness generation procedure makes crucial use of a lossy trapdoor function (LTDF) family $(G_{\mathsf{Loss}}, G_{\mathsf{Inj}}, S, F, F^{-1})$ with special properties. This special LTDF family, referred to as a *bijective, $(D,\alpha)$-admissible* LTDF family (with potentially seed-dependent domains) has the property that injective branches are each *bijective*, the output space has size approximately $D$, and the lossy branches are *very lossy* (to avoid a "bad" set of fractional size $\alpha$). We refer the reader to Section 3.2.4 and Definition 3.2.16 for a formal discussion of the required conditions.

We now define the second randomness generation procedure. Let $(G_{\mathsf{Loss}}, G_{\mathsf{Inj}}, S, F, F^{-1})$ be a bijective, $(D,\alpha)$-admissible LTDF family for $D = \binom{4k}{k}$ and $\alpha = \binom{3k}{k}/\binom{4k}{k}$, as in Theorem 3.2.17(1). Suppose each party $P_j$ is associated with a corresponding LTDF seed $\sigma_j$ that is known to all parties (i.e., contained in the CRS). At each point when $P_j$ must generate randomness for sampling the set $J \subset [4k]$, he executes the procedure LTDF-Sample-$J$ given below.

LTDF-Sample-$J(\sigma_j)$:

1. Sample a random value $r \leftarrow S(\sigma_j)$ from the domain $D_{\sigma_j}$ of the LTDF function corresponding to seed $\sigma_j$.
2. Output the evaluation $F(\sigma_j, r)$ as the desired randomness.

Loosely speaking, the simulation works as follows. For each honest party, the simulator will sample an *injective* seed $\sigma_j$, together with an inversion trapdoor. This will allow the simulator to efficiently explain any desired randomness output value. In contrast, each corrupted party will be given a *lossy* seed $\sigma_j$, whose corresponding function image will be sufficiently small that with high probability it will not hit the small "special" random set.

We now elaborate on these two arguments.

*Simulator can retroactively force any output for honest $P_j$.* By the definition of a bijective admissible LTDF family (Definition 3.2.16, Properties 1 & 2), we know that every injective function $F(\sigma, \cdot)$ for $\sigma \leftarrow G_{\mathsf{Inj}}(1^k)$ is *bijective* onto the corresponding output space, and the *size* of this output space is at least $\binom{4k}{k}$. Thus, given a trapdoor $\tau_j$ for an injective branch $F(\sigma_j, \cdot)$, the simulator can "force" any desired output $r \in [\binom{4k}{k}]$, by simply inverting $r' = F^{-1}(\sigma_j, \tau_j, r)$, and taking $r'$ as the randomness sampled to generate $r$. Note that since $F(\sigma_j, \cdot)$ is a bijection, a random choice of output $r$ corresponds to a random choice of input $r'$, and will thus yield the correct distribution.

*Semi-malicious $P_j$ cannot force "bad" output.* We now prove that with overwhelming probability over the honest receiver's choice of $I \subset [4k]$, a semi-malicious sender $P_j$ in the OT protocol will be unable to choose a set $J \subset [4k]$ as the output of LTDF-Sample-$J(\sigma_j)$ for which $I \cap J = \emptyset$, since such a $J$ simply does not *exist* within the range of his (lossy) LTDF function $F(\sigma_j, \cdot)$.

**Claim 3.4.9** (Choosing the set $J$). *Let $(G_{\mathsf{Loss}}, G_{\mathsf{Inj}}, S, F, F^{-1})$ be a bijective, $(D, \alpha)$-admissible LTDF family for $D = \binom{4k}{k}$ and $\alpha = \binom{3k}{k}/\binom{4k}{k}$, and let $\sigma \leftarrow G_{\mathsf{Loss}}(1^k)$ be an arbitrary* lossy *LTDF seed. Associate the first $\binom{4k}{k}$ elements in the output space of the corresponding function $F(\sigma, \cdot)$ with the collection of sets $A \subset [4k]$ of size $k$ (and denote this collection as $\mathbf{A}_k$). Then there exists a negligible function $\mu(k)$ for which*

$$\Pr_{I \leftarrow \mathbf{A}_k}[\exists J \in \mathbf{A}_k : (I \cap J = \emptyset) \wedge (J = F(\sigma, x) \text{ for some } x \in D_\sigma)] < \mu(k).$$

*Proof.* Consider any fixed value $A \in \mathbf{A}_k$ that lies in the image of the lossy branch, $F(\sigma, D_\sigma)$. For this $A$, the probability that a randomly selected set $I \leftarrow \mathbf{A}_k$ does *not* intersect $A$ is exactly $\binom{3k}{k}/\binom{4k}{k}$. Thus, by a union bound, the probability that there exists *any* $A \in F(\sigma, D_\sigma)$ for which $A \cap I = \emptyset$ is bounded by $|F(\sigma, D_\sigma)| \cdot \binom{3k}{k}/\binom{4k}{k}$. Finally, by the definition of a bijective admissible LTDF family (Definition 3.2.16, Property 3), it holds that this probability is negligible in $k$. □

## Preventing Ciphertexts that Decrypt to *Wrong* Messages

As the final case, we must ensure that a corrupted sender-receiver pair cannot coordinate to produce a (secret key, encryption of $m$) pair that decrypts to the *wrong* message $1 - m$. By the perfect correctness of the underlying encryption scheme, it suffices to show that none of the underlying obliviously sampled ciphertexts "accidentally" decrypt to the wrong special message $M_{1-m}$. Namely, the adversary will choose a secret key $u \in \mathbb{Z}_p$ and will obliviously sample a ciphertext $(x, y) \in \mathbb{G}^2$, and we must guarantee that, with high probability over the randomly chosen message $M_{1-m}$ fixed in the CRS, the adversary will not be able to choose his values $u, (x, y)$ such that $\frac{y}{x^u} = M_{1-m}$.

To achieve this, the secret key $u \in \mathbb{Z}_p$ and each component of the the obliviously generated ciphertext $(x, y) \in \mathbb{G}^2$ will be sampled using the same LTDF-based randomness generation procedure as in the previous subsection, but with a different choice of parameters $(D, \alpha)$. Here, the target output space size $D$ will be set to $p = |\mathbb{G}| = 2^k$. We will choose the $\alpha$ parameter so that the *total* collection of possible values of $\frac{y}{x^u}$, over all possible combinations of $u, x, y$ chosen from the range of the the sender and receiver's (lossy) LTDF functions, form a negligible fraction of the entire space of possible values (i.e., the full message space of the underlying encryption scheme). Thus, when the "special" messages $M_0, M_1$ are chosen randomly as part of the initial CRS, the probability that they will fall into this small set of "bad" attainable messages is negligible. As proved below (in Claim 3.4.10), this holds when $\alpha$ is set to $2^{-k/3}$.

We now define the third randomness generation procedure. Let $(G_{\mathsf{Loss}}, G_{\mathsf{Inj}}, S, F, F^{-1})$ be a bijective, $(D, \alpha)$-admissible LTDF family for $D = 2^k$ and $\alpha = 2^{-k/3}$, as given in Theorem 3.2.17(2). Suppose each party $P_j$ is associated with a LTDF seed $\sigma_j$ contained in the CRS and known to all parties. Party $P_j$ executes the following procedure LTDF-Sample-$\mathbb{G}$ each time he must generate

LTDF-Sample-$\mathbb{G}(\sigma_j)$:

1. Sample a random value $r \leftarrow S(\sigma_j)$ from the domain $D_{\sigma_j}$ of the LTDF function corresponding to seed $\sigma_j$.
2. Output the evaluation $F(\sigma_j, r)$ as the desired randomness.

randomness for (1) sampling a secret key $u$ (when acting as receiver in the OT), or (2) selecting each of the two components in an obliviously sampled ciphertext $(x, y)$.

The simulation approach will be identical to that in the previous subsection (for choosing the set $J$). For each honest party, the simulator will sample an injective seed $\sigma_j$, together with an inversion trapdoor. For each corrupted party, the CRS will be a lossy seed $\sigma_j$, whose corresponding function image will be sufficiently small that it will not hit the small "special" random subset of possible randomness values. For completeness, we again elaborate on these two points.

*Simulator can retroactively force any output for honest $P_j$.* By the definition of a bijective admissible LTDF family (Definition 3.2.16, Properties 1 & 2), every injective function $F(\sigma, \cdot)$ for $\sigma \leftarrow G_{\mathsf{Inj}}(1^k)$ is *bijective* onto the corresponding output space, and the *size* of this output space is at least $|\mathbb{G}| = 2^k$. Thus, given a trapdoor $\tau_j$ for an injective branch $F(\sigma_j, \cdot)$, the simulator can "force" any desired output $r \in [2^k]$, by simply inverting $r' = F^{-1}(\sigma_j, \tau_j, r)$, and taking $r'$ as the randomness sampled to generate $r$. Note that since $F(\sigma_j, \cdot)$ is a bijection, a random choice of output $r$ corresponds to a random choice of input $r'$, yielding the correct distribution.

*Semi-malicious $P_j$ cannot force "bad" output.* We now prove that with overwhelming probability over the choice of each "special" message $M_0$ and $M_1$ in the CRS, a corrupted sender-receiver pair *cannot* generate a secret key $u \in \mathbb{Z}_p$ and obliviously sampled ciphertext $(x, y) \in \mathbb{G} \times \mathbb{G}$ for which $\frac{y}{x^u} = M_0$ (or $M_1$), simply because such a triple $u, x, y$ does not *exist* within the ranges of their (lossy) LTDF functions $F(\sigma_R, \cdot)$ (for $u$) and $F(\sigma_S, \cdot)$ (for $x, y$).

**Claim 3.4.10** (Preventing *incorrect* decryptions). *Let $(G_{\mathsf{Loss}}, G_{\mathsf{Inj}}, S, F, F^{-1})$ be a bijective, $(D, \alpha)$-admissible LTDF family for $D = 2^k$ and $\alpha = 2^{-k/3}$, and let $\sigma_R, \sigma_S \leftarrow G_{\mathsf{Loss}}(1^k)$ be arbitrary* lossy *seeds. Then there exists a negligible function $\mu(k)$ for which*

$$\Pr_{M \leftarrow \mathbb{G}}\left[\exists\ u \in F(\sigma_R, D_{\sigma_R}), \exists\ a, b \in F(\sigma_S, D_{\sigma_S}) : \left(\frac{b}{a^u} = M\right)\right] < \mu(k).$$

*Proof.* For any fixed lossy seeds $\sigma_R, \sigma_S$, the fraction of values in $\mathbb{G}$ that are attainable via $\frac{b}{a^u}$ is

$$\frac{1}{|\mathbb{G}|} \cdot \left|\left\{\frac{b}{a^u} : u \in F(\sigma_R, D_{\sigma_R}), a, b \in F(\sigma_S, D_{\sigma_S})\right\}\right| \leq \frac{1}{2^k} \cdot |F(\sigma_R, D_{\sigma_R})| \cdot |F(\sigma_S, D_{\sigma_S})|^2.$$

But, by the $(2^k, 2^{-k/3})$-admissibility of the LTDF family (see Definition 3.2.16, Property 3), we know that $2^{-k/3}|F(\sigma, D_\sigma)| < \mu(k)$ for some negligible function $k$, and for every lossy seed $\sigma \leftarrow G_{\mathsf{Loss}}(1^k)$.

Therefore, it holds that

$$\frac{1}{2^k} \cdot |F(\sigma_R, D_{\sigma_R})| \cdot |F(\sigma_S, D_{\sigma_S})|^2 < (\mu(k))^3,$$

which is also negligible in $k$. Therefore, for a randomly chosen $M \leftarrow \mathbb{G}$, the probability that $M$ will fall in the set of attainable values $\frac{b}{a^u}$ is negligible, as desired. □

## Combining the Pieces

We now explicitly define the procedures for generating randomness for each relevant application in the OT protocol. These are given in Figure 3.1.

$\mathsf{Rand\text{-}KeyGen}(\sigma^{(2)}, (g_1, g_2, g_3, g_4))$: Sample $I \leftarrow \mathbf{A}_k$. For each $i \in I$, sample randomness to be used for $\mathsf{Gen}$, by executing $u_i \leftarrow \mathsf{LTDF\text{-}Sample\text{-}}\mathbb{G}(\sigma^{(2)})$. For each $j \in [4k]\backslash I$, sample randomness to be used for *obliviously* sampling a public key, by executing $r_j \leftarrow \mathsf{DDH\text{-}Rerand}(g_1, g_2, g_3, g_4)$.
Output $(I, \{u_i\}_{i \in I}, \{r_j\}_{j \in [4k]\backslash I})$.

$\mathsf{Rand\text{-}OblivKeyGen}(g_1, g_2, g_3, g_4)$: For each $i \in [4k]$, sample randomness to be used for obliviously sampling a public key, by executing $r_i \leftarrow \mathsf{DDH\text{-}Rerand}(g_1, g_2, g_3, g_4)$.
Output $\{r_i\}_{i \in [4k]}$.

$\mathsf{Rand\text{-}Enc}(\sigma^{(1)}, \sigma^{(2)})$: Sample randomness for choosing $J \in \mathbf{A}_k$, by executing $J \leftarrow \mathsf{LTDF\text{-}Sample\text{-}}J(\sigma^{(1)})$. For each $j \in J$, sample encryption randomness $\mathsf{rand}_j \leftarrow \{0,1\}^*$. For each $i \notin J$, sample randomness for obliviously sampling a ciphertext, by running two executions $x_i, y_i \leftarrow \mathsf{LTDF\text{-}Sample\text{-}}\mathbb{G}(\sigma^{(2)})$.
Output $(J, \{\mathsf{rand}_j\}_{j \in J}, \{(x_i, y_i)\}_{i \in [4k]\backslash J})$.

**Figure 3.1:** The procedures for generating randomness for key generation, oblivious key generation, and encryption, to be used in the OT protocol.

Combining the earlier claims, the following lemmas hold:

**Lemma 3.4.11** (Corrupt sender, honest receiver). *Let $\sigma^{(1)}$ and $\sigma^{(2)}$ be arbitrary lossy LTDF seeds. Then with overwhelming probability over $M_0, M_1 \leftarrow \mathbb{G}$ and an honest execution of* $\mathsf{rnd}_G \leftarrow \mathsf{Rand\text{-}KeyGen}$, *it holds for* every *possible output* $\mathsf{rnd}_E$ *of* $\mathsf{Rand\text{-}Enc}(\sigma^{(1)}, \sigma^{(2)})$ *(potentially using malicious randomness) that* $\mathsf{Dec}_{\mathsf{pk}}(c) = m$, *where* $(\mathsf{pk}, \mathsf{sk}) = \mathsf{NCGen}(1^k; \mathsf{rnd}_G)$ *and* $c = \mathsf{NCEnc}_{\mathsf{pk}}(m; \mathsf{rnd}_E)$.

*Proof.* Follows from Claims 3.4.9 and 3.4.10. □

**Lemma 3.4.12** (Honest sender, corrupt receiver). *Let $\vec{g} = (g_1, g_2, g_3, g_4)$ be a* non-*DDH tuple. Then* any *obliviously sampled public key* $\mathsf{pk} = \mathsf{ONCGen}(\mathsf{rand}_O)$ *for which* $\mathsf{rnd}_O$ *is generated via* $\mathsf{rnd}_O \leftarrow \mathsf{Rand\text{-}OblivKeyGen}(\vec{g})$ *is necessarily* lossy.

*Proof.* Follows from Claims 3.4.7 and 3.4.4. □

**Lemma 3.4.13** (Corrupt sender and receiver). *Let $\sigma^{(1)}$ and $\sigma^{(2)}$ be arbitrary lossy LTDF seeds. Then with overwhelming probability over $M_0, M_1 \leftarrow \mathbb{G}$, it holds for* every *possible pair of outputs* $(\mathsf{pk}, \mathsf{sk}) = \mathsf{Gen}(1^k; \mathsf{rnd}_G)$ *and* $c = \mathsf{NCEnc}_{\mathsf{pk}}(m; \mathsf{rnd}_E)$ *for which* $\mathsf{rnd}_G \leftarrow \mathsf{Rand\text{-}KeyGen}(\sigma^{(2)}, \vec{g})$ *and* $\mathsf{rnd}_E \leftarrow \mathsf{Rand\text{-}Enc}(\sigma^{(1)}, \sigma^{(2)})$, *that* $\mathsf{Dec}_{\mathsf{sk}}(c) \in \{m, \perp\}$

*Proof.* Follows from Claim 3.4.10. □

### 3.4.4 Our OT Protocol

In what follows, we use the augmented NCE scheme from subection 3.4.2, together with the randomness generation procedures from the previous subsection to construct a 1-out-of-4 OT protocol that is secure against adaptive auxiliary information in the semi-malicious setting. This OT protocol will play a major role in achieving MPC that is secure in the same model, constructed in Section 3.5.

**Our 1-out-of-4 OT protocol for semi-malicious parties.**
CRS: Uniformly random message $M_0, M_1 \leftarrow \mathbb{G}$. For each party $P_i$: (injective) LTDF seeds $\sigma_i^{(1)}, \sigma_i^{(2)}$, and (non-DDH) tuple $\vec{g}^i = (g_1^i, g_2^i, g_3^i, g_4^i)$.

1. The receiver $P_i$, on input $b \in [4]$, does the following:
   1. For $b \in [4]$, sample a standard NCE key pair $(e_b, d_b)$ via $\mathsf{Rand\text{-}KeyGen}(\sigma_i^{(2)}, \vec{g}^i)$. For each $b' \in [4] \setminus \{b\}$, obliviously sample an NCE public key $e_{b'}$ via $\mathsf{Rand\text{-}OblivKeyGen}(\vec{g}^i)$.
   2. The receiver sends $(e_1, \ldots, e_4)$ to the sender.
3. The sender $P_j$, on input $m_1, \ldots, m_4$, where each $m_i \in \mathbb{G}$, and upon receiving the message $(e_1, \ldots, e_4)$ from the receiver, does the following:
   1. For each $b' \in [4]$, encrypt the message $m_{b'}$ under public key $e_{b'}$. This is done by executing $c_{b'} \leftarrow \mathsf{Rand\text{-}Enc}(m_{b'}, e_{b'}, \sigma_j^{(1)}, \sigma_j^{(2)})$.
   2. Send $(c_1, \ldots, c_4)$.
3. The receiver decrypts $c_b$ using the secret key $d_b$

We do not prove security of the above OT protocol here. Instead, the security properties of the above OT protocol will be proven directly (as necessary) in the security proof of our MPC protocol.

## 3.5 Our MPC Protocol in the Semi-Malicious Model

In this section, we construct an MPC protocol $\Pi$ that is secure against adaptive auxiliary information the semi-malicious model (as in Definition 3.3.3).

Let $P_1, \ldots, P_n$ be $n$ parties that wish to securely compute a function $f : \{0,1\}^n \to \{0,1\}$. We assume for simplicity of notation that each party $P_i$ takes as input a single bit $x_i \in \{0,1\}$, and the function $f$ outputs a single bit $f(x_1, \ldots, x_n)$. We assume for concreteness that party $P_n$ receives the output. All our results (and proofs) extend readily to the case where the inputs and/or outputs are strings, and where each party $P_i$ may receive a different output $f_i(x_1, ..., x_n)$.

**Theorem 3.5.1.** *For every efficiently computable function $f$, there exists a protocol $\Pi$ in the CRS model for computing $f$, that is secure against adaptive auxiliary information (as per Definition 3.3.3).*

The rest of this section is devoted to the proof of Theorem 3.5.1.

We start by presenting the protocol $\Pi$ in Section 3.5.1. Then, we prove that $\Pi$ is secure. Namely, we fix any PPT adversary $\mathcal{A}$ that corrupts a set of parties $M \subseteq \{P_1, \dots, P_n\}$ and may request adaptive auxiliary information on the secret states of honest parties. In Section 3.5.2, we construct a simulator $\mathcal{S}$ that corrupts the same set of parties, and simulates the real world execution of $\mathcal{A}$. Finally, in Section 3.5.3 we prove that, indeed, the real world is indistinguishable from the simulated (ideal) world.

### 3.5.1 Protocol for Semi-Malicious Adversaries

Our protocol $\Pi$ is essentially the honest-but-curious GMW protocol, instantiated with the semi-malicious OT protocol from Section 3.4, with the following modifications:

1. The parties do not generate all their randomness in the onset of the protocol, but rather they generate their randomness "online," whenever needed. This will be important to achieve security against adaptive auxiliary information.
2. In the output stage, rather than simply sending all the shares of each output wire to the party receiving that output, the parties will first "re-randomize" their shares.

Recall that our OT protocol is in the CRS model.

**Our Protocol.** Let $\{\mathsf{crs}_1, \dots, \mathsf{crs}_n\}$ be the common random string, where $\mathsf{crs}_i$ is used by party $P_i$ in the OT protocol, as described in Section 3.4.

1. **Input-Sharing Stage:** Party $P_i$ shares its input $x_i$ with all parties, by choosing $n$ random bits $x_i^1, \dots, x_i^n \leftarrow \{0,1\}$ such that $\bigoplus_{j=1}^n x_i^j = x_i$. Then, $P_i$ sends $x_i^j$ to each party $P_j$, and stores $x_i^i$.
2. **Circuit Evaluation:** The parties now proceed to evaluate the circuit $C$ in a gate-by-gate manner (as in GMW). Let $\alpha$ and $\beta$ be the bit-values corresponding to the input wires of a given gate. Then, each party $P_i$ holds bits $\alpha_i$, $\beta_i$ such that $\sum_{i=1}^n \alpha_i = \alpha$ and $\sum_{i=1}^n \beta_i = \beta$. The gate is evaluated in the following manner (depending upon whether it is an addition gate or a multiplication gate):
   - *Addition Gate*: In this case, each party $P_i$ simply sets its share of the bit value corresponding to the output wire of the gate to be $\gamma_i = \alpha_i + \beta_i$. Thus, we have that $\sum_{i=1}^n \gamma_i = \sum_{i=1}^n (\alpha_i + \beta_i) = \alpha + \beta = \gamma$, as required.
   - *Multiplication Gate*: In this case, each party needs to compute its own share of $\gamma = (\sum_{i=1}^n \alpha_i)(\sum_{i=1}^n \beta_i)$. The main idea behind the computation for multiplication gate is

the following equation (we refer the reader to Section 3.2.2 in [Gol98] for a justification of this equation):

$$\left(\sum_{i=1}^{n}\alpha_i\right)\cdot\left(\sum_{i=1}^{n}\beta_i\right) = n\cdot\sum_{i=1}^{n}\alpha_i\cdot\beta_i + \sum_{1\le i<j\le n}(\alpha_i+\alpha_j)(\beta_i+\beta_j)\ .$$

Let $\delta$ and $\eta$ denote the first and second sum respectively on the right side in the above equation (thus, we have that $\gamma = \delta+\eta$). Note that each party $P_i$ can compute a share $\delta_i$ of the first sum locally by simply computing $n\cdot\alpha_i\cdot\beta_i$. To compute the shares of the second sum, the parties engage in multiple executions of the 1-out-of-4 OT protocol described in Section 3.4. More specifically, for each pair $i$ and $j$ (such that $1\le i<j\le n$), parties $P_i$ and $P_j$ engage in an execution of the 1-out-of-4 OT protocol in the following manner. Let $\eta_{i,j}+\eta_{j,i} = (\alpha_i+\alpha_j)(\beta_i+\beta_j)$. Party $P_i$ chooses a random bit $\eta_{i,j}$, and prepares the following table:

| Value of $(\alpha_j,\beta_j)$ | Receiver input $b$ | Receiver output $\eta_{j,i}$ |
|---|---|---|
| $(0,0)$ | 1 | $o_1 = \eta_{i,j}+(\alpha_i+0)(\beta_i+0)$ |
| $(0,1)$ | 2 | $o_2 = \eta_{i,j}+(\alpha_i+0)(\beta_i+1)$ |
| $(1,0)$ | 3 | $o_3 = \eta_{i,j}+(\alpha_i+1)(\beta_i+0)$ |
| $(1,1)$ | 4 | $o_4 = \eta_{i,j}+(\alpha_i+1)(\beta_i+1)$ |

$P_i$ and $P_j$ now engage in an execution of the 1-out-of-4 OT protocol, where the input of $P_i$ are the values $o_1,o_2,o_3,o_4$ and the input of $P_j$ is value $b = 2^1\cdot\alpha_j+2^0\cdot\beta_j+1\in[4]$. Upon receiving the output $o_b$, party $P_j$ sets $\eta_{j,i} = o_b$.
After completion of all the OT executions, each party $P_i$ first computes $\eta_i = \bigoplus_{j\ne i}\eta_{i,j}$ and then computes $\gamma_i = \delta_i\oplus\eta_i$ as its share of the output wire of the gate.

**3. Output Stage:** Each party $P_i$ holds a share of the output wire of the circuit $C$. Denote by $\alpha_i$ the share of party $P_i$. The parties do the following:

1. Each party $P_i$ generates $n-1$ random key pairs $(e_{\ell,i},d_{\ell,i})\leftarrow \mathsf{NCGen}(1^k)$, one for each $\ell\in[n]\setminus\{i\}$, and sends $e_{\ell,i}$ to party $P_\ell$. Note that the encryption key $e_{\ell,i}$ will be used for messages sent from party $P_\ell$ to $P_i$.
2. Each party $P_i$ chooses at random $r_{i,1},\dots,r_{i,n}$ such that $\bigoplus_{\ell=1}^{n} r_{i,\ell} = \alpha_i$.
3. Each party $P_i$ sends the ciphertext $c_{i,\ell}\leftarrow \mathsf{NCEnc}_{e_{i,\ell}}(r_{i,\ell})$ to each party $P_\ell$.
4. Each party $P_i$ decrypts the ciphertexts he received from each party $P_\ell$ by computing $r_{\ell,i} = \mathsf{NCDec}_{d_{\ell,i}}(c_{\ell,i})$ for every $\ell\ne i$.
5. Each party $P_i$ sends the share $r'_i \triangleq \bigoplus_{\ell=1}^{n} r_{\ell,i}$ to the party $P_n$ receiving the output.

The party $P_n$ xor's all the shares $r'_i$ he receives, and sets the output to be $y = \bigoplus_{i=1}^{n} r'_i$.

This concludes the description of our protocol $\Pi$.

In order to prove that the protocol $\Pi$ is secure against adaptive auxiliary information, we need to prove that for every semi-malicious *real world* adversary $\mathcal{A}$ corrupting a set of parties

$M \subseteq \{P_1, \ldots, P_n\}$ there exists a *ideal world* simulator $\mathcal{S}$ corrupting the same set of parties $M$, such that for any auxiliary input $z \in \{0,1\}^*$, any input vector $\vec{x}$, it holds that:

$$\left\{\mathsf{IDEAL}^{f}_{\mathcal{S},M}(1^k, \vec{x}, z)\right\}_{k \in N} \approx_c \left\{\mathsf{REAL}^{\Pi}_{\mathcal{A},M}(1^k, \vec{x}, z)\right\}_{k \in N}.$$

To this end, fix any semi-malicious real world adversary $\mathcal{A}$ corrupting a set of parties $M$. In what follows, we construct a corresponding simulator $\mathcal{S}$.

### 3.5.2 Description of the Simulator

**Notation.** We let $G$ denote the number of gates in the circuit being computed, and we denote these gates by $\{1, \ldots, G\}$, where the first $n$ gates $\{1, \ldots, n\}$ correspond to the $n$ input gates. For every gate $g \in [G]$, we let $v_g$ denote the (output) value of the gate when computed on the inputs $(x_1, \ldots, x_n)$. (Note that the value of the $j$th input gate for a corrupted party $j \in M$ is the input value selected by $\mathcal{A}$ given $\{x_i\}_{i \in M}$, and thus $v_j$ may not be equal to the original input $x_j$). For each party $P_i$ and for each gate $g \in [G]$, we denote by $\mathsf{state}_i(g)$ the state of $P_i$ after computing gate $g$. We do not want to assume erasures, and thus we assume that $\mathsf{state}_i(g+1)$ contains $\mathsf{state}_i(g)$ for every gate $g \in [G-1]$. We denote the joint state of all parties after computing gate $g$ by

$$\mathsf{state}(g) = (\mathsf{state}_1(g), \ldots, \mathsf{state}_n(g)).$$

Our simulator tries to simulate all the states $\mathsf{state}_i(g)$. Note that he can easily simulate these states for the semi-malicious parties, since semi-malicious parties act honestly, except that they may choose their inputs and randomness arbitrarily. Indeed, the simulator can easily extract the selected inputs of the semi-malicious parties, and their randomness at each step, which is enough to simulate their states step by step.

On the other hand, the simulator cannot fully simulate the states of *honest* parties, since, for example, $\mathsf{state}_i(g)$ contains the input $x_i$, which the simulator does not know. However, as we describe below, he does succeed in simulating these states "partially". More specifically, as in the GMW protocol, $\mathsf{state}_i(g)$ consists of the input $x_i$ of party $P_i$, all the messages received by party $P_i$, and the randomness generated and used by $P_i$ during the computation. Each simulated $\mathsf{state}_i(g)$ will have all the above components, but with unknowns from $\{v_g\}_{g \in [G]}$ (which, in turn, can be efficiently computed from the inputs $x_1, ..., x_n$). Namely, once these unknowns are instantiated, the simulated $\mathsf{state}_i(g)$ is computational indistinguishable from the "real" $\mathsf{state}_i(g)$. Moreover, viewing the simulated $\mathsf{state}_i(g)$ as a function of $\{v_g\}$, this function is efficiently computable.

Jumping ahead, this enables the simulator to simulate the adversary's auxiliary information queries as follows: Each time the adversary asks for an auxiliary information query of the form $L(\mathsf{state}(g))$, the simulator asks for the auxiliary information query $L'$, defined as follows: $L'$ has the simulated $\mathsf{state}(g)$ hard-wired into it, and on input $(x_1, ..., x_n)$, the function $L'$ computes the value of all the unknowns $\{v_g\}$ that appear in $\mathsf{state}(g)$, and then outputs $L(\mathsf{state}(g))$.

**The Simulator $\mathcal{S}$.**

**1. Generating** crs. Sample random messages $M_0, M_1 \leftarrow \mathbb{G}$. For each honest party $P_i$, sample *injective* LTDF seeds with inversion trapdoors $(\sigma_i^{(1)}, \tau_i^{(1)}) \leftarrow G_{\mathsf{Inj}}^{(1)}$, $(\sigma_i^{(2)}, \tau_i^{(2)}) \leftarrow G_{\mathsf{Inj}}^{(2)}$, and sample a random DDH tuple by sampling $\alpha, \beta, \gamma \leftarrow \mathbb{Z}_p^*$ and taking $\vec{g}_i = (g^\alpha, g^\beta, g^\alpha\gamma, g^\beta\gamma) \in \mathbb{G}^4$. For each corrupted party $P_j$, sample *lossy* LTDF seeds $\sigma_j^{(1)} \leftarrow G_{\mathsf{Loss}}^{(1)}$, $\sigma_j^{(2)} \leftarrow G_{\mathsf{Loss}}^{(2)}$, and sample a random *non*-DDH tuple $\vec{g}_j = (g_1, g_2, g_3, g_4) \leftarrow \mathbb{G}^4$ (e.g., by rejection sampling). Output $\{M_0, M_1, (\sigma_1^{(1)}, \sigma_1^{(2)}, \vec{g}_1), ..., (\sigma_n^{(1)}, \sigma_n^{(2)}, \vec{g}_n)\}$ as the common reference string. (Note that $\{(\alpha_i, \beta_i, \gamma_i)\}_{i \in [n] \setminus M}$ and $\{\tau_i^{(1)}, \tau_i^{(2)}\}_{i \in [n] \setminus M}$ are kept by the simulator as trapdoors).

**2. Input-Sharing Stage:** Recall that we denote the $i$th input gate by gate $i$. The simulator first initializes $\mathsf{state}_i(0) = (\mathsf{crs}, v_i)$ for every party $P_i$, where $v_i$ is set as follows. For each corrupt party $P_i$, the value of $v_i$ is determined by extracting the input $x_i'$ from the adversary's special auxiliary tape. For an honest party $P_i$, $v_i$ is set to be the original input $x_i$; note that the simulator thus does not know $v_i$ explicitly, but can efficiently compute it given $x_i$.

The simulator simulates each party $P_i$ in the input-sharing stage by running the following for $i = 1, ..., n$:

- If party $P_i$ is corrupted, the simulator uses $\mathcal{A}$'s selected input $x_i'$ and randomness for this step (which are written on $\mathcal{A}$'s special auxiliary tapes) to simulate $P_i$.
- Otherwise, if $P_i$ is honest, the simulator runs the following steps:
  1. Choose randomness $s_{i,1}, \ldots, s_{i,n} \leftarrow \{0,1\}$ such that $\sum_{i=1}^n s_{i,j} = 0$.
  2. For every $j \neq i$, send $s_{i,j}$ to $P_j$. Note that if $P_j$ is an honest party, then $\mathcal{S}$ simply keeps a record of $s_{i,j}$; otherwise, if $P_j$ is corrupted, then $\mathcal{S}$ sends $s_{i,j}$ to $\mathcal{A}$.
  3. For every $j \neq i$, set the share of each party $P_j$ to be $s_{i,j}$, and set the share of party $P_i$ to be $s_{i,i} \oplus v_i$, where recall that $v_i = x_i$ is the value of gate $i$.
  4. For every $j \neq i$, set $\mathsf{state}_j(i) = \mathsf{state}_j(i-1) \cup \{s_{i,j}\}$, and set $\mathsf{state}_i(i) = \mathsf{state}_i(i-1) \cup \{s_{i,i} \oplus v_i, r_i\}$, where $r_i \triangleq \{s_{i,j}\}_{j \neq i}$ is the simulated randomness used by $P_i$ to secret share his input.

**Adaptive Auxiliary Information Queries.** Upon receiving an adaptive auxiliary information query $L$ from $\mathcal{A}$, the simulator $\mathcal{S}$ sends the following query $L'$ to its auxiliary information oracle. We assume for simplicity, and without loss of generality, that all the auxiliary information queries that are sent during the input preparation stage are sent at the end of this stage, namely after $\mathsf{state}(n)$ was computed. The function $L'$ has the value $\mathsf{state}(n)$ (as defined above) hardwired into it. It takes as input the actual input values $(x_1, \ldots, x_n)$, it computes the unknowns $\{v_i\}_{i \notin M} = \{x_i\}_{i \notin M}$ (which are the unknowns that appear in $\mathsf{state}(n)$), then it computes the actual value $\mathsf{state}(n)$ and outputs $L(\mathsf{state}(n))$.

**3. Circuit Evaluation:** Recall that the circuit evaluation is done in a gate-by-gate (and level-by-level) manner. We describe how to simulate $\mathsf{state}_i(g)$, for every party $P_i$ and for every $g \in [G]$, assuming we already simulated $\mathsf{state}_i(g-1)$ for every party $P_i$.[7]

We explain the simulation of $\mathsf{state}_i(g)$ separately for addition gates and multiplication gates.

**Addition gate:** Suppose that the $g^{th}$ gate is an addition gate, and let $(\alpha_1, \ldots, \alpha_n)$ denote the simulated shares of its left child, and let $(\beta_1, \ldots, \beta_n)$ denote the simulated shares of its right child, where $(\alpha_i, \beta_i)$ are the simulated shares of party $P_i$ (where both $\alpha_i$ and $\beta_i$ are present in $\mathsf{state}_i(g-1)$). For each $i \in [n]$, the simulator $\mathcal{S}$ computes $\gamma_i = \alpha_i + \beta_i$ and sets

$$\mathsf{state}_i(g) = \mathsf{state}_i(g-1) \cup \{\gamma_i\}.$$

From our induction assumption, $\mathsf{state}_i(g-1)$ (and, in particular, $\alpha_i, \beta_i$) is an efficiently computable function of the unknowns $\{v_g\}$. This implies that $\gamma_i$, and thus $\mathsf{state}_i(g)$, is also efficiently computable from the $\{v_g\}$.

**Adaptive Auxiliary Information Queries.** Upon receiving an adaptive auxiliary information query $L$ from $\mathcal{A}$, the simulator $\mathcal{S}$ sends the query $L'$ to his auxiliary information oracle, where $L'$ is defined as follows. Suppose that the query $L$ was sent at the end of the simulation of the $g^{th}$ gate, and thus the adversary expects in return the value $L(\mathsf{state}(g))$. This value is computed by first using the inputs $(x_1, \ldots, x_n)$ to compute the unknowns $\{v_g\}$ that appear in $\mathsf{state}(g)$, and then computing $L(\mathsf{state}(g))$.

**Multiplication gate:** Suppose the $g^{th}$ gate is a multiplication gate. Let $(\alpha_1, \ldots, \alpha_n)$ denote the simulated shares of its left child, and let $(\beta_1, \ldots, \beta_n)$ denote the simulated shares of its right child

Recall that in the real computation of a multiplication gate, each pair of parties $P_i$ and $P_j$ runs an OT protocol with inputs that correspond to their shares $(\alpha_i, \beta_i)$ and $(\alpha_j, \beta_j)$, respectively. We consider four different cases:

1. $P_j$ is a corrupted sender, and $P_i$ is an honest receiver. In this case, $\mathcal{S}$ needs to simulate the honest receiver $P_i$.

2. $P_i$ is an honest sender, and $P_j$ is a corrupted receiver. In this case, $\mathcal{S}$ needs to simulate the honest sender $P_i$.

3. Both sender $P_i$ and receiver $P_j$ are honest. In this case, $\mathcal{S}$ needs to simulate *both* the honest sender and receiver.

4. Both sender and receiver are corrupted. In this case, $\mathcal{S}$ can read $\mathcal{A}$'s randomness for each step (from the special randomness tape) and simulate appropriately.

**Remark.** Note that at the beginning of the computation of gate $g$, the state of $P_i$ is the same as his state at the end of the computation of gate $g-1$. We abuse notation, and initially set $\mathsf{state}_i(g) \leftarrow \mathsf{state}_i(g-1)$ for every party $P_i$. This is an abuse of notation, since we typically use $\mathsf{state}_i(g)$ to denote the state of party $P_i$ at the *end* of the computation of gate $g$. During each OT

[7]Recall that we have already simulated $\{\mathsf{state}_i(g)\}_{g=1}^{n}$ in the input preparation stage, where the first $n$ gates denote the input gates.

protocol, the simulator updates $\mathsf{state}_i(g)$ to include the randomness used and the messages received by $P_i$ during the OT protocol, as described below.

In what follows, we assume for the sake of simplicity of notation, and without loss of generality, that the decryption key of the NCE scheme simply consists of all randomness used in the key generation procedure. Namely, $(e, d) \leftarrow \mathsf{NCGen}(1^k, d)$. Note that this is without loss of generality since our adaptive auxiliary information model does not assume erasures: i.e., this randomness is already contained as part of each party's secret state anyway, which can be leaked on. (Also recall that the NCE simulator algorithms already simulate all randomness used during key generation, as opposed to simply simulating the decryption key).

**Case 1: Simulating honest receiver $P_i$.** Note that $\mathcal{S}$ must simulate the actions of $P_i$ in the OT without knowing the true value of $P_i$'s input $b$ (since $b$ likely depends on previous values $\{v_g\}$ that $\mathcal{S}$ does not know). At a high level, this is done as follows. When $P_i$ is supposed to generate a single encryption-decryption key pair $(e_b, d_b)$ and sample the remaining three encryption keys $e_\ell$ obliviously, the simulator will generate *all four* key pairs with corresponding decryption keys. He simulates the coin tossing protocol (run by all parties to generate $P_i$'s randomness for this step) in such a way that he can later "explain" the values $(e_1, ..., e_4)$ together with the correct $d_b$. In order to simulate answers to auxiliary information queries made on $P_i$, the simulator $\mathcal{S}$ maintains a simulation of $P_i$'s secret state as a function of $b$ (and thus a function of the original unknown values $\{v_g\}$). For each adaptive auxiliary input query $\mathcal{A}$ requests, the simulator $\mathcal{S}$ makes a corresponding query to his ideal world oracle that plugs in the correct values of the unknowns $\{v_g\}$ to $P_i$'s simulated secret state and then computes $\mathcal{A}$'s auxiliary information function.

Explicitly, the simulator $\mathcal{S}$ does the following:

1. For *every* value of $b' \in [4]$, generate a key pair using $\mathsf{NCGen}$. That is, sample randomness $\mathsf{rand}_{b'}$, evaluate $\mathsf{rnd}_{b'} = \mathsf{Rand\text{-}KeyGen}(\sigma_i^{(2)}, \vec{g}_i; \mathsf{rand}_{b'})$, and take $(e_{b'}, d_{b'}) = \mathsf{NCGen}(1^k; \mathsf{rnd}_{b'})$.

   The simulator sends $(e_1, \ldots, e_4)$ as the simulated message sent from $P_i$ to $P_j$ in the OT protocol.

2. Simulate the secret state of $P_i$ as a function of his secret input $b = 2^1 \cdot \alpha_i + 2^0 \cdot \beta_i + 1$, as follows. (Note that the simulator may not know $b$; however, it can be efficiently computed from $(e_1, \ldots, e_4, d_1, \ldots, d_4)$ and from $P_i$'s input shares $(\alpha_i, \beta_i)$, which can in turn be efficiently computed from the simulated $\mathsf{state}_i(g-1)$ and the values $\{v_g\}$.)

   For $b \in [4]$ corresponding to the secret input $b \in [4]$, the simulator simply appends $\mathsf{rand}_b$ as the randomness selected by $P_i$ to ultimately generate an standard NCE key pair.

   For each $b' \in [b] \setminus \{b\}$, the simulator must explain the public key $e_{b'}$ as being generated via the oblivious key generation algorithm $\mathsf{ONCGen}$, with randomness generated from the $\mathsf{DDH\text{-}Rerand}$ procedure. Consider one such $b'$. Parse $e_{b'} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_{4k})$ as $4k$ public keys in the underlying lossy encryption scheme. For each $\ell \in [4k] \setminus I_{b'}$, the key was already properly obliviously generated using $\mathsf{DDH\text{-}Rerand}$. For each remaining $\ell \in I_{b'}$, the simulator did not sample $\mathsf{pk}_\ell = (g^{\alpha'}, g^{\beta'}, g^{\alpha' u}, g^{\beta' u})$ using $\mathsf{DDH\text{-}Rerand}$; but, he knows the discrete logarithm information $\alpha', \beta', u$ of these target values (as this is part of the generation process), and he also knows (as part of the CRS generation trapdoor) the discrete logarithm values $\alpha, \beta, \gamma$ of

$\vec{g}_i$ CRS vector of party $P_i$. Thus, the simulator can "explain" $\mathsf{pk}_\ell$ as being generated via $\mathsf{DDH\text{-}Rerand}$, with rerandomization exponents $\alpha^{-1}\alpha, \beta^{-1}\beta$, and $\gamma^{-1}u$.

3. Upon receiving the ciphertexts $(c_1, \ldots, c_4)$ from the semi-malicious sender $P_j$, the simulator computes $o_i = \mathsf{NCDec}_{d_i}(c_i)$ for every $i \in [4]$. If for any $i$, the output is $\perp$ (which is the case if the ciphertext $c_i$ is malformed[8]), then simulate $P_i$ sending an abort message to all honest parties, and thus all honest parties abort.

4. The simulator also updates the state of each corrupted party $P_\ell$ as dictated by the random tape of $\mathcal{A}$.

**Adaptive Auxiliary Information Queries.** Upon receiving an adaptive auxiliary information query $L$ from $\mathcal{A}$, the simulator $\mathcal{S}$ sends the corresponding query $L'$, which first computes the unknowns $\{v_g\}$ that appear in $\mathsf{state}(g)$, and then computes $L(\mathsf{state}(g))$.

**Case 2: Simulating honest sender $P_i$.** Here, $\mathcal{S}$ must simulate the actions of $P_i$ as the *sender* in the OT without knowing $P_i$'s input shares $(\alpha_i, \beta_i)$. Recall that in the OT protocol, the messages $m_1, ..., m_4$ held by $P_i$ correspond to $\eta \oplus (\alpha_i + \alpha_j)(\beta_i + \beta_j)$ for the four possible values of the receiver $P_j$'s input shares $(\alpha_j, \beta_j)$, where $\eta$ is a fixed random value that serves as $P_i$'s secret share of the desired output (i.e., $(\alpha_i + \alpha_j)(\beta_i + \beta_j)$). Note that, because of the mask $\eta$, any *one* of the output messages $o_\ell$ is uniformly distributed. At a high level, the simulator $\mathcal{S}$ will select the output message $o_b$ learned by the (semi-malicious) receiver $P_j$ *at random*, and will somehow simulate the three remaining messages (and all other pieces of $P_i$'s secret state) as a function of the original unknown values $\{v_g\}$. To simulate answers to auxiliary information queries requested by $\mathcal{A}$, the simulator $\mathcal{S}$ makes a corresponding auxiliary information query to his ideal world oracle that plugs in the correct values of the unknowns $\{v_g\}$ to the expression of $P_i$'s secret state and then computes $\mathcal{A}$'s auxiliary information function.

Explicitly, the simulator $\mathcal{S}$ does the following:

1. Let $(e_1, e_2, e_3, e_4)$ denote the four public keys sent by the semi-malicious receiver $P_j$. Let $b \in [4]$ denote the input of the semi-malicious receiver $P_j$ in the OT protocol. Note that $b$ is well-defined from the input shares $(\alpha_j, \beta_j)$ of the party $P_j$, and moreover, note that the simulator can easily extract these shares.

2. Choose a random bit $o_b$ (as the output to be received by $P_j$ in the OT protocol), and honestly follow the encryption procedure: i.e., sample randomness $r_b \leftarrow \mathsf{Rand\text{-}Enc}(\sigma_i^{(1)}, \sigma_i^{(2)})$, and compute $c_b = \mathsf{NCEnc}_{e_b}(o_b; r_b)$.

   Let $\eta = o_b + (\alpha_i + \alpha_j)(\beta_i + \beta_j)$, where $b = 2^1 \cdot \alpha_j + 2^0 \cdot \beta_j + 1$ corresponds to the shares $(\alpha_j, \beta_j)$ that were extracted from the semi-malicious receiver $P_j$. The simulator would like to send

$$(\mathsf{NCEnc}_{e_1}(m_1), \ldots, \mathsf{NCEnc}_{e_4}(m_4))$$

[8]For the particular NCE scheme that we use, this can occur, e.g., if some of the $4k$ ciphertexts of the underlying lossy encryption scheme decrypt to $M_0$ and some decrypt to $M_1$. See Section 3.4.2 for details.

to the receiver, where for each $b' = 2^1 \cdot \delta_0 + 2^0 \cdot \delta_1 + 1 \in [4]$ (where $\delta_0, \delta_1 \in \{0,1\}$),

$$m_{b'} = \eta + (\alpha_i + \delta_0)(\beta_i + \delta_1).$$

(Note that, for $b' = b$, we have $m_b = o_b$). However, the simulator may not know $(m_1, \dots, m_4)$, since he may not know the shares $(\alpha_i, \beta_i)$ of the honest sender $P_i$.

Instead, the simulator does the following: For every $b' \in [4]$ such that $b' \neq b$, run the *alternative simulator* $\mathsf{NCSim}'$ of the augmented NCE scheme, with input $e_{b'}$, to compute a tuple $(c_{b'}, r^0_{E,b'}, r^1_{E,b'}) \leftarrow \mathsf{NCSim}'(e_{b'})$, which he can later open to either value 0 or 1.

3. Simulate the secret state of the honest party to explain the randomness $(r^{m_1}_{E,1}, \dots, r^{m_4}_{E,4})$. Recall that this randomness should supposedly have been generated via $\mathsf{Rand\text{-}Enc}(\sigma_i^{(1)}, \sigma_i^{(2)}$, which makes a sequence of calls to the LTDF functions $F(\sigma_i^{(1)}, \cdot)$ and $F(\sigma_i^{(2)}, \cdot)$. For each such supposed LTDF evaluation, the simulator determines a preimage randomness that "explains" the desired output by using the inversion trapdoors $\tau_i^{(1)}, \tau_i^{(2)}$ (generated during the CRS simulation).

   The complete collection of preimage randomness values is added to the simulated secret state of party $P_i$. Denote this by $\mathsf{rand}_1, \dots, \mathsf{rand}_4$).

4. Simulate the sender $P_i$ sending $(c_1, \dots, c_4)$ to $P_j$.

5. The simulator also updates the state of each corrupted party $P_\ell$ as dictated by the random tape of $\mathcal{A}$.

**Adaptive Auxiliary Information Queries.** Upon receiving an adaptive auxiliary information query $L$ from $\mathcal{A}$, the simulator $\mathcal{S}$ sends the corresponding auxiliary information query $L'$, which first computes the unknowns $\{v_g\}$ that appear in $\mathsf{state}(g)$, and then computes $L(\mathsf{state}(g))$.

**Case 3: Simulating honest sender $P_i$ and honest receiver $P_j$.** Note that in this case the simulator does not know *both* the shares $(\alpha_i, \beta_i)$ of the honest sender *and* the shares $(\alpha_j, \beta_j)$ of the honest receiver. Loosely speaking, the simulator will proceed by combining the strategies from the previous two cases. Explicitly, simulator $\mathcal{S}$ does the following:

1. Run the simulator $\mathsf{NCSim}$ of the (augmented) NCE scheme to compute

$$(e_\ell, c_\ell, d^0_\ell, r^0_{E,\ell}, d^1_\ell, r^1_{E,\ell}) \leftarrow \mathsf{NCSim}(1^k),$$

   for each $\ell \in [4]$. Later in the simulation, the keys $(e_1, ..., e_4)$ will be used as the message sent by the receiver $P_j$, and the ciphertexts $(c_1, ..., c_4)$ will be used as the responding message sent by the sender $P_i$.

2. Choose a random bit $\eta \leftarrow \{0,1\}$ to serve as the secret share of $(\alpha_i + \alpha_j)(\beta_i + \beta_j)$ kept by the sender, $P_i$. For every $\ell = 2^1 \cdot \delta_0 + 2^0 \cdot \delta_1 + 1 \in [4]$, (where $\delta_0, \delta_1 \in \{0,1\}$), let $m_\ell = \eta + (\alpha_i + \delta_0)(\beta_i + \delta_1)$. The tuple $(m_1, ..., m_4)$ are the messages held by the sender in the OT protocol. (Note that the simulator may not know $(m_1, ..., m_4)$, but they are efficiently computable given $\eta$ and the unknown input shares $(\alpha_i, \beta_i)$ of the sender.)

3. Denote by $b$ the input of receiver $P_j$ in the OT protocol: that is, $b = 2^1 \cdot \alpha_j + 2^0 \cdot \beta_j + 1$, where $(\alpha_j, \beta_j)$ denote the input shares of $P_j$.

   Simulate the secret state of the receiver $P_j$ so as to explain the encryption key $e_b$ as being generated by NCGen (with randomness from Rand-KeyGen), and the remaining keys $e_{b'}$ for $b' \in [4] \setminus \{b\}$ as being generated by ONCGen (with randomness from Rand-OblivKeyGen), as described in Case 1 above.

4. Send $(e_1, e_2, e_3, e_4)$ to the sender $P_i$ on behalf of the receiver $P_j$.

5. In response, send the ciphertexts $(c_1, \ldots, c_4)$ (that were also computed earlier by running NCSim) on behalf of the sender $P_i$ to the receiver $P_j$.

   The simulator now simulates the secret state of the sender $P_j$ to be consistent with each ciphertext $c_\ell$ as the output of NCEnc encrypting the correct message $m_\ell$ (and with randomness generated via Rand-Enc$(\sigma_i^{(1)}, \sigma_i^{(2)})$. This is done exactly as described in Case 2 above.

6. The simulator also updates the state of each corrupted party $P_\ell$ as dictated by the random tape of $\mathcal{A}$.

**Adaptive Auxiliary Information Queries.** Upon receiving an adaptive auxiliary information query $L$ from $\mathcal{A}$, the simulator $\mathcal{S}$ sends the corresponding query $L'$, which first computes the unknowns $\{v_g\}$ that appear in state$(g)$, and then computes $L(\text{state}(g))$.

**Case 4: Simulating for corrupt sender and receiver.** Let $(\alpha_i, \beta_i)$ and $(\alpha_j, \beta_j)$ be the input shares of the corrupt sender $P_i$ and receiver $P_j$. Recall that the simulator $\mathcal{S}$ can easily extract these shares from the adversary. Let $\eta$ be the random bit that was selected by the sender $P_i$ (which can be read from the adversary's special random tape). Collectively, these values define the receiver's input $b$, the sender's messages $m_1, ..., m_4$, and the "correct" output of the OT: namely, the message $m_b$.

The simulator $\mathcal{S}$ simulates the OT execution as follows. $\mathcal{S}$ honestly simulates the actions of each honest party in the coin-tossing protocol. For each corrupted party, $\mathcal{S}$ reads the randomness used in each step of the protocol from the adversary's special random tape, and simulates exactly. Let $m'$ be the final output of the OT, as determined by this simulation. If $m'$ is *not* equal to the correct output $m_b$, then the simulator $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ updates the state of each party according to his simulation. Namely, $\mathcal{S}$ updates the state of each corrupted party $P_\ell$ as dictated by the random tape of $\mathcal{A}$.

**3. Output Stage:** Denote by $\alpha_i$ the share of each party $P_i$ corresponding to the output wire. Recall that in the output re-randomization stage, each party $P_i$ performs the following steps: (a) generating an NCE key pair $(e_{\ell,i}, d_{\ell,i})$ for each other party $P_\ell$ and broadcasting all the encryption keys $\{e_{\ell,i}\}_{\ell \neq i}$, (b) secret sharing his output share $\alpha_i$ as $\alpha_i = \alpha_{i,1} \oplus \cdots \oplus \alpha_{i,n}$, and (c) sending each generated share $\alpha_{i,\ell}$ to the corresponding party $P_\ell$, encrypted under $P_\ell$'s key $e_{i,\ell}$. At a high level, the simulator $\mathcal{S}$ will use the adversary $\mathcal{A}$ to simulate all the corrupted parties, and will simulate the actions of each honest party $P_i$, by choosing *random* secret shares $\alpha_{i,\ell}$ of $P_i$'s output share (and acting honestly) for *all but one share*. For the remaining secret share, $\mathcal{S}$ instead generates *simulated* keys and ciphertexts for the NCE scheme, which he can later open to any value he wishes.

In what follows, suppose that $P_n$ (the party who receives the final output) is honest. The case where $P_n$ is adversarial is very similar, and is addressed later.

We note that the simulator knows the output share $\alpha_j$ for each corrupted party $P_j$, and for every honest party $P_i$, the share $\alpha_i$ is a polynomial-time computable function of $\{v_g\}$. The simulator $\mathcal{S}$ simulates the "re-randomization" of the shares as follows.

1. For each honest party $P_i$, s.t. $i \neq n$, sample $(e_{\ell,i}, d_{\ell,i}) \leftarrow \mathsf{NCGen}(1^k)$ for each $\ell \in [n] \setminus \{i\}$, and send $e_{\ell,i}$ to each party $P_\ell$. Note that key $e_{\ell,i}$ will be used for messages sent from party $P_\ell$ to $P_i$.

2. For party $P_n$:

   (a) Sample $(e_{\ell,n}, d^0_{\ell,n}, d^1_{\ell,n}, c_{\ell,n}, r^0_{E,\ell,n}, r^1_{E,\ell,n}) \leftarrow \mathsf{NCSim}(1^k)$ for each honest party $P_\ell$ s.t. $\ell \neq n$, and send $e_{\ell,n}$.

   (b) Sample $(e_{\ell,n}, d_{\ell,n}) \leftarrow \mathsf{NCGen}(1^k)$ for each corrupted party $P_\ell$, and send $e_{\ell,n}$ to $P_\ell$.

3. For each corrupted party $P_\ell$, simulate $\mathcal{A}$ to obtain $e_{i,\ell}$ for each $\ell \in [n] \setminus \{\ell\}$. These keys will be used for messages sent from $P_i$ to corrupted party $P_\ell$.

4. For each honest party $P_i$, choose at random $r_{i,1}, \ldots, r_{i,n-1} \leftarrow \{0,1\}$. These will correspond to *all but one* secret share of $P_i$'s output share $\alpha_i$.

   Let
   $$r_{i,n} \triangleq \alpha_i \oplus \left( \bigoplus_{\ell=1}^{n-1} r_{i,\ell} \right),$$
   so that
   $$\alpha_i = \bigoplus_{\ell=1}^{n} r_{i,\ell}.$$
   Note that the simulator may not know $r_{i,n}$ since he may not know $\alpha_i$. However, he could compute $r_{i,n}$ efficiently if he was given $\{v_g\}$ (since $\alpha_i$ itself is an efficiently-computable function of $\{v_g\}$).

5. For each honest party $P_i$ and for each $\ell \in [n-1] \setminus \{i\}$, simulate $P_i$'s message to $P_\ell$ honestly: i.e., choose randomness $u_{i,\ell} \leftarrow \{0,1\}^*$ and send the ciphertext $c_{i,\ell} = \mathsf{NCEnc}_{e_{\ell,i}}(r_{i,\ell}, u_{i,\ell})$ to party $P_\ell$.

6. For each honest party $P_i$ $(i \neq n)$, send $c_{i,n}$ (as computed in step 2) to party $P_n$.

   Jumping ahead, the ciphertext $c_{i,n}$ should decrypt to $r_{i,n}$, and the simulator can equivocate on the randomness of this ciphertext if needed, since the *correct* encryption randomness $r^{r_{i,n}}_{E,i,n}$ is a function of $r_{i,n}$, which in turn is a function of $\{v_g\}$.

7. For each party $P_i$, let
$$r'_i \triangleq \bigoplus_{\ell=1}^{n} r_{\ell,i}.$$
Note that the simulator knows $r'_i$ for every $i \in [n-1]$, but does not know $r'_n$. Fortunately, he could compute $r'_n$ efficiently if he was given $\{v_g\}$ (which he will do to answer leakage queries).

8. For each party $P_i$ ($i \neq n$), simulate $P_i$ sending the share $r'_i$ to party $P_n$, who is the party receiving the output.

The simulator sends the inputs $(x_i)_{i\in M}$ of the corrupted parties to the trusted party, and the trusted party returns
$$y = f(x_1, \ldots, x_n)$$
to the honest party $P_n$.

For each honest party $P_i$ ($i \neq n$), set

$$\mathsf{state}_i(\mathsf{output}) =$$
$$\left(\mathsf{state}_i(G), \{e_{\ell,i}, d_{\ell,i}\}_{\ell\in[n]\setminus\{i\}}, \{e_{i,\ell}\}_{\ell\in[n]\setminus\{i\}}\{r_{i,\ell}\}_{\ell=1}^{n}, \{c_{i,\ell}, u_{i,\ell}\}_{\ell\in[n-1]\setminus\{i\}}, (c_{i,n}, r_{E,i,n}^{r_{i,n}}), \{c_{\ell,i}\}_{\ell\neq i}\right).$$

For party $P_n$, set

$$\mathsf{state}_n(\mathsf{output}) = \left(\mathsf{state}_n(G), \{e_{i,n}, d_{i,n}\}_{i\in M}, \{e_{i,n}, d_{i,n}^{r_{i,n}}\}_{i\in H}, \{r_{n,\ell}\}_{\ell=1}^{n}, \{c_{n,\ell}, u_{n,\ell}\}_{\ell\in[n-1]}, \{c_{\ell,n}\}_{\ell\neq n}\right),$$

where $H$ refers to the set of honest parties $P_i$ (s.t. $i \neq n$) and $M$ refers to the set of corrupted parties.

**Leakage Queries.** Upon receiving a auxiliary information query $L$ from $\mathcal{A}$, the simulator $\mathcal{S}$ sends the auxiliary information query $L'$, which first computes the unknowns $\{v_g\}$ that appear in $\mathsf{state}(g)$, and then computes $L(\mathsf{state}(g))$.

**Remark.** In the above simulation we assumed that the party $P_n$, who receives the output, is honest. If $P_n$ is malicious, then the simulator chooses an arbitrary honest party, say $P_1$, and does the same as above, while replacing the role of $P_n$ with $P_1$. The main difference is that now $P_1$ needs to send his *unknown* re-randomized share to $P_n$. The simulator uses the output $y$ of the trusted party to compute this re-randomized share as follows:
$$r'_1 \triangleq y \oplus \left(\bigoplus_{i=1}^{n} r'_i\right),$$
and simulates party $P_1$ sending $r'_1$ to party $P_n$, who then supposedly outputs $\bigoplus_{i=1}^{n} r'_i = y$.

### 3.5.3 Proof of Indistinguishability

In this section, we prove that the output of the ideal-world experiment with the simulator $\mathcal{S}$ (from the previous section) is, indeed, computationally indistinguishable from the output of the real-world experiment with adversary $\mathcal{A}$.

Let us start by introducing some notation. In Section 3.5.2, we slightly abused notation, and denoted by $\mathsf{state}_i(g)$ both the simulated state and the real state. In this section, it is important to distinguish between the two, and hence we denote by $\mathsf{state}_i(g)$ the real state and by $\mathsf{state}'_i(g)$ the simulated state. Let $\vec{x} = (x_1, ..., x_n)$ be the set of all parties' inputs, and $z$ be the auxiliary information given to $\mathcal{A}$ (where these values may depend efficiently on the CRS). We denote

$$\mathsf{state}_i(\mathsf{output}) = (s_{i,\mathsf{init}}, s_{i,1}, s_{i,2}, \ldots, s_{i,G}, s_{i,\mathsf{out}}),$$

where $s_{i,\mathsf{init}} = ((\mathsf{crs}_i, h_i), x_i, z_i)$ is the initial state of $P_i$ (where $z_i = \emptyset$ for honest $P_i$ and $z_i = z$ for corrupted $P_i$); $s_{i,g}$ is the information added to $\mathsf{state}_i(g-1)$ after the computation of gate $g$ (i.e., $\mathsf{state}_i(g) = (\mathsf{state}_i(g-1), s_{i,g})$); and $s_{i,\mathsf{out}}$ is the information added to $\mathsf{state}_i(G)$ in the output stage. (Note that the dependence on $\vec{x}$ and $z$ is implicit).

In contrast, we denote

$$\mathsf{state}'_i(\mathsf{output}) = (\mathsf{sim}_{i,\mathsf{init}}, \mathsf{sim}_{i,1}, \mathsf{sim}_{i,2}, \ldots, \mathsf{sim}_{i,G}, \mathsf{sim}_{i,\mathsf{out}}),$$

where $\mathsf{sim}_{i,\mathsf{init}} = ((\mathsf{crs}'_i, h_i), x_i, z_i)$ is the simulated initial state of $P_i$ (in particular, where $\mathsf{crs}'$ is generated using the simulation algorithm $(\mathsf{crs}', \mathsf{trap}) \leftarrow \mathsf{crsSim}_{\mathsf{eq}}(1^k)$ for honest parties); $\mathsf{sim}_{i,g}$ is the simulated version of $s_{i,g}$; and $\mathsf{sim}_{i,\mathsf{out}}$ is the simulated version of $s_{i,\mathsf{out}}$. More specifically, recall that the simulator $\mathcal{S}$ fully simulates the states for the semi-malicious parties, and simulates the states of honest parties as a function of the unknown inputs $\{x_i\}_{i \notin M}$. For corrupted parties $P_i$, the value $\mathsf{state}'_i$ is defined to be the simulated state of $P_i$ generated by $\mathcal{S}$. For honest parties $P_i$, the value $\mathsf{state}'_i$ is defined to be the simulated state of $P_i$ generated by $\mathcal{S}$, with the values of the true inputs $\{x_i\}_{i \notin M}$ plugged in. (We emphasize, however, that the simulator does not have access to these inputs $\{x_i\}_{i \notin M}$, except when answering adaptive auxiliary information queries).

Note that in order to prove that the output of the ideal-world experiment with simulator $\mathcal{S}$ is computationally indistinguishable from the output of the real-world experiment with adversary $\mathcal{A}$, it suffices to prove that, for every set of inputs $\vec{x}$ and every auxiliary input $z$,

$$(\mathsf{state}_i(\mathsf{output}))_{i=1}^n \overset{c}{\cong} (\mathsf{state}'_i(\mathsf{output}))_{i=1}^n .$$

In particular, these distributions contain the inputs and outputs of every party, and the entire views of corrupted parties throughout the experiment, including adaptive auxiliary information.

For every $g \in \{1, 2 \ldots, G\}$, we denote by

$$\begin{aligned} s_g &\triangleq (s_{1,g}, \ldots, s_{n,g}), & \mathsf{sim}_g &\triangleq (\mathsf{sim}_{1,g}, \ldots, \mathsf{sim}_{n,g}), \\ s_{\mathsf{init}} &\triangleq (s_{1,\mathsf{init}}, \ldots, s_{n,\mathsf{init}}), & \mathsf{sim}_{\mathsf{init}} &\triangleq (\mathsf{sim}_{1,\mathsf{init}}, \ldots, \mathsf{sim}_{n,\mathsf{init}}), \\ s_{\mathsf{out}} &\triangleq (s_{1,\mathsf{out}}, \ldots, s_{n,\mathsf{out}}), & \mathsf{sim}_{\mathsf{out}} &\triangleq (\mathsf{sim}_{1,\mathsf{out}}, \ldots, \mathsf{sim}_{n,\mathsf{out}}). \end{aligned}$$

Thus, it suffices to prove the following statement.

**Theorem 3.5.2.** *For the values defined above, it holds for every set of inputs $\vec{x}$ and every auxiliary input $z$ (which may depend efficiently on the CRS),*

$$\left(s_{\mathsf{init}}, \{s_g\}_{g=1}^{G}, s_{\mathsf{out}}\right) \overset{c}{\cong} \left(\mathsf{sim}_{\mathsf{init}}, \{\mathsf{sim}_g\}_{g=1}^{G}, \mathsf{sim}_{\mathsf{out}}\right).$$

**Remark 3.5.3.** We note that, in Theorem 3.5.2, each component of the distribution vectors can be viewed as a distribution that depends on all previous components. That is, each $s_g$ in the vector $\left(s_{\mathsf{init}}, \{s_g\}_{g=1}^{G}, s_{\mathsf{out}}\right)$ corresponds to the distribution of information added to state$(g-1)$ as a result of the real-world protocol execution of gate $g$ *in which each party $P_i$ enters with view* $(s_{i,\mathsf{init}}, s_{i,1}, ..., s_{i,g-1})$. This will become important later, when we wish to consider "mixed" distribution vectors, e.g. $(\mathsf{sim}_{\mathsf{init}}, \mathsf{sim}_1, ..., \mathsf{sim}_g, s_{g+1}, ..., s_G, s_{\mathsf{out}})$, where the first values are generated using the simulator, and the remaining values are generated by a real-world experiment execution.

*Proof of Theorem 3.5.2.* We prove the theorem by way of three lemmas, in which we replace the output of the CRS generation, the circuit evaluation, and the output re-randomization phases by their simulated versions, one by one.

**Lemma 3.5.4** (CRS Generation). *The simulator $S$ correctly simulates the CRS generation stage. That is, for every set of inputs $\vec{x}$ and every auxiliary input $z$ (which may depend efficiently on the CRS),*

$$\left(s_{\mathsf{init}}, \{s_g\}_{g=1}^{G}, s_{\mathsf{out}}\right) \overset{c}{\cong} \left(\mathsf{sim}_{\mathsf{init}}, \{s_g\}_{g=1}^{G}, s_{\mathsf{out}}\right).$$

*Proof.* Recall that $s_{\mathsf{init}} = ((\mathsf{crs}_i, h_i), x_i, z_i)$ and $\mathsf{sim}_{\mathsf{init}} = ((\mathsf{crs}'_i, h_i), x_i, z_i)$, where $\mathsf{crs}'$ includes *simulated* CRS values. Recall that the CRS simulation involves replacing the random non-DDH tuples $(g_1, g_2, g_3, g_4)$ of honest parties with random DDH tuples, and replacing the random injective LTDF seeds $\sigma$ of corrupted parties with randomly sampled *lossy* seeds. The lemma thus follows immediately by the DDH assumption, the indistinguishability of injective and lossy LTDF seeds, and a standard hybrid argument. □

**Lemma 3.5.5** (Circuit Evaluation). *The simulator $S$ correctly simulates the circuit evaluation stage. That is, for every set of inputs $\vec{x}$ and every auxiliary input $z$ (which may depend efficiently on the CRS),*

$$\left(\mathsf{sim}_{\mathsf{init}}, \{s_g\}_{g=1}^{G}, s_{\mathsf{out}}\right) \overset{c}{\cong} \left(\mathsf{sim}_{\mathsf{init}}, \{\mathsf{sim}_g\}_{g=1}^{G}, s_{\mathsf{out}}\right).$$

This step constitutes the bulk of the proof of Theorem 3.5.2.

*Proof.* We prove the lemma by a standard hybrid argument. For every $\ell \in \{1, 2 \dots, G+1\}$, consider the $\ell^{th}$ hybrid of distributions

$$\mathcal{D}_\ell = \left(\mathsf{sim}_{\mathsf{init}}, \left(\{\mathsf{sim}_g\}_{g=1}^{\ell-1}, \{s_g\}_{g=\ell}^{G}\right), s_{\mathsf{out}}\right),$$

(using the notation described in Remark 3.5.3). By definition, we have that

$$\mathcal{D}_1 = \left(\mathsf{sim}_{\mathsf{init}}, \{s_g\}_{g=1}^{G}, s_{\mathsf{out}}\right) \quad \text{and} \quad \mathcal{D}_{G+1} = \left(\mathsf{sim}_{\mathsf{init}}, \{\mathsf{sim}_g\}_{g=1}^{G}, s_{\mathsf{out}}\right).$$

Thus to prove the lemma, it suffices to prove that for every $\vec{x}, z$, and for every $\ell \in \{1, 2 \ldots, G\}$,

$$\mathcal{D}_\ell \overset{c}{\cong} \mathcal{D}_{\ell+1}.$$

To this end, fix $\ell \in [G]$. Note that if gate $\ell$ is an input gate (i.e., $\ell \in [n]$), then the simulation of the $\ell^{th}$ gate is perfect, and thus

$$\mathcal{D}_\ell = \mathcal{D}_{\ell+1}.$$

Similarly, if the $\ell^{th}$ gate is an addition gate, then the simulation is also perfect, and hence

$$\mathcal{D}_\ell = \mathcal{D}_{\ell+1}.$$

The non-trivial case is where the $\ell^{th}$ gate is a multiplication gate. Recall that in this case, the simulator simulates an OT protocol between every pair of parties. (He also computes $n\alpha_i\beta_i$ for each party $P_i$, but this is done exactly as is done in the real world, and hence for the sake of simplicity is ignored here.)

We denote by $\mathsf{OT}_{i,j}$ the state generated by the OT protocol between parties $P_i$ and $P_j$ , and we denote the corresponding simulated state by $\mathsf{SimOT}_{i,j}$. Thus, to prove that $\mathcal{D}_\ell \overset{c}{\cong} \mathcal{D}_{\ell+1}$, it suffices to prove that

$$(\mathsf{sim}_{\mathsf{init}}, \mathsf{sim}_1, \ldots, \mathsf{sim}_{\ell-1}, (\mathsf{OT}_{i,j})_{1\le i<j\le n}, s_{\ell+1}, \ldots, s_G, s_{\mathsf{out}}) \overset{c}{\cong}$$
$$(\mathsf{sim}_{\mathsf{init}}, \mathsf{sim}_1, \ldots, \mathsf{sim}_{\ell-1}, (\mathsf{SimOT}_{i,j})_{1\le i<j\le n}, s_{\ell+1}, \ldots, s_G, s_{\mathsf{out}}).$$

This is proven via an inner hybrid argument. For the simplicity of notation, we denote $A = (\mathsf{sim}_{\mathsf{init}}, \mathsf{sim}_1, \ldots, \mathsf{sim}_{\ell-1})$ and we denote $B = (s_{\ell+1}, \ldots, s_G, s_{\mathsf{out}})$. We also denote $(\mathsf{OT}_{i,j})_{1\le i<j\le n}$ as an ordered sequence $(\mathsf{OT}_1, \ldots, \mathsf{OT}_m)$. Similarly, we denote $(\mathsf{SimOT}_{i,j})_{1\le i<j\le n}$ as an ordered sequence $(\mathsf{SimOT}_1, \ldots, \mathsf{SimOT}_m)$. According to this notation, we need to prove that

$$(A, \mathsf{OT}_1, \ldots, \mathsf{OT}_m, B) \overset{c}{\cong} (A, \mathsf{SimOT}_1, \ldots, \mathsf{SimOT}_m, B).$$

This is proven via a hybrid argument. For every $\ell \in \{0, 1 \ldots, m\}$, let the $\ell^{th}$ hybrid distribution be

$$\mathcal{U}_\ell = (A, \mathsf{SimOT}_1, \ldots, \mathsf{SimOT}_\ell, \mathsf{OT}_{\ell+1}, \ldots, \mathsf{OT}_m, B).$$

Note that

$$\mathcal{U}_0 = (A, \mathsf{OT}_1, \ldots, \mathsf{OT}_m, B),$$

and

$$\mathcal{U}_m = (A, \mathsf{SimOT}_1, \ldots, \mathsf{SimOT}_m, B).$$

Thus, it suffices to prove that for every $\ell \in [m]$,

$$\mathcal{U}_{\ell-1} \overset{c}{\cong} \mathcal{U}_\ell.$$

To this end, fix any $\ell \in [m]$. We need to prove that

$$\left(A, (\mathsf{SimOT}_i)_{i=1}^{\ell-1}, \mathsf{OT}_\ell, (\mathsf{OT}_i)_{i=\ell+1}^{m}, B\right) \overset{c}{\cong} \left(A, (\mathsf{SimOT}_i)_{i=1}^{\ell-1}, \mathsf{SimOT}_\ell, (\mathsf{OT}_i)_{i=\ell+1}^{m}, B\right). \quad (3.1)$$

We distinguish between the following four cases.

1. **Case 1.** The $\ell^{th}$ OT protocol is between a corrupted sender $P_j$ and an honest receiver $P_i$.
2. **Case 2.** The $\ell^{th}$ OT protocol is between an honest sender $P_i$ and a corrupted receiver $P_j$.
3. **Case 3.** The $\ell^{th}$ OT protocol is between an honest sender $P_i$ and an honest receiver $P_j$.
4. **Case 4.** The $\ell^{th}$ OT protocol is between a corrupted sender $P_i$ and a corrupted receiver $P_j$.

We prove that Equation (3.1) holds for each of the four cases above.[9]

**Case 1.** Recall that the simulated distribution differs from the real version in two respects: (1) $P_i$ aborts in the simulation if *any* of the four ciphertexts $(c_1, ..., c_4)$ is malformed (whereas in the real execution he aborts only if $c_b$ is malformed), and (2) the alleged randomness of $P_i$ is chosen in a different fashion.

Let $E^*$ be the event that at least one ciphertext $c_{b'}$, where $b' \neq b$, sent by the corrupted sender $P_j$ is such that $\mathsf{NCDec}_{d_{b'}}(c_{b'}) = \perp$. We first show that $\Pr[E^*] = \mathsf{negl}(k)$. Recall that the sender is semi-malicious, and thus follows the prescribed protocol (though perhaps using arbitrarily selected randomness). In particular, each ciphertext $c_{b'}$, for $b' \in [4]$, is generated by running the honest encryption algorithm $\mathsf{NCEnc}$ on the message $m_{b'}$ with some value of encryption randomness $r_{E,b'}$ that is generated as the outcome of the procedure $\mathsf{Rand\text{-}Enc}(\sigma_j^{(1)}, \sigma_j^{(2)})$. Thus, by Lemma 3.4.11, it holds with overwhelming probability over the selection of the CRS messages $M_0, M_1 \leftarrow \mathbb{G}$, and honestly generated secret key $d_{b'}$, that $\mathsf{Dec}d_{b'}(c_{b'}) = m$, as desired.

Now, assume that event $E^*$ does not occur. We argue that the simulated randomness of $P_i$ has the same distribution as the corresponding honestly generated randomness. Recall that in an honest execution, $P_i$ samples only the public key $e_b$ using $\mathsf{NCGen}$, and obliviously samples the remaining three. In the simulation, $\mathcal{S}$ sampled all four using $\mathsf{NCGen}$. For each $e_{b'}, b' \neq b$, and each $\ell \in I_{b'}$, this translated to "explaining" the underlying public key $\mathsf{pk}_\ell = (g_1, g_2, g_1^u, g_2^u)$ as being generated via $\mathsf{DDH\text{-}Rerand}(\vec{g}_i)$. But, since $\vec{g}_i$ is a DDH tuple, the two distributions of output public keys are identical: namely, they both produce uniformly random DDH tuples. Further, since the simulator knows the discrete logarithms of the elements in the tuple $\vec{g}_i$, and also in the target tuple $\mathsf{pk}_\ell$ (since these were part of its generation process), he can efficiently produce the appropriate randomness (i.e., rerandomization exponents) that serve as the simulated randomness of $P_i$ (as described in Claim 3.4.7).

**Case 2.** In this case, the sender $P_i$ is honest and the receiver $P_j$ is corrupted. Recall the only differences between the simulated distribution and a real execution are the way in which the ciphertexts $(c_1, ..., c_4)$ are generated and the alleged encryption randomness values are sampled. We begin by showing that the simulated values $\{(c_{b'}, r_{E,b'}^{m_{b'}})\}_{b' \in [4]}$ are statistically indistinguishable from honestly generated ones, where $r_{E,b'}^{m_{b'}}$ constitutes the encryption randomness used in $\mathsf{NCEnc}$. Then, as the final step, we show that the process of arriving at these encryption randomness values (via $\mathsf{Rand\text{-}Enc}$) is properly simulated.

[9]Note that, in particular, this will imply the security of the OT protocol from Section 3.4.4 against adaptive auxiliary information (in the semi-malicious model).

First, consider $(c_b, r_{E,b})$ for the $b \in [4]$ defined by the corrupted receiver $P_j$'s input shares $(\alpha_j, \beta_j)$. Note that in $\mathsf{OT}_\ell^1$, the correct output message $m_b$ is distributed uniformly, since it is a random secret share of the value $(\alpha_i+\alpha_j)(\beta_i+\beta_j)$. Thus, for $b$, the simulated value $m_b = o_b$ (which is chosen at random), ciphertext $c_b \leftarrow \mathsf{NCEnc}_{e_b}(o_b)$, and corresponding encryption randomness all have exactly the right distribution.

Now, consider $b' \in [4] \setminus b$. Given the value $m_b = o_b$, the remaining three values $m_{b'}$ are defined as in $\mathsf{OT}_\ell^1$. Thus, it remains to show that the simulated ciphertext-randomness pairs $(c_{b'}, r_{E,b'}^{m_{b'}})$ are indistinguishable from honestly generated ones for the same messages. If the encryption keys $e_{b'}$ were *honestly* sampled (obliviously), then we would immediately get computational indistinguishability from the standard indistinguishability of simulated ciphertexts in the NCE scheme. However, the receiver $P_j$ is *semi-malicious*, and thus may attempt to choose randomness for key generation in such a way that security no longer holds (e.g., if he has side information that allows him to decrypt). We now show this cannot occur.

Recall that our NCE scheme from Section 3.4.2 has the additional property that almost all public keys generated by $\mathsf{ONCGen}$ are *lossy*. Further, recall that $P_j$ does not have full control over the randomness used to generate the public keys $e_{b'}$, but rather must use the output of $\mathsf{Rand\text{-}OblivKeyGen}(\vec{g}_j)$. Since $P_j$'s CRS tuple $\vec{g}_i$ is a *non*-DDH tuple, it holds by Lemma 3.4.12 that *any* resulting public key $e_{b'}$ that he may generate in this fashion is necessarily lossy, meaning that the pair $(c_{b'}, r_{E,b'}^{m_{b'}})$ generated by the alternative simulator is *statistically* indistinguishable from the honestly generated pair $(c, r)$ as in an honest execution. Since the distributions are statistically close, they are indistinguishable even given any side information on $e_{b'}$ that the adversary may have.

It remains to show that the simulator properly simulates the process of generating these randomness values $r_{E,b'}^{m_{b'}}$ via an execution of $\mathsf{Rand\text{-}Enc}(\sigma_i^{(1)}, \sigma_i^{(2)})$. But, recall that $\sigma_i^{(1)}$ and $\sigma_i^{(2)}$ are *injective* LTDF seeds (which necessarily describe *bijective* functions, since the LTDF family we use is a bijective $(D, \alpha)$-admissible one), and the simulator knows their corresponding inversion trapdoors $\tau_i^{(1)}, \tau_i^{(2)}$ (generated during the simulation of the CRS). Therefore, the simulator will succeed in computing the corresponding preimages for the simulation of the randomness selected by $P_i$ and used in $\mathsf{Rand\text{-}Enc}(\sigma_i^{(1)}, \sigma_i^{(2)})$ in order to generate $r_{E,b'}^{m_{b'}}$. And, further, since the values of $r_{E,b'}^{m_{b'}}$ each by themselves are distributed randomly, the resulting simulated preimages will also have the correct (random) distribution.

**Case 3.** In this case, both the sender $P_i$ and the receiver $P_j$ are honest. This case follows from combining the arguments in the two previous cases. We sketch the proof below.

Recall that in the simulation, the encryption keys, ciphertexts, simulated key generation randomness, and simulated encryption randomness are all generated via the NCE simulator, $\mathsf{NCSim}$. By the indistinguishability properties of the NCE simulator, it holds that this collection of values is indistinguishable from the corresponding collection formed by executing $\mathsf{NCGen}$ and $\mathsf{NCEnc}$ on these corresponding randomness values. Further, since standard public keys (generated via $\mathsf{NCGen}$) are indistinguishable from obliviously sampled ones (generated via $\mathsf{ONCGen}$), this collection of values is indistinguishable from a corresponding collection of values produced by honestly following the OT procedure without the randomness generation steps (i.e., $\mathsf{Rand\text{-}KeyGen}$, $\mathsf{Rand\text{-}OblivKeyGen}$, and $\mathsf{Rand\text{-}Enc}$). (Recall that this honest procedure amounts to the receiver sampling three of his public keys obliviously, and the sender generating encryptions of each of his messages). Finally, from the arguments above and in Section 3.4.3, given that the above distributions are indistinguishable, it holds that the collection of public keys, ciphertexts, and *preimage randomness* used to generate the above key generation randomness (via Rand-KeyGen and Rand-OblivKeyGen) and encryption randomness (via Rand-Enc) is also indistinguishable from the honestly generated distribution.

**Case 4.** This is the case where both the sender $P_i$ and the receiver $P_j$ in $\mathsf{OT}_\ell$ are corrupted. Note that in this case the simulation is perfect, except that the simulator $\mathcal{S}$ may abort in cases where honest parties in a real execution would not abort.

Recall that $\mathcal{S}$ aborts if the output of the receiver in the OT is not equal to the *correct* output value $m_b$, as determined by the parties' input shares $(\alpha_i, \beta_i), (\alpha_j, \beta_j)$. This happens only if the output of the receiver in the OT is equal to $(1-m_b)$. In a real execution, the shares $(\alpha_i, \beta_i), (\alpha_j, \beta_j)$ are unknown to honest parties, and so they will not be able to identify if the receiver $P_j$ receives a valid, but *incorrect* output value, $(1 - m_b)$.[10] We now argue that this event occurs with only negligible probability.

Recall that the parties $P_i, P_j$ are semi-malicious, and thus must follow the protocol correctly, but may choose their randomness arbitrarily. Thus, the only way they can generate an incorrect output $m'$ in the OT is if they are able to force randomness for the NCE key generation and encryption such that the ciphertext $c_b$ corresponding to the correct message $m_b$ decrypts to $(1-m_b)$. However, recall that the semi-malicious parties do not have full control over the randomness used for these tasks, but rather must use the output of $\mathsf{Rand\text{-}Enc}(\sigma_i^{(2)}, \vec{g}_i)$ and $\mathsf{Rand\text{-}KeyGen}(\sigma_j^{(1)}, \sigma_j^{(2)})$, respectively. By Lemma 3.4.13, since $\sigma_j^{(1)}, \sigma_j^{(2)}$, and $\sigma_i^{(1)}$ are each *lossy* seeds for the LTDF, it holds with overwhelming probability over the choice of the CRS messages $M_0, M_1$ that for *any* choice of randomness used by $\mathcal{A}$ (on behalf of $P_i, P_j$), that the corresponding ciphertext-key pair will decrypt to either the correct message $m_b$, or to $\perp$.

This concludes the proof of Lemma 3.5.5.

□

**Lemma 3.5.6** (Output Stage). *The simulator $\mathcal{S}$ correctly simulates the output re-randomization stage. That is, for every set of inputs $\vec{x}$ and auxiliary input $z$,*

$$\left(\mathsf{sim}_{\mathsf{init}}, \{\mathsf{sim}_g\}_{g=1}^{G}, s_{\mathsf{out}}\right) \overset{c}{\cong} \left(\mathsf{sim}_{\mathsf{init}}, \{\mathsf{sim}_g\}_{g=1}^{G}, \mathsf{sim}_{\mathsf{out}}\right).$$

*Proof.* Suppose there exist inputs $\vec{x}$, auxiliary input $z$, and an adversary $\mathcal{B}$ that distinguishes between the above two distributions with probability $\epsilon$. We first consider the case where the party $P_n$, who receives the output, is honest. The case where $P_n$ is adversarial is very similar and is addressed later.

Denote by $\alpha_i$ the share of each party $P_i$ corresponding to the output wire. Let $H$ denote the set of honest parties not including $P_n$. Without loss of generality, assume that the set $H$ consists of parties $P_1, \ldots, P_{|H|}$. Then, we consider a set of inner hybrids $\mathcal{H}_0, \ldots, \mathcal{H}_{|H|}$, where $\mathcal{H}_i$ is described as follows:

[10]Note that if the output of the receiver is $\perp$ (instead of $m_b$), then parties will abort in both the simulation and the real execution.

**Distribution $\mathcal{H}_i$:** This distribution is $\left(\mathsf{sim}_{\mathsf{init}}, \{\mathsf{sim}_g\}_{g=1}^G, \mathsf{sim}_{\mathsf{out}}^i\right)$, where $\mathsf{sim}_{\mathsf{out}}^i$ is generated in the same manner as $s_{\mathsf{out}}$ (in the real execution), except the following differences:

1. For each honest party $P_j$ s.t. $j \in [1, i]$, $P_n$ samples $(e_{j,n}, d_{j,n}^0, d_{j,n}^1, c_{j,n}, r_{E,j,n}^0, r_{E,j,n}^1) \leftarrow \mathsf{NCSim}(1^k)$, and sends $e_{j,n}$ to party $P_j$.
2. For each honest party $P_j$ s.t. $j \in [1, i]$, choose at random $r_{j,1}, \ldots, r_{j,n-1} \leftarrow \{0,1\}$. These will correspond to *all but one* secret share of $P_j$'s output share $\alpha_j$. Let
$$r_{j,n} \triangleq \alpha_j \oplus \left(\bigoplus_{\ell=1}^{n-1} r_{j,\ell}\right),$$
so that
$$\alpha_j = \bigoplus_{\ell=1}^{n} r_{j,\ell}.$$
Note that the (hybrid) simulator may not know $r_{j,n}$ since he may not know $\alpha_j$. However, he could compute $r_{j,n}$ efficiently if he was given $\{v_g\}$ (since $\alpha_j$ itself is an efficiently-computable function of $\{v_g\}$).
3. For each honest party $P_j$ s.t. $j \in [1, i]$, send $c_{j,n}$ (as computed in step 1) to party $P_n$.
4. For the remaining honest parties $P_j$ such that $j \in [i+1, ..., |H|]$, generate secret shares $\{r_{j,\ell}\}$ and ciphertexts $c_{j,\ell} = \mathsf{NCEnc}_{e_{j,\ell}}(r_{j,\ell}; u_{j,\ell})$ honestly.
5. $\mathsf{sim}_{\mathsf{out}}^i = \mathsf{sim}_{1,\mathsf{out}}^i, ..., \mathsf{sim}_{n,\mathsf{out}}^i$, where for each honest party $P_j$ s.t. $j \in [1, i]$, we have:
$$\mathsf{sim}_{j,\mathsf{out}}^i = \left(\{e_{\ell,j}, d_{\ell,j}\}_{\ell \in [n] \setminus \{j\}}, \{e_{j,\ell}\}_{\ell \in [n] \setminus \{j\}}, \{r_{j,\ell}\}_{\ell=1}^n, \{c_{j,\ell}, u_{j,\ell}\}_{\ell \in [n-1] \setminus \{j\}}, (c_{j,n}, r_{E,j,n}^{r_{j,n}}), \{c_{\ell,j}\}_{\ell \neq j}\right),$$
while for party $P_n$, we have:
$$\mathsf{state}_n(\mathsf{output}) = \left(\mathsf{state}_n(G), \{e_{j,n}, d_{j,n}^{r_{j,n}}\}_{j=1}^i, \{e_{j,n}, d_{j,n}\}_{j=i+1}^{n-1}, \{r_{n,\ell}\}_{\ell=1}^n, \{c_{n,\ell}, u_{n,\ell}\}_{\ell \in [n-1]}, \{c_{\ell,n}\}_{\ell \neq n}\right).$$
For each remaining party $P_\ell$, $\mathsf{sim}_{\ell,\mathsf{out}}^i$ is defined in the same manner as in $s_{\ell,\mathsf{out}}$.

This completes the description of the distribution $\mathcal{H}_i$. Note that by definition, we have that $\mathsf{sim}_{\mathsf{out}}^0 = s_{\mathsf{out}}$ and $\mathsf{sim}_{\mathsf{out}}^{|H|} = \mathsf{sim}_{\mathsf{out}}$, and thus
$$\mathcal{H}_0 = \left(\mathsf{sim}_{\mathsf{init}}, \{\mathsf{sim}_g\}_{g=1}^G, s_{\mathsf{out}}\right) \quad \text{and} \quad \mathcal{H}_{|H|} = \left(\mathsf{sim}_{\mathsf{init}}, \{\mathsf{sim}_g\}_{g=1}^G, \mathsf{sim}_{\mathsf{out}}\right).$$

By a standard hybrid argument, we have that $\exists i \in [|H|]$ such that $\mathcal{B}$ distinguishes between the distributions $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$ (and thus the distributions $\mathsf{sim}_{\mathsf{out}}^{i-1}$ and $\mathsf{sim}_{\mathsf{out}}^i$) with probability at least $\frac{\epsilon}{|H|}$. However, note that the only difference between $\mathsf{sim}_{\mathsf{out}}^{i-1}$ and $\mathsf{sim}_{\mathsf{out}}^i$ is that in $\mathsf{sim}_{\mathsf{out}}^i$ the key pair $(e_{i,n}, d_{i,n}^{r_{i,n}})$ and the corresponding ciphertext and randomness pair $(c_{i,n}, r_{E,i,n}^{r_{i,n}})$, were chosen using the NCE simulator, whereas in $\mathsf{sim}_{\mathsf{out}}^{i-1}$ this is done using the standard $\mathsf{NCGen}$ and $\mathsf{NCEnc}$ algorithms. Thus, computational indistinguishability follows immediately from the security of NCE scheme, i.e., the assumption that for every bit $b$ the distribution

$$\{(e, c, d, r_E) : (e, d) \leftarrow \mathsf{NCGen}(1^k), c \leftarrow \mathsf{NCEnc}_e(b; r_E)\}$$

is computationally indistinguishable from the distribution

$$\{(e, c, d^b, r_E^b) : (e, c, d^0, d^1, r_E^0, r_E^1) \leftarrow \mathsf{NCSim}(1^k)\}.$$

Formally, one can use $\mathcal{B}$ to break this indistinguishability with probability $\frac{\epsilon}{|H|}$, by embedding an instance $(e, c, d, r_E)$ in $\mathsf{sim}_{\mathsf{out}}^{i-1}$, which results with the same distribution as $\mathsf{sim}_{\mathsf{out}}^{i-1}$ if $(e, c, d, r_E)$ was generated using $\mathsf{NCGen}$ and $\mathsf{NCEnc}$, and results in the same distribution as $\mathsf{sim}_{\mathsf{out}}^{i}$ if $(e, c, d, r_E)$ was generated using the NCE simulator $\mathsf{NCSim}$. This completes the proof for the case where $P_n$ is honest.

If $P_n$ is malicious, then we can choose an honest party, say $P_1$, and use the same proof idea as above, while replacing the role of $P_n$ with $P_1$. The main difference is that now $P_1$ would need to send his *unknown* re-randomized share to $P_n$. The (hybrid) simulator uses the output $y$ of the trusted party to compute this re-randomized share as follows:

$$r_1' \triangleq y \oplus \left( \bigoplus_{i=1}^{n} r_i' \right),$$

where $r_i'$ is the re-randomized share of party $P_i$ (see protocol description) and simulates party $P_1$ sending $r_1'$ to party $P_n$, who then supposedly outputs $\bigoplus_{i=1}^{n} r_i' = y$. □

Combining Lemmas 3.5.4, 3.5.5, and 3.5.6 implies Theorem 3.5.2, thus concluding the proof of Theorem 3.5.1.

□

## 3.6 From Semi-Malicious to Malicious

In the previous sections, we constructed a protocol that is secure against adaptive auxiliary information in the *semi-malicious* model. In particular, we restricted ourselves to the class of adversaries who may choose their input and random coins arbitrarily, but who otherwise follow the specification of the protocol honestly. Now, we would like to consider the setting in which the adversary is allowed to behave in an *arbitrarily malicious* manner. Towards this end, we give a compiler that, given a protocol $\Pi$ secure in the semi-malicious model, generates a protocol $\Pi'$ secure in the fully malicious model. Our compiler is very similar to the original compiler of [GMW87], but adapted to the adaptive auxiliary information setting.

### 3.6.1 Overview of the Compiler

Similar to [GMW87], the key idea for the compiler is to have each party commit to its input and random coins, and after each step prove in zero knowledge that it is indeed following the protocol.

More specifically, for every message sent in $\Pi$, the same message is sent in $\Pi'$, but is accompanied by a commitment to the "newly" flipped random coins used in the generation of this message (which could be maliciously chosen for adversarial parties) and a ZK proof guaranteeing that the message sent is consistent with the party's input, its "previous" and "newly" flipped random coins,[11] and all the previous messages sent in the protocol.

However, we will need the commitment scheme and ZK proof system to have additional properties. First of all, since we allow adaptive auxiliary information, we will require the commitment scheme and ZK proof system to be secure in this setting. Further, we will need the commitment scheme to be *equivocal* and *extractable*. Equivocation is used to achieve security against adaptive auxiliary information, when the simulator must commit to an unknown value on behalf of an honest party and may need to equivocate to properly simulate the auxiliary information. The extractability requirement guarantees that the simulator can extract the messages committed to by the malicious adversary in $\Pi'$. The reason we need extractability is that our simulator for the underlying protocol $\Pi$ (in the semi-malicious model) is assumed to have access to the adversary's input and random coins in order to properly simulate, and our simulator in the *malicious* model must be able to feed these values to him.

Fortunately, in the CRS model, such an extractable, non-interactive equivocal commitment scheme [FS89, GOS06a, GS08], and a NIZK proof system [GJS11, BCH11] both exist. Using these primitives as described above (and as in [GMW87]), the compiler generates a protocol secure against fully malicious adversaries even in the setting of adaptive auxiliary information. In fact, using the very recently proposed extension of the UC framework for the setting of adaptive auxiliary information [BCH11], we can argue that the compiler described above (implemented with UC secure versions of the required components) achieves the stronger notion of UC security. This allows us to argue that our protocol is secure against adaptive auxiliary information even in the setting of arbitrary composition. For simplicity of exposition, however, we first focus on the construction and proof in the standalone model, and relegate discussion of the UC setting to Section 3.7.3. We also recall the extended UC framework of [BCH11] in Section 3.7.1.

### 3.6.2 The Compiler

We now formally describe the compiler. Our construction will make use of two building blocks:

- An extractable equivocal commitment scheme $(\mathsf{crsGen}_{\mathsf{com}}, \mathsf{Com}, \mathsf{Rec})$ with simulator algorithms $\mathsf{Sim}_{\mathsf{eq}} = (\mathsf{crsSim}_{\mathsf{eq}}, \mathsf{comSim}_{\mathsf{eq}})$ and extraction algorithms $(\mathsf{crsGen}_E, E)$.
- A leakage-resilient NIZK argument system $(K, P, V)$, with simulator $(\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$, for the language

$$\mathcal{L}^i_\Pi = \{((m, \overline{m}), (\mathsf{crs}^{\mathsf{com}}, \mathsf{com}^{\mathsf{input}}, C^{\mathsf{rand}})) \mid \exists (x, R, u^{\mathsf{input}}, U^{\mathsf{rand}}) \text{ s.t. } m = \Pi_i(x, R, \overline{m}),$$
$$\mathsf{com}^{\mathsf{input}} = \mathsf{Com}(\mathsf{crs}^{\mathsf{com}}, x; u^{\mathsf{input}}),\ c^{\mathsf{rand}}_j = \mathsf{Com}(\mathsf{crs}^{\mathsf{com}}, r_j; u_j)\ \forall j\},$$

where $C^{\mathsf{rand}} = \{c^{\mathsf{rand}}_1, ..., c^{\mathsf{rand}}_t\}$, $R = \{r_1, ..., r_t\}$, and $U^{\mathsf{rand}} = \{u_1, ..., u_t\}$.

[11]Observe that commitments to all the "previous" and "newly" flipped random coins must have been provided at some point in the protocol.

That is, $((m, \overline{m}), (\mathsf{crs}^{\mathsf{com}}, \mathsf{com}^{\mathsf{input}}, C^{\mathsf{rand}})) \in \mathcal{L}_{\Pi}^{i}$ if there exists an input $x$ consistent with $\mathsf{com}^{\mathsf{input}}$, and randomness $R$ consistent with the commitments $C^{\mathsf{rand}} = \{c_1^{\mathsf{rand}}, \ldots, c_t^{\mathsf{rand}}\}$, such that $m$ is the correct next message generated by a party $P_i$ in protocol $\Pi$, given $\overline{m}$ as the the transcript of $\Pi$-messages sent so far, and input $x$ and randomness $R$. Here, the witness $(x, R, u^{\mathsf{input}}, U^{\mathsf{rand}})$ contains all secret information that supports this claim: namely, the input $x$, random tape $R$ of $P_i$ up to this point, and the randomness $u^{\mathsf{input}}, U^{\mathsf{rand}}$ that was used to generate these commitments.

Given a protocol $\Pi$, the compiled protocol $\mathsf{Comp}(\Pi)$ is described in Figure 3.2. In the description of $\mathsf{Comp}(\Pi)$, when any party is instructed to abort, he first sends the message "abort" to all parties, and then exits the protocol.

**Remark 3.6.1.** Note that both the extractable, non-interactive equivocal commitments, and the leakage resilient NIZK argument system are in the CRS model. In our compilation, for simplicity, we use a separate CRS for each party; however, we remark that this is not essential. In fact, in our final construction, in the UC setting (Section 3.7.3), the same compilation is actually done with a single CRS.

**Theorem 3.6.2.** *Suppose the protocol $\Pi$ is secure against adaptive auxiliary information in the presence of* semi-malicious *adversaries, and assume the existence of an extractable equivocal commitment scheme and leakage-resilient NIZK argument system as described above. Then $\mathsf{Comp}(\Pi)$ performs the same functionality as $\Pi$ and is secure against adaptive auxiliary information in the presence of* malicious *adversaries.*

*Proof.* Note that $\mathsf{Comp}(\Pi)$ performs the same functionality as $\Pi$ by definition. Now, suppose for contradiction there exists a malicious adversary $\mathcal{A}$, a set of inputs $\vec{x}$, and auxiliary input $z$ (where $\vec{x}$ and $z$ may depend efficiently on the CRS) such that security of $\mathsf{Comp}(\Pi)$ does not hold in the setting of adaptive auxiliary information. We construct a corresponding *semi-malicious* adversary $\mathcal{A}_{\Pi}$ for $\Pi$, which breaks the security of the underlying protocol $\Pi$ with respect to the same inputs $\vec{x}, z$.

We now describe the semi-malicious adversary $\mathcal{A}_{\Pi}$ against $\Pi$, who uses $\mathcal{A}$ as a black box. In order to use $\mathcal{A}$, the adversary $\mathcal{A}_{\Pi}$ must simulate messages sent by honest parties in $\mathsf{Comp}(\Pi)$, and simulate responses to $\mathcal{A}$'s auxiliary information queries. Similar to the simulator in Section 3.5, $\mathcal{A}_{\Pi}$ maintains a simulated version $\mathsf{state}_i$ of the secret state of each honest party $P_i$ in $\mathsf{Comp}(\Pi)$. Although $\mathcal{A}_{\Pi}$ may not know this state entirely (for instance, it includes the secret input of $P_i$), the complete state can be expressed as an efficiently computable function of $P_i$'s input $x_i$ and randomness $R_i$ in $\Pi$. The simulated state will be used to answer the auxiliary infomration queries made by $\mathcal{A}$ (see Leakage Queries description below).

The semi-malicious adversary $\mathcal{A}_{\Pi}$ for $\Pi$:

**CRS Generation.** $\mathcal{A}_{\Pi}$ receives a value $\mathsf{crs}_{\Pi}$ for the protocol $\Pi$. In order to use the malicious adversary $\mathcal{A}$ (who expects a CRS for the *compiled* protocol $\mathsf{Comp}(\Pi)$), he generates the following additional values:

**$\mathsf{Comp}(\Pi)$**

Let $\mathcal{P} = \{P_1, ..., P_n\}$ denote the set of all parties participating in $\Pi$, with inputs $\{x_1, ..., x_n\}$.

1. **CRS Generation:** Sample $\mathsf{crs}_\Pi \leftarrow \mathsf{crsGen}_\Pi(1^k)$ for the semi-malicious protocol $\Pi$. For each party $P_i \in \mathcal{P}$, generate a pair of CRS values $(\mathsf{crs}_i^{\mathsf{com}}, \mathsf{crs}_i^{\mathsf{NIZK}})$, where $\mathsf{crs}_i^{\mathsf{com}} \leftarrow \mathsf{crsGen}_{\mathsf{com}}(1^k)$ is for the extractable equivocal commitment scheme, and $\mathsf{crs}_i^{\mathsf{NIZK}} \leftarrow K(1^k)$ is for the leakage-resilient NIZK proof system. Output $(\mathsf{crs}_\Pi, \{\mathsf{crs}_i^{\mathsf{com}}, \mathsf{crs}_i^{\mathsf{NIZK}}\}_{i\in[n]})$.
2. **Input Commitment:** Each party $P_i$ commits to his input, $\mathsf{com}_i \leftarrow \mathsf{Com}(\mathsf{crs}_i^{\mathsf{com}}, x_i; u_i^{\mathsf{input}})$, stores the randomness $u_i^{\mathsf{input}}$ that was used, and sends $\mathsf{com}_i$ to all parties.
3. **Protocol Execution:** The parties emulate the protocol $\Pi$ as follows.

   Each time a party $P_i$ must send a message in $\Pi$, he performs the following steps.

   (a) Let $\overline{m}$ be the series of $\Pi$-messages that were sent so far in the protocol execution.

   If computing the $\Pi$-message requires generating random coins, $P_i$ samples an appropriate random value $r_{i,t}$, appends $r_{i,t}$ to its list of random coins generated during the protocol
   $$R_i = \{r_{i,1}, ..., r_{i,t}\},$$
   computes a commitment
   $$c_{i,t} \leftarrow \mathsf{Com}(\mathsf{crs}_i^{\mathsf{com}}, r_{i,t}; u_{i,t}),$$
   and appends the randomness $u_{i,t}$ that was used to its list
   $$U_i^{\mathsf{rand}} = \{u_{i,1}, ..., u_{i,t}\}.$$

   (b) In addition to the resulting outgoing message $m$ that $P_i$ sends in $\Pi$, $P_i$ also sends the commitment $c_{i,t}$ and a proof (using the LR-NIZK proof system) that $m$ was computed correctly. That is, $P_i$ sends $(m, c_{i,t}, \pi_{i,t})$, where
   $$\pi_{i,t} \leftarrow P(\mathsf{crs}_i^{\mathsf{NIZK}}, ((m, \overline{m}), (\mathsf{crs}_i^{\mathsf{com}}, \mathsf{com}_i, C_i^{\mathsf{rand}})), (x_i, R_i, u_i^{\mathsf{input}}, U_i^{\mathsf{rand}})).$$

   Each time party $P_i$ receives a message $(m, c, \pi)$ that was sent by $P_j$, he performs the following steps:

   (a) Let $\overline{m}$ be the series of $\Pi$-messages that were received in the protocol execution so far.

   (b) Let $C_j^{\mathsf{rand}}$ be the series of commitments to randomness sent by $P_j$ during the protocol execution so far, including the current value $c$.

   (c) $P_i$ verifies that the proof $\pi$ is correct. That is, he verifies that
   $$1 = V\left(\mathsf{crs}_j^{\mathsf{NIZK}}, \pi, ((m, \overline{m}), (\mathsf{crs}_j^{\mathsf{com}}, \mathsf{com}_j, C_j^{\mathsf{rand}}))\right).$$
   If the condition fails, then $P_i$ aborts. Otherwise, $P_i$ appends $m$ to $\overline{m}$ and proceeds to the next step in $\Pi$.
4. **Output:** Each party $P_i$ outputs as instructed in $\Pi$.

**Figure 3.2:** The compiled protocol $\mathsf{Comp}(\Pi)$.

- For each corrupted party $P_j$, the semi-malicious adversary $\mathcal{A}_\Pi$ honestly generates a CRS value for the NIZK proof system $\mathsf{crs}_j^{\mathsf{NIZK}} \leftarrow K(1^k)$, and uses the extractor algorithm to generate a CRS for the commitment scheme $(\mathsf{crs}_j^{\mathsf{com}}, \mathsf{trap}_j^E) \leftarrow \mathsf{crsGen}_E(1^k)$.

- For each honest party $P_i$, the semi-malicious adversary $\mathcal{A}_\Pi$ generates *simulated* CRS values: $(\mathsf{crs}_i^{\mathsf{NIZK}}, \tau_i) \leftarrow \mathcal{S}_1(1^k)$ and $(\mathsf{crs}_i^{\mathsf{com}}, \mathsf{trap}_i) \leftarrow \mathsf{crsSim}_{\mathsf{eq}}(1^k)$.

$\mathcal{A}_\Pi$ sends $\mathsf{CRS} \triangleq (\mathsf{crs}_\Pi, \{\mathsf{crs}_i^{\mathsf{com}}, \mathsf{crs}_i^{\mathsf{NIZK}}\}_{i\in[n]})$ to $\mathcal{A}$. For each party $P_i$, initialize $\mathsf{state}_i = \mathsf{CRS}$.

**Input Phase.**

1. For each honest party $P_i$ in $\mathsf{Comp}(\Pi)$, $\mathcal{A}_\Pi$ generates a *simulated* commitment

$$(\mathsf{com}_i, \mathsf{dec}_i^0, \mathsf{dec}_i^1) \leftarrow \mathsf{comSim}_{\mathsf{eq}}(\mathsf{crs}_i^{\mathsf{com}}, \mathsf{trap}_i),$$

   which supposedly is a commitment to $P_i$'s (unknown) input $x_i$, and gives $\mathsf{com}_i$ to $\mathcal{A}$.

   For each honest party $P_i$, update $\mathsf{state}_i \leftarrow (\mathsf{state}_i, x_i, \mathsf{dec}_i^{x_i}, \{\mathsf{com}_j\}_{j\notin M})$. Note that $\mathcal{A}_\Pi$ may not know these values directly, but can compute them efficiently given the input values $\{x_i\}_{i\notin M}$.

2. $\mathcal{A}_\Pi$ uses $\mathcal{A}$ to choose input values $x_j$ for *corrupt parties* $P_j$, as follows. Once $\mathcal{A}$ receives $\{\mathsf{com}_i\}_{i\notin M}$, he responds with a commitment $\mathsf{com}_j$ for each malicious party $P_j$. From these commitments, the semi-malicious adversary $\mathcal{A}_\Pi$ extracts the underlying inputs

$$x_j = E(\mathsf{crs}_j^{\mathsf{com}}, \mathsf{com}_j, \mathsf{trap}_j^E),$$

   using the trapdoors he generated during the CRS generation phase. $\mathcal{A}_\Pi$ records these values $\{x_j\}_{j\in M}$ as his inputs to $\Pi$. Note that these inputs $x_j$ may depend on the CRS (as sampled above), but that they can be efficiently computed as a function of $\mathsf{crs}_\Pi$.

   For each honest party $P_i$, update $\mathsf{state}_i \leftarrow (\mathsf{state}_i, \{\mathsf{com}_j\}_{j\in M})$.

**Protocol Execution.** The semi-malicious adversary $\mathcal{A}_\Pi$ acts as a middle man between the honest parties in the execution of $\Pi$ and the malicious adversary $\mathcal{A}$ (in a simulated execution of $\mathsf{Comp}(\Pi)$), translating between $\Pi$-messages and $\mathsf{Comp}(\Pi)$-messages as follows.

- **Each time an honest party $P_i$ sends a message $m$ in $\Pi$:** The semi-malicious adversary $\mathcal{A}_\Pi$ forwards the message $m$ along to $\mathcal{A}$, together with:
  - A *simulated* commitment $c_{i,t}$, where $(c_{i,t}, d_{i,t}^0, d_{i,t}^1) \leftarrow \mathsf{comSim}_{\mathsf{eq}}(\mathsf{crs}_i^{\mathsf{com}}, \mathsf{trap}_i)$, which supposedly commits to the randomness used by $P_i$ to generate the message $m$, and
  - A *simulated* NIZK proof $\pi_{i,t} \leftarrow \mathcal{S}_2(\mathsf{crs}_i^{\mathsf{NIZK}}, \tau_i, ((m, \overline{m}), (\mathsf{crs}_i^{\mathsf{com}}, \mathsf{com}_i, C_i^{\mathsf{rand}})); u_{\mathsf{NIZK}})$ that the message $m$ was computed correctly.

- **Each time $\mathcal{A}$ sends a message $(m, c, \pi)$ on behalf of $P_j$ in $\mathsf{Comp}(\Pi)$:** The semi-malicious adversary $\mathcal{A}_\Pi$ does the following.

1. $\mathcal{A}_\Pi$ begins by verifying that the proof $\pi$ is correct. That is, he appends $c$ to the vector $C_j^{\mathsf{rand}}$ and verifies whether
$$1 = V(\mathsf{crs}_j^{\mathsf{NIZK}}, \pi, ((m, \overline{m}), (\mathsf{crs}_j^{\mathsf{com}}, \mathsf{com}_j^{\mathsf{input}}, C_j^{\mathsf{rand}}))).$$
If the condition fails, then $\mathcal{A}_\Pi$ aborts.
2. If $\pi$ verifies correctly, then $\mathcal{A}_\Pi$ extracts the randomness $r_{j,t}$ from $\mathcal{A}$'s commitment,
$$r_{j,t} \leftarrow E(\mathsf{crs}_j^{\mathsf{com}}, c, \mathsf{trap}_j^E),$$
and sends the message $m$ in protocol $\Pi$ on behalf of party $P_j$. The semi-malicious adversary $\mathcal{A}_\Pi$ appends $r_{j,t}$ into its special random tape $R_j = (r_{j,1}, ..., r_{j,t})$ as the randomness that he used to generate $P_j$'s message $m$.

**Auxiliary Information Queries** In order to answer auxiliary information queries made by $\mathcal{A}$ in the (simulated) execution of $\mathsf{Comp}(\Pi)$, the $\Pi$-adversary $\mathcal{A}_\Pi$ maintains a simulated version of the secret state of honest parties in $\mathsf{Comp}(\Pi)$ as a function of the unknown inputs $\{x_i\}_{i \notin M}$ and random tapes $\{R_i\}_{i \notin M}$ of the honest parties in $\Pi$. Answers to $\mathcal{A}$'s auxiliary information queries can then be computed directly via an auxiliary information query of $\mathcal{A}_\Pi$ in the execution of $\Pi$. Note that this procedure is analogous to the method of simulating auxiliary information that was used by the simulator $\mathcal{S}$ in the semi-malicious setting.

We have already described how $\mathcal{A}_\Pi$ simulates the secret state of honest parties $\{\mathsf{state}_i\}_{i \notin M}$ during the CRS generation and input phases. It now remains to describe how $\{\mathsf{state}_i\}_{i \notin M}$ is updated during the course of the protocol.

For each message $(m, c_{j,t}, \pi_{j,t})$ sent by a *corrupted* party $P_j$, adversary $\mathcal{A}_\Pi$ updates $\mathsf{state}_i \leftarrow (\mathsf{state}_i, (m, c_{j,t}, \pi_{j,t}))$ for each $i \notin M$.

Each time $\mathcal{A}_\Pi$ simulates an *honest* party $P_i$ sending a message triple $(m, c_{i,t}, \pi_{i,t})$, the adversary $\mathcal{A}_\Pi$ must update $\mathsf{state}_i$ to include the randomness that was supposedly used to generate these values. In particular, $\mathcal{A}_\Pi$ must simulate:

1. Randomness $r_{i,t}$ used by $P_i$ to generate the $\Pi$-message $m$,
2. Randomness used to generate the commitment $c_{i,t}$ to $r_{i,t}$, and
3. Randomness $\mathsf{rand}_{\mathsf{NIZK}}$ used to generate the NIZK proof $\pi_{i,t}$ that $m$ was computed correctly.

This simulation is done as follows.

For (a) and (b), $\mathcal{A}_\Pi$ simply updates the secret state of $P_i$ as $\mathsf{state}_i \leftarrow (\mathsf{state}_i, r_{i,t}, d_{i,t}^{r_{i,t}})$, where $r_{i,t}$ is the randomness that was used by the honest party $P_i$ to generate the message $m$ in $\Pi$. Since $r_{i,t}$ is contained as part of $P_i$'s random tape $R_i$ up to this point, the values $r_{i,t}$ and $d_{i,t}^{r_{i,t}}$ are efficiently computable functions of $R_i$.

Randomness $\mathsf{rand}_{\mathsf{NIZK}}$ of type (c) can be simulated consistently as a function of the corresponding witness $w_i$, by the leakage resilience of the NIZK argument system, where $w_i = (x_i, R_i, u_i^{\mathsf{input}}, U_i^{\mathsf{rand}})$ consists of the input and all past randomness used by $P_i$ in $\mathsf{Comp}(\Pi)$. From above, we have that the simulated version of $w_i$ is efficiently computable given $x_i$ and $R_i$: namely,
$$w_i = (x_i, R_i, \mathsf{dec}_i^{x_i}, (d_{i,1}^{r_{i,1}}, ..., d_{i,t}^{r_{i,t}})).$$

Now, define $L_{\mathsf{id}}$ to be the function

$$L_{\mathsf{id}}(w_i, \mathsf{rand}_{\mathsf{NIZK}}) \triangleq \mathsf{rand}_{\mathsf{NIZK}}.$$

Then, treating $L_{\mathsf{id}}$ as a "leakage function" on $(w_i, \mathsf{rand}_{\mathsf{NIZK}})$, define $L'$ to be the corresponding modified leakage function, as per Definition 3.2.12:

$$L' \leftarrow \mathcal{S}_3(\mathsf{crs}_i^{\mathsf{NIZK}}, \tau_i, ((m, \overline{m}), (\mathsf{crs}_i^{\mathsf{com}}, \mathsf{com}_i^{\mathsf{input}}, C_i^{\mathsf{rand}})), u_{\mathsf{NIZK}}, L_{\mathsf{id}}). \tag{3.2}$$

Note that $\tau_i$ is the trapdoor that was sampled together with $\mathsf{crs}_i^{\mathsf{NIZK}}$,

$$((m, \overline{m}), (\mathsf{crs}_i^{\mathsf{com}}, \mathsf{com}_i^{\mathsf{input}}, C_i^{\mathsf{rand}}))$$

is the relevant instance, and $u_{\mathsf{NIZK}}$ is the randomness that was used by the simulator $\mathcal{S}_2$ to generate the simulated proof $\pi'_{i,t}$. Then, by the leakage resilience of the NIZK argument system (see Definition 3.2.12), we have that

$$L'(w_i) \overset{c}{\cong} L_{\mathsf{id}}(w_i, \mathsf{rand}_{\mathsf{NIZK}}),$$

and thus $L'(w_i) \overset{c}{\cong} \mathsf{rand}_{\mathsf{NIZK}}$. So, in order to simulate the randomness $\mathsf{rand}_{\mathsf{NIZK}}$ of type (c), the adversary $\mathcal{A}_\Pi$ updates $\mathsf{state}_i$ as

$$\mathsf{state}_i \leftarrow (\mathsf{state}_i, L'(w_i)).$$

We remark that $L'$ and $L'(w_i)$ are efficiently computable given $x_i, R_i$.

We have now described how the adversary $\mathcal{A}_\Pi$ updates the simulated secret state of honest parties at every stage of the simulated protocol $\mathsf{Comp}(\Pi)$. At any point when $\mathcal{A}_\Pi$ receives an auxiliary information query $L$ from $\mathcal{A}$, the semi-malicious adversary $\mathcal{A}_\Pi$ sends the auxiliary information query $L''$ to his own oracle in protocol $\Pi$, where $L''$ is defined to first plug in the unknowns $\{x_i\}_{i\notin M}, \{R_i\}_{i\notin M}$ that appear in the current state $\triangleq \{\mathsf{state}_i\}_{i\notin M}$, and then compute $L(\mathsf{state})$.

**Output** The semi-malicious adversary $\mathcal{A}_\Pi$ outputs in $\Pi$ whatever $\mathcal{A}$ outputs (in the simulated execution of $\mathsf{Comp}(\Pi)$).

This concludes the description of the $\Pi$-adversary $\mathcal{A}_\Pi$.

We now prove Theorem 3.6.2 in two steps, using $\mathcal{A}_\Pi$ constructed above. First, we show that the output distribution $\mathsf{REAL}_{\mathcal{A},M}^{\mathsf{Comp}(\Pi)}(1^k, \vec{x}, z)$ generated by running $\mathcal{A}$ in $\mathsf{Comp}(\Pi)$ is computationally indistinguishable from $\mathsf{REAL}_{\mathcal{A}_\Pi,M}^{\Pi}(1^k, \vec{x}, z)$, generated by running $\mathcal{A}_\Pi$ in $\Pi$ (we refer the reader to Section 3.3 for the definition of these distributions). Then, we show that (with overwhelming probability) the adversary $\mathcal{A}_\Pi$ is, in fact, semi-malicious. With these two lemmas, together with the semi-malicious security of $\Pi$, the theorem will follow.

We begin by proving the first of these statements.

**Lemma 3.6.3.** *Let $\mathcal{A}$ be the malicious adversary for protocol Comp$(\Pi)$, and $\mathcal{A}_\Pi$ the corresponding adversary for protocol $\Pi$, described above. Then for every set of inputs $\vec{x}$ and auxiliary input $z$,*

$$\mathsf{REAL}_{\mathcal{A},M}^{\mathit{Comp}(\Pi)}(1^k, \vec{x}, z) \overset{c}{\cong} \mathsf{REAL}_{\mathcal{A}_\Pi,M}^{\Pi}(1^k, \vec{x}, z).$$

*Proof.* We prove the lemma via a sequence of intermediate hybrid experiments, beginning with the real-world execution of $\mathsf{Comp}(\Pi)$ with the malicious adversary $\mathcal{A}$, and ending with the adversary $\mathcal{A}_\Pi$ in an execution of $\Pi$.

**Hybrid 0** The malicious adversary $\mathcal{A}$ interacts with honest parties in an execution of $\mathsf{Comp}(\Pi)$.

**Hybrid 1** Similar to Hybrid 0, with the following changes for each honest party $P_i$:

- Party $P_i$'s CRS value $\mathsf{crs}_i^{\mathsf{NIZK}}$ for the NIZK argument system is sampled using the simulator algorithm, $(\mathsf{crs}_i^{\mathsf{NIZK}}, \tau_i) \leftarrow \mathcal{S}_1(1^k)$.
- For each message $(m, c_{i,t}, \pi_{i,t})$ sent by an honest party $P_i$, the accompanying NIZK proof $\pi_{i,t}$ that $m$ was computed correctly is replaced by a *simulated* proof,

$$\pi'_{i,t} \leftarrow \mathcal{S}_2(\mathsf{crs}_i^{\mathsf{NIZK}}, \tau_i, ((m, \overline{m}), (\mathsf{crs}_i^{\mathsf{com}}, \mathsf{com}_i^{\mathsf{input}}, C_i^{\mathsf{rand}})); u_{\mathsf{NIZK}}).$$

The secret state of $P_i$ is updated as $\mathsf{state}_i \leftarrow (\mathsf{state}_i, L'(w_i))$, where

$$w_i = (x_i, R_i, u_i^{\mathsf{input}}, U_i^{\mathsf{rand}})$$

is the witness for the NIZK, and $L'$ is the modified "leakage" function given in Equation (3.2). Note that $L'(w_i)$ simulates the randomness allegedly used by $P_i$ to produce the NIZK $\pi'_{i,t}$. (See Auxiliary Information Queries discussion above.)

**Hybrid 2** Similar to Hybrid 1, with the following changes for each honest party $P_i$:

- Party $P_i$'s CRS value $\mathsf{crs}_i^{\mathsf{com}}$ for the commitment scheme is sampled using the simulator algorithm, $(\mathsf{crs}_i^{\mathsf{com}}, \mathsf{trap}_i) \leftarrow \mathsf{crsSim}_{\mathsf{eq}}(1^k)$.
- In the input phase, each commitment $\mathsf{com}_i$ made by $P_i$ to his input is replaced by a *simulated* commitment $\mathsf{com}'_i$, where

$$(\mathsf{com}'_i, \mathsf{dec}_i^0, \mathsf{dec}_i^1) \leftarrow \mathsf{comSim}_{\mathsf{eq}}(\mathsf{crs}_i^{\mathsf{com}}, \mathsf{trap}_i).^{12}$$

The randomness $u_i^{\mathsf{input}}$ allegedly used by $P_i$ to generate $\mathsf{com}'_i$ is simulated by $\mathsf{dec}_i^{x_i}$. In particular, $\mathsf{dec}_i^{x_i}$ is used in the place of $u_i^{\mathsf{input}}$ in the witness $w_i = (x_i, R_i, u_i^{\mathsf{input}}, U_i^{\mathsf{rand}})$ when simulating randomness used to generate NIZKs later in the protocol (see Hybrid 1).
- For each message $(m, c_{i,t}, \pi'_{i,t})$ sent by $P_i$, the commitment $c_{i,t}$ to the randomness $r_{i,t}$ used by $P_i$ to generate the message $m$ is replaced by a *simulated* commitment $c'_{i,t}$, where

$$(c'_{i,t}, d^0_{i,t}, d^1_{i,t}) \leftarrow \mathsf{comSim}_{\mathsf{eq}}(\mathsf{crs}_i^{\mathsf{com}}, \mathsf{trap}_i).^{13}$$

Each value $u_{i,t}$ allegedly used by $P_i$ to generate $c'_{i,t}$ is simulated by $d_{i,t}^{r_{i,t}}$. In particular, $d_{i,t}^{r_{i,t}}$ is used in the place of $u_{i,t}$ in $w_i$ when simulating randomness used to generate NIZKs (see Hybrid 1).

[12] Recall that $\mathsf{com}'_i, \mathsf{dec}_i^0, \mathsf{dec}_i^1$ are *vectors* of commitment/decommitment information, as described in Remark 4.4.9.
[13] Recall that $c'_{i,t}, d^0_{i,t}, d^1_{i,t}$ are *vectors* of commitment/decommitment information, as described in Remark 4.4.9.

Note that $\mathsf{dec}_i^{x_i}, d_{i,t}^{r_{i,t}}$ are efficiently computable functions of $P_i$'s input $x_i$ and the randomness $R_i = \{r_{i,1}, ..., r_{i,t}\}$ used by $P_i$ to compute his $\Pi$-messages.

**Hybrid 3** Same as Hybrid 2, except that for each *corrupted* party $P_j$, the CRS value for the commitment scheme is sampled using the extractor algorithm, $(\mathsf{crs}_j^{\mathsf{com}}, \mathsf{trap}_j^E) \leftarrow \mathsf{crsGen}_E(1^k)$.

**Hybrid 4** The adversary $\mathcal{A}_\Pi$ corresponding to $\mathcal{A}$ interacts with honest parties in protocol $\Pi$.

For each Hybrid $i$, we denote by

$$\mathsf{Hybrid}^i_{\mathcal{A},M}(1^k, \vec{x}, z)$$

the distribution of inputs $\vec{x}, z$ and outputs of all parties running Hybrid $i$, where $\mathcal{A}$ corrupts the set of parties $M$. We now show that the output of each of the above hybrid experiments is computationally indistinguishable (and, in particular, the outputs of Hybrid 3 and 4 are equivalent), thus implying Lemma 3.6.3.

**Claim 3.6.4.** $\mathsf{Hybrid}^0_{\mathcal{A},M}(1^k, \vec{x}, z) \overset{c}{\cong} \mathsf{Hybrid}^1_{\mathcal{A},M}(1^k, \vec{x}, z)$.

*Proof.* The claim follows immediately by the leakage resilience and zero knowledge of the argument system (Definition 3.2.12) together with a standard hybrid argument, where the changes of Hybrid 1 are introduced one honest party at a time. □

**Claim 3.6.5.** $\mathsf{Hybrid}^1_{\mathcal{A},M}(1^k, \vec{x}, z) \overset{c}{\cong} \mathsf{Hybrid}^2_{\mathcal{A},M}(1^k, \vec{x}, z)$.

*Proof.* The claim follows directly by the equivocation property of the equivocal commitment scheme (Corollary 4.4.5) together with a standard hybrid argument, where the changes of Hybrid 2 are introduced one honest party at a time. □

**Claim 3.6.6.** $\mathsf{Hybrid}^2_{\mathcal{A},M}(1^k, \vec{x}, z) \overset{c}{\cong} \mathsf{Hybrid}^3_{\mathcal{A},M}(1^k, \vec{x}, z)$.

*Proof.* The claim holds by the indistinguishability of simulated CRS values generated by the extractor $\mathsf{crsGen}_E$ of the extractable equivocal commitment scheme (Definition 4.4.8), together with a standard hybrid argument. □

**Claim 3.6.7.** $\mathsf{Hybrid}^3_{\mathcal{A},M}(1^k, \vec{x}, z) \equiv \mathsf{Hybrid}^4_{\mathcal{A},M}(1^k, \vec{x}, z)$.

*Proof.* In Hybrid 3, the adversary $\mathcal{A}$ interacts with honest parties in a modified version of protocol $\mathsf{Comp}(\Pi)$, in which the honest parties send their $\Pi$-messages along with *simulated* commitments and NIZKs. In Hybrid 4, $\mathcal{A}_\Pi$ interacts with honest parties in $\Pi$ and emulates $\mathcal{A}$. More specifically, $\mathcal{A}_\Pi$ uses the messages of honest parties in $\Pi$ to simulate the view of $\mathcal{A}$ in Hybrid 3, and then executes $\mathcal{A}$ in his head to determine his next actions.

Since the inputs $\vec{x}, z$ are the same in both hybrids, and the output of honest parties in both hybrids is exactly the output dictated by $\Pi$, it remains to show that $\mathcal{A}_\Pi$ (in Hybrid 4) properly simulates the view of $\mathcal{A}$ in Hybrid 3. We now prove this is the case.

**CRS generation:** $\mathcal{A}_\Pi$ simulates the CRS perfectly.

**Input phase:** In Hybrid 3, each honest party generates and sends a simulated commitment to their input; auxiliary information queries made by $\mathcal{A}$ at this step are answered using the corresponding decommitment information $\mathsf{dec}_i^{x_i}$. In Hybrid 4, $\mathcal{A}_\Pi$ generates simulated commitments and simulates auxiliary information in an identical fashion. Recall that $\mathcal{A}_\Pi$ reproduces auxiliary information on $\mathsf{dec}_i^{x_i}$ by making the corresponding query on $x_i$ to his oracle in $\Pi$. Thus, $\mathcal{A}_\Pi$'s simulation in this phase is perfect.

**Protocol Execution:** Note that $\mathcal{A}_\Pi$'s simulation of $\mathcal{A}$'s view up to this point is perfect. We show this property is maintained throughout the protocol execution by induction.

*Each time $\mathcal{A}$ receives a message $(m, c, \pi)$ from an honest party $P_i$ in Hybrid 3:* Since $P_i$ is honest, he will also send a message $m$ with the same distribution to $\mathcal{A}_\Pi$ in Hybrid 4. Given this message $m$, $\mathcal{A}_\Pi$ in Hybrid 4 simulates a corresponding commitment $c'$ and NIZK $\pi'$, with an identical distribution to the simulated values $(c, \pi)$ generated by $P_i$ in Hybrid 3, and sends the tuple $(m, c', \pi')$ to $\mathcal{A}$. Thus, this simulation is perfect. Further, $\mathcal{A}_\Pi$ simulates $\mathcal{A}$'s auxiliary infomration perfectly, using his own oracle for $\Pi$.

*Each time $\mathcal{A}$ sends a message $(m, c, \pi)$ on behalf of $P_j$ in $\mathsf{Comp}(\Pi)$ in Hybrid 3:* If the proof $\pi$ does not correctly verify, then the honest parties in Hybrid 3 abort, and the protocol concludes. By induction, the simulated message $(m', c', \pi')$ generated by $\mathcal{A}_\Pi$ in Hybrid 4 is identically distributed to the one sent by $\mathcal{A}$ in Hybrid 3. In this case, the proof $\pi'$ will also not verify properly, and so $\mathcal{A}_\Pi$ simulates the honest parties aborting. Otherwise, in both hybrids the honest parties receive $\Pi$-message $m$ and proceed to the next step.

**Each time an auxiliary information query is made during the protocol execution:** $\mathcal{A}_\Pi$ simulates responses to $\mathcal{A}$'s auxiliary information queries exactly as they are answered in Hybrid 3.

Thus, $\mathcal{A}_\Pi$ perfectly simulates the view of $\mathcal{A}$ in Hybrid 3, and the claim holds.

□

We now bring the hybrids together. Recall that

$$\mathsf{REAL}^{\mathsf{Comp}(\Pi)}_{\mathcal{A},M}(1^k, \vec{x}, z) \equiv \mathsf{Hybrid}^0_{\mathcal{A},M}(1^k, \vec{x}, z), \text{ and}$$
$$\mathsf{REAL}^{\Pi}_{\mathcal{A}_\Pi,M}(1^k, \vec{x}, z) \equiv \mathsf{Hybrid}^4_{\mathcal{A},M}(1^k, \vec{x}, z).$$

Hence, Lemma 3.6.3 follows from Claims 3.6.4-3.6.7.

□

We now prove that $\mathcal{A}_\Pi$ is a semi-malicious $\Pi$-adversary for the same inputs $\vec{x}, z$. To do so, we must prove that at each step of the protocol, the message sent by $\mathcal{A}_\Pi$ is the correct message as dictated by $\Pi$, given the current protocol transcript, and the choice of inputs $\{x'_j\}_{j \in M}$ and randomness contained in $\mathcal{A}_\Pi$'s special tapes.

**Lemma 3.6.8.** *Let $\mathcal{A}$ be s malicious adversary for* Comp$(\Pi)$ *in the adaptive auxiliary information model. Then for the set of inputs $\vec{x}$ and auxiliary input $z$ in* Comp$(\Pi)$*, the corresponding adversary $\mathcal{A}_\Pi$ (as constructed above) is a* semi-malicious *adversary for $\Pi$ with the same inputs $\vec{x}$ and auxiliary input $z$.*

*Proof.* We must prove that $\mathcal{A}_\Pi$ follows the protocol $\Pi$ *honestly*, given his chosen inputs and randomness. We do so by a sequence of intermediate hybrid experiments.

**Hybrid 4** The adversary $\mathcal{A}_\Pi$ interacts with honest parties in an execution of $\Pi$ (i.e., the same as Hybrid 4 from the previous lemma).

**Hybrid 5** The same as Hybrid 4, except that the experiment immediately ends in fail at any point that a corrupted party $P_j$ sends a message $(m, c_{j,t}, \pi_{j,t})$ such that the proof $\pi_{j,t}$ verifies properly but the message $m$ is *not* the correct next message in the protocol $\Pi$ (for some consistent input and randomness). Namely,

$$1 = V(\mathsf{crs}_j^{\mathsf{NIZK}}, \pi, ((m, \overline{m}), (\mathsf{crs}_j^{\mathsf{com}}, \mathsf{com}_j, C_j^{\mathsf{rand}}))),$$

but

$$((m, \overline{m}), (\mathsf{crs}_j^{\mathsf{com}}, \mathsf{com}_j, C_j^{\mathsf{rand}})) \notin \mathcal{L}_\Pi^j.$$

**Hybrid 6** Same as Hybrid 5, except that the experiment also ends in fail at any point that a corrupted party $P_j$ sends a message $(m, c_{j,t}, \pi_{j,t})$ such that the proof $\pi_{j,t}$ verifies properly, but the message $m$ is not the correct next message in $\Pi$ given the *extracted* input and randomness from $P_j$'s commitments. That is, $\pi$ verifies correctly (as above), but

$$m \neq \Pi_j(x_j, R_j, \overline{m}),$$

where $\overline{m}$ is the current transcript of $\Pi$-messages up to this point and

$$\begin{cases} x_j \triangleq E(\mathsf{crs}_j^{\mathsf{com}}, \mathsf{com}_j, \mathsf{trap}_j^E), \\ R_j \triangleq (r_{j,1}, ..., r_{j,t}) \text{ for } r_{j,t'} \triangleq E(\mathsf{crs}_j^{\mathsf{com}}, c_{j,t'}, \mathsf{trap}_j^E). \end{cases}$$

Note that Hybrid 6 restricts the adversary to semi-malicious behavior in $\Pi$ for the set of inputs and randomness extracted from his commitments. We now show that, since $\mathcal{A}$ is computationally bounded (and thus so is $\mathcal{A}_\Pi$), the outputs of Hybrids 4-6 are statistically close.

**Claim 3.6.9.** $\mathsf{Hybrid}^4_{\mathcal{A}_\Pi,M}(1^k, \vec{x}, z) \overset{s}{\cong} \mathsf{Hybrid}^5_{\mathcal{A}_\Pi,M}(1^k, \vec{x}, z)$.

*Proof.* The claim follows directly from the *soundness* of the leakage-resilient NIZK argument system, together with a union bound. Namely, the probability that a computationally bounded adversary can generate a proof for a false statement is negligible, and so Hybrid 5 will only end in fail with negligible probability. □

**Claim 3.6.10.** $\mathsf{Hybrid}^5_{\mathcal{A}_\Pi,M}(1^k, \vec{x}, z) \overset{s}{\cong} \mathsf{Hybrid}^6_{\mathcal{A}_\Pi,M}(1^k, \vec{x}, z)$.

*Proof.* Clearly whenever Hybrid 5 ends in fail, Hybrid 6 also fails. It remains to show that the probability of causing fail in Hybrid 6 but not in Hybrid 5 is negligible. Recall that this event occurs when $\mathcal{A}$ sends a message $(m, c_{j,t}, \pi_{j,t})$ such that $\pi_{j,t}$ correctly verifies and the corresponding instance

$$((m, \overline{m}), (\mathsf{crs}_j^{\mathsf{com}}, \mathsf{com}_j, C_j^{\mathsf{rand}}))$$

*does* lie in the language $\mathcal{L}_\Pi^j$ for some witness, but this witness is not correctly extracted from $\mathcal{A}$'s commitments by the extraction algorithm $E$. But, by the extractability property of the commitment scheme (Definition 4.4.8), the probability of this event is negligible.

□

Together, Claims 3.6.9 and 3.6.10 imply that with overwhelming probability, the Π-adversary $\mathcal{A}_\Pi$ is a *semi-malicious* adversary with respect to the set of inputs and randomness extracted from his commitments, and thus Lemma 3.6.8 holds.

□

Theorem 3.6.2 follows.

□

## 3.7 Universally Composable MPC Secure Against Adaptive Auxiliary Information

In the previous sections, we constructed an MPC protocol that is secure against adaptive auxiliary information in the *stand-alone* setting. In particular, we did this by first restricting ourselves to *semi-malicious* adversaries. Then, in Section 3.6, we gave a compiler that, given a protocol Π secure in the semi-malicious model, generates a protocol Π′ secure in the fully malicious model. This compiler is very similar to the original compiler of [GMW87], but adapted to the adaptive auxiliary information setting. In this section, we give a similar compiler that yields a protocol secure in the more demanding UC setting.

**Outline.** Our compiler in the UC setting directly builds upon the compiler of [CLOS02] using techniques demonstrated in the stand-alone case (in Section 3.6). We start by describing the steps involved in the construction of a UC secure protocol [CLOS02].

**Step 1** The first step is to construct a commit-and-prove functionality that allows parties to commit to their secret values (in a hiding way) and to prove (in zero-knowledge) some theorem about these secret values to other parties.

**Step 2** The commit-and-prove functionality is extended to a multi commit-and-prove functionality. This functionality allows one party to commit and send proofs to multiple parties at the same time.

**Step 3** Finally, a protocol secure in the semi-honest setting is compiled with the above described multi commit-and-prove functionality, to yield a protocol secure in the fully malicious setting. The argument involves use of the UC composition theorem [Can00].

We need to adapt the above steps to the setting of adaptive auxiliary informatiomn. To this end, we proceed as follows.

**Step 1** We start by giving a protocol that securely realizes the "leaky" commit-and-prove functionality (i.e., secure against adaptive auxiliary information), denoted by $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$. This follows directly from [GJS11] which in turn relies on [GOS06a].

**Step 2** Next, we will extend the commit-and-prove functionality to a multi commit-and-prove functionality. The non-interactive nature of the protocol realizing $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$ allows us immediately to extend it to realize the leaky multi commit-and-prove functionality, denoted by $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$.

**Step 3** Finally, we take a protocol secure in the semi-malicious setting and give a compiler that uses a multi commit-and-prove functionality and generates a UC secure protocol in the setting of adaptive auxiliary information. This step is very similar to [CLOS02]. However, the argument uses (in a black-box manner) the UC composition theorem in the setting of adaptive auxiliary information [BCH11].

**Roadmap.** In Section 3.7.1 we present the UC framework in the setting of adaptive auxiliary information. In Section 3.7.2 we provide ideal specifications for some basic tasks. Additionally we provide details on constructing a protocol to securely realize $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$ given a protocol that securely realizes $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$. This takes care of Step 2 given Step 1. In Section 3.7.3 we present our compiler, which is needed for Step 3. Finally, in Section 3.7.4 we construct a protocol that realizes $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$ functionality, which is needed for Step 1.

## 3.7.1 UC Framework with Adaptive Auxiliary Information

Bitansky et. al. [BCH11] initiated the study of UC framework in the setting of adaptive auxiliary information. In this section, as in the stand-alone case, we consider adversaries in the UC setting that can obtain joint adaptive auxiliary information on the secret states of all parties participating in a session. We start by giving a short introduction to the UC framework and its extension to our setting. We do not give full details and present the framework in a way that eases the understanding of our results. For full details see [Can00, BCH11].

Following [GMR89, Gol01], a protocol is represented as an interactive Turing machine (ITM), which represents the program to be run within each participant.

**Basic UC.** The basic UC framework considers realization of an "ideal specification" $\mathcal{F}$ by a "real implementation" $\pi$. The requirement is that for any "real world attacker" $\mathcal{A}$ against the implementation $\pi$ there exists an "ideal world attacker" $\mathcal{S}$ (also referred to as the simulator) against the specification $\mathcal{F}$, such that an "environment" $\mathcal{Z}$ that interacts with $\mathcal{S}$ and $\mathcal{F}$ has essentially the same view as in an interaction with $\mathcal{A}$ and $\pi$. In other words, the adversarial effect on any (potentially large) system $\mathcal{Z}$ executed with $\pi$ cannot be worse than the adversarial effect on the same system where $\pi$ is replaced by $\mathcal{F}$.

The basic UC model lets the environment $\mathcal{Z}$ determine the inputs to the parties running the protocol and see the outputs generated by these parties, and also allows free communication between the environment and the adversary. Additionally, the adversary typically has full control over the communication between parties, and the ability to "corrupt" parties. Corruption is modeled as just another interface available to the adversary, where it can send a message "you are corrupted" to any party. In case of "passive" (or "semi-honest") corruption, the party responds to this message by handing its entire input state to the adversary. To model "active" (or "malicious") corruption, the party also changes the program that it is running from then on. Note that in this paper we restrict ourselves to "static" corruptions, i.e., the adversary can corrupt parties only before the protocol starts.

**UC with Adaptive Auxiliary Information.** A natural approach to modeling adaptive auxiliary information within the UC framework (both in the "ideal specification" and the "real implementation") is by allowing the adversary to send a "leak L" messages to a *process*[14] (where $L$ is some function), and have that process reply with $L(s)$ where $s$ is its internal state.

**The leakage aggregator.** The modeling approach described above limits the adversary in obtaining auxiliary information on individual processes only. However, in real life auxiliary information might actually be more complicated and involve multiple processes. To account for this property of real life auxiliary information, a new "global entity" called the *leakage aggregator* is introduced. The aggregator $\mathcal{G}$ can access the entire state of all the components in the system. The adversary is allowed to make auxiliary information queries to sets of processes and not just individual ones. An auxiliary information query specifies a function $L$ and a set of processes $P = \{p_1, \ldots, p_t\}$. This query is forwarded to the aggregator, who evaluates $L(s_1, \ldots, s_t)$ (where $s_1, \ldots, s_t$ are the states of the processes $p_1, \ldots, p_t$, respectively) and returns the result to the adversary.

We now discuss the above ideas a bit more formally. In the setting of auxiliary information we allow the adversary $\mathcal{A}$ to perform *auxiliary information* instructions on the joint local states of multiple processes executing within potentially different participants. An auxiliary information request (by $\mathcal{A}$) is assumed to specify a function $L$, represented by a circuit, and a set of processes. As a consequence a new ITM called an aggregator $\mathcal{G}$ is invoked. The aggregator obtains the states of the processes on which the auxiliary information is requested, applies $L$ to these states, and returns the result to $\mathcal{A}$.

In this paper we consider adversaries that are able to obtain auxiliary information on the entire joint state of all processes corresponding to some specified sessions. Therefore in our setting, we assume that the adversary specifies "session identifiers" $\text{SID}_1, \ldots, \text{SID}_t$, and leakage function is applied to the internal states of all the honest processes that correspond to the sessions $\text{SID}_1, \ldots, \text{SID}_t$.

More formally, an auxiliary information operation is handled as follows: First the adversary sends a query $(\text{LEAK}, L, \text{SID}_1, \ldots, \text{SID}_t)$ to $\mathcal{G}$, where $L$ is the auxiliary information function and $\text{SID}_1, \ldots, \text{SID}_t$ are the sessions in which auxiliary information is being leaked. Then $\mathcal{G}$ obtains the states of all honest processes that correspond to each of these sessions, applies $L$ to these states and

[14]Recall that a protocol is represented as an interactive Turing machine (ITM), which represents the program to be run within each participant. Each execution of a protocol is referred to as a *session*. Consider a specific participant taking part in a specific session. The program executing in the specific participant taking part in the specific session is referred to as a *process*.

returns the result to $\mathcal{A}$. Finally, $\mathcal{G}$ reports the output length of the function $L$ to all the processes whose state is included in the evaluation, and reports the session identifiers and the output length to the environment.

**Remark.** In [BCH11], it is assumed that processes are tagged with "party identifiers" PID, and joint auxiliary information is allowed from all the processes that have the same PID. This models a real world setting in which information that depends on the entire state of a physical device (including all the processes that are currently running on it), is leaked.

**UC emulation with auxiliry information.** Protocol emulation is defined just as in the standard UC framework. More specifically, we require that for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish between an interaction with $\mathcal{A}$ and the implementation protocol, or an interaction with $\mathcal{S}$ and the ideal specification. However, unlike the standard UC setting, the composition theorem in the setting of auxiliary information holds only if the simulator is *leakage-oblivious*, a notion introduced in [BCH11].

Loosely speaking a simulator is said to be leakage-oblivious if the ideal world auxiliary information obtained by the simulator is used *only* for the purposes of simulating auxiliary information queries of the real adversary. More formally, we require that the simulator $\mathcal{S}$ has a special subroutine $\widetilde{\mathcal{S}}$ for handling auxiliary information queries. Whenever $\mathcal{S}$ receives from the environment a request to apply an auxiliary information function $L$ for processes $p_1, \ldots, p_t$, $\widetilde{\mathcal{S}}$ is invoked to produce a "state translation circuit" $T$. The circuit $T$ is meant to transform the internal states of processes in the specification protocols into its states in the implementation protocols. Once $T$ is produced, the aggregator is given the composed auxiliary information circuit $L \circ T$. When the result is returned, it is forwarded directly to the environment and $\mathcal{S}$ returns to its state prior to the leakage event. This simulator is referred to as a *leakage-oblivious simulator.*

**Definition 3.7.1** (Strong UC-emulation [BCH11]). Let $\pi$ be real protocol in the auxiliary information model and $\phi$ be an ideal functionality, and let $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ and $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the view of the environment $\mathcal{Z}$ when interacting with $\mathcal{S}$ and $\phi$, or with $\mathcal{A}$ and $\pi$ (as formally defined in [BCH11]). Then $\pi$ strongly UC-emulates $\phi$ against adaptive auxiliary information if there exists a leakage-oblivious simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

Let $\pi$ be an implementation and $\mathcal{F}$ be a specification. Also let $\rho = \rho[\pi]$ be a protocol that includes subroutine calls to $\pi$. Below we denote by $\rho^{\pi}$ the system where the subroutine calls to $\pi$ are actually processed by $\pi$, and we denote by $\rho^{\mathcal{F}}$ the system where these subroutine calls are processed by $\mathcal{F}$.

Next, we recall the extended UC composition theorem, proved in [BCH11].

**Theorem 3.7.2** (UC-composition with adaptive auxiliary information [BCH11]). *Let $\rho, \pi, \mathcal{F}$ be protocols as above, all modular up to auxiliary information, such that $\pi$ UC-emulates $\mathcal{F}$ with a leakage-oblivious simulator. Then $\rho^{\pi}$ UC-emulates $\rho^{\mathcal{F}}$. Furthermore, it does so with a leakage-oblivious simulator.*

Here, a protocol is said to be "modular up to auxiliary information" if it only interacts with its caller, its subroutines, the adversary and the aggregator. Note that the UC composition theorem in [Can00] relies on all the processes being "modular," namely a process can only interact with its caller, its subroutines and the adversary.

### 3.7.2 Basic UC Functionalities

In this subsection we will present definitions of some of the basic UC functionalities that we use in our final compilation. We follow the notation of [BCH11] and denote ideal functionalities that allow for auxiliary information by $\mathcal{F}^{+\mathsf{lk}}$.

#### Common Random String Model.

In the common reference string (CRS) model [CF01, CLOS02], all parties in the system obtain from a trusted party a common reference string, which is sampled according to a pre-specified distribution $\mathcal{D}$. The common reference string is referred to as the *CRS*. In the UC framework, this is modeled by an ideal functionality $\mathcal{F}_{\mathsf{CRS}}$ that samples a string $\rho$ from the pre-specified distribution $\mathcal{D}$ and sets $\rho$ as the CRS. $\mathcal{F}_{\mathsf{CRS}}$ is described in Figure 3.3.

**Functionality $\mathcal{F}_{\mathsf{CRS}}$ parameterized by distribution $\mathcal{D}$**

1. Upon activation with session id sid proceed as follows. Sample $\rho = \mathcal{D}(r)$, where $r$ denotes uniform random coins, and send $(\mathsf{crs}, \mathsf{sid}, \rho)$ to the adversary.
2. Upon receiving $(\mathsf{crs}, \mathsf{sid})$ from some party send $(\mathsf{crs}, \mathsf{sid}, \rho)$ to that party.

**Figure 3.3:** The Common Reference String Functionality.

**Functionality $\mathcal{F}_{\mathsf{BC}}$**

$\mathcal{F}_{\mathsf{BC}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary.

1. Upon receiving a message $(\mathsf{broadcast}, \mathsf{sid}, P_i, \mathcal{P}, x)$ from $P_i$, where $\mathcal{P}$ is a set of parties, send $(\mathsf{broadcast}, \mathsf{sid}, P_i, \mathcal{P}, x)$ to all parties in $\mathcal{P}$ and the adversary, and halt.

**Figure 3.4:** The ideal broadcast functionality $\mathcal{F}_{\mathsf{BC}}$ [CLOS02].

#### Authenticated Broadcast

In order to obtain our result, we assume that each set of parties who engage in a protocol execution has access to an *authenticated* broadcast channel. We next recall the definition for this functionality as defined in [CLOS02]. This functionality is modeled by the ideal broadcast functionality, denoted $\mathcal{F}_{\mathsf{BC}}$, and defined in Figure 3.4.

Goldwasser and Lindell [GL02] showed that the $\mathcal{F}_{\mathsf{BC}}$ functionality can be UC realized for any number of corrupted parties assuming *ideal authenticated channels* (without any setup assumption). Since the parties don't have a private state in this functionality, and since the entire message $(\mathsf{broadcast}, \mathsf{sid}, P_i, \mathcal{P}, x)$ is public to all the parties including the adversary, any protocol that securely realizes the $\mathcal{F}_{\mathsf{BC}}$ functionality, is secure even in the setting of adaptive auxiliary information.

## UC Leaky Commit-and-Prove functionality

**Functionality $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$**

$\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$, parameterized by a value $k$ and a relation $\mathcal{R}$, proceeds as follows:

1. Upon receiving an input $(\mathsf{commit}, \mathsf{sid}, \mathsf{input}, P, V, w)$ proceed as follows. Ignore the message if this is not the first message received (corresponding to session id sid) or $P$ and $V$ do not match the recorded values for session id sid. Else, append the value $w$ (where $w \in \{0,1\}^*$) to the list $\overline{w}$, record $P$ and $V$ and send $|w|$ to the adversary. (Initially, the list $\overline{w}$ is empty.) Once the adversary acknowledges, output $(\mathsf{receipt}, \mathsf{sid}, \mathsf{input}, P, V)$ to $V$.

   On input $(\mathsf{commit}, \mathsf{sid}, \mathsf{random}, P, V, r)$, perform the same steps with $r$ (instead of $w$), and append $r$ to the list $\overline{r}$ (which is initially empty).

2. Upon receiving an input $(\mathsf{prove}, \mathsf{sid}, P, V, x)$ (ignore the message if $P$ and $V$ do not match the recorded values for session id sid) from party $P$, check whether $\mathcal{R}(x, (\overline{w}, \overline{r}))$. If so, then send $(\mathsf{verified}, \mathsf{sid}, P, V, x)$ to the adversary, and once the adversary allows, send the same message to $V$.

3. **Auxiliary Information:** Given a message $(\mathsf{leak}, P)$, give the aggregator the ideal state $(\overline{w}, \overline{r})$. Continue behaving as above.

**Figure 3.5:** The leaky commit-and-prove functionality $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$.

In this section we define the ideal leaky commit-and-prove functionality. Informally this functionality allows parties to commit to a sequence of values and later prove properties about the committed values. The functionality remains secure even in the setting where an adversary can learn auxiliary information about the committed values (in the sense as considered in this paper).

The functionality allows for multiple concurrent two-party interactions, where each interaction consists of two phases: a $\mathsf{commit}$ phase, where the receiver of the commitment obtains a "commitment" to an unknown value, and a $\mathsf{prove}$ phase, in which the receiver, on the senders request, obtains a "receipt" for the fact that the committed values satisfy the specified relation $\mathcal{R}$. The security guarantees are: (a) The committed values remain secret. (b) After the commit phase, the committer cannot change the actual value that he committed to. (c) Finally, the committer cannot provide a "receipt" for a false theorem. Furthermore, correctness requires that a committer and a receiver that follow the protocol can successfully commit and prove true theorems about the committed values. The functionality $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$ is defined formally in Figure 3.5. The ideal state of the sender consists of the "committed" values, while the ideal state of the receiver is empty.

We provide a construction of $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$ in Section 3.7.4. Our construction is very similar to the adaptive UC NIZK construction from [GOS06a, GJS11]. We stress that this construction is *non-interactive*, a property that is crucial to extending the construction to the multi receiver/verifier setting. We note that in the adaptive UC NIZK construction of [GOS06a, GJS11], each NIZK proof consists of a commit-and-proof, which makes it easily adaptable to our setting. Since we do not use the construction of [GOS06a] as a black-box (our functionality is different than an adaptive NIZK functionality), for the sake of completeness we provide the construction in Section 3.7.4. We follow techniques from [GJS11, BCH11] in order to argue security against adaptive auxiliary information.

**Proposition 3.7.3.** *Under the decisional linear assumption in the setting of bilinear groups there exists a (non-interactive) protocol that UC realizes $\mathcal{F}_{CP}^{+lk}$ with a leakage-oblivious simulator in the $\mathcal{F}_{CRS}$-hybrid model.*

**Functionality $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$**

$\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$ running with parties $P_1, \ldots, P_n$ and an adversary, and parameterized by a value $k$ and a relation $\mathcal{R}$, proceeds as follows:

1. Upon receiving an input $(\mathsf{commit}, \mathsf{sid}, \mathsf{input}, P, \mathcal{V}, w)$ proceed as follows. Ignore the message if this is not the first message received (corresponding to session id sid) or $P$ and $\mathcal{V}$ do not match the recorded values for session id sid. Else append the value $w$ (where $w \in \{0,1\}^*$) to the list $\overline{w}$ (which is initially empty), record $P$ and $\mathcal{V}$ and send $|w|$ to the adversary. Once the adversary acknowledges, output $(\mathsf{receipt}, \mathsf{sid}, \mathsf{input}, P, \mathcal{V})$ to all parties in $\mathcal{V}$. However, if a commit message has previously been received, then first check that the recorded set of parties is $\mathcal{V}$ and that the sender is $P$.

   On input $(\mathsf{commit}, \mathsf{sid}, \mathsf{random}, P, \mathcal{V}, r)$, perform the same steps with $r$ (instead of $w$), and append $r$ to the list $\overline{r}$ (which is initially empty).

2. Upon receiving an input $(\mathsf{prove}, \mathsf{sid}, P, \mathcal{V}, x)$ (ignore the message if $P$ and $\mathcal{V}$ do not match the recorded values for session id sid) from party $P$ ($\mathcal{V}$ is the set of intended verifiers), check whether $\mathcal{R}(x, (\overline{w}, \overline{r}))$. If so then send $(\mathsf{verified}, \mathsf{sid}, P, \mathcal{V}, x)$ to the adversary, and once it allows send the same message to every party in $\mathcal{V}$.

3. **Auxiliary Information:** Given a message $(\mathsf{leak}, P)$, give the aggregator the ideal state $(\overline{w}, \overline{r})$. Continue behaving as above.

**Figure 3.6:** The leaky multi commit-and-prove functionality $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$.

## UC Leaky Multi Commit-and-Prove functionality

In this section, we extend the commit-and-prove functionality to a *multi* commit-and-prove functionality. This extension allows a single party to generate commitments/proofs to many receivers/verifiers. The formal definition is provided in Figure 3.6.

The key observation (as made in [CLOS02] in the context of commitments) in realizing the $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$ functionality is that the protocol that UC realizes the functionality $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$ is *non-interactive.* Therefore, this extension is obtained by simply having the committer/prover broadcast the commitment/proof string of the protocol to all the participating parties using the broadcast channel. We refer the reader to [CLOS02] for the complete proof. The commitment/proof string is broadcasted using the $\mathcal{F}_{\mathsf{BC}}$ functionality which ensures that only one message is broadcast using a given session identifier. This ensures that all the parties receive the same commitment/proof for any session identifier $\mathsf{sid}$.

**Proposition 3.7.4.** *Assuming that there exists a (non-interactive) protocol that UC realizes $\mathcal{F}_{CP}^{+lk}$ with a leakage-oblivious simulator, then we can construct a protocol that UC realizes $\mathcal{F}_{CP}^{+lk,1:M}$ with a leakage-oblivious simulator in the $(\mathcal{F}_{CP}^{+lk}, \mathcal{F}_{BC})$-hybrid model.*

## 3.7.3 The Compiler

In this section we describe our compiler. Parts of this section have been taken verbatim from [CLOS02]. Our compiler, given a multi-party protocol $\Pi$ that is secure against adaptive auxiliary information in the semi-malicious model, generates a "functionally equivalent" protocol $\mathsf{Comp}(\Pi)$ that is universally composable and secure against adaptive auxiliary information in the malicious model. Our compiler is constructed in the $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$-hybrid model. A description of our compiler is given in Figure 3.7. We note that each party has to commit and prove statements to all other parties during the protocol execution. In order to do this, each party $P_i$ uses a separate invocation of $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$, with the session ID $\mathsf{sid}_i$. The protocol should make sure that these session ID's are unique as long as the session ID of the current copy of $\mathsf{Comp}(\Pi)$ is unique. This can be done by setting $\mathsf{sid}_i = \mathsf{sid} \circ i$ where $\mathsf{sid}$ is the session ID of the current copy of $\mathsf{Comp}(\Pi)$. Since in our case the semi-malicious adversary is allowed to choose its random coins on the fly, commitments to all the randomness cannot be generated at the beginning of the protocol. This is the main deviation of our compiler from the multi-party secure computation compiler of [CLOS02].

**Remark.** We stress that in this section we consider reactive functionalities that have multiple stages which are separately activated. We stress that each activation of $\Pi$ corresponds to one stage of execution of the protocol. This is done in order to deal with reactive functionalities.

**Comp($\Pi$)**

Let $\mathcal{P}$ denote the set of all parties participating in $\Pi$. Party $P_i$ proceeds as follows (the code for all the other parties is analogous):

1. **Activation due to new input:** When activated with input $(\mathsf{sid}, x)$, party $P_i$ proceeds as follows.
   - (a) *Input commitment:* $P_i$ sends $(\mathsf{commit}, \mathsf{sid}_i, \mathsf{input}, P_i, \mathcal{P}, x)$ to $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$ which adds $x$ to the list of inputs $\overline{x_i}$ (this list is initially empty and contains $P_i$'s inputs from all the previous activations of $\Pi$). (At this point all other parties $P_j \in \mathcal{P}$ receive the message $(\mathsf{receipt}, \mathsf{sid}_i, \mathsf{input}, P_i, \mathcal{P})$ from $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$. $P_i$ then proceeds to the next step.)
   - (b) *Protocol computation:* Let $\overline{m}$ be the series of $\Pi$-messages that were broadcast in all the activations of $\Pi$ until now ($\overline{m}$ is initially empty). $P_i$ runs the code of $\Pi$ on its input list $\overline{x_i}$, messages $\overline{m}$, random tape $\overline{r_i}$ and new random coins $r$. If $\Pi$ instructs $P_i$ to broadcast a message, $P_i$ proceeds to Steps 1c and 1d.
   - (c) *Randomness commitment:* $P_i$ sends $(\mathsf{commit}, \mathsf{sid}_i, \mathsf{random}, P_i, \mathcal{P}, r)$ to $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$ which adds $r$ to its list of random coins $\overline{r_i}$ (this list is initially empty and contains $P_i$'s random coins from all the previous activations of $\Pi$). (At this point all other parties $P_j \in \mathcal{P}$ receive the message $(\mathsf{receipt}, \mathsf{sid}_i, \mathsf{random}, P_i, \mathcal{P})$ from $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$. $P_i$ then proceeds to the next step (Step 1d).)
   - (d) *Outgoing message transmission:* For each outgoing message $m$ that $P_i$ sends in $\Pi$, $P_i$ sends $(\mathsf{prove}, \mathsf{sid}_i, P_i, \mathcal{P}, (m, \overline{m}))$ to $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$ with the relation $\mathcal{R}_\Pi$ defined as follows:

     $$\mathcal{R}_\Pi = \{((m, \overline{m}), (\overline{x_i}, \overline{r_i})) \mid m = \Pi(\overline{x_i}, \overline{r_i}, \overline{m})\}$$

     In other words, $P_i$ proves that $m$ is the correct next message generated by $\Pi$ when the input sequence is $\overline{x_i}$, the random tape is $\overline{r_i}$, and the series of broadcast $\Pi$-messages equals $\overline{m}$. (Recall that all elements of $\overline{r_i}$ and $\overline{x_i}$ were committed to by $P_i$ in the past, using commit activations of $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$ with identifier $\mathsf{sid}_i$.)
2. **Activation due to incoming message:** Upon receiving a message $(\mathsf{verified}, \mathsf{sid}_j, P_j, \mathcal{P}, (m, \overline{m}))$ that is sent by $P_j$, party $P_i$ verifies that the following condition holds:
   - $\overline{m}$ equals the series of $\Pi$-messages that were broadcast in all the activations until now. ($P_i$ knows these messages because all parties see all messages sent.)

   If the condition fails, then $P_i$ ignores the messages. Otherwise, $P_i$ appends $m$ to $\overline{m}$ and proceeds as in Steps 1b, 1c and 1d above.
3. **Output:** Whenever $\Pi$ generates an output value, $\mathsf{Comp}(\Pi)$ generates the same output value.

**Figure 3.7:** The compiled protocol Comp($\Pi$).

**Proposition 3.7.5** (Multi-party protocol Compiler). *Let $\Pi$ be a multi-party protocol and let Comp($\Pi$) be the protocol obtained by applying the compiler of Figure 3.7 to $\Pi$. Then, for every* malicious *adversary $\mathcal{A}$ that interacts with Comp($\Pi$) in the $\mathcal{F}_{CP}^{+lk,1:M}$- hybrid model, there exists a* semi-malicious *adversary $\mathcal{A}_\Pi$ that interacts with* $\Pi$, *such that for every environment $\mathcal{Z}$,*

$$\mathrm{EXEC}_{\Pi,\mathcal{A}',\mathcal{Z}} \equiv \mathrm{EXEC}^{\mathcal{F}_{CP}^{+\mathsf{lk},1:M}}_{Comp(\Pi),\mathcal{A},\mathcal{Z}}.$$

***Proof (sketch).*** We construct a semi-malicious adversary $\mathcal{A}_\Pi$ from the malicious adversary $\mathcal{A}$. Adversary $\mathcal{A}_\Pi$ runs the protocol $\Pi$ while internally simulating an execution of $\mathsf{Comp}(\Pi)$ for $\mathcal{A}$. The key point is that $\mathcal{A}$ is forced to send all messages via $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$ that verifies their correctness. Thus, essentially, $\mathcal{A}$ must behave in a semi-malicious way, and can be simulated by a truly semi-malicious party $\mathcal{A}_\Pi$. $\mathcal{A}_\Pi$ runs a simulated copy of $\mathcal{A}$, and proceeds as follows.

**Simulating the communication with $\mathcal{Z}$:** The input values received by $\mathcal{A}_\Pi$ from $\mathcal{Z}$ are written on $\mathcal{A}$'s input tape, and the output values of $\mathcal{A}$ are copied to $\mathcal{A}_\Pi$'s own output tape.

**Simulating an activation due to new input:** When the first message of an activation of $\Pi$ is sent, $\mathcal{A}_\Pi$ internally simulates for $\mathcal{A}$ the appropriate stage in $\mathsf{Comp}(\Pi)$. This is done as follows. Let $P_i$ be the activated party with a new input. If $P_i$ is not corrupted, then $\mathcal{A}_\Pi$ internally passes $\mathcal{A}$ the message $(\mathsf{receipt}, \mathsf{sid}_i, \mathsf{input}, P_i, \mathcal{P})$ that $\mathcal{A}$ expects to receive from $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk},1:\mathsf{M}}$ (where $\mathcal{P}$ is the set of all parties participating in $\Pi$). If $P_i$ is corrupted, then $\mathcal{A}_\Pi$ receives a message $(\mathsf{commit}, \mathsf{sid}_i, \mathsf{input}, P_i, \mathcal{P}, x)$ from $\mathcal{A}$ (who controls $P_i$). $\mathcal{A}_\Pi$ adds $x$ to its list $\overline{x_i}$ of inputs received from $P_i$ and passes $\mathcal{A}$ the string $(\mathsf{receipt}, \mathsf{sid}_i \circ i, \mathsf{input}, P_i, \mathcal{P})$. Furthermore, $\mathcal{A}_\Pi$ sets $P_i$'s input tape to equal $x$.

**Dealing with messages sent by honest parties:** If an uncorrupted party $P_i$ sends a message $m$ in $\Pi$ to a corrupted party (controlled by $\mathcal{A}_\Pi$), then $\mathcal{A}_\Pi$ prepares simulated messages of $\mathsf{Comp}(\Pi)$ to give to $\mathcal{A}$. Specifically, $\mathcal{A}_\Pi$ passes $\mathcal{A}$ the messages $(\mathsf{receipt}, \mathsf{sid}_i \circ i, \mathsf{random}, P_i, \mathcal{P})$ and $(\mathsf{verified}, \mathsf{sid}_i, P_i, \mathcal{P}, (m, \overline{m}))$

**Dealing with messages sent by corrupted parties:** When $\mathcal{A}$ sends a $\mathsf{Comp}(\Pi)$-message from a corrupted party, $\mathcal{A}_\Pi$ translates this to the appropriate message in $\Pi$. That is, $\mathcal{A}_\Pi$ obtains the messages $(\mathsf{commit}, \mathsf{sid}_i, \mathsf{random}, P_i, \mathcal{P}, r)$ and $(\mathsf{prove}, \mathsf{sid}_i, P_i, \mathcal{P}, (m, \overline{m}))$ from $\mathcal{A}$, in the name of a corrupted party $P_i$. It then adds $r$ to its list $\overline{r_i}$. Then, $\mathcal{A}_\Pi$ checks that the series of broadcasted $\Pi$-messages is indeed $\overline{m}$. $\mathcal{A}_\Pi$ also checks that $m = \Pi(\overline{x_i}, \overline{r_i}, \overline{m})$. If yes, then it delivers the message $m$ by writing it on the semi-malicious party $P_i$'s outgoing communication tape in $\Pi$ and adds $m$ to the list $\overline{m}$. Otherwise, $\mathcal{A}_\Pi$ does nothing.

**Dealing with auxiliary information:** When the simulated $\mathcal{A}$ makes an auxiliary information query on all the past inputs and the random tapes of honest parties in $\Pi$, then $\mathcal{A}_\Pi$ externally makes a auxiliary information query on the inputs and the random tapes of honest parties. Note that honest parties have no additional internal state in $\mathsf{Comp}(\Pi)$ besides what it had in $\Pi$.

We claim that $\mathcal{Z}$'s view of an interaction with $\mathcal{A}_\Pi$ and $\Pi$ is distributed identically to its view of an interaction with $\mathcal{A}$ and $\mathsf{Comp}(\Pi)$. This follows in a way very similar to the proof of compilation as in [CLOS02] and so we skip the details. We note that the primary difference between their setting and ours is the ability of the adversary to request auxiliary information. But, since the internal secret state of parties are the same in $\mathsf{Comp}(\Pi)$ and $\Pi$, such auxiliary information can be simulated by simply forwarding the request. □

### 3.7.4 Realizing $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$

The functionality $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$ can be directly UC-realized using the UC NIZK construction of Groth, Ostrovsky and Sahai [GOS06a]. $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$ has some syntactic differences from their construction, and for the sake of completeness we recall the construction and proof here. Parts of this section have been taken verbatim from [GOS06a].

#### Tools

We will use the following cryptographic tools.

**Definition 3.7.6** (Encryption with pseudorandom ciphertexts). A public-key cryptosystem $(K_{\text{pseudo}}, E, D)$ has pseudorandom ciphertexts of length $\ell_E(k)$ if for all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} &\Pr\left[(pk, \mathrm{DK}) \leftarrow K_{\text{pseudo}}(1^k) : \mathcal{A}^{E_{pk}(\cdot)}(pk) = 1\right] \\ \approx\ &\Pr\left[(pk, \mathrm{DK}) \leftarrow K_{\text{pseudo}}(1^k) : \mathcal{A}^{R_{pk}(\cdot)}(pk) = 1\right], \end{aligned} \tag{3.3}$$

where $R_{pk}(m)$ runs $c \leftarrow \{0,1\}^{\ell_E(k)}$ and every time returns a fresh $c$. We require that the cryptosystem has errorless decryption.

Pseudorandom cryptosystems of length $\ell_E(k)$ can be constructed from any trapdoor permutation over domain $\{0,1\}^{\ell_E(k)-1}$, as we can use the Goldreich-Levin hard-core bit [GL89] of a trapdoor permutation to make a one-time pad. Trapdoor permutations over $\{0,1\}^{\ell_E(k)-1}$ can, in turn, be constructed from the RSA assumption assuming $\ell_E(k)$ is large enough [CFGN96], or from other special number theoretic assumptions as described in [GOS06a].

**Tag-based simulation-sound trapdoor commitment** A tag-based commitment scheme has four algorithms, $(K_{\text{tag-com}}, \text{commit}, \text{Tcom}, \text{Topen})$. The key generation algorithm $K_{\text{tag-com}}$ produces a commitment key $ck$ as well as a trapdoor key $\mathrm{TK}$. There is a commitment algorithm that takes as input the commitment key $ck$, a message $m$, and any tag $\mathtt{tag}$, and outputs a commitment $c = \text{commit}_{ck}(m, \mathtt{tag}; r)$. To open a commitment $c$ with tag $\mathtt{tag}$ we reveal $m$ and the randomness $r$. Anybody can now verify $c = \text{commit}_{ck}(m, \mathtt{tag}; r)$. As usual, the commitment scheme must be both hiding and binding.

In addition to these two algorithms, there is also a pair of trapdoor algorithms Tcom, Topen that allow us to create an equivocal commitment and later open this commitment to any value we prefer. We create an equivocal commitment and an equivocation key as $(c, \mathrm{EK}) \leftarrow \text{Tcom}_{\mathrm{TK}}(\mathtt{tag})$. Later we can open it to any message $m$ as $r \leftarrow \text{Topen}_{\mathrm{EK}}(c, m, \mathtt{tag})$, such that $c = \text{commit}_{ck}(m, \mathtt{tag}; r)$. We require that equivocal commitments and openings are indistinguishable from real openings. For all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} &\Pr\left[(ck, \mathrm{TK}) \leftarrow K_{\text{tag-com}}(1^k) : \mathcal{A}^{\mathcal{R}(\cdot,\cdot)}(ck) = 1\right] \\ \approx\ &\Pr\left[(ck, \mathrm{TK}) \leftarrow K_{\text{tag-com}}(1^k) : \mathcal{A}^{\mathcal{O}(\cdot,\cdot)}(ck) = 1\right], \end{aligned} \tag{3.4}$$

where $\mathcal{R}(m, \mathtt{tag})$ returns a randomly selected randomizer and $\mathcal{O}(m, \mathtt{tag})$ computes $(c, \text{EK}) \leftarrow \text{Tcom}_{\text{TK}}(m, \mathtt{tag})$; $r \leftarrow \text{Topen}_{\text{EK}}(c, m, \mathtt{tag})$ and returns $r$. Both oracles ignore tags that have already been submitted once.

The tag-based simulation-soundness property means that a commitment using tag remains binding even if we have made equivocations for commitments using different tags. For all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned}\Pr\Big[&(ck, \text{TK}) \leftarrow K(1^k); (c, \mathtt{tag}, m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(ck) : \mathtt{tag} \notin Q \text{ and} \\ &c = \text{commit}_{ck}(m_0, \mathtt{tag}; r_0) = \text{commit}_{ck}(m_1, \mathtt{tag}; r_1) \text{ and } m_0 \neq m_1\Big] \approx 0,\end{aligned} \tag{3.5}$$

where $\mathcal{O}(\text{commit}, \mathtt{tag})$ computes $(c, \text{EK}) \leftarrow \text{Tcom}_{\text{TK}}(\mathtt{tag})$, returns $c$ and stores $(c, \mathtt{tag}, \text{EK})$, and $\mathcal{O}(\text{open}, c, m, \mathtt{tag})$ returns $r \leftarrow \text{Topen}_{ck}(\text{EK}, c, m, \mathtt{tag})$ if $(c, \mathtt{tag}, \text{EK})$ has been stored, and where $Q$ is the list of tags for which equivocal commitments have been made by $\mathcal{O}$.

The term tag-based simulation-sound commitment comes from Garay, MacKenzie and Yang [GMY06], while the definition presented here is from MacKenzie and Yang [MY04]. The latter paper offers a construction based on one-way functions.

## Construction

We now bring these tools together to realize the leaky commit-and-prove functionality $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$.

Groth, Ostrovsky and Sahai [GOS06a] provide a construction of UC NIZKs with honest prover state reconstruction. As shown in [GJS11], this construction can be proven to be secure against adaptive auxiliary information. In their protocol, the prover commits to the witness using a "special" commitment scheme and proves by a leakage-resilient NIZK proof system that the committed values satisfy the required relation. The properties required by the "special" commitment scheme are that: (a) it is equivocal, i.e., it has the property that the simulator can "open" a commitment it makes to any desired value later; (b) at the same time, an adversarial sender cannot "cheat" in these commitments (i.e., open it to a value different from the one originally committed); and (c) further, the simulator can in fact *extract* the values that an adversary commits to in its commitments. The first two properties are satisfied by a tag-based simulation-sound trapdoor commitment scheme, as described in the previous section. Finally in order to achieve extractability, these commitments are additionally accompanied by appropriate encryptions of the randomness used to commit, where the encryption scheme has pseudorandom ciphertexts. Appropriately generated encryptions *suffice* for achieving extractability, while still maintaining the hiding property of the commitments.

In order to implement the commit-and-prove functionality, we decouple the generation of the commitments and the proof. In other words, the prover does not need to provide all the commitments and the proofs together in one shot. The prover can provide commitments under the special commitment scheme to "appropriate values" and later provide NIZK proofs about relationships among the committed values. The correctness, soundness and zero-knowledge properties for our protocol follow in a way identical to the [GOS06a] proof. We are only left to argue that our new protocol is secure against adaptive auxiliary information.

Observe that the simulator can equivocate on the committed values to any value of its choice and, as in [GJS11], this equivocation ability is used only in answering the auxiliary information queries. In particular, the equivocation ability is used in the auxiliary information queries in order to open the commitments to a value that is unknown to the simulator. However, this value itself is *fixed* (at the time commitment is made). Therefore, all auxiliary information queries will use the same fixed value in generating responses for the auxiliary information queries and in effect use the same opening for the committed value. Furthermore, since the NIZK proof system is also leakage resilient, the auxiliary information queries on the NIZK proof system can be reduced to auxiliary information queries on the witnesses of these proof statements, which essentially correspond to the "openings" of commitments that the simulator provides. The complete proof follows in a way very similar to the proof in [GJS11, GOS06a].

We now describe he construction of the protocol. Let $(K, P, V)$ be a LR-NIZK (Definition 3.2.12). Let $(K_{\text{pseudo}}, E, D)$ be an encryption scheme with pseudorandom ciphertexts, and let $(K_{\text{tag-com}}, \text{commit}, \text{Tcom}, \text{Topen})$ be a tag-based simulation-sound trapdoor commitment scheme. The resulting protocol can be seen in Figure 3.8. We have modified the description of the protocol to match our definition of $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$. We use the notation from Section 3.7.4.

**Theorem 3.7.7.** *The protocol $\phi$ described in Figure 3.8 UC-realizes the functionality $\mathcal{F}_{CP}^{+lk}$ in the $\mathcal{F}_{CRS}$-hybrid model.*

**Protocol $\phi$: Realization of $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$ Functionality**

- **Common reference string generation:**
  1. $(ck, \text{TK}) \leftarrow K_{\text{tag-com}}(1^k)$
  2. $(pk, \text{DK}) \leftarrow K_{\text{pseudo}}(1^k)$
  3. $(\sigma, \tau) \leftarrow S_1(1^k)$
  4. Return $\Sigma = (ck, pk, \sigma)$
- **Commit to input** On input $(\mathsf{commit}, \mathsf{sid}, \mathsf{input}, P, V, w)$: Let $\ell$ be the length of $w$.
  1. Compute an extractable commitment to $w$ using the technique of [GOS06a]. That is,
     (a) For $i = 1$ to $\ell$ select $r_i$ at random and let $c_i := \text{commit}_{ck}(w_i, \mathtt{tag}; r_i)$, where $\mathtt{tag} := \mathsf{sid}$.
     (b) For $i = 1$ to $\ell$ select $R_{w_i}$ at random and set $c_{i,w_i} := E_{pk}(r_i; R_{w_i})$ and choose $c_{i,1-w_i}$ as a random string.
     (c) Let the *overall commitment* $c := (c_1, c_{1,0}, c_{1,1}, \ldots, c_\ell, c_{\ell,0}, c_{\ell,1})$.
  2. Add $w$ to the list $\overline{w}$ and $c$ to the list $\overline{c}$. (The lists $c, w$ will initially be empty.) And send $(\mathsf{sid}, \mathsf{input}, P, V, c)$ to $V$. On receiving $(\mathsf{sid}, \mathsf{input}, P, V, c)$ $V$ outputs $(\mathsf{receipt}, \mathsf{sid}, \mathsf{input}, P, V)$.
- **Commit to randomness.** On input $(\mathsf{commit}, \mathsf{sid}, \mathsf{random}, P, V, r)$: Perform the same steps as above. In this case, generate the overall commitment $d$ to $r$, add $r$ to the list $\overline{r}$, add $d$ to the list $\overline{d}$ and send $(\mathsf{sid}, \mathsf{random}, P, V, d)$ to $V$. On receiving $(\mathsf{sid}, \mathsf{random}, P, V, d)$ $V$ outputs $(\mathsf{receipt}, \mathsf{sid}, \mathsf{random}, P, V)$.
- **Prove.** On input $(\mathsf{prove}, \mathsf{sid}, P, V, x)$ such that $\mathcal{R}(x, (\overline{w}, \overline{r}))$ do
  1. Create an LR-NIZK proof $\pi$ for the statement that there exists $\overline{w}$ and $\overline{r}$ and randomness such that $\overline{c}$ and $\overline{d}$ are lists of overall commitments to elements of $\overline{w}$ and $\overline{r}$, respectively, and $\mathcal{R}(x, (\overline{w}, \overline{r}))$.
  2. Generate $\Pi = (\mathtt{tag}, x, \pi)$ and send $(\mathsf{proof}, \mathsf{sid}, P, V, \Pi)$ to $V$
- **Verify.** On input $(\mathsf{proof}, \mathsf{sid}, P, V, \Pi)$:
  1. Parse $\Pi = (\mathtt{tag}, x, \pi)$
  2. Verify the LR-NIZK proof $\pi$ with respect to $x$, $\mathtt{tag}$ and $\overline{c}$ and $\overline{d}$.
  3. Output $(\mathsf{verified}, \mathsf{sid}, P, V, x)$ if the check works out.

**Figure 3.8:** Protocol $\phi$ for realizing $\mathcal{F}_{\mathsf{CP}}^{+\mathsf{lk}}$.

# Chapter 4

# Multi-Party Computation Secure Against Continual Memory Leakage

In this work, we construct a multi-party computation (MPC) protocol that is secure even if a malicious adversary, in addition to corrupting 1-$\epsilon$ fraction of all parties (for an arbitrarily small constant $\epsilon > 0$), can *leak* information about the secret state of each honest party. This leakage can be *continuous* for an unbounded number of executions of the MPC protocol, computing different functions on the same or different set of inputs. We assume a (necessary) "leak-free" pre-processing stage.

In contrast to the results discussed in Chapter 3, we achieve leakage resilience *without weakening the security guarantee* of classical MPC. Namely, an adversary who is given leakage on honest parties' states, is guaranteed to learn nothing beyond the input and output values of corrupted parties. This differs from previous works on leakage in the multi-party protocol setting, which weaken the security notion, and only guarantee that a protocol which leaks $\ell$ bits about the parties' secret states, yields at most $\ell$ bits of leakage on the parties private *inputs*. For some functions, such as voting, such leakage can be detrimental.

Our result relies on standard cryptographic assumptions, and our security parameter is polynomially related to the number of parties.

We begin by providing a technical overview of our construction in Section 4.1. The preliminaries appear in Section 4.2. A formal description of our adversarial model and security definition appears in Section 4.3. In Section 4.4, we describe how to achieve *weakly* leakage-resilient MPC for randomized functionalities within this model (building atop the weakly leakage-resilient MPC protocol for deterministic functionalities from Chapter 3). Our final leakage-resilient MPC protocol construction appears in Section 4.5, and the proof of security appears in Section 4.6.

## 4.1 Overview of Our Construction

In this section, we provide an overview of our leakage-resilient MPC protocol construction.

**Starting point: "Only Computation Leaks" (OCL) Compiler** As discussed in Section 1.3.2, it is known how to convert any circuit into one that is secure in the *only computation leaks* (OCL) model, where one assumes that secret information does not leak when merely stored in memory, but that any information used during a computation may leak. Specifically, Goldwasser and Rothblum [GR12] construct an efficient compiler that takes any circuit (with some secret values hard-wired) and converts it into a leakage-resilient one, consisting of several *modules*, each of which performs a specific sub-computation. The security guarantee is that an adversary, who at any point of time throughout the computation obtains bounded leakage from the "currently active" module, does not learn any more information than having black-box access to the circuit.

In light of this result, a natural first idea towards realizing our goal of constructing leakage-resilient MPC protocols is the following. Let $P_1, \dots, P_n$ denote the set of all parties, and let $U_{\vec{x}}$ be a universal circuit that has the secret input vector $\vec{x}$ of all the parties hard-wired into it, and on input a circuit $f$, outputs $U_{\vec{x}}(f) = f(x)$. Then, very roughly, the (candidate) MPC protocol works as follows. First, in the "leak free" pre-processing phase, apply the OCL compiler of [GR12] on circuit $U_{\vec{x}}$ to obtain a set of modules $\mathsf{Sub}_1, \dots, \mathsf{Sub}_n$ such that on any input $f$, the "compiled" circuit (consisting of $\mathsf{Sub}_1, \dots, \mathsf{Sub}_n$) outputs $U_{\vec{x}}(f) = f(\vec{x})$. Next, in the computation phase, in order to securely compute a function $f$, each party $P_i$ *emulates* the module $\mathsf{Sub}_i$ (such that the computation of $\mathsf{Sub}_i$ is performed by party $P_i$), where the input of $\mathsf{Sub}_1$ is $f$, and the output of $\mathsf{Sub}_n$ is the protocol output $f(\vec{x})$. Finally, in the update phase, the parties update their respective modules by running the update algorithm of the OCL compiler.

Now, assuming that we can reduce (independent) leakage on each party to (independent) leakage on its corresponding module, one may hope that the above MPC protocol achieves the "desired" security properties: in particular, privacy of the inputs that were "encoded" in the pre-processing phase. Unfortunately, as we explain below, this is not the case. Nevertheless, as will be evident from the forthcoming discussion, the above approach serves as a good starting point towards realizing our goal.

**OCL Compiler vs. LR-MPC.** There are two main differences between the setting of leakage-resilient MPC (LR-MPC) and an OCL compiler.

1. The first difference is perhaps best illustrated by the fact that an OCL compiler only guarantees security against an *external* adversary who can obtain leakage from the modules. In contrast, in the setting of LR-MPC, we wish to guarantee security against an *internal* adversary, who may also corrupt a subset of the parties.

   More concretely, recall that the security of the OCL compiler crucially relies on the assumption that an external adversary can only obtain *bounded, independent* leakage on each module. Further, in order for the correctness of the "compiled" circuit to hold, each module must perform its computation as specified. As a result, the above approach, at best, yields an MPC protocol that is secure when *all* the parties are honest (not even semi-honest) but can be leaked upon by an external adversary. Specifically, note that if an internal adversary can corrupt some of the parties, then we can no longer guarantee correctness of computation, and even worse, an adversary may be able to obtain *joint* leakage on multiple modules, and learn the *entire* secret state of modules corresponding to corrupted parties, thus violating both of the above stated requirements.

2. The second difference between the OCL compiler and the leakage-resilient MPC setting is that in the OCL setting, the communication between the modules is assumed to be private (but may be leaked), and leakage is assumed to happen "in order"; i.e., only a module which is currently computing can be leaked upon. On the other hand, in the leakage-resilient MPC setting, the entire communication is to be known to the adversary, and moreover, leakage on any party can happen at any time.

**Emulating Modules via *Weakly* LR-MPC.** Our key idea to circumvent the first problem stated above is to emulate each $\mathsf{Sub}_i$ by a designated *set* of parties $S_i = \{P_{i_1}, \ldots, P_{i_\ell}\}$, instead of a single party $P_i$. More concretely, we secret share $\mathsf{Sub}_i$ between $P_{i_1}, \ldots, P_{i_\ell}$, who then run a specific MPC protocol $\Pi$ to jointly emulate the (functionality of) module $\mathsf{Sub}_i$. Now, note that as long as some portion of the parties in the designated set $S_i$ is honest, the emulation of $\mathsf{Sub}_i$ will be "correct", and if leakage on each honest party is bounded, then we can expect the leakage on the module $\mathsf{Sub}_i$ to be bounded as well. Furthermore, if all of the designated sets $S_i$ for the modules $\mathsf{Sub}_i$ are disjoint (i.e., no party is contained within two different sets), then the leakage on each module will be independent, as required. However, note that since we are in the setting of leakage, in order for the above idea to work, we need the MPC protocol $\Pi$ to satisfy some form of leakage-resilience. Thus, a priori, it seems that we haven't made any progress at all.

Our next crucial observation is that protocol $\Pi$ in fact only needs to a satisfy a *weaker* form of leakage-resilience. Specifically, we only require that leakage on the secret state of each party $P_{i_\ell}$ executing protocol $\Pi$ (to emulate $\mathsf{Sub}_i$) can be *"reduced"* to leakage on the module $\mathsf{Sub}_i$. (We stress that this suffices since the OCL compiler allows bounded leakage on each module.) More generally, this translates to constructing an MPC protocol such that the leakage on the secret states of the honest parties in the real world can be reduced to leakage on the inputs of the honest parties in the ideal world. Fortunately, an MPC protocol satisfying the above (weak) form of leakage-resilience was recently constructed by Boyle et al. [BGJ+13] (as described in Chapter 3).

However, this result cannot be plugged in directly: the protocol of [BGJ+13] is for *deterministic* functionalities (and, in fact, it is impossible to achieve this definition for randomized functionalities if the adversary can attain "joint" leakage on the collective states of all honest parties [CLL+13]). However, in the present adversarial model, where the adversary may only leak *independently* on the secret state of each honest party, security for randomized functionalities can be achieved. In Section 4.4, we construct a form of strong leakage-resilient coin-tossing protocol and prove that combining it with the protocol of [BGJ+13] yields weakly-leakage resilient MPC for randomized functionalities, as required.[1]

**Using an LDS Compiler Instead of an OCL Compiler.** Our key idea to circumvent the second problem stated above is to use an LDS compiler instead of an OCL compiler. The LDS (leaky distributed system) model was introduced in [BCG+11], and it strengthens the OCL model

[1]At this point, an advanced reader may question whether a weakly leakage-resilient MPC protocol (such as the modified protocol of Boyle et al. [BGJ+13]), in conjunction with a leakage-resilient secret sharing scheme, directly yields a leakage-resilient MPC protocol in our model. Unfortunately, this is not the case since the simulator of Boyle et al. requires *joint* leakage on the honest party inputs, even when the real world adversary makes *disjoint* leakage queries on the secret states of honest parties.

in two ways (which are exactly the strengthening we need). First, in the OCL model, the leakage happens is a certain ordering (based on the order of computation). The LDS model strengthens the power of the adversary, by allowing him to leak from the sub-computations in any order he wishes. Moreover, he can leak a bit from $\mathsf{Sub}_i$, then leak a bit from $\mathsf{Sub}_j$, and based on the leakage values, leak again on $\mathsf{Sub}_i$. So, the adversary controls which $\mathsf{Sub}_i$ he wishes to leak from. In addition, in the LDS model, the adversary can view and control the entire communication between the modules. We refer the reader to Section 4.2.5 for details on the LDS compiler.

By using an LDS compiler, as opposed to an OCL compiler, we get around the second problem mentioned above.

**Reducing Number of Parties via FHE.** An important issue that was overlooked in the previous discussion is the following. The only known OCL compiler that does not rely on leak-free hardware [GR12], and thus the only known LDS compiler without leak-free hardware, suffers from the drawback that the number of modules in the "compiled" circuit is linear in the size of the original circuit. As a result, when we apply the LDS compiler on $U_{\vec{x}}$, whose size grows with $|\vec{x}|$, the number of resultant modules is more than the number of parties! Thus, a priori, it is not even clear how to realize the above approach.

In order to resolve the above problem, we make crucial use of fully homomorphic encryption (FHE) in the following manner. Specifically, instead of simply applying the LDS compiler to $U_{\vec{x}}$, we now first compute a key pair $(\mathsf{pk}, \mathsf{sk})$ for an FHE scheme, and then apply the LDS compiler to the decryption circuit $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$ with the secret key sk hardwired. Note that the number of resultant modules is now *independent* of the number of parties. Now, in a non-interactive input phase (that is also "leak-free"), the parties $P_i$ encrypt their respective inputs $x_i$ under the public key pk, and publish the resulting ciphertexts $\hat{x}_i$. Then, whenever the parties wish to compute a functionality $f$ over their inputs, they homomorphically evaluate $\hat{y}_f \triangleq \mathsf{Eval}_{\mathsf{pk}}((\hat{x}_1, ..., \hat{x}_n), f)$, and collectively evaluate the compiled decryption circuit on the value $\hat{y}_f$ in the manner described above.

**Missing Pieces.** Despite our best attempt to cover the main ideas in the above discussion, some important issues still remain undiscussed. For example, it is not immediately clear how to choose the designated sets of parties $S_i$ such that at least one of the parties in each set $S_i$ is honest, and each set $S_i$ is independent. Very roughly, to address this problem, we employ (an adapted version of) the committee election protocol of Feige [Fei99] to divide the parties into several committees, one for each module. Then, by careful choice of parameters, we are able to obtain the desired guarantees. We refer the reader to the technical sections for more details.

## 4.2 Preliminaries

### 4.2.1 Non-Interactive Zero Knowledge

**Definition 4.2.1.** [FLS90, BFM88, BSMP91]: $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{proof}}))$ is an *efficient adaptive NIZK proof system* for a language $L \in \mathsf{NP}$ with witness relation $\mathcal{R}$ if $\mathsf{Gen}, \mathsf{P}, \mathsf{V}, \mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{proof}}$ are all ppt algorithms, and there exists a negligible function $\mu$ such that for all $k$ the following three requirements hold.

- **Completeness**: For all $x, w$ such that $\mathcal{R}(x,w) = 1$, and for all strings $\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$,
$$\mathsf{V}(\mathsf{crs}, x, \mathsf{P}(x, w, \mathsf{crs})) = 1.$$
- **Adaptive Soundness**: For all adversaries $\mathcal{A}$, if $\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$ is sampled uniformly at random, then the probability that $\mathcal{A}(\mathsf{crs})$ will output a pair $(x, \pi)$ such that $x \notin L$ and yet $\mathsf{V}(\mathsf{crs}, x, \pi) = 1$, is at most $\mu(k)$.
- **Adaptive Zero-Knowledge**: For all ppt adversaries $\mathcal{A}$,
$$\left| \Pr[\mathsf{Exp}_{\mathcal{A}}(k) = 1] - \Pr[\mathsf{Exp}_{\mathcal{A}}^{\mathcal{S}}(k) = 1] \right| \leq \mu(k),$$
where the experiment $\mathsf{Exp}_{\mathcal{A}}(k)$ is defined by:
$$\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$$
$$\text{Return } \mathcal{A}^{\mathsf{P}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs})$$
and the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathcal{S}}(k)$ is defined by:
$$(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k)$$
$$\text{Return } \mathcal{A}^{S'(\mathsf{crs},\mathsf{trap},\cdot,\cdot)}(\mathsf{crs}),$$
where $S'(\mathsf{crs}, \mathsf{trap}, x, w) = \mathcal{S}^{\mathsf{proof}}(\mathsf{crs}, \mathsf{trap}, x)$.

We next define the notion of a NIZK proof of knowledge.

**Definition 4.2.2.** Let $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{proof}}))$ be an efficient adaptive NIZK proof system for an NP language $L \in \mathsf{NP}$ with a corresponding NP relation $\mathcal{R}$. We say that $\Pi$ is a *proof-of-knowledge* if there exists a ppt algorithm $E$ such that for every ppt adversary $\mathcal{A}$,
$$\Pr[\mathcal{A}(\mathsf{crs}) = (x, \pi) \text{ and } E(\mathsf{crs}, \mathsf{trap}, x, \pi) = w^* \text{ s.t. } \mathsf{V}(\mathsf{crs}, x, \pi) = 1 \text{ and } (x, w^*) \notin \mathcal{R}] = \mathsf{negl}(k),$$
where the probabilities are over $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k)$, and over the random coin tosses of the extractor algorithm $E$.

**Remark.** There is a standard way to convert any NIZK proof system $\Pi$ to a NIZK proof-of-knowledge system $\Pi'$. The idea is to append to the crs a public key pk corresponding to any semantic secure encryption scheme. Thus, the common reference string corresponding to $\Pi'$ is of the form $\mathsf{crs}' = (\mathsf{crs}, \mathsf{pk})$. In order to prove that $x \in L$ using a witness $w$, choose randomness $r \leftarrow \{0,1\}^{\mathsf{poly}(k)}$, compute $c = \mathsf{Enc}_{\mathsf{pk}}(w, r)$ and compute a NIZK proof $\pi$, using the underlying NIZK proof system $\Pi$, that $(\mathsf{pk}, x, c) \in L'$, where
$$L' \triangleq \{(\mathsf{pk}, x, c) : \exists (w, r) \text{ s.t. } (x, w) \in \mathcal{R} \text{ and } c = \mathsf{Enc}_{\mathsf{pk}}(w, r)\}.$$
Let $\pi' = (\pi, c)$ be the proof.

The simulator will generate a simulated $\mathsf{crs}'$ by generating $(\mathsf{crs}, \mathsf{trap})$ using the underlying simulator $\mathcal{S}^{\mathsf{crs}}$, and by generating a public key pk along with a corresponding secret key sk. Thus, $\mathsf{trap}' = (\mathsf{trap}, \mathsf{sk})$. The extractor algorithm $E$, will extract a witness for $x$ from a proof $\pi' = (\pi, c)$ by using sk to decrypt the ciphertext $c$.

**Lemma 4.2.3** ([FLS90]). *Assuming the existence of enhanced trapdoor permutations, there exists an efficient adaptive NIZK proof of knowledge for all languages in NP.*

### 4.2.2 Equivocal Commitments

Informally speaking, a bit-commitment scheme is *equivocal* if it satisfies the following additional requirement. There exists an efficient simulator that outputs a fake commitment such that: (a) the commitment can be decommitted to both 0 and 1, and (b) the simulated commitment and decommitment pair is indistinguishable from a real pair. We now formally define the equivocability property for bit-commitment schemes in the CRS model.

The following definition is adapted from [FS89, CIO98].

**Definition 4.2.4.** A non-interactive bit-commitment scheme $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Rec})$ in the CRS model is said to be an *equivocal bit-commitment scheme in the CRS model* if there exists a PPT simulator algorithm $\mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{com}})$ such that $\mathcal{S}^{\mathsf{crs}}$ takes as input the security parameter $1^k$ and outputs a CRS and trapdoor pair, $(\mathsf{crs}, \mathsf{trap})$; and $\mathcal{S}^{\mathsf{com}}$ takes as input such a pair $(\mathsf{crs}, \mathsf{trap})$ and generates a tuple $(\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1)$ of a commitment string $c$ and two decommitments $\mathsf{eqdec}^0$ and $\mathsf{eqdec}^1$ (for 0 and 1), such that the following holds.

1. For every $b \in \{0, 1\}$ and every $(\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1) \leftarrow \mathcal{S}^{\mathsf{com}}(\mathsf{crs}, \mathsf{trap})$, it holds that
$$\mathsf{Rec}(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}^b) = b.$$
2. For every $b \in \{0, 1\}$, the random variables
$$\{(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}) : \mathsf{crs} \leftarrow \mathsf{Gen}(1^k), (\mathsf{eqcom}, \mathsf{eqdec}) \leftarrow \mathsf{Com}(\mathsf{crs}, b)\}$$
and
$$\{(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}^b) : (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k), (\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1) \leftarrow \mathcal{S}^{\mathsf{com}}(\mathsf{crs}, \mathsf{trap})\}$$
are computationally indistinguishable.

**Claim 4.2.5.** *Let* $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Rec}, \mathcal{S})$ *where* $\mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{com}})$, *be a non-interactive bit-commitment scheme in the CRS model (as in Definition 4.4.4). Then the distributions*
$$\{(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}) : (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k), (\mathsf{eqcom}, \mathsf{eqdec}) \leftarrow \mathsf{Com}(\mathsf{crs}, b)\}$$
*and*
$$\{(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}^b) : (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k), (\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1) \leftarrow \mathcal{S}^{\mathsf{com}}(\mathsf{crs}, \mathsf{trap})\}$$
*are computationally indistinguishable.*

*Proof.* The proof follows from the following simple analysis.
$$\{(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}) : (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k), (\mathsf{eqcom}, \mathsf{eqdec}) \leftarrow \mathsf{Com}(\mathsf{crs}, b)\} \approx$$
$$\{(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}) : \mathsf{crs} \leftarrow \mathsf{Gen}(1^k), (\mathsf{eqcom}, \mathsf{eqdec}) \leftarrow \mathsf{Com}(\mathsf{crs}, b)\} \approx$$
$$\{(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}^b) : (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k), (\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1) \leftarrow \mathcal{S}^{\mathsf{com}}(\mathsf{crs}, \mathsf{trap})\},$$
where the first equation follows from the fact that $\{\mathsf{crs} : \mathsf{crs} \leftarrow \mathsf{Gen}(1^k)\}$ is computationally indistinguishable from $\{\mathsf{crs} : (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k)\}$, and the second equation follows from Definition 4.4.4. $\square$

**Reusable CRS.** Note that the simulator algorithms $\mathcal{S}^{\mathsf{crs}}$ and $\mathcal{S}^{\mathsf{com}}$ are described as separate algorithms in the Definition 4.4.4 to highlight that it is not necessary to create a separate CRS for every equivocal commitment, i.e., the CRS is *reusable*. In this case, Definition 4.4.4 can be extended in a straightforward manner to consider indistinguishability of an honestly generated tuple consisting of crs and polynomially many commitment, decommitment pairs, from a simulated tuple.

**Lemma 4.2.6** ([CLOS02]). *Assuming the existence of one-way functions, there exists an equivocal bit commitment in the (reusable) CRS model.*

**String Equivocal Commitments.** For our purposes, we actually use *string* equivocal commitment schemes. Note that such a scheme can be easily constructed by simply repeating the above bit commitment scheme in *parallel*. More specifically, a commitment to a string of length $n$ is a vector $(\mathsf{eqcom}_1, ..., \mathsf{eqcom}_n)$, with corresponding decommitment vector $(\mathsf{eqdec}_1, ..., \mathsf{eqdec}_n)$. The simulator algorithm $\mathcal{S}^{\mathsf{com}}$ produces a commitment vector and a *pair* of decommitment vectors $d^0 = (\mathsf{eqdec}_1^0, ..., \mathsf{eqdec}_n^0)$, $d^1 = (\mathsf{eqdec}_1^1, ..., \mathsf{eqdec}_n^1)$. A decommitment to any particular bit string $a = (a, ..., a_n)$ can be formed by selecting the appropriate decommitment values $(\mathsf{eqdec}_1^{a_1}, \ldots, \mathsf{eqdec}_n^{a_n})$.

### 4.2.3 The Elect Protocol

As part of our protocol, we elect disjoint committees, and need the guarantee that the number of parties elected is not too large, and that (with overwhelming probability in $k$) a constant fraction of each committee is honest. Such a protocol can be obtained using the technique of Feige's lightest bin committee election protocol [Fei99].

Feige's protocol selects a single committee of approximate size $\ell$ out of $n$ parties.[2] It consists of a single round, in which each party $P_i$ chooses and broadcasts a random integer "bin" $b_i \in \left[\frac{n}{\ell}\right]$. The elected committee consists of the parties in the "lightest bin": that is, those parties $P_j$ whose value $b_j$ was selected by the smallest number of total parties. Feige demonstrated that no set of malicious parties $M \subset [n]$ of size $(1-\epsilon)n$ can force a committee $\mathcal{E}$ to be elected for which $|\mathcal{E} \cap M|$ is significantly greater than $(1-\epsilon)\ell$, by using a Chernoff bound to argue that each bin contains nearly $\epsilon n$ *honest* parties.

**Lemma 4.2.7** (Feige). *For any constant* $(1-\epsilon) > 0$ *and any* $\ell < n$*, Feige's lightest bin protocol is a 1-round public-coin protocol for electing a committee* $\mathcal{E}$ *such that for any set of malicious parties* $M \subset [n]$ *of size* $t = (1-\epsilon)n$,

1. $|\mathcal{E}| \leq \ell$,
2. $\Pr[|\mathcal{E} \setminus M| \leq (\epsilon - \eta)\ell] < \frac{n}{\ell} e^{-\frac{\eta^2 \ell}{2\epsilon}}$ $\forall$ *constant* $\eta > 0$,
3. $\Pr\left[\frac{|\mathcal{E} \cap M|}{|\mathcal{E}|} \geq (1-\epsilon) + \eta\right] < \frac{n}{\ell} e^{-\frac{\eta^2 \ell}{2\epsilon}}$ $\forall$ *constant* $\eta > 0$.

*Proof.* Given as Lemma 2.2.9 in Chapter 2. □

[2]In Feige's original work [Fei99], he considered the specific case of $\ell = \log n$. For our purpose, we need to elect committees whose size depends on the security parameter (to achieve negligible error), and thus we consider general $\ell$.

We will use the following simple corollary within Section 4.4.5.

**Corollary 4.2.8.** *For any constant $(1-\epsilon) > 0$ and $\ell = \sqrt{n}$, Feige's lightest bin protocol is a 1-round public-coin protocol for electing a committee $\mathcal{E}$ such that $|\mathcal{E}| \leq \sqrt{n}$ and for any set of malicious parties $M \subset [n]$ of size $t = (1-\epsilon)n$,*

$$\Pr[|\mathcal{E} \setminus M| = 0] < ne^{-\sqrt{n}/2}.$$

Now, suppose we wish to elect $m$ disjoint committees, each of size approximately $k$, where $k$ is the security parameter, and where the number of parties $n$ is at least $n \geq mk^2$. We consider the following protocol, Elect. In a single round, each party samples and broadcasts a random value $x_i \leftarrow [n/k]$. The resulting committees are precisely the $m$ lightest bins. Namely, suppose the lightest bin is $\ell_1$, the second lightest bin is $\ell_2$, etc. Then $\mathcal{E}_j = \{P_i : x_i = \ell_j\}$, for $j = 1, ..., m$.

**Lemma 4.2.9.** *Let $n \geq mk^2$, and let $M \subset [n]$ be any subset of corrupted parties of size $(1-\epsilon)n$. Then for any constant $\eta > 0$, there exists a negligible function $\nu(k)$ in $k$ such that the protocol* Elect *yields a collection of $m$ committees $\{\mathcal{E}_j\}_{j=1}^m$ where*

1. $\Pr\left[\exists j : |\mathcal{E}_j| \leq (\epsilon - \eta)k\right] \leq \nu(k)$, *and*
2. $\Pr\left[\exists j : \frac{|\mathcal{E}_j \cap M|}{|\mathcal{E}_j|} > (1-\epsilon) + \eta\right] \leq \nu(k)$,

*where the probabilities are taken over the randomness of all parties. Moreover,*

$$|\mathcal{E}_1 \cup \ldots \cup \mathcal{E}_m| \leq mk.$$

*Proof.* For each bin $b$ and honest party $i$, we define the indicator variable $X_{i,b}$ to be 1 if and only if party $i$ selects bin $b$. Since we consider only honest parties, this is a Bernoulli random variable with $p = \frac{k}{n}$. For a particular bin $b$, we can now bound the probability that few honest parties selected this bin as compared to the expected value $\epsilon k$.

$$\begin{aligned}\Pr\left[\sum_{i \notin M} X_{i,b} < (\epsilon - \eta)k\right] &= \Pr\left[\sum_{i \notin M} X_{i,b} < \left(1 - \frac{\eta}{\epsilon}\right)\epsilon k\right] \\ &< e^{-(\epsilon)k(\frac{\eta}{\epsilon})^2/2} \\ &= e^{-\frac{\eta^2 k}{2\epsilon}},\end{aligned}$$

where the second inequality holds by a Chernoff bound.[3] Taking a union bound, the probability that *any* bin $b$ has fewer than $(\epsilon - \eta)k$ honest parties will be

$$\Pr[\exists \text{ Bin } b : \sum_{i \notin M} X_{i,b} < (\epsilon - \eta)k] < \frac{n}{k} e^{-\frac{\eta^2 k}{2\epsilon}}.$$

[3]Exact Chernoff bound used: For $X_1, ..., X_n$ independent Bernoulli random variables and $\mu = \mathbb{E}[\sum_i X_i]$, then for $0 < \delta < 1$, it holds that $\Pr[\sum_i X_i < (1-\delta)\mu] < e^{-\mu\delta^2/2}$.

Note that since $n = \mathsf{poly}(k)$, this probability is negligible in $k$ for any constant $\eta$, implying property (1).

It remains to show that $|\mathcal{E}_j| \leq (1+o(1))k$ for every elected committee $\mathcal{E}_j$ (which, together with the above, will imply property (2)).

Suppose $|\mathcal{E}_i| = k + \ell$ for an elected committee $\mathcal{E}_i$. We will argue that with overwhelming probability, $\ell \in o(k)$. Since $\mathcal{E}_i$ is one of the $m$ lightest bins, it must be that each of the remaining $\frac{n}{k} - m$ bins has size at least $k + \ell$. Now, we know that with overwhelming probability in $k$, each of the $m$ elected bins $\mathcal{E}_i$ has at least $(\epsilon - \frac{\eta}{2})k$ honest parties. This means that (with overwhelming probability), the total number of parties in all *non-elected* bins can be no greater than

$$n - m\left(\epsilon - \frac{\eta}{2}\right)k.$$

Thus, since the number of non-elected parties is at least $(\frac{n}{k} - m)(k + \ell)$, we must have with overwhelming probability that

$$\begin{aligned}
&\left(\frac{n}{k} - m\right)(k+\ell) \leq n - m\left(\epsilon - \frac{\eta}{2}\right)k. \\
\implies &\left(\frac{n}{k} - m\right)\ell \leq n - m\left(\epsilon - \frac{\eta}{2}\right)k - (n - km). \\
&\qquad = km - km\left(\epsilon - \frac{\eta}{2}\right) \\
\implies &\ell \leq \frac{km - km\left(\epsilon - \frac{\eta}{2}\right)}{\left(\frac{n}{k} - m\right)} \\
&\quad \leq k\left(\frac{1 - \epsilon + \eta/2}{k-1}\right) \qquad \text{since } n \geq mk^2 \\
&\quad \in o(k).
\end{aligned}$$

This implies that each elected committee $\mathcal{E}_i$ must satisfy

$$\begin{aligned}
|\mathcal{E}_i| &\leq k + \ell \\
&\in (1 + o(1))k,
\end{aligned}$$

as desired.

We next prove that

$$\left|\mathcal{E}_1 \cup \ldots \cup \mathcal{E}_m\right| \leq mk$$

To this end, we order the $n/k$ bins by their weight, and denote the resulting ordered bins by $B_1, \ldots, B_{n/k}$ (where $B_1$ is the lightest and $B_{n/k}$ is the heaviest). Think of the first $m$ bins (i.e., the $m$ lightest bins) as a super-bin $\mathsf{Super}_1$, the next $m$ bins as a super-bin $\mathsf{Super}_2$, etc. Thus, there are $\lceil \frac{n}{mk} \rceil$ super-bins. Note that $\mathcal{B}_1$ is the lightest of the first $\lfloor \frac{n}{mk} \rfloor$ super-bins,[4] and thus is of size at most $mk$, as desired. □

[4] If $\frac{n}{mk}$ is not an integer then the last super-bin may contain less than $m$ bins, and thus may be lighter than the first super-bin.

### 4.2.4 Fully Homomorphic Encryption

A fully homomorphic public-key encryption scheme (FHE) consists of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$. The first three are the standard key generation, encryption and decryption algorithms of a public key scheme. The additional algorithm $\mathsf{Eval}$ is a deterministic polynomial-time algorithm that takes as input a public key $\mathsf{pk}$, a ciphertext $\hat{x} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x)$ and a circuit $C$, and outputs a new ciphertext $c = \mathsf{Eval}_{\mathsf{pk}}(\hat{x}, C)$ such that $\mathsf{Dec}_{\mathsf{sk}}(c) = C(x)$, where $\mathsf{sk}$ is the secret key corresponding to the public key $\mathsf{pk}$. It is required that the size of $c$ depends polynomially on the security parameter and the length of the output $C(x)$, but is otherwise independent of the size of the circuit $C$.

Several such FHE schemes have been constructed, starting with the seminal work of Gentry [Gen09]. Recently, new schemes were presented by Brakerski, Gentry and Vaikuntanathan [BV11, BGV11] that achieve greater efficiency and are based on the LWE assumption. We note that in these schemes, the size of the public key depends linearly on the depth of the functions being evaluated. As a result, the complexity of our preprocessing phase depends on the maximum depth of functions that we would like to compute. This issue can be avoided all together if we assume that the schemes of [BV11, BGV11] are circular secure.

For our construction, we need an FHE scheme with the following additional property, which we refer to as *certifiability.* Loosely speaking, an FHE scheme is said to be certifiable, if there is an efficient algorithm that takes as input a random string $r$ and tests whether it is "good" to use $r$ as randomness in the encryption algorithm $\mathsf{Enc}$. More precisely, a certifiable FHE scheme is associated with a set $R$, which consists of all the "good" random strings, such that (1) a random string is in $R$ with overwhelming probability; and (2) The $\mathsf{Eval}$ algorithm and the decryption algorithm $\mathsf{Dec}$ are correct on ciphertexts that use randomness from $R$ to encrypt. A formal definition follows.

**Definition 4.2.10.** A FHE scheme is said to be *certifiable* if there exists a subset $R \subseteq \{0,1\}^{\mathsf{poly}(k)}$ of possible randomness values for which the following hold.

1. $\Pr[r \in R] = 1 - \mathsf{negl}(k)$, where the probability is over uniformly sampled $r \leftarrow \{0,1\}^{\mathsf{poly}(k)}$.

2. There exists an efficient algorithm $\mathcal{A}_R$ such that $\mathcal{A}_R(r) = 1$ for $r \in R$ and 0 otherwise.

3. We have

$$\Pr_{\mathsf{pk},\mathsf{sk}}\left[\begin{array}{c} \forall b_1,\ldots,b_n \in \{0,1\},\ \forall r_1,\ldots,r_n \in R, \\ \forall \text{ poly-size circuits } f : \{0,1\}^n \rightarrow \{0,1\} \\ \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{pk}}(f, c_1,\ldots,c_n)) = f(b_1,\ldots,b_n), \\ \text{where } c_i = \mathsf{Enc}_{\mathsf{pk}}(b_i; r_i) \end{array}\right] = 1 - \mathsf{negl}(k).$$

We note that this property holds, for example, for the schemes of [BV11, BGV11]. For the readers who are familiar with these constructions, the set of "good" randomness $R$ corresponds to encrypting with sufficiently "small noise."

### 4.2.5 Leaky Distributed Systems

One of the tools in our construction is a compiler that converts any circuit $C$ (with secrets) into a collection of sub-computations (or "modules") $\mathsf{Sub}_1, \ldots, \mathsf{Sub}_m$, whose sequential evaluation evaluates the circuit $C$, and which is secure in the *leaky distributed systems* (LDS) model, a model recently introduced by Bitansky *et. al.* [BCG+11].

Before we describe this compiler, let us recall prerequisite prior works [JV10, GR10, GR12], which construct such a compiler in the *"only computation leaks"* (OCL) model. In particular, these works demonstrate a compiler that takes a circuit $C$ and converts it into a circuit $C'$ consisting of $m$ disjoint, ordered sub-computations $\mathsf{Sub}_1, \ldots, \mathsf{Sub}_m$, where the input to sub-computation $\mathsf{Sub}_i$ depends only on the output of earlier sub-computations. Each of these sub-computations $\mathsf{Sub}_i$ is modeled as a non-uniform randomized poly-size circuit, with a "secret state." It was proven that no information about the circuit $C$ is leaked, even if each of these sub-computations is leaky. More specifically, the adversary can request to see a bounded-length function of each $\mathsf{Sub}_i$ (separately), and these leakage functions may be adaptively chosen.

These works also consider the continual leakage setting, where leakage occurs over and over again in time. In this setting, the secret state of each $\mathsf{Sub}_i$ must be continually updated or refreshed. To this end, after each computation, all the $\mathsf{Sub}_i$'s update their secret state by running a randomized protocol Update. We stress that leakage may occur during each of these update protocols, and that such leakage may be a function of both the current secret state and the randomness used by the Update procedure.

In this work, we use such a compiler which is secure in the LDS model [BCG+11]. The LDS model strengthens the OCL model in two ways. First, in the LDS model, the adversary is allowed to *view and control the entire communication between modules*; in contrast, the OCL model assumes the communication between modules is kept secret from the adversary, and that the messages are generated honestly. Second, in the LDS model, the adversary may leak adaptively on each module *in any order.* For instance, the adversary may leak a bit from $\mathsf{Sub}_i$, then a bit from $\mathsf{Sub}_j$, and based on the results, leak again on $\mathsf{Sub}_i$. In contrast, the OCL model only allows the adversary to request leakage information from the module that is currently computing. In particular, this restricts the adversary to leak on modules in order (i.e., first leak from $\mathsf{Sub}_1$, then from $\mathsf{Sub}_2$, etc.).

In what follows, we describe the LDS compiler of [BCG+11] in more detail. Similar to the OCL circuit compiler, the LDS circuit compiler initializes each module with some secret state, and thereafter the modules can receive messages, send messages, generate fresh randomness, and maintain a local state. To evaluate the circuit $C$, an input $v$ is given to $\mathsf{Sub}_1$ and the modules communicate to jointly compute $C(v)$, which is eventually output by $\mathsf{Sub}_m$.

**Remark 4.2.11.** For the sake of simplicity of notation, we assume (without loss of generality) that the module $\mathsf{Sub}_i$ only sends messages to $\mathsf{Sub}_{i+1}$ (where we define $\mathsf{Sub}_{m+1} \triangleq \mathsf{Sub}_1$). Moreover, we assume for simplicity that during each computation, where $C$ is evaluated on some input $v$, each module $\mathsf{Sub}_i$ sends a *single* message to $\mathsf{Sub}_{i+1}$, and that $\mathsf{Sub}_m$ does not send a message to any module, and simply outputs $C(v)$. This assumption indeed holds for the LDS compiler of [BCG+11] which is based on [GR12]. We note that this assumption is not needed for our result to be correct, but it simplifies the notation.

The LDS model considers an adversary who interacts with and leaks from the modules in attempt to learn $C$. As in the OCL model, the adversary is allowed to freely choose the inputs $v$ to the computation, and to choose one submodule at a time and evaluate a leakage function on its inner state and randomness. There is no limit to the number of times each component can be chosen to leak during the lifetime of the system, as long as the total rate of leakage from each component is not too high. In addition, the leakage functions can be chosen adaptively, as a function of all information learned up to that point. However, in contrast to the OCL model, the adversary is allowed to leak on the modules in any order. The adversary also has the additional power to see and control all the communication between the modules.

More explicitly, the LDS model considers continual leakage, where the lifetime of each submodule is partitioned into time periods. At the end of each time period, the modules "refresh" their inner state by applying a (possibly distributed) Update procedure, after which they erase their previous state. As with the rest of the computation, the Update procedure is also exposed to leakage, and the adversary controls the exchange of messages during the update.

**Definition 4.2.12** (Leaky Distributed Systems (LDS) Model). In a $\lambda$-*bounded* LDS *attack*, a PPT adversary $\mathcal{A}$ interacts with modules $(\mathsf{Sub}_1, ..., \mathsf{Sub}_m)$ by adaptively performing any sequence of the following actions:

- $\mathsf{Interact}(j, \mathsf{msg})$: For $j \in [m]$, send the message $\mathsf{msg}$ to the $j$'th submodule, $\mathsf{Sub}_j$, and receive the corresponding reply. Note that the modules are message-driven: they become activated when they receive a message from the attacker, at which point they compute and send the result, and then wait for additional messages.

- $\mathsf{Leak}(j, L)$: For $j \in [m]$ and a poly-size leakage function $L : \{0,1\}^* \to \{0,1\}$, if strictly fewer than $\lambda$ queries of the form $\mathsf{Leak}(j, \cdot)$ have been made so far, $\mathcal{A}$ receives the evaluation of $L$ on the secret state of the $j$'th submodule, $\mathsf{Sub}_j$. Otherwise, $\mathcal{A}$ receives $\perp$.

In a *continual* $\lambda$-LDS *attack*, the adversary $\mathcal{A}$ repeats a $\lambda$-bounded LDS attack polynomially many times, where between every two consecutive attacks the secret states of the modules are updated. The update is done by running a distributed Update protocol among all the modules. We also allow $\mathcal{A}$ to leak during the Update procedure, where the leakage function takes as input both the current secret state of $\mathsf{Sub}_j$ and the randomness it uses during the Update procedure.

We denote by time period $t$ of submodule $\mathsf{Sub}_j$ the time period between the beginning of the $(t-1)$'st Update procedure and the end of the $t$'th Update procedure in that submodule (note that these time periods are overlapping).[5] We allow the adversary $\mathcal{A}$ to leak at most $\lambda$ bits from each $\mathsf{Sub}_j$ during each (local) time period.

We refer to such an adversary $\mathcal{A}$ as an $\lambda$-LDS *adversary*, and denote the output of $\mathcal{A}$ in such an attack by $\mathcal{A}[\lambda : \mathsf{Sub}_1, ..., \mathsf{Sub}_m : \mathsf{Update}]$.

We say that the collection of modules $(\mathsf{Sub}_1, ..., \mathsf{Sub}_m)$ is $\lambda$-*secure* in the LDS model if for any $\lambda$-LDS adversary $\mathcal{A}$ interacting with the modules as described above, there exists a PPT simulator who simulates the output of $\mathcal{A}$.

**Definition 4.2.13** (LDS-Secure Circuit Compiler). We say that $(\mathcal{C}, \mathsf{Update})$ is a $\lambda$-LDS *secure circuit compiler* if for any circuit $C$ and $(\mathsf{Sub}_1, ..., \mathsf{Sub}_m) \leftarrow \mathcal{C}(C)$, the following two properties hold:

[5] Intuitively, time period $t$ is the entire time period where the $t$'th updated secret states can be leaked. Note that during the $t$'th Update procedure, both the $(t-1)$'st and the $t$'th secret state may leak, which is why the time periods are overlapping.

1. **Correctness:** The collection of modules $(\mathsf{Sub}_1, ..., \mathsf{Sub}_m)$ maintain the functionality of $C$ when all the messages between them are delivered intact.

2. **Secrecy:** For every PPT $\lambda$-LDS adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$, such that for any ensemble of poly-size circuits $\{C_n\}$ and any auxiliary input $z \in \{0,1\}^{\mathsf{poly}(n)}$:
$$\left\{\mathcal{A}(z)[\lambda : \mathsf{Sub}_1, ..., \mathsf{Sub}_m : \mathsf{Update}]\right\}_{n\in\mathbb{N}, C\in C_n} \approx_c \left\{\mathcal{S}^C(z, 1^{|C|})\right\}_{n\in\mathbb{N}, C\in C_n},$$
where $\mathcal{S}$ only queries $C$ on the inputs $\mathcal{A}$ sends to the first module, $\mathsf{Sub}_1$.

**Theorem 4.2.14** ([BCG$^+$11]). *Assuming the existence of a non-committing encryption scheme and a $\lambda$-OCL circuit compiler which compiles a circuit $C$ to $m(|C|)$ modules, there exists a $\lambda$-LDS secure circuit compiler $(\mathcal{C}, \mathsf{Update})$ for which $\mathcal{C}(C)$ has the same number of modules, $m(|C|)$.*

A very recent work of Goldwasser and Rothblum [GR12] constructs a $\lambda$-OCL circuit compiler with the following properties.

**Theorem 4.2.15** ([GR12]). *For any security parameter $k$, there (unconditionally) exists a $\lambda$-OCL secure circuit compiler for $\lambda = \tilde{\Omega}(k)$, that takes any circuit $C$ into a collection of $O(|C|)$ modules, each of size $O(k^3)$.*

**Remark 4.2.16** (Folklore). If one additionally assumes the existence of a fully homomorphic encryption (FHE) scheme, then there exists a $\lambda$-LDS secure circuit compiler $(\mathcal{C}, \mathsf{Update})$ such that for every poly-size circuit $C$, the number of output sub-computations $\mathsf{Sub}_1, ..., \mathsf{Sub}_m$ generated by $\mathcal{C}$ is polynomial in the security parameter of the FHE scheme and *independent* of the size of $C$.

### 4.2.6 *Weakly* Leakage-Resilient MPC

Our construction of a leakage-resilient MPC protocol in the preprocessing model (as described in Section 4.3.2), uses as a building block an MPC protocol that is leakage-resilient with respect to a *weaker* notion of secrecy (where the ideal world is weakened), as was recently constructed in [BGJ$^+$13]. For lack of a better name, we call it *weakly* leakage-resilient MPC. Below, we present a simplified (and weaker) version of the security definition achieved by [BGJ$^+$13], which suffices for our construction.[6]

Very briefly, the security definition in [BGJ$^+$13] follows the ideal/real world paradigm. They consider a real-world execution where an adversary, in addition to corrupting a number a parties, can obtain leakage information on the joint secret states of the honest parties at any point during the protocol execution. Leakage queries may be adaptively chosen based on all information received up to that point (including responses to previous leakage queries), and are computed on the joint secret states of all the honest parties.

Note that in the absence of a leakage-free preprocessing phase, one cannot hope to realize the standard ideal world security in the presence of leakage attacks, since leakage directly on the inputs

[6]Among the differences: The work of [BGJ$^+$13] considered a more general definition to capture, e.g., "noisy" and computationally hard-to-invert leakage, whereas here we will only require a more simplistic bounded-output leakage model.

of honest parties cannot be simulated. To this end, [BGJ$^+$13] consider an ideal world experiment where in addition to learning the output of the function evaluation, the simulator is also allowed to request leakage on the inputs of all the honest parties jointly. Below, we describe the ideal and real world experiments and give the formal security definition from [BGJ$^+$13].

**Ideal World.** We first describe the ideal world experiment, where $n$ parties $P_1, \dots, P_n$ interact with an ideal functionality for computing a function $f$. An adversary may corrupt any subset $M \subset \mathcal{P}$ of the parties. As in the standard MPC ideal world experiment, the parties send their inputs to the ideal functionality and receive the output of $f$ evaluated on all inputs. The main difference from the standard ideal world experiment is that the adversary is also allowed to make *leakage queries* on the inputs of the honest parties. Such queries are evaluated on the *joint* collection of all parties' inputs. The ideal world execution proceeds as follows.

**Inputs:** Each party $P_i$ obtains an input $x_i$. The adversary is given auxiliary input $z$ and selects a subset of parties $M \subset \mathcal{P}$ to corrupt.

**Sending inputs to trusted party:** Each honest party $P_i$ sends its input $x_i$ to the ideal functionality. For each corrupted party $P_i \in M$, the adversary may select any value $x'_i$ and send it to the ideal functionality.

**Trusted party computes output:** Let $x'_1, \dots, x'_n$ be the inputs that were sent to the ideal functionality. The ideal functionality computes $f(x'_1, \dots, x'_n)$.

**Adversary learns output:** The ideal functionality first sends the evaluation $f(x'_1, ..., x'_n)$ to the adversary. The adversary replies with either continue or abort.

**Honest parties learn output:** If the message is abort, the ideal functionality sends $\perp$ to all honest parties. If the adversary's message was continue, then the ideal functionality sends the function evaluation $f(x'_1, \dots, x'_n)$ to all honest parties.

**Leakage queries on inputs:** The adversary may send (adaptively chosen) leakage queries in the form of efficiently computable functions $L_j$ (described as a circuit). On receiving such a query, the ideal functionality computes $L_j(x'_1, \dots, x'_n)$ and returns the output to the adversary.

**Outputs:** Honest parties output their inputs and the messages they obtained from the ideal functionality. Malicious parties may output an arbitrary PPT function of their initial input (auxiliary input and random-tape) and the message it has obtained from the ideal functionality.

An ideal world adversary $\mathcal{S}$ who obtains a total of $\lambda$ bits of leakage is referred to as a *$\lambda$-leakage ideal adversary.* The overall output of the ideal-world experiment consists of all the inputs and values received by honest parties from the ideal functionality, together with the output of the adversary, and is denoted by $\mathsf{W\text{-}IDEAL}^{f}_{\mathcal{S},M}(1^k, \vec{x}, z)$.

**Real World.** The real-world experiment begins by first choosing a common random string $\mathsf{crs}$. Then, each party $P_i$ receives an input $x_i$ and the adversary $\mathcal{A}$ receives auxiliary input $z$. These values can depend arbitrarily on the $\mathsf{crs}$, but need to be efficiently computable given the $\mathsf{crs}$. However, for the sake of simplicity of notation, throughout this section we assume that these values are independent of the $\mathsf{crs}$.

The adversary $\mathcal{A}$ selects any arbitrary subset $M \subset \mathcal{P}$ of the parties to corrupt. Each corrupted party $P_i \in M$ hands over its input to $\mathcal{A}$. The parties $P_1, \ldots, P_n$ now engage in an execution of a real $n$-party protocol $\Pi$. The adversary $\mathcal{A}$ sends all messages on behalf of the corrupted parties, and may follow an arbitrary polynomial-time strategy. In contrast, the honest parties follow the instructions of $\Pi$. Furthermore, at any point during the protocol execution, the adversary may make leakage queries of the form $L$ and learn $L(\mathsf{state}_{\mathcal{P}\setminus M})$, where $\mathsf{state}_{\mathcal{P}\setminus M}$ denotes the concatenation of the protocol states $\mathsf{state}_i$ of each honest party $P_i$. We allow the adversary to choose the leakage queries adaptively.

Honest parties have the ability to toss fresh coins at any point in the protocol, and at that point these coins are added to the state of that party. At the conclusion of the protocol execution, each honest party $P_i$ generates an output according to $\Pi$. Malicious parties may output an arbitrary PPT function of the view of $\mathcal{A}$.

An adversary $\mathcal{A}$ who obtains at most $\lambda$ bits of leakage is referred to as a *$\lambda$-leakage real adversary.* Let $\mathsf{Gen}_{\mathsf{WLR}}$ denote the CRS generation algorithm. Further, let $\mathsf{W\text{-}REAL}^{\Pi}_{\mathcal{A}}(1^k, \mathsf{crs}, \vec{x}, z)$ be the random variable that denotes the values output by the parties at the end of the protocol $\Pi$ (using $\mathsf{crs} \leftarrow \mathsf{Gen}_{\mathsf{WLR}}(1^k)$ as the CRS). Then, the overall output of the real-world experiment is defined as the tuple $(\mathsf{crs}, \mathsf{W\text{-}REAL}^{\Pi}_{\mathcal{A},M}(1^k, \mathsf{crs}, \vec{x}), z)$.

**Security Definition.** We now state the formal security definition.

**Definition 4.2.17** ($\lambda$-Weakly Leakage-Resilient MPC). A protocol $\Pi$ evaluating a functionality $f$ is a *$\lambda$-weakly leakage-resilient* MPC protocol if for every PPT $\lambda$-leakage real adversary $\mathcal{A}$, there exists a $\lambda$-leakage ideal adversary $\mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{exec}})$, corrupting the same parties as $\mathcal{A}$, such that for every input vector $\vec{x}$, every auxiliary input $z \in \{0,1\}^*$, and every subset $M \subset \mathcal{P}$, it holds that

$$\left\{\mathsf{crs}, \mathsf{W\text{-}IDEAL}^{f}_{\mathcal{S}^{\mathsf{exec}}(\mathsf{crs},\mathsf{trap}),M}(1^k, \vec{x}, z)\right\}_{k \in N} \approx_c \left\{\mathsf{crs}', \mathsf{W\text{-}REAL}^{\Pi}_{\mathcal{A},M}(1^k, \mathsf{crs}', \vec{x}, z)\right\}_{k \in N},$$

where $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k)$, and $\mathsf{crs}' \leftarrow \mathsf{Gen}(1^k)$.

**Theorem 4.2.18** ([BGJ+13]). *Based on the DDH assumption, for every poly-size function $f$, for every leakage bound $\lambda \in \mathbb{N}$, and any number of parties and corrupted parties, there exists a protocol $\Pi$ in the common random string model for computing $f$ that is $\lambda$-weakly leakage resilient as per Definition 4.2.17.*

**Remark 4.2.19.** We note that Theorem 4.2.18 holds even if we allow the input vector $\vec{x}$ and the auxiliary input $z$ to be arbitrary poly-time computable functions of the $\mathsf{crs}$. We eliminated this dependency from Definition 4.2.17 only for the sake of simplicity of notation.

**Remark 4.2.20** (Standalone vs. UC Security). The main result in [BGJ+13] actually achieves a stronger notion of *universally composable (UC)* security, at the cost of additionally relying on the decisional linear assumption over bilinear groups. Indeed, their UC-secure WLR-MPC construction relies on a leakage-resilient UC-NIZK system, whose only known construction [GJS11, GOS06b] is based on the decisional linear assumption in the bilinear groups setting.

However, for the present work, it suffices to obtain a "standalone" secure construction of WLR-MPC. Thus, it is possible to replace the UC-NIZK system with a standalone secure *interactive* weakly leakage-resilient ZKPoK system. This, in turn, can be based on the DDH assumption. The resulting WLR-MPC achieves standalone security based on only the DDH assumption in the CRS model.

**Security against disjoint leakage** In Definition 4.2.17, the real-world adversary $\mathcal{A}$ is allowed to obtain *joint* leakage on the secret states of the honest parties. In the present work, we consider a weaker adversarial model, in which the leakage on each honest party in the real world is *disjoint* (i.e., $\mathcal{A}$ is not allowed to leak on the joint secret states of the honest parties). Theorem 4.2.18 clearly still applies to this setting. However, we note that the *ideal world* guarantee does not become stronger when we consider this set of restricted adversaries: that is, even to simulate such adversaries, the simulator $\mathcal{S}$ needs *joint* leakage on the inputs of all the honest parties.[7]

**Security for randomized functions** We note that Theorem 4.2.18 holds for *deterministic* functionalities $f$. In this work, we need to use a weak leakage-resilient MPC protocol for *randomized* functions (since the modules in the OCL leakage-resilient circuit compute randomized functions). In Section 4.4, we show that in our setting, where leakage in the real world is disjoint, the number of parties is polynomially related to the security parameter, and a constant fraction of the parties are honest, then we can construct weak leakage resilient protocols for randomized functions.

## 4.3 Our Model

In this section, we present the MPC model and the security definition considered in this paper. We start by giving a brief overview of our model and then proceed to give a formal description of the same.

*Overview.* We consider the setting of $n$ parties $\mathcal{P} = \{P_1, ..., P_n\}$ who wish to jointly compute any ppt function over their private inputs. Specifically, we consider the case where the parties wish to perform *arbitrarily many* evaluations of functions of their choice. We refer to a protocol that allows computation of multiple functions (over a given set of inputs) as a *multi-function MPC protocol.* Unlike the standard MPC setting, we consider security of a multi-function MPC protocol against "leaky" adversaries that may (continuously) leak on the secret state of each honest party during the protocol execution.

To formally define security, we turn to the real/ideal paradigm. Very briefly, we consider a real-world execution where an adversary, who corrupts any arbitrary number of parties in the system, may additionally obtain arbitrary bounded, independent leakage on the secret state of

[7]In fact, if we could simulate real-world adversaries that obtain only *disjoint* leakage queries, with a simulator that obtains only *disjoint* leakage queries, then this would almost immediately give us a result similar to ours: An MPC protocol with preprocessing that is secure against continual leakage.

each honest party. However, unlike the recent works on leakage-resilient interactive protocols [GJS11, BCH11, BGK11, DHP11, BGJ$^+$13], we consider the *standard* ideal world model, where the adversary does not learn *any* information on the honest party inputs.

Note that if we do not put any restriction on the real-world adversary, and in particular, if he is allowed to obtain leakage *throughout* the protocol execution, then it is impossible to realize the standard ideal world model, since the adversary may simply leak on the inputs of the honest parties, while this information cannot be simulated in the ideal world. With this in mind, we (necessarily) allow for a "leak-free" one-time *preprocessing stage* that happens at the beginning of the real-world execution. Furthermore, to withstand *continual* leakage attacks, we (necessarily) allow for periodic *updates* of the secret values of the parties. We allow leakage to occur during this update procedure as usual.

We now proceed to give a formal description of our model in the remainder of this section. In Section 5.3, we describe the ideal world experiment. Next, in Section 5.3, we describe the real world experiment. Finally, in Section 5.3, we present our security definition.

Throughout this work, we assume that the functions to be evaluated on parties' inputs give the same output to all parties. This is for simplicity of exposition, since otherwise, if the output itself is a secret value (given to an honest party) then this value can be leaked. This can be handled by complicating our security guarantees, and, indeed, one can tweak our construction to ensure that the adversary learns *only* leakage information on such outputs. However, for the sake of simplicity, we choose to avoid this issue.

### 4.3.1 Ideal World

In the ideal world, each party $P_i$ sends her input $x_i$ to a trusted third party. Whenever the adversary $\mathcal{A}$ sends a poly-size circuit $f$ to the trusted party, it sends back $f(x_1, \ldots, x_n)$. Since we consider the case of dishonest majority, we can only obtain security with abort: i.e., the adversary first receives the function output $f(x_1, \ldots, x_n)$, and then chooses whether the honest parties also learn the output, or to prematurely abort. The adversary can query the trusted party many times with various functions $f_j$. Moreover, these functions can be adaptively chosen, based on the outputs of previous functions. The ideal world model is formally described below.

**Inputs:** Each party $P_i$ obtains an input $x_i$. The adversary is given auxiliary input $z$. He selects a subset of the parties $M \subset \mathcal{P}$ to corrupt, and is given the inputs $x_\ell$ of each party $P_\ell \in M$.

**Sending inputs to trusted party:** Each honest party $P_i$ sends its input $x_i$ to the ideal functionality. For each corrupted party $P_i \in M$, the adversary may select any value $x'_i$ and send it to the ideal functionality.

**Trusted party computes output:** Let $x'_1, ..., x'_n$ be the inputs that were sent to the trusted party. Then, the following is repeated for any (unbounded) polynomial number of times:

- **Function selection:** The adversary chooses a poly-size circuit $f_j$, and sends it to the ideal functionality.

- **Adversary learns output:** The ideal functionality sends the evaluation $f_j(x'_1, ..., x'_n)$ to the adversary. The adversary replies with either continue or abort.
- **Honest parties learn output:** If the adversary's message was abort, then the trusted party sends $\perp$ to all honest parties. Otherwise, if the adversary's message was continue, then it sends the function output $f_j(x'_1, ..., x'_n)$ to all honest parties.

**Outputs:** Honest parties output all the messages they obtained from the ideal functionality. Malicious parties may output an arbitrary PPT function of the adversary's view.

The overall output of the ideal-world experiment consists of the outputs of all parties. For any ideal-world adversary $\mathcal{S}$ with auxiliary input $z \in \{0,1\}^*$, any input vector $\vec{x}$, any set of functions $\{f_j\}_{j=1}^{p}$ chosen by the adversary, and security parameter $k$, we denote the output of the corresponding ideal-world experiment by

$$\mathsf{IDEAL}_{\mathcal{S},M}\left(1^k, \vec{x}, z, \{f_j\}_{j=1}^{p}\right).$$

Note that this is a slight abuse of notation since the functions $\{f_j\}_{j=1}^{p}$ may be chosen adaptively.

## 4.3.2 Real World

The real world execution begins by an adversary $\mathcal{A}$ selecting any arbitrary subset of parties $M \subset \mathcal{P}$ to corrupt. The parties then engage in an execution of a real $n$-party multi-function MPC protocol $\Pi = (\Pi_{\mathsf{Pre}}, \Pi_{\mathsf{Input}}, \Pi_{\mathsf{Online}})$ that consists of three stages, namely, (a) a *preprocessing phase*, (b) an *input phase*, and (c) an *online phase*, as described below. We assume that honest parties have the ability to toss fresh coins at any point. Throughout the execution of $\Pi$, the adversary $\mathcal{A}$ sends all messages on behalf of the corrupted parties, and may follow an arbitrary polynomial-time strategy. In contrast, the honest parties follow the instructions of $\Pi$. Furthermore, at any point (except during the preprocessing and the input phases) during the protocol execution, the adversary may leak on the *entire* secret state of each honest parties, via an *MPC leakage query*, defined as follows.

**Definition 4.3.1.** A *MPC leakage query* is defined by $\mathsf{Leak}(i, L)$, where $i \in [n]$ and $L : \{0,1\}^* \rightarrow \{0,1\}$ is a poly-size circuit. When an adversary sends a leakage query $\mathsf{Leak}(i, L)$, he receives the evaluation of $L$ on the *entire* secret state of party $P_i$.

We now formally describe the different phases in the protocol.

**Preprocessing phase:** This phase is *interactive* and *leak-free*, and is run only once. It is independent of the inputs of the parties, and is independent of the functions that will later be evaluated. Thus, this phase can be run in the beginning of time, before the parties even know what their inputs are, or what functions they would like to evaluate.

We assume that no leakage occurs during the run of this preprocessing phase, but we do allow leakage to occur as soon as the preprocessing phase ends. At the end of this phase each party $P_i$ has an (initial) secret state $\mathsf{state}_1^{P_i}$.

**Input phase:** This phase is *non-interactive* and *leak-free*, and depends only on the inputs $x_1, ..., x_n$ (independent of the functions to be computed). Whenever a party $P_i$ gets (or chooses) a secret input $x_i$, she does some local computation which may depend on her secret input $x_i$ and on her secret state $\mathsf{state}_1^{P_i}$. She then sends a message to all parties, and erases her secret input $x_i$.

We assume that the party $P_i$ is not leaked upon during the execution of this phase. However, leakage may occur between the preprocessing phase and the input phase, and leakage may occur immediately after the input phase.

We emphasize that each party can change her input as often as she wants by simply re-running the input phase with the new input.[8]

**Online phase:** This phase takes place in a *leaky* environment. During this phase, the parties carry out an unbounded number of function evaluations on their inputs, and update their respective secret states. At any point during this phase, $\mathcal{A}$ may make adaptively-chosen leakage queries, as per Definition 4.3.1, in the manner as described below.

Whenever $\mathcal{A}$ wishes to compute a function $f_j$ (represented as a poly-size circuit), all parties execute the function evaluation protocol $\Pi_{\mathsf{Comp}}$, described below. Whenever $\mathcal{A}$ wants the honest parties to update their secret states, all parties execute the update protocol $\Pi_{\mathsf{Update}}$, described below. We let $\Pi_{\mathsf{Online}} = (\Pi_{\mathsf{Comp}}, \Pi_{\mathsf{Update}})$. We begin at leakage time period $\ell = 1$; after each update procedure, $\ell$ is incremented.

- **Computation procedure:**
  1. All parties execute protocol $\Pi_{\mathsf{Comp}}(f_j)$, where honest parties $P_i$ act in accordance with input $\mathsf{state}_\ell^{P_i}$. Note that the secret state of parties may change during the execution of this protocol, as dictated by $\Pi_{\mathsf{Comp}}$.
  2. At the conclusion of the computation phase, each honest party $P_i$ outputs his final message of the protocol (which should correspond to the evaluation of $f_j$). Malicious parties may output an arbitrary PPT function of the view of $\mathcal{A}$.
- **$\ell^{th}$ Update procedure:**
  1. All parties execute protocol $\Pi_{\mathsf{Update}}$, where honest parties $P_i$ act in accordance with input $\mathsf{state}_\ell^{P_i}$.
  2. At the conclusion of the update phase, each honest party $P_i$ sets $\mathsf{state}_{\ell+1}^{P_i}$ to be $P_i$'s output from $\Pi_{\mathsf{Update}}$. Each honest $P_i$ erases $\mathsf{state}_\ell^{P_i}$.
  3. Increment $\ell \leftarrow \ell + 1$.

**Leakage:** Initialize each $\mathsf{leaked}_\ell$ to 0. Each leakage query $(i, L)$ made by $\mathcal{A}$ during the $\ell^{\text{th}}$ time period is answered as follows.

- During the **computation** phase: if $\mathsf{leaked}_\ell \geq \lambda$, then $\mathcal{A}$ receives $\emptyset$. Otherwise, $\mathcal{A}$ receives the evaluation of $L$ on the current secret state of party $P_i$, and $\mathsf{leaked}_\ell \leftarrow \mathsf{leaked}_\ell + 1$.

[8]For simplicity, in the security proof in Section 4.6, we assume that the parties run the input phase only once, however the proof extends readily to the case that the parties rerun the input phase many times with different inputs.

- In $\ell$th **update** phase: if *either* $\mathsf{leaked}_\ell \geq \lambda$ *or* $\mathsf{leaked}_{\ell+1} \geq \lambda$, then $\mathcal{A}$ receives $\emptyset$. Otherwise, $\mathcal{A}$ receives the evaluation of $L$ on the current secret state of party $P_i$, and both $\mathsf{leaked}_\ell \leftarrow \mathsf{leaked}_\ell + 1$ and $\mathsf{leaked}_{\ell+1} \leftarrow \mathsf{leaked}_{\ell+1} + 1$.

We emphasize that the $\mathcal{A}$'s leakage queries may be made on any party, adaptively chosen based on all information received up to that point (including responses to previous leakage queries). The only restriction is that the number of bits leaked between the execution of any two consecutive update protocols is bounded. Note that the leakage queries made during the $\ell$'th update phase (where parties transition between their $\ell$'th and $(\ell+1)$'st secret states) are counted against *both* the $\ell$'th and $(\ell+1)$'st time period, where the $\ell$'th time period is the time period where the party stores her $\ell$'th secret state. The reason for this "double counting" is that during the $\ell$'th update phase, the adversary can leak both on the $\ell$'th secret state and on the $\ell+1$'st secret state of the party.

We refer to an adversary who corrupts $t$ parties $M \subset \mathcal{P}$ and makes up to $\lambda$ leakage queries in each time period as a $(t,\lambda)$-*continual leakage adversary.*

For any adversary $\mathcal{A}$ with auxiliary input $z \in \{0,1\}^*$, any inputs $\{x_i\}_{i=1}^n$, any set of functions $\{f_j\}_{j=1}^p$ chosen (adaptively) by the adversary, and any security parameter $k$, we denote the output of the multi-function MPC protocol $\Pi = (\Pi_{\mathsf{Pre}}, \Pi_{\mathsf{Input}}, \Pi_{\mathsf{Online}})$ by

$$\mathsf{REAL}^{\Pi}_{\mathcal{A},M}\left(1^k, \vec{x}, z, \{f_j\}_{j=1}^p\right).$$

Loosely speaking, we say that a protocol $\Pi$ is a *leakage-resilient* multi-function MPC protocol if any adversary, who corrupts a subset of parties, receives leakage information as described above, and runs the protocol with honest parties on any (unbounded) sequence of functions $f_1, ..., f_p$, gains *no information* about the inputs of the honest parties beyond the output of the functions $f_j(x_1, ..., x_n)$ for $j = 1, ..., p$. We formalize this in the next subsection.

### 4.3.3 Security Definition

In what follows, we formally define our model of security; i.e., what it means for a real-world protocol to emulate the desired ideal world.

**Definition 4.3.2** (Leakage-Resilient MPC). A multi-function evaluation protocol $\Pi = (\Pi_{\mathsf{Pre}}, \Pi_{\mathsf{Input}}, \Pi_{\mathsf{Online}})$ is said to be *$\lambda$-leakage-resilient against $t$ malicious parties* if for every PPT $(t,\lambda)$-continual leakage MPC adversary $\mathcal{A}$ in the real world, there exists a PPT adversary $\mathcal{S}$ corrupting the same parties in the ideal world such that for every input vector $\vec{x}$, every auxiliary input $z$, and any (adaptively chosen) set of functions $\{f_j\}_{j=1}^p$ where $p = \mathsf{poly}(k)$, it holds that

$$\mathsf{IDEAL}_{\mathcal{S},M}\left(1^k, \vec{x}, z, \{f_j\}_{j=1}^p\right) \approx_c \mathsf{REAL}^{\Pi}_{\mathcal{A},M}\left(1^k, \vec{x}, z, \{f_j\}_{j=1}^p\right).$$

Note that we do *not* allow the simulator to request leakage on honest parties' inputs in the ideal world, as was done in [BCH11, DHP11, BGJ+13], and thus model a stronger notion of secrecy than what was achieved in prior works.[9]

[9]With the (necessary) addition of a one-time leak-free preprocessing phase.

## 4.4 Weakly Leakage-Resilient MPC for *Randomized* Functions

In our final leakage-resilient MPC protocol, parties will be required to jointly evaluate randomized functions via a weakly leakage-resilient (WLR) MPC protocol. However, the WLR-MPC protocol from [BGJ$^+$13] is guaranteed to be fully simulatable only for *deterministic* computations. In this section, we describe the challenges in achieving WLR-MPC for randomized functions, and then provide a construction for such functions to be used as a tool within our LR-MPC protocol.

In a setting where leakage is not a concern, there is no significant distinction between MPC protocols for deterministic and randomized functions. Indeed, given a protocol for deterministic functions, one can achieve MPC for any randomized functionality $f$ by having each party submit an additional random string $r_i$ as input to an MPC protocol for the *deterministic* function $f'((x_1, r_1), \dots, (x_n, r_n)) := f(x_1, \dots, x_n; \bigoplus_i r_i)$ that evaluates $f$ using randomness equal to $\bigoplus_i r_i$. However, if the adversary is able to *leak* information on the secret states of honest parties before selecting his strings $r_j$, such direct transformations break down. For example, in the transformation proposed above, the adversary can skew the final combined randomness $\bigoplus_i r_i$ by leaking on the honest parties' strings $r_i$, and then choosing the $r_j$s of corrupted parties adaptively. If the adversary is able to leak information on the joint secret state of honest parties (even a single bit), then one will run into the same problem for *any* possible transformation: indeed, collectively generating unbiased randomness within such a leakage model is simply impossible [CLL$^+$13].

One can circumvent this impossibility by considering real-world adversaries who leak *independently* on the secret states of different honest parties. For example, if the leakage on honest parties is independent and bounded, then replacing the xor function in the example above with a robust multi-source extractor (see Section 4.4.3) will guarantee that the adversary cannot skew the resulting random string. This yields meaningful guarantees for the resulting protocol (e.g., on the correctness of the protocol output distribution) that will likely suffice for a number of applications.

However, for us, the challenge runs even deeper. The primary issue is a subtle, yet crucial requirement of simulation-based security that will be required in order to use the WLR-MPC protocol within a larger protocol. Namely, in our case it is not enough to ensure that the parties evaluate $f$ with unbiased randomness. Rather, the simulator must be able to force a *particular* random string to be used.

To be more explicit, we will require the following real/ideal-world security definition. In the real world, parties communicate via the protocol, and leakage occurs independently on the secret states of honest parties. In the ideal world, parties submit their inputs $x_i$ to a trusted third party. The trusted party samples a uniform random string $R$, responds with the evaluation $f(x_1, \dots, x_n; R)$ of $f$ on the received inputs using randomness $R$, and answers leakage queries on the collection of secret values $(x_i, \dots, x_n, R)$. Now, to achieve security, the simulator in the ideal world must be able to properly simulate the interactions of honest parties to be consistent with this output $f(x_1, \dots, x_n; R)$; In particular, he must simulate consistently with the selected random string $R$. Achieving this notion of security will be crucial for our application, where we wish to use the WLR-MPC protocol within a larger protocol.[10]

[10]Specifically, we will be using the WLR-MPC protocol to execute the modules of a LDS-compiled circuit, and must compose the corresponding simulators appropriately. Given black-box access to the circuit, the LDS simulator will simulate the output and leakage on each LDS module for some fixed choice of randomness $R$. Given this information,

In order to achieve this definition, we will need a protocol that allows parties in the real world to collectively generate randomness in a "fully simulatable" fashion, in the sense that the simulator can "force" any desired outcome $R$ for the coin toss.

In this section, we show how to achieve fully simulatable coin tossing within a setting where leakage in the real world occurs independently on honest parties' secret states, the number of parties is polynomially related to the security parameter, a constant fraction of which are honest, and honest parties have the ability to erase information from their secret state. In turn, we show how to use such a coin tossing protocol to achieve WLR-MPC for randomized functions within the same setting. This is precisely the setting required to plug the WLR-MPC into our final leakage-resilient MPC protocol.

We begin in Section 4.4.1 by formally presenting the required security definition for WLR-MPC for randomized functions. In Section 4.4.2 we formally define the required "fully simulatable" coin tossing protocol. We introduce some tools to be used in the construction in Section 4.4.3. In Section 4.4.4 we present the coin tossing protocol construction and proof. Then in Section 4.4.5, we describe how to use this coin tossing protocol to achieve WLR-MPC for randomized functions.

### 4.4.1 Security Definition: WLR-MPC for Randomized Functionalities

In this section, we formally define the notion of WLR-MPC that will be required for our final LR-MPC protocol construction. Note that the definition which follows is essentially the same as Definition 4.2.17, with the following changes:

- We now consider *randomized* functions $f$. In the ideal world, the trusted party selects the (uniform) randomness $R$ for evaluating $f$, and answers leakage queries on the honest parties' inputs $x_i$ *and* the random string $R$.
- Leakage in the real world is assumed to take place *independently* on the secret state of each honest party.
- Honest parties may now periodically *erase* portions of their secret state. (Note that this is anyway required in the setting of continual leakage).

**Ideal World.** We first describe the ideal world experiment, where $n$ parties $P_1, \dots, P_n$ interact with an ideal functionality for computing a randomized functionality $f$. An adversary may corrupt any subset $M \subset \mathcal{P}$ of the parties. As in the standard MPC ideal world experiment, the parties send their inputs to the ideal functionality and receive the output of $f$ evaluated on all inputs and some uniform random string $R$. The main difference from the standard ideal world experiment is that the adversary is also allowed to make *leakage queries* on the inputs of the honest parties and random string $R$. Such queries are evaluated on the *joint* collection of all parties' inputs and $R$. Formally, the ideal world execution proceeds as follows.

**Inputs:** Each party $P_i$ obtains an input $x_i$. The adversary is given auxiliary input $z$ and selects a subset of parties $M \subset \mathcal{P}$ to corrupt.

the WLR-MPC simulator must be able to simulate the WLR-MPC execution of each module consistent with this output and leakage (and thus with the chosen random string $R$).

**Sending inputs to trusted party:** Each honest party $P_i$ sends its input $x_i$ to the ideal functionality. For each corrupted party $P_i \in M$, the adversary may select any value $x'_i$ and send it to the ideal functionality.

**Trusted party computes output:** Let $x'_1, \ldots, x'_n$ be the inputs that were sent to the ideal functionality. The ideal functionality samples a uniform random string $R$ and computes $f(x'_1, \ldots, x'_n; R)$.

**Adversary learns output:** The ideal functionality first sends the evaluation $f(x'_1, ..., x'_n; R)$ to the adversary. The adversary replies with either continue or abort.

**Honest parties learn output:** If the message is abort, the ideal functionality sends $\perp$ to all honest parties. If the adversary's message was continue, then the ideal functionality sends the function evaluation $f(x'_1, \ldots, x'_n; R)$ to all honest parties.

**Leakage queries on inputs:** The adversary may send (adaptively chosen) leakage queries in the form of efficiently computable functions $L_j$ (described as a circuit). On receiving such a query, the ideal functionality computes $L_j(x'_1, \ldots, x'_n, R)$ and returns the output to the adversary.

**Outputs:** Honest parties output their inputs and the messages they obtained from the ideal functionality. Malicious parties may output an arbitrary PPT function of their initial input (auxiliary input and random-tape) and the message it has obtained from the ideal functionality.

An ideal world adversary $\mathcal{S}$ who obtains a total of $\lambda$ bits of leakage is referred to as a $\lambda$-*leakage ideal adversary.* The overall output of the ideal-world experiment consists of all the inputs and values received by honest parties from the ideal functionality, together with the output of the adversary, and is denoted by $\mathsf{W\text{-}IDEAL}^{f}_{\mathcal{S},M}(1^k, \vec{x}, z)$.

**Real World.** The real-world experiment begins by first choosing a common random string crs. Then, each party $P_i$ receives an input $x_i$ and the adversary $\mathcal{A}$ receives auxiliary input $z$. These values can depend arbitrarily on the crs, but need to be efficiently computable given the crs. However, for the sake of simplicity of notation, throughout this section we assume that these values are independent of the crs.

The adversary $\mathcal{A}$ selects any arbitrary subset $M \subset \mathcal{P}$ of the parties to corrupt. Each corrupted party $P_i \in M$ hands over its input to $\mathcal{A}$. The parties $P_1, \ldots, P_n$ now engage in an execution of a real $n$-party protocol $\Pi$. The adversary $\mathcal{A}$ sends all messages on behalf of the corrupted parties, and may follow an arbitrary polynomial-time strategy. In contrast, the honest parties follow the instructions of $\Pi$. Furthermore, at any point during the protocol execution, the adversary may make leakage queries of the form $(L, i)$ and learn $L(\mathsf{state}_i)$, where $\mathsf{state}_i$ denotes the current secret state of party $P_i$. We allow the adversary to choose the leakage queries adaptively.

Honest parties have the ability to toss fresh coins at any point in the protocol, and at that point these coins are added to the state of that party. In addition, honest parties have the ability to erase portions of their secret state; such information can no longer be accessed in future computations, and is removed from $\mathsf{state}_i$ for future leakage queries. At the conclusion of the protocol execution, each honest party $P_i$ generates an output according to $\Pi$. Malicious parties may output an arbitrary PPT function of the view of $\mathcal{A}$.

An adversary $\mathcal{A}$ who obtains at most $\lambda$ bits of leakage is referred to as a *$\lambda$-independent-leakage real adversary.* Let $\mathsf{Gen}_{\mathsf{WLR}}$ denote the CRS generation algorithm. Further, let $\mathsf{W\text{-}REAL}^{\Pi}_{\mathcal{A}}(1^k, \mathsf{crs}, \vec{x}, z)$ be the random variable that denotes the values output by the parties at the end of the protocol $\Pi$ (using $\mathsf{crs} \leftarrow \mathsf{Gen}_{\mathsf{WLR}}(1^k)$ as the CRS). Then, the overall output of the real-world experiment is defined as the tuple $(\mathsf{crs}, \mathsf{W\text{-}REAL}^{\Pi}_{\mathcal{A},M}(1^k, \mathsf{crs}, \vec{x}), z)$.

**Security Definition.** We now state the formal security definition of WLR-MPC for randomized functions.

**Definition 4.4.1** ($\lambda$-Weakly Leakage-Resilient MPC for Randomized Functionalities). A protocol $\Pi$ evaluating a randomized functionality $f$ is a *$\lambda$-weakly leakage-resilient* MPC protocol if for every PPT $\lambda$-independent-leakage real adversary $\mathcal{A}$, there exists a $\lambda$-leakage ideal adversary $\mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{exec}})$, corrupting the same parties as $\mathcal{A}$, such that for every input vector $\vec{x}$, every auxiliary input $z \in \{0,1\}^*$, and every subset $M \subset \mathcal{P}$, it holds that

$$\left\{\mathsf{crs}, \mathsf{W\text{-}IDEAL}^{f}_{\mathcal{S}^{\mathsf{exec}}(\mathsf{crs},\mathsf{trap}),M}(1^k, \vec{x}, z)\right\}_{k\in N} \approx_c \left\{\mathsf{crs}', \mathsf{W\text{-}REAL}^{\Pi}_{\mathcal{A},M}(1^k, \mathsf{crs}', \vec{x}, z)\right\}_{k\in N},$$

where $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k)$, and $\mathsf{crs}' \leftarrow \mathsf{Gen}(1^k)$.

### 4.4.2 Security Definition: Fully Simulatable Leakage-Resilient Coin Tossing

Converting from an MPC protocol for deterministic functions to one for general randomized functions requires constructing a multi-party coin tossing protocol that is "fully simulatable," in the sense that the simulator can "force" any desired outcome for the coin toss, simulating the protocol appropriately to yield this outcome. We begin by formalizing the desired coin tossing object.

**Ideal Functionality $\mathcal{F}^{\mathsf{rand}}_i$.** We wish to emulate an ideal functionality that samples a random bit and sends it to a *single* party. Formally, define the ideal functionality $\mathcal{F}^{\mathsf{rand}}_i$ that takes no inputs, samples a random bit $b$, and sends $b$ as output *only* to $P_i$. For adversary (simulator) $\mathcal{S}$, denote the output of all parties (including $\mathcal{S}$) in the ideal world experiment by $\mathsf{Ideal}_{\mathcal{F}^{\mathsf{rand}}_i}(\mathcal{S})$.

**Real World Experiment.** Recall that in the real world, the adversary is allowed to leak independently on the entire secret state of each honest party throughout the protocol execution. The output of the experiment consists of the outputs of all parties, where each honest party outputs a value as dictated by the protocol, and each corrupted party can output an arbitrary PPT function of the view of the adversary. For protocol $\Pi_{\mathsf{rand}}$ and adversary $\mathcal{A}$, denote the output of the real world experiment as $\mathsf{Real}_{\Pi_{\mathsf{rand}}}(\mathcal{A})$.

**Security Definition** The desired coin tossing protocol must satisfy the standard simulatability requirement, with respect to the ideal and real world experiments defined above.

**Definition 4.4.2.** Protocol $\Pi_i$ is said to be a *fully simulatable, $\lambda$-strongly leakage-resilient coin tossing protocol* generating a coin toss for party $i$ if for every real world adversary $\mathcal{A}$ who leaks at most $\lambda$ bits, there exists a simulator $\mathcal{S}$ in the ideal world, such that

$$\mathsf{Ideal}_{\mathcal{F}_i^{\mathsf{rand}}}(\mathcal{S}) \overset{c}{\cong} \mathsf{Real}_{\Pi_{\mathsf{rand}}}(\mathcal{A}).$$

Our protocol construction will achieve the above strongly leakage-resilient definition in the case that the party $P_i$ receiving the output bit is a *malicious* party. If the recipient $P_i$ of the random bit is an *honest* party, then this strong leakage-resilient definition is not achievable. This is because the adversary's leakage query may reveal the secret output bit of $P_i$, which cannot be correctly simulated given only the outputs of malicious parties (in this case, the empty strings).

However, in this honest-$P_i$ scenario our protocol will achieve the next best thing. Namely, we can guarantee simulatability of the protocol execution *up to the point* that a leakage query occurs; then, if *at this point* the simulator is given the output bit $b$, the simulation can be *consistently* continued, with the correct output distribution.

More formally, we consider the following slightly weaker definition.

**Ideal Functionality $\mathcal{F}_i^{\mathsf{rand},+L}$.** This ideal functionality takes no inputs, samples a random bit $b$ and sends $b$ as output to a *single* party $P_i$, as before. However, in addition, $\mathcal{F}_i^{\mathsf{rand},+L}$ accepts a leakage query from the adversary and replies with the output bit $b$. For adversary (simulator) $\mathcal{S}$, denote the output of all parties (including $\mathcal{S}$) in the ideal world experiment by $\mathsf{Ideal}_{\mathcal{F}_i^{\mathsf{rand},+L}}(\mathcal{S})$.

**Real World Experiment.** The real world experiment is unchanged.

**Security Definition**

**Definition 4.4.3.** Protocol $\Pi_i$ is said to be a *fully simulatable, $\lambda$-weakly leakage-resilient coin tossing protocol* generating a coin toss for party $i$ if for every real world adversary $\mathcal{A}$ who leaks at most $\lambda$ bits, there exists a simulator $\mathcal{S}$ in the ideal world who may only query $\mathcal{F}_i^{\mathsf{rand},+L}$, such that

$$\mathsf{Ideal}_{\mathcal{F}_i^{\mathsf{rand},+L}}(\mathcal{S}) \overset{c}{\cong} \mathsf{Real}_{\Pi_{\mathsf{rand}}}(\mathcal{A}).$$

Moreover, $\mathcal{S}$ sends its leakage query to $\mathcal{F}_i^{\mathsf{rand},+L}$ only after $\mathcal{A}$ makes a leakage query.

## 4.4.3 Preliminaries

We now formally define the underlying tools that will be used in our construction.

### Extractable Equivocal Commitments

Informally speaking, a bit-commitment scheme is *equivocal* if it satisfies the following additional requirement. There exists an efficient simulator that outputs a fake commitment such that: (1) the commitment can be decommitted to both 0 and 1, and (2) the simulated commitment and decommitment pair is indistinguishable from a real pair. We now formally define the equivocability property for bit-commitment schemes in the CRS model.

The following definition is adapted from [FS89, CIO98].

**Definition 4.4.4.** A non-interactive bit-commitment scheme $(\mathsf{crsGen}, \mathsf{Com}, \mathsf{Rec})$ in the CRS model is said to be an *equivocal bit-commitment scheme in the CRS model* if there exists a PPT simulator algorithm $\mathsf{Sim}_{\mathsf{eq}} = (\mathsf{crsSim}_{\mathsf{eq}}, \mathsf{comSim}_{\mathsf{eq}})$ such that $\mathsf{crsSim}_{\mathsf{eq}}$ takes as input the security parameter $1^k$ and outputs a CRS and trapdoor pair, $(\mathsf{crs}, \mathsf{trap})$; and $\mathsf{comSim}_{\mathsf{eq}}$ takes as input such a pair $(\mathsf{crs}, \mathsf{trap})$ and generates a tuple $(\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1)$ of a commitment string $\mathsf{eqcom}$ and two decommitments $\mathsf{eqdec}^0$ and $\mathsf{eqdec}^1$ (for 0 and 1), such that the following holds.

1. For every $b \in \{0,1\}$ and every $(\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1) \leftarrow \mathsf{comSim}_{\mathsf{eq}}(\mathsf{crs}, \mathsf{trap})$, it holds that
$$\mathsf{Rec}(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}^b) = b.$$

2. For every $b \in \{0,1\}$, the random variables
$$\{(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}) : \mathsf{crs} \leftarrow \mathsf{crsGen}(1^k), (\mathsf{eqcom}, \mathsf{eqdec}) \leftarrow \mathsf{Com}(\mathsf{crs}, b)\}$$
and
$$\{(\mathsf{crs}, \mathsf{eqcom}, \mathsf{eqdec}^b) : (\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{crsSim}_{\mathsf{eq}}(1^k), (\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1) \leftarrow \mathsf{comSim}_{\mathsf{eq}}(\mathsf{crs}, \mathsf{trap})\}$$
are computationally indistinguishable.

**Reusable CRS.** Note that the simulator algorithms $\mathsf{crsSim}_{\mathsf{eq}}$ and $\mathsf{comSim}_{\mathsf{eq}}$ are described as separate algorithms in the Definition 4.4.4 to highlight that it is not necessary to create a separate CRS for every equivocal commitment, i.e., the CRS is *reusable*. In this case, Definition 4.4.4 can be extended in a straightforward manner to consider indistinguishability of an honestly generated tuple consisting of CRS and polynomially many commitment, decommitment pairs, from a simulated tuple. Explicitly, we will use the following property.

**Corollary 4.4.5.** *For any polynomial $p(k)$, and any collection of bits $\{b_1, ..., b_{p(k)}\}$, the following distributions are indistinguishable:*
$$\left\{\left(\mathsf{crs}, \{\mathsf{eqcom}_i, \mathsf{eqdec}_i\}_{i=1}^{p(k)}\right) : \mathsf{crs} \leftarrow \mathsf{crsGen}(1^k), (\mathsf{eqcom}_i, \mathsf{eqdec}_i) \leftarrow \mathsf{Com}(\mathsf{crs}, b_i)\right\}$$
*and*
$$\left\{\left(\mathsf{crs}, \{\mathsf{eqcom}_i, \mathsf{eqdec}_i^{b_i}\}_{i=1}^{p(k)}\right) : \begin{array}{l}(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{crsSim}_{eq}(1^k) \\ (\mathsf{eqcom}_i, \mathsf{eqdec}_i^0, \mathsf{eqdec}_i^1) \leftarrow \mathsf{comSim}_{eq}(\mathsf{crs}, \mathsf{trap})\end{array}\right\}.$$

*Proof.* Follows from Definition 4.4.4 by a standard hybrid argument. □

**Theorem 4.4.6** ([FS89, CLOS02]). *Assuming the existence of one-way functions, there exists an equivocal bit commitment scheme in the (reusable) CRS model.*

**Remark 4.4.7.** Throughout this paper, we assume that our equivocal commitment scheme has two additional properties: namely, that the decommitment information eqdec contains *all randomness* used to generate the corresponding commitment, and that the commitment scheme is *statistically binding*. Indeed, the scheme of Feige and Shamir [FS89] satisfies these additional properties.

**Definition 4.4.8.** An *extractable* equivocal commitment scheme in the CRS model is an equivocal commitment scheme $(\mathsf{crsGen}, \mathsf{Com}, \mathsf{Rec}, \mathsf{Sim}_{\mathsf{eq}})$ (as in Definition 4.4.4) with additional algorithms $(\mathsf{crsGen}_E, E)$ such that

$$\{\mathsf{crs} \leftarrow \mathsf{crsGen}_E(1^k)\} \overset{c}{\cong} \{\mathsf{crs} \leftarrow \mathsf{crsGen}(1^k)\},$$

and for all PPT adversaries $\mathcal{A}$,

$$\Pr[(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{crsGen}_E(1^k); \mathsf{com} \leftarrow \mathcal{A}(\mathsf{crs}); y \leftarrow E(\mathsf{crs}, \mathsf{com}, \mathsf{trap}) : \exists\ \mathsf{dec}, x \neq y \\ \text{s.t. } x = \mathsf{Rec}(\mathsf{crs}, \mathsf{dec}, \mathsf{com})] \leq \mathsf{negl}(k).$$

**Remark 4.4.9.** For our purposes, we want an extractable equivocal commitment scheme for *strings*. Note that such a scheme can be easily constructed by simply repeating the above bit commitment scheme in *parallel* (as was done explicitly in the previous sections). In this section, we abbreviate this notation, and denote a commitment to a string of length $m$ as a *vector* $\mathsf{eqcom} = (\mathsf{eqcom}_1, ..., \mathsf{eqcom}_m)$, with corresponding decommitment vector $\mathsf{eqdec} = (\mathsf{eqdec}_1, ..., \mathsf{eqdec}_m)$. The simulator algorithm $\mathcal{S}^{\mathsf{com}}$ produces a commitment vector and a *pair* of decommitment vectors $d^0 = (\mathsf{eqdec}_1^0, ..., \mathsf{eqdec}_m^0)$, $d^1 = (\mathsf{eqdec}_1^1, ..., \mathsf{eqdec}_m^1)$. A decommitment to any particular bit string $a = (a, ..., a_m)$ can be formed by selecting the appropriate decommitment values $(\mathsf{eqdec}_1^{a_1}, \ldots, \mathsf{eqdec}_n^{a_m})$, and will be denoted by $d^a$.

Without loss of generality, we assume that the decommitment information is simply the randomness used for commitment.

In [GOS06a], it is shown how to construct an extractable equivocal commitment scheme from any equivocal commitment scheme, given any encryption scheme with pseudorandom ciphertexts. In particular, the following statement holds.

**Theorem 4.4.10** ([GS08, GOS06a, FS89]). *There exists an extractable equivocal commitment scheme in the CRS model, based on any trapdoor permutation.*

## Robust Multi-Source Extractor

A multi-source extractor takes as input several independent sources, each with sufficient amount of entropy, and outputs a string that is statistically close to uniform. In this work, we need a multi-source extractor that extracts randomness even if some of the sources are "malicious," but independent of the "honest" ones. Such an extractor, which we refer to as a robust multi-source extractor, was constructed by Kamp, Rao, Vadhan and Zuckerman [KRVZ06]. The notion of entropy that is used is *min-entropy.* A random variable $X \subseteq \{0,1\}^n$ is said to have min entropy $k$, denoted by $H_\infty(X) = k$, if for every $x \in \{0,1\}^n$, $\Pr[X = x] \leq \frac{1}{2^k}$, and is said to have min-entropy rate $\alpha$ if $H_\infty(X) \geq \alpha n$.

**Theorem 4.4.11** ([KRVZ06]). *For any constant $\delta > 0$ and every $n \in \mathbb{N}$, there is a polynomial-time computable robust multi-source extractor* $\mathsf{Ext} : \left(\{0,1\}^d\right)^n \to \{0,1\}^m$ *that takes as input $n$ independent sources, each in $\{0,1\}^d$, and produces an $m$-bit string that is $\epsilon$-close to uniform, as long as the min-entropy rate of the combined sources is $\delta$, and where $m = 0.99\delta nd$ and $\epsilon = 2^{-\Omega((nd)/\log^3(nd))}$.*

### 4.4.4 Construction: Fully Simulatable Leakage-Resilient Coin Tossing.

We now present the desired coin tossing protocol, $\Pi_{i^*}^{\mathsf{rand}}$. We describe the protocol for generating a single random (secret) output bit; an analogous protocol for *strings* can be constructed by repeating the one-bit protocol sequentially.

Our construction is in the CRS model (for the use of extractable equivocal commitments).

At a high level, the protocol works as follows. The parties begin by collectively generating $k^2$ random candidate bits, one of which will eventually be selected as the output. Each bit is generated by having each party $P_i$ sample and commit to a secret random string $r_i$; although the $r_i$'s are not revealed, they implicitly define a bit $b = \mathsf{Ext}(r_1, ..., r_{|\mathcal{E}_{\mathsf{rand}}|})$, where $\mathsf{Ext}$ is a robust multi-source extractor (see Section 4.4.3).

Next, the parties collectively select one of these candidate bits as the final output. This is done by eliminating random candidates one at a time. In each iteration, the parties collectively generate a random index $\ell \in [k^2]$ in the same manner as above (i.e., by each sampling a random string and committing, and then extracting a random index $\ell$ from all the strings), and the $\ell$th candidate is eliminated.

A formal description of $\Pi_i^{\mathsf{rand}}$ is given in Figure 4.1.

**Theorem 4.4.12.** *Suppose* $(\mathsf{Com}, \mathsf{Rec})$ *is an extractable equivocal commitment scheme in the CRS model and* $\mathsf{Ext}$ *is a robust randomness extractor. Then for any malicious adversary* $\mathcal{A}$ *corrupting* $(1-\epsilon)$ *fraction of all parties,* $\Pi_{\mathsf{rand}}$ *is a* $\lambda$*-weakly leakage-resilient fully simulatable coin tossing protocol in the CRS model, as in Definition 4.4.3, for leakage parameter* $\lambda = k$.

**Remark 4.4.13.** The protocol $\Pi_{i^*}^{\mathsf{rand}}$ in fact securely emulates a somewhat stronger ideal functionality, which not only outputs a random bit $b$ to party $i^*$, but also outputs a commitment to $b$ to all parties (using an extractable equivocal commitment scheme), and provides party $i^*$ with the corresponding decommitment information. Namely, it emulates the functionality $F_{i^*,\mathsf{com}}^{\mathsf{rand},+L}$ which has party $P_{i^*}$'s CRS value $\mathsf{crs}_{i^*}$ hardcoded, and does the following:

Functionality $F_{i^*,\mathsf{com}}^{\mathsf{rand},+L}$

Input: None.

Compute:

1. Sample a random bit $b \leftarrow \{0, 1\}$.
2. Generate an equivocal commitment to $b$ with respect to $P_{i^*}$'s CRS: i.e., $(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{Com}(\mathsf{crs}_{i^*}, b)$.

Output: To party $P_{i^*}$: $b, \mathsf{decom}$. To all parties: $\mathsf{com}$.

Indeed, recall that in $\Pi_i^{\mathsf{rand}}$ (see Figure 4.1), all parties except $P_{i^*}$ open their contribution to the selected candidate bit $\ell^*$, while $P_{i^*}$'s contribution $r_{i^*}^{\ell^*}$ to the bit remains secret. All parties hold a commitment $\mathsf{com}_{i^*}^{\ell^*}$ to $r_{i^*}^{\ell^*}$ with respect to $\mathsf{crs}_{i^*}$, and party $P_{i^*}$ holds both the value of $r_{i^*}^{\ell^*}$ and the corresponding decommitment information $\mathsf{decom}_{i^*}^{\ell^*}$.

**Coin Toss $\Pi_{i^*}^{\mathsf{rand}}$:**
Party $P_i$ performs the following steps:

1. (Generate $k^2$ candidate output bits)
   Repeat for $\ell = 1, ..., k^2$:

   Sample $r_i^\ell \leftarrow \{0,1\}^*$.
   Publish commitment $\mathsf{com}_i^\ell$ where $(\mathsf{com}_i^\ell, \mathsf{decom}_i^\ell) \leftarrow \mathsf{Com}(1^k, \mathsf{crs}_i, r_i^\ell)$.

2. (Iteratively eliminate output candidates until one remains)
   Repeat for $m = k^2, (k^2 - 1), ..., 2$:

   (a) Sample a random string $\mathsf{elim}_i^m \leftarrow \{0,1\}^*$. Publish commitment $c_i^m$ where
   $$(c_i^m, d_i^m) \leftarrow \mathsf{Com}(1^k, \mathsf{crs}_i, \mathsf{elim}_i^m).$$

   (b) After receiving a commitment from each party, reveal $\mathsf{elim}_i$ to all parties by sending $d_i^m$.

   (c) If any party's $\mathsf{elim}_{i'}^m$ does not agree with its published commitment $c_{i'}^m$, then $\mathsf{abort}$.

   (d) Otherwise, compute $\mathsf{elim}^m = \mathsf{Ext}(\mathsf{elim}_1^m, ..., \mathsf{elim}_{|\mathcal{E}_{\mathsf{rand}}|}^m) \in [m]$. (Note that the number of remaining candidates to choose from decreases in each iteration). The candidate output with index $\mathsf{elim}^m$ is eliminated.

3. Once all but one candidate $\ell^* \in [k^2]$ has been eliminated:

   (a) If $P_i$ is *not* the designated coin toss recipient $P_{i^*}$, then reveal the $\ell^*$th share $r_i^{\ell^*}$ generated in Step 1, by publishing $\mathsf{decom}_i^\ell$.

   (b) If $P_i$ is the designated recipient $P_{i^*}$, then wait until all parties have revealed their shares. If any $r_i^{\ell^*}$ does not agree with the corresponding commitment $\mathsf{com}_i^{\ell^*}$, then $\mathsf{abort}$. Otherwise, output
   $$b = \mathsf{Ext}(r_1^{\ell^*}, ..., r_{|\mathcal{E}_{\mathsf{rand}}|}^{\ell^*}).$$

**Figure 4.1:** The protocol $\Pi_{i^*}^{\mathsf{rand}}$ for generating a random bit known only to party $P_{i^*}$.

Thus, in the protocol $\mathsf{com}_{i^*}^{\ell^*}$ corresponds to the desired output commitment $\mathsf{com}$ (given to all parties), and $\mathsf{decom}_{i^*}^{\ell^*}$ corresponds to the desired output decommitment information $\mathsf{decom}$ (given only to party $P_{i^*}$).

**Overview of Simulator $\mathcal{S}$.** Consider Step 1 of the protocol, consisting of $k^2$ "generation phases" for candidate output bits $b^1, ..., b^{k^2}$. We say that the adversary leaks *during* a generation phase if he makes a leakage query after the honest parties have committed to their secret random string but before at least one corrupted party commits to his own string.

If the adversary leaks on the honest parties *during* the generation phase of some value $b^\ell$, then the simulator cannot necessarily force a desired output for $b^\ell = \mathsf{Ext}(r_1^\ell, ..., r_|^\ell \mathcal{E}_{\mathsf{rand}}|)$. This is because the simulator's simulated responses to leakage queries on honest parties' states (including their inputs $r_i^\ell$) may tie him down to specific values (for example, if the adversary leaks a collision-resistant hash applied to a party's secret state, it will be infeasible for the simulator to find any alternative preimage). However, these $r_i^\ell$ values must be chosen by the simulator before the corrupted parties $P_j$ choose their inputs $r_j^\ell$ to the generation procedure, and thus the extracted output $\mathsf{Ext}(r_1^\ell, ..., r_{|\mathcal{E}_{\mathsf{rand}}|}^\ell)$ cannot be fixed a priori. Hence, when this situation occurs, our simulator will take this generation phase as a loss and simply choose random values $r_i^\ell$ for the inputs of honest parties. We will refer to such generation phases as *bad*.

However, if the adversary does not leak during a particular $b^\ell$ generation phase, then the simulator can extract the inputs $r_j^\ell$ of the corrupt parties and choose the $r_i^\ell$'s for the honest parties to force any desired output value $b^\ell$. Since the adversary cannot leak the entire $r_i^\ell$'s of all honest parties (and thus the $r_i^\ell$'s are independent entropy sources), $b^\ell = \mathsf{Ext}(r_1^\ell, ..., r_{|\mathcal{E}_{\mathsf{rand}}|}^\ell)$ still appears uniform, and so the adversary cannot distinguish between the case where the $r_i$s are sampled truly at random and the case where they are sampled by the simulator as above. We refer to these generation phases as *good*.

The simulator will use the latter technique to "cheat" in some of the generation phases. Namely for each good generation phase in Step 1, the simulator $\mathcal{S}$ will force the output $b^\ell$ to be the *target* output bit. Note that in the case that the party $P_i$ receiving the output is honest, then the target bit $b_T$ is not explicitly known by the simulator a priori. However, the simulator remains able to force both $b^\ell = 0$ and $b^\ell = 1$ until the point when a leakage query is made by the adversary, at which point the simulator can learn the target $b_T$ by making a leakage query to his own oracle.

In Step 2, the simulator will follow a similar cheating strategy to ensure that the undesired bit $(1 - b_T)$ is never the final output. For each good elimination round (during which the adversary does not leak), the simulator will force the selected index $\ell$ to come from a *skewed* distribution, so that on one hand the distribution is indistinguishable from random in the eyes of the adversary, and on the other hand it eliminates all undesirable candidates $(1 - b_T)$ (if such exist). More explicitly, the simulator classifies the candidate bits $b^\ell = \mathsf{Ext}(b_1^\ell, ..., b_{|\mathcal{E}_{\mathsf{rand}}|}^\ell)$ into three different types, described in Figure 4.2.

With high probability, the simulator will be able to force a skewed distribution in which all candidates of Type BB are eliminated. Note that the adversary cannot distinguish BB candidates from BG, since he can only leak independently on each party's random string $r_i^\ell$ (and thus the extracted output bit still appears uniform). Hence, the simulator chooses a random candidate to eliminate, and if it is BG he replaces it with a random BB candidate, which will be undetectable

**Types of candidate bits:**

- **Type G:** Those resulting from "good" generation phases. (The simulator can force these to be the desired output $b_T$).
- **Type BG:** Those resulting from "bad" generation phases, but which happen to result in the good output $b_\ell = b_T$.
- **Type BB:** Those resulting from "bad" generation phases that have the bad output $b_\ell = (1 - b_T)$.

**Figure 4.2:** Characterizations of candidate bits $b^\ell = \mathsf{Ext}(b_1^\ell, ..., b_{|\mathcal{E}_{\mathsf{rand}}|}^\ell)$.

by the adversary.

We now proceed to prove Theorem 4.4.12.

*Proof of Theorem 4.4.12.* We first explicitly construct the simulator $\mathcal{S}$ and then prove that it simulates correctly.

**The simulator $\mathcal{S}$:**

0. Denote the set of corrupted parties by $C$. Simulate the parties' CRS values for the extractable equivocal commitment scheme as follows. For each honest party $P_i$, $i \notin C$, sample $\mathsf{crs}_i \leftarrow \mathsf{crs}\mathcal{S}_{\mathsf{eq}}(1^k)$ using the simulation algorithm. For each corrupt party $P_j$, $i \in C$, sample $\mathsf{crs}_j \leftarrow \mathsf{crsGen}(1^k)$ honestly.

1. The simulator $\mathcal{S}$ simulates the honest parties in Step 1 as follows. Let $\mathsf{BAD} = \emptyset$ and set $\mathsf{state}_i = \emptyset$ for each honest party $P_i$.

   For each $\ell = 1, ..., k^2$, and for each honest party $P_i$, generate a *simulated* equivocal commitment (allegedly to a random string $r_i$),

   $$(\mathsf{com}_i^\ell, \mathsf{decom}_i^{\ell,0}, \mathsf{decom}_i^{\ell,1}) \leftarrow \mathsf{Sim}_{\mathsf{eq}}(1^k, \mathsf{crs}_i^{\mathsf{com}}, \mathsf{trap}_i),$$

   and publish $\mathsf{com}_i^\ell$. (Recall that these values are each *vectors* of commitment/decommitment information).

   If the adversary makes a leakage query $(i, L)$, then do the following.

   - Query the leakage oracle to learn the desired target output bit, $b_T$.
   - If the leakage query occurs *after* the simulator has published the (simulated) commitments $\mathsf{com}_i^\ell$ for honest parties, but *before* all commitments for corrupted parties have been published for this $\ell$, then

     (a) Add index $\ell$ to the set $\mathsf{BAD}$. This candidate will be either Type BG or BB (not G).

(b) For each honest party $P_i$, sample a *random* value $r_i^\ell$ and update the secret state of $P_i$ as $\mathsf{state}_i \leftarrow \mathsf{state}_i \cup (r_i^\ell, \mathsf{decom}_i^{\ell, r_i^\ell})$.[11]

- For every $1 \leq \ell' < \ell$ for which the honest parties' strings $r_i^{\ell'}$ have not yet been chosen by the simulator, do the following:

  (a) For each corrupted party $P_j$, extract $r_j^{\ell'}$ from the adversary's commitment,

$$r_j^{\ell'} \leftarrow E(\mathsf{crs}_j^{\mathsf{com}}, \mathsf{com}_j^{\ell'}, \mathsf{trap}_j^E).$$

  (b) Sample random values $\{r_i^{\ell'}\}_{i \notin C}$ for the honest parties, subject to the constraint

$$b_T = \mathsf{Ext}(r_1^{\ell'}, \ldots, r_{|\mathcal{E}_{\mathsf{rand}}|}^{\ell'}).$$

  (Since the output value is a single bit, this can be done, e.g., by rejection sampling). Update the secret state of each honest party $P_i$ as $\mathsf{state}_i \leftarrow \mathsf{state}_i \cup (r_i^{\ell'}, \mathsf{decom}_i^{\ell', r_i^{\ell'}})$.

2. In Step 2, the simulator $\mathcal{S}$ determines a skewed distribution of eliminations, as follows. $\mathcal{S}$ begins by partitioning the indices $[k^2]$ into two bins, $\mathsf{Bin}_1$ and $\mathsf{Bin}_2$. The simulator will cheat (when he can) by eliminating indices in $\mathsf{Bin}_1$ first and then indices in $\mathsf{Bin}_2$. Recall the three types of candidate bit generation phases G, BG, and BB (see Figure 4.2). The indices $\ell \in [k^2]$ are partitioned into the two bins as follows:

   (a) Type BB indices are placed in $\mathsf{Bin}_1$.

   (b) Type BG indices are placed in $\mathsf{Bin}_2$.

   (c) Each type G index is assigned to either $\mathsf{Bin}_1$ or $\mathsf{Bin}_2$ at random.

   We note that at this point the simulator knows for each index $\ell \in [k^2]$ whether it is of type BB, BG, or G. This is because we are in one of two cases: either all candidates are of type G (if the adversary did not leak during any of the candidate bit generation phases), or the adversary requested a leakage query during at least one of the candidate generation phases, at which time the simulator will have learned the target bit $b_T$ from his own leakage oracle.

   $\mathcal{S}$ now simulates the actions of honest parties during the eliminations. Initialize $\mathsf{BAD}_{\mathsf{elim}} = \emptyset$. This set will identify the collection of *bad* elimination rounds (i.e., elimination rounds during which the adversary leaks). The simulator $\mathcal{S}$ iterates the following steps for $m = k^2, (k^2-1), \ldots, 2$, until a single candidate $\in [k^2]$ remains:

   For each honest party $P_i$, generate a *simulated* equivocal commitment (allegedly to a random string $\mathsf{elim}_i^m$),

$$(c_i^m, d_i^{m,0}, d_i^{m,1}) \leftarrow \mathsf{Sim}_{\mathsf{eq}}(1^k, \mathsf{crs}_i^{\mathsf{com}}, \mathsf{trap}_i),$$

   and publish $c_i^m$.

   - If the adversary requests leakage at this point, then

[11]Recall that the decommitment information is assumed to contain all randomness used in generating the commitment (see Remark 4.4.7).

(a) Add index $m$ to the set $\mathsf{BAD}_{\mathsf{elim}}$. In this $m$th elimination step, the simulator will not be able to cheat.

(b) For each honest party $P_i$, (honestly) sample a random value $\mathsf{elim}_i^m$, update the secret state of $P_i$ as $\mathsf{state}_i \leftarrow \mathsf{state}_i \cup (\mathsf{elim}_i^m, d_i^{m,\mathsf{elim}_i^m})$, and send $\mathsf{elim}_i^m$ on behalf of $P_i$.

- Otherwise, assuming no leakage occurs before the corrupted parties each publish their commitments, perform the following steps:

(a) For each corrupted party $P_j$, extract $\mathsf{elim}_j^m$ from the adversary's commitment,

$$\mathsf{elim}_j^m \leftarrow E(\mathsf{crs}_j^{\mathsf{com}}, c_j^m, \mathsf{trap}_j^E).$$

(b) The simulator will now sample values for $\mathsf{elim}_i^m$ for honest parties $P_i$ to yield a skewed distribution on the resulting candidate index. Namely, he will begin by eliminating random candidates from $\mathsf{Bin}_1$, and then once $\mathsf{Bin}_1$ is entirely eliminated he will eliminate random candidates from $\mathsf{Bin}_2$. This is done as follows:

i. Choose a random index $\ell \in \mathsf{Bin}_1$. If $\mathsf{Bin}_1 = \emptyset$, then choose $\ell \in \mathsf{Bin}_2$ at random.

ii. Sample random values $\{\mathsf{elim}_i^m\}_{i \notin C}$ for the honest parties, subject to the constraint

$$\ell = \mathsf{Ext}(\mathsf{elim}_1^m, \ldots, \mathsf{elim}_{|\mathcal{E}_{\mathsf{rand}}|}^m).$$

(Since the extractor has short output length $\log(k^2)$, this can be done, e.g., by rejection sampling).

iii. Update the secret state of each honest party $P_i$ as $\mathsf{state}_i \leftarrow \mathsf{state}_i \cup (\mathsf{elim}_i^m, d_i^{m,\mathsf{elim}_i^m})$.

**Proof of Indistinguishability** We consider a sequence of intermediate hybrids. The output of each hybrid experiment consists of the outputs of all parties, where honest parties output in accordance with the dictated protocol, and malicious parties may output any efficiently computable function of the view of the adversary. For every Hybrid $t$ adversary $\mathcal{A}_t$ with auxiliary input $z \in \{0,1\}^*$, we denote the output of the corresponding hybrid $t$ experiment by

$$\mathsf{HYB}_t\left(\mathcal{A}_t, 1^k, z\right).$$

We now go through the hybrids one by one, beginning with Hybrid 0. In each step we show that for any adversary $\mathcal{A}_t$ running in Hybrid $t$, there exists an adversary $\mathcal{A}_{t+1}$ running in Hybrid $(t+1)$ such that

$$\mathsf{HYB}_t\left(\mathcal{A}_t, 1^k, z\right) \overset{c}{\cong} \mathsf{HYB}_{t+1}\left(\mathcal{A}_{t+1}, 1^k, z\right).$$

**Hybrid 0** The real-world execution of $\Pi_{\mathsf{rand}}$ with adversary $\mathcal{A}$, who leaks up to $k$ bits on the secret state of honest parties.

**Hybrid 1** Simulate the $\mathsf{crs}_i^{\mathsf{com}}$'s for honest parties $P_i$ using the equivocation simulator, and give them the trapdoor $\mathsf{trap}_i$. Simulate the $\mathsf{crs}_i^{\mathsf{com}}$ for corrupted parties $P_i$ using the extraction simulation algorithm.

**Claim 4.4.14.** $\mathsf{HYB}_0\left(\mathcal{A}_0, 1^k, z\right) \overset{c}{\cong} \mathsf{HYB}_1\left(\mathcal{A}_1, 1^k, z\right)$.

*Proof.* This holds by the indistinguishability of simulated CRS values for the equivocal commitment scheme, together with a standard hybrid argument. □

**Hybrid 2** Same as Hybrid 1, except that during Step 1 and Step 2 every time an honest party $P_i$ is supposed to sample a random value $u$ (corresponding to $r_i^\ell$ or $\mathsf{elim}_i^m$) and commit to it, he now samples $u$ as before, but generates a *simulated* commitment

$$(c, d^0, d^1) \leftarrow \mathsf{Sim}_{\mathsf{eq}}(1^k, \mathsf{crs}_i^{\mathsf{com}}, \mathsf{trap}_i).$$

The party publishes the commitment $c$ and updates his secret state as $\mathsf{state}_i \leftarrow \mathsf{state}_i \cup (u, d^u)$.

**Claim 4.4.15.** $\mathsf{HYB}_1\left(\mathcal{A}_1, 1^k, z\right) \overset{c}{\cong} \mathsf{HYB}_2\left(\mathcal{A}_2, 1^k, z\right)$.

*Proof.* This holds by the indistinguishability of simulated equivocal commitments, together with a standard hybrid argument. □

**Hybrid 3** Same as Hybrid 2, except that the experiment immediately ends in $\mathsf{fail}$ if a corrupted party $P_j$ provides decommitment information $(\mathsf{value}', \mathsf{dec})$ that "opens" any of his previous commitments $\mathsf{com}$ to any value $\mathsf{value}'$ *not equal* to the corresponding extracted value,

$$\mathsf{value} = E(\mathsf{crs}_j^{\mathsf{com}}, \mathsf{com}, \mathsf{trap}_j^E).$$

**Claim 4.4.16.** $\mathsf{HYB}_2\left(\mathcal{A}_2, 1^k, z\right) \equiv \mathsf{HYB}_3\left(\mathcal{A}_3, 1^k, z\right)$.

*Proof.* This holds by the extraction property of the extractable equivocal commitment scheme, together with a standard hybrid argument. □

**Hybrid 4** Similar to Hybrid 3, except that the secret states of the honest parties in Step 1 are updated differently, and leakage queries are answered with respect to the simulated states.

In the rounds of Step 1 where the adversary leaks, there is no change—the simulation is performed honestly. In rounds without leakage, the simulator "cheats" and sets the secret states of honest parties in such a way to force the target output bit $b_T$ as much as possible. Note that in some cases, if the party $P_\alpha$ receiving the output bit is honest, then $b_T$ is not explicitly known to the simulator at this point. However, the simulator will maintain the simulated secret state of parties as an efficient function of $b_T$, and at the point that a leakage query is made by the adversary (at which time the simulator must have concrete values for the secret states that he can provide leakage on), the simulator will query the target bit $b_T$ from his leakage oracle.

Explicitly, for each $\ell$ for which the adversary doesn't leak, the simulator does the following:

- Extract the shares $r_j^\ell$ of corrupted parties as

$$r_j^\ell = E(\mathsf{crs}_j^{\mathsf{com}}, \mathsf{com}_j^\ell, \mathsf{trap}_j^E).$$

- Choose random $\{r_i^\ell\}_{i \notin C}$ for honest parties, subject to $\mathsf{Ext}(r_1^\ell, \ldots, r_{|\mathcal{E}_{\mathsf{rand}}|}^\ell) = b_T$.
- Update the secret state of each honest party $P_i$ as $\mathsf{state}_i \leftarrow \mathsf{state}_i \cup (r_i^\ell, \mathsf{decom}_i^{\ell, r_i^\ell})$, where $(\mathsf{com}_i^\ell, \mathsf{decom}_i^{\ell,0}, \mathsf{decom}_i^{\ell,1})$ was the simulated equivocal commitment triple generated in Step 1 for index $\ell$.

**Claim 4.4.17.** $\mathsf{HYB}_3\left(\mathcal{A}_3, 1^k, z\right) \equiv \mathsf{HYB}_4\left(\mathcal{A}_4, 1^k, z\right)$.

*Proof.* Clearly if all the simulated shares of multiple rounds $\ell$ are revealed, then this cheating will be detected, since the collection of output values will be unnaturally biased toward the same value $b_T$. However, in our case this cheating simulation will go unnoticed, since the only information the adversary learns about the simulated shares is (1) limited leakage on individual shares, and (2) the complete set of shares $\{r_i^\ell\}_{i \notin C}$ for *one* value of $\ell$. We now argue that this information on the simulated shares is actually statistically indistinguishable from the corresponding honest distribution.

First, consider leakage on the shares. Let $\mathsf{shares}^\ell = \{r_i^\ell\}_{i \notin C}$ be the distribution of honestly generated shares for all honest parties $i \notin C$ in round $\ell$. Denote by $\mathsf{shares}^\ell{}_{\mathsf{Sim}}(b_T)$ the corresponding distribution of simulated shares, as a function of the target bit $b_T$. We first argue that, for a given round $\ell$, the distribution of simulated leakage is statistically close to the distribution of honestly generated leakage. Let $\mathsf{Leak}(\mathsf{shares}^\ell)$ and $\mathsf{Leak}(\mathsf{shares}^\ell_{\mathsf{Sim}}(b_T))$ denote the outputs of a leakage attack on the corresponding sets of shares. Recall that $\mathsf{Leak}(\mathsf{shares})$ includes adaptively chosen, but *independent* leakage on each of the individual shares, and has output length bounded by $k$ bits. This means the shares $r_i^\ell$ remain independent sources that together have at least $N - k - \log^2 k$ bits of entropy with overwhelming probability. Thus, by the properties of the robust multi-source extractor (see Theorem 4.4.11), we have that, even conditioned on leakage, the distribution of the extracted bit in the honest case is statistically close to the 1-bit uniform distribution:

$$\{\mathsf{Ext}(r_1^\ell, \ldots, r_{|\mathcal{E}_{\mathsf{rand}}|}) \mid \mathsf{Leak}(\mathsf{shares}^\ell)\} \overset{s}{\cong} U_1,$$

Equivalently, the distribution of leakage on shares yielding extracted output 0 is statistically close to the corresponding distribution on shares yielding extracted output 1. Now, the distribution of simulated shares corresponds exactly to one of these (for whichever value is $b_T$). The honest distribution of shares is simply formed by randomly sampling from one of these two distributions—i.e., yielding a random extracted output. Thus, it holds that the leakage on the simulated shares is statistically close to the same leakage on honestly generated shares.

This relation also extends to multiple rounds $\ell$, by a straightforward hybrid argument.

So the only problem is if the honest parties must ever reveal these shares in their entirety. In our setting, however, one of two events will occur:

- If the final chosen index $\ell$ corresponds to a "bad" round, in which the simulator acted honestly, then the cheating shares will never be revealed, and the revealed distribution of shares will be exactly correct.

- In the other case, the simulator must reveal the complete shares only for *one* of the cheating instances. But, the distribution of a single set of simulated shares actually has the correct distribution, since by itself the forced output $b_T$ of the extraction is uniformly distributed.

$\square$

**Hybrid 5** Same as Hybrid 4, except that in the elimination phase, instead of choosing truly random strings $\mathsf{elim}_i^m$ (as prescribed by the protocol), the strings of honest parties are chosen as follows.

- First, choose a *random* ordering $(\ell_1, ..., \ell_{k^2})$ of the elements of $[k^2]$.
- For each round $m$ of elimination in which the adversary does not leak, the simulator chooses $\mathsf{elim}_i^m$'s in such a way to *force* the resulting eliminated index to be the first value $\ell_{m'}$ of the ordering $(\ell_1, ..., \ell_{k^2})$ that has not yet been eliminated. That is, he extracts the $\mathsf{elim}_j^m$'s of corrupted parties as

$$\mathsf{elim}_j^m = E(\mathsf{crs}_j^{\mathsf{com}}, c_j^m, \mathsf{trap}_j^E),$$

and then chooses random $\{\mathsf{elim}_i^m\}_{i \notin C}$ subject to the constraint $\mathsf{Ext}(\mathsf{elim}_1^m, ..., \mathsf{elim}_{|\mathcal{E}_{\mathsf{rand}}|}^m) = \ell_{m'}$ for the first of $\ell_1, \ell_2, ...$ that has not already been eliminated.
- For the rounds of elimination in which the adversary *does* leak, the simulator selects $\mathsf{elim}_i^m$ at random (honestly).

**Claim 4.4.18.** $\mathsf{HYB}_4\left(\mathcal{A}_4, 1^k, z\right) \overset{s}{\cong} \mathsf{HYB}_5\left(\mathcal{A}_5, 1^k, z\right)$.

*Proof.* In Hybrid 4, the strings $\{\mathsf{elim}_i^m\}_{i \notin C}$ are selected at random, and the output is computed as $\mathsf{elim}^m = \mathsf{Ext}(\mathsf{elim}_1^m, ..., \mathsf{elim}_{|\mathcal{E}_{\mathsf{rand}}|}) \in [m]$. In Hybrid 5, (whenever possible) the simulator chooses the *output value* $\mathsf{elim}^m \in [m]$ first at random, and then samples $\{\mathsf{elim}_i^m\}_{i \notin C}$ randomly, subject only to the condition that they yield (the random) output $\mathsf{elim}^m$.

But, by the properties of the robust multi-source extractor, the honest output distribution $\mathsf{Ext}(\mathsf{elim}_1^m, ..., \mathsf{elim}_{|\mathcal{E}_{\mathsf{rand}}|})$ is itself statistically close to uniform, given the view of the adversary. Thus, first sampling a uniform output $\in [m]$ and then reverse sampling yields a distribution that is statistically close.

$\square$

**Hybrid 6** Exactly the same as Hybrid 5, except that instead of choosing a *random* ordering $(\ell_1, ..., \ell_{k^2})$ to be forced during the elimination phase (as much as possible), the ordering $(\ell_1, ..., \ell_{k^2})$ to be forced is selected as follows:

- First, all indices $\ell \in [k^2]$ are partitioned into two bins: $\mathsf{Bin}_1, \mathsf{Bin}_2$. All indices of Type BB are placed in $\mathsf{Bin}_1$, and all indices of Type BG are placed in $\mathsf{Bin}_2$. Each index of Type G is assigned a bin at random. (Note that the type of each index is fully determined at this point of the execution).

- The ordering $(\ell_1, ..., \ell_{k^2})$ is selected by choosing elements of $\mathsf{Bin}_1$ at random until it is empty, and then choosing elements of $\mathsf{Bin}_2$ at random until it is empty.

**Claim 4.4.19.** $\mathsf{HYB}_5\left(\mathcal{A}_5, 1^k, z\right) \overset{s}{\cong} \mathsf{HYB}_6\left(\mathcal{A}_6, 1^k, z\right)$.

*Proof.* Note that the only change between Hybrid 5 and 6 is the distribution of the ordering $(\ell_1, ..., \ell_{k^2})$ that is forced by the simulator during the elimination phase. It thus suffices to prove that the skewed distribution is indistinguishable from the honest (random) distribution.

**Lemma 4.4.20.** *Let $D_1$ be the distribution over orderings of $[k^2]$ formed by:*

$$(\textit{random permutation of } \mathsf{Bin}_1)||(\textit{random permutation of } \mathsf{Bin}_2),$$

*where* $\mathsf{Bin}_1$ *and* $\mathsf{Bin}_2$ *are as chosen by the simulator, and* $||$ *denotes concatenation of orderings. Then given* $\mathsf{view}(\mathcal{A})$, *$D_1$ is statistically close to uniform.*

*Proof.* Note that choosing an ordering of $[k^2]$ uniformly is equivalent to first assigning *all indices* $\ell \in [k^2]$ randomly to a bin $\mathsf{Bin}_1$ or $\mathsf{Bin}_2$, and then choosing random ordering of each bin. The only difference between this procedure and the one defining $D_1$ is that the indices $\ell \in [k^2]$ corresponding to "bad" candidate bits are artificially placed in $\mathsf{Bin}_1$ or $\mathsf{Bin}_2$ based on the value of the corresponding extracted bit $b_\ell = \mathsf{Ext}(r_1^\ell, ..., r_{|\mathcal{E}_{\mathsf{rand}}|}^\ell)$ (i.e., whether the bit is of Type BB or BG). But, as we have argued earlier, by the property of the robust multi-source extractor, given the view of the adversary, the bit $b_\ell$ is statistically close to uniform. Indeed, this is because the view of the adversary contains only the strings $r_j^\ell$ of corrupted parties $P_j$ and leakage on the $r_i^\ell$'s of honest parties (in addition to other information that is independent of $b_\ell$). Therefore, conditioned on the view of the adversary, the distribution $D_1$ will also be statistically close to uniform. □

□

**Hybrid 7** The ideal-world experiment. That is, the same as Hybrid 6, except the final output of the protocol is replaced by the true output $b_T$, instead of whatever is dictated by the parties' views up to that point (note that this is only relevant when an *honest* party $P_i$ is the recipient of the output bit).

**Claim 4.4.21.** $\mathsf{HYB}_6\left(\mathcal{A}_6, 1^k, z\right) \overset{s}{\cong} \mathsf{HYB}_7\left(\mathcal{A}_7, 1^k, z\right)$.

*Proof.* We must prove that with overwhelming probability, the output of the simulated execution is the target output bit $b_T$. It suffices to show that the probability of a Type BB candidate being selected in the simulation is negligible, since these are the only candidates that yield an output of $(1 - b_T)$.

Recall that $|\mathsf{BAD}_{\mathsf{elim}}|$ is the number of bad elimination rounds, in which the simulator could not force the skewed distribution and instead a random index $\ell$ was eliminated. In the simulation, the indices that are eliminated will then be: (a) $|\mathsf{BAD}_{\mathsf{elim}}|$ $(\leq k)$ *random* candidates $\in [k^2]$, and (b) $(k^2-1)-|\mathsf{BAD}_{\mathsf{elim}}|$ random elements from $\mathsf{Bin}_1$ (if not yet empty). Since all Type BB candidates are necessarily contained in $\mathsf{Bin}_1$, as long as the size of $\mathsf{Bin}_1$ is no greater than $(k^2-1)-|\mathsf{BAD}_{\mathsf{elim}}|$, we will be guaranteed that all Type BB candidates are eliminated. This corresponds to $\mathsf{Bin}_2$ containing at least $|\mathsf{BAD}_{\mathsf{elim}}|$ elements. We now show that with overwhelming probability, $\mathsf{Bin}_2$ will contain at least this many Type G indices. That is, if for all $\ell$ of Type G we define $X_\ell$ to be the indicator variable corresponding to index $\ell$ falling into $\mathsf{Bin}_2$, then we have

$$\begin{aligned}\Pr[|\mathsf{Bin}_2|\geq|\mathsf{BAD}_{\mathsf{elim}}|] &\geq \Pr[|\mathsf{Bin}_2|\geq k]\\ &= \Pr\Big[\sum_{\ell\text{ Type G}} X_\ell \geq k\Big]\\ &\geq 1-e^{-\left(\frac{k^2-k}{2}\right)\left(\frac{k+1}{k-1}\right)^2/2} \qquad \text{by a Chernoff bound.}^{12}\\ &\geq 1-e^{k^2/4}.\end{aligned}$$

□

□

### 4.4.5 Achieving WLR-MPC for Randomized Functions

We now describe how to utilize the coin tossing protocol from the previous section to achieve WLR-MPC for randomized functions in the setting where leakage occurs independently on each honest party's secret state, and where there are many parties, a constant fraction of which are honest.

This protocol will use two tools as a black box:

1. A $\lambda$-WLR-MPC protocol $\Pi_g^{\mathsf{det}}$ for *deterministic* functions $g$ in the CRS model, as in Theorem 4.2.18 (based on DDH).

2. A $\lambda$-weakly leakage-resilient fully simulatable coin tossing protocol $\Pi_{i^*}^{\mathsf{rand}}$ in the CRS model, as in Theorem 4.4.12 (based on any trapdoor permutation).

**Theorem 4.4.22.** *Fix any constants $\epsilon,\delta>0$, and assume the existence of the two above tools with leakage parameter $\lambda$. Then for every probabilistic polynomial-time function $f$, there exists a $\lambda$-weakly leakage resilient MPC protocol $\Pi_f$ as in Definition 4.4.1 (assuming erasures and independent real-world leakage) for evaluating $f$, in the common random string model, tolerating $t=(1-\epsilon)n$ corrupted parties. Security of the protocol holds with probability $1-\mathsf{negl}(k)-ne^{-\sqrt{n}/2}$, where $k$ is the security parameter.*

We begin by giving a high-level overview of the protocol $\Pi_f$, which is described in detail in Figure 4.3.

As the first step in the protocol, the parties $P_1,\ldots,P_n$ will elect two committees from amongst themselves: one committee $C_{\mathsf{rand}}$ will serve the role of generating secret randomness to be used

[12]Explicit Chernoff bound used: $\Pr[\sum X_\ell<(1-\delta)\mu]<e^{-\mu\delta^2/2}$ for $0<\delta<1$, where $\mu$ is the mean of $\sum X_\ell$. In our case, $\mu=\frac{1}{2}(k^2-k)$ and $\delta=\frac{k+1}{k-1}$.

in the evaluation of the functionality $f$; the second committee $C_{\mathsf{WLR}}$ takes everyone's inputs and uses the communicated randomness as an additional input in order to evaluate $f$ *deterministically*, using the original WLR-MPC protocol (for deterministic functions).[13]

More formally, after the two committees are elected, all parties secret share their inputs $x_i$ among the parties in $C_{\mathsf{WLR}}$. In order to guarantee correctness and input independence, they provide commitments to each share to the whole committee. Then, all parties erase their original input and intermediate information from the secret sharing procedure.

Next, the committee $C_{\mathsf{rand}}$ repeatedly executes the weakly leakage-resilient fully simulatable coin tossing protocol $\Pi_{i^*}^{\mathsf{rand}}$ from Section 4.4.4 in order to generate randomness for $f$.[14] In each execution, one party $P_{i^*}$ receives a secret random bit and decommitment information, and all other parties receive a public commitment to this bit. Party $P_{i^*}$ then secret shares this bit (together with the corresponding decommitment information) among the parties in $C_{\mathsf{WLR}}$, and erases his state once more. For each bit of randomness required by $f$, the committee $C_{\mathsf{rand}}$ will execute this process $\Pi_{i^*}^{\mathsf{rand}}$ once for every $i^* \in C_{\mathsf{rand}}$; the final random bit used will be the xor of the resulting bits (to ensure the final random bit remains secret from the adversary).

Finally, the committee $C_{\mathsf{WLR}}$ uses this information to reconstruct all parties inputs, reconstruct the generated randomness, verify the correctness of all values, and then evaluate $f$ on the corresponding inputs and randomness via the underlying WLR-MPC protocol for *deterministic* functions.

*Proof of Theorem 4.4.22.* We consider a sequence of intermediate hybrids. The output of each hybrid experiment consists of the outputs of all parties, where honest parties output in accordance with the dictated protocol, and malicious parties may output any efficiently computable function of the view of the adversary. For every adversary $\mathcal{A}_\ell$ with auxiliary input $z \in \{0,1\}^*$ running in hybrid experiment $\ell$ with initial inputs $\vec{x}$, we denote the output of the corresponding hybrid $\ell$ experiment by

$$\mathsf{HYB}_\ell\left(\mathcal{A}_\ell, 1^k, z, \{x_i\}_{i=1}^n\right).$$

We now go through the hybrids one by one, beginning with Hybrid 0. In each step we show that for any adversary $\mathcal{A}_\ell$ running in Hybrid $\ell$, there exists an adversary $\mathcal{A}_{\ell+1}$ running in Hybrid $(\ell+1)$ such that

$$\mathsf{HYB}_\ell\left(\mathcal{A}_\ell, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_{\ell+1}\left(\mathcal{A}_{\ell+1}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

**Hybrid 0.** The real world: i.e., the adversary interacts with honest parties in the real-world experiment running $\Pi_f$.

**Hybrid 1. (Abort if committee election fails).**

[13]We remark that the $ne^{-\sqrt{n}/2}$ term in the failure probability of the overall protocol corresponds to the probability that the elected committees are completely composed of corrupted parties.

[14]It is important for security that $\Pi_{i^*}^{\mathsf{rand}}$ is repeated *sequentially*, and that the parties in $C_{\mathsf{rand}}$ have erased their original inputs $x_i$ before executing $\Pi_{i^*}^{\mathsf{rand}}$, as the weakly leakage-resilient coin tossing protocol does not enjoy "leakage oblivious" simulation (as defined by [BCH11]), and thus does not directly compose with concurrent protocol executions.

**WLR-MPC for Randomized Functionality $f$**
Parties $P_1, \ldots, P_n$.
CRS: $\{\mathsf{crs}_i\}_{i \in [n]}$ common random string for each party for the equivocal commitment scheme.

1. Elect committees: Run Feige committee election protocol among all $n$ parties to elect a committee $C_{\mathsf{WLR}}$ of approximate size $\log^2 n$. Run a second execution of the Feige committee election protocol among the remaining parties $[n] \setminus C_{\mathsf{WLR}}$ to elect a second, disjoint committee $C_{\mathsf{rand}}$ of approximate size $\log^2 n$. Denote $n_{\mathsf{WLR}} = |C_{\mathsf{WLR}}|$ and $n_{\mathsf{rand}} = |C_{\mathsf{rand}}|$.

2. Parties secret share inputs among committee $C_{\mathsf{WLR}}$:

   Each party $P_i \in [n]$ performs the following steps:

   (a) Secret share input $x_i$ as $(x_i^1, \ldots, x_i^{n_{\mathsf{WLR}}}) \leftarrow \mathsf{Share}(x_i)$ using the $n_{\mathsf{WLR}}$-out-of-$n_{\mathsf{WLR}}$ xor secret sharing scheme.

   (b) For each share $x_i^j$, generate a commitment to the share using the extractable equivocal commitment scheme: $(\mathsf{com}_i^j, \mathsf{dec}_i^j) \leftarrow \mathsf{Com}(\mathsf{crs}_i, x_i^j)$. Denote $\overline{\mathsf{com}}_i = (\mathsf{com}_i^1, \ldots, \mathsf{com}_i^{n_{\mathsf{WLR}}})$.

   (c) To each party $P_j \in C_{\mathsf{WLR}}$, send the collection of information $(x_i^j, \mathsf{dec}_i^j, \overline{\mathsf{com}}_i)$.

   (d) Erase all information up to this point (including the original input $x_i$).

   Note that at the conclusion of this step, the secret state of each party $P_j \in C_{\mathsf{WLR}}$ consists of one secret share and decommitment value $(x_i^j, \mathsf{dec}_i^j)$ from each party $P_i \in [n]$, in addition to a collection of commitments $\{\overline{\mathsf{com}}_i\}_{i \in [n]}$ to *all* the distributed secret shares. The secret state of each party $P_i \notin C_{\mathsf{WLR}}$ is empty.

3. $C_{\mathsf{rand}}$ committee generates randomness for evaluation of $f$:

   Let $r$ denote the number of bits of randomness required by an execution of $f$. Repeat the following steps sequentially for each value of $\ell = 1, \ldots, r$, and for each $i^* = 1, \ldots, [n_{\mathsf{rand}}]$ (in order to collectively sample $r$ random bits "in the well" for each party in $C_{\mathsf{rand}}$):

   (a) Execute the fully simulatable $\lambda$-weakly leakage resilient coin tossing protocol $\Pi_{i^*}^{\mathsf{rand}}$ (see Figure 4.1) in order to sample a random bit $b_{\ell,i^*}$ "in the well" for party $P_{i^*} \in C_{\mathsf{rand}}$. As a result, all parties $P_i \in C_{\mathsf{rand}}$ receive a commitment $\mathsf{com}_{\ell,i^*}$ to $b_{\ell,i^*}$, and party $P_{i^*}$ learns $b_{\ell,i^*}$ together with the corresponding decommitment information $\mathsf{decom}_{\ell,i^*}$.

   (b) Each party $P_i \in C_{\mathsf{rand}}$ sends $\mathsf{com}_{\ell,i^*}$ to all parties $P_j \in C_{\mathsf{WLR}}$.

   (c) Party $P_{i^*}$ secret shares the pair $(b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*})$ among the parties in the committee $C_{\mathsf{WLR}}$, by executing the procedure in Step 2 with $(b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*})$ in the place of $x_i$. Denote the corresponding secret shares, commitments, and decommitments as $\{s_{\ell,i^*}^j\}_{j \in [n_{\mathsf{WLR}}]}, \{c_{\ell,i^*}^j\}_{j \in [n_{\mathsf{WLR}}]}, \{d_{\ell,i^*}^j\}_{j \in [n_{\mathsf{WLR}}]}$.

   (d) All parties in $C_{\mathsf{rand}}$ erase their secret state.

4. $C_{\mathsf{WLR}}$ committee evaluates $f$ on this randomness:

   The parties of $C_{\mathsf{WLR}}$ execute the $\lambda$-weakly leakage resilient MPC protocol $\Pi_g^{\mathsf{det}}$ for *deterministic* function $g$ defined in Figure 4.4, using as input the secret shares, commitments, and decommitment information generated above. Output the result of this execution, answer.

**Figure 4.3:** WLR-MPC protocol $\Pi_f$ for evaluating a randomized functionality $f$.

**Modified *deterministic* functionality $g$**

Input: Each party $P_j \in C_{\mathsf{WLR}}$: $\left\{x_i^j, \mathsf{decom}_i^j, \overline{\mathsf{com}}_i\right\}_{i\in[n]}$, $\left(\mathsf{com}_{\ell,i^*}, \left\{s_{\ell,i^*}^j, d_{\ell,i^*}^j, \overline{c}_{\ell,i^*}\right\}\right)_{\ell\in[r],i^*\in[n_{\mathsf{rand}}]}$.

Compute: If any party submits commitment information $\{\overline{\mathsf{com}}_i\}_i, \{\mathsf{com}_{\ell,i^*}\}_{\ell,i^*}, \{\overline{c}_{\ell,i^*}\}_{\ell,i^*}$ in disagreement with that submitted by another party, then $g$ terminates and outputs $\perp$. Otherwise, we will denote the agreeing values by $\{\overline{\mathsf{com}}_i\}_i = \{(\mathsf{com}_i^1, \ldots, \mathsf{com}_i^{n_{\mathsf{WLR}}})\}_i, \{\mathsf{com}_{\ell,i^*}\}_{\ell,i^*}$, and $\{\overline{c}_{\ell,i^*}\}_{\ell,i^*} = \{(c_{\ell,i^*}^1, \ldots, c_{\ell,i^*}^{n_{\mathsf{WLR}}})\}_{\ell,i^*}$.

1. For each $i \in [n]$, reconstruct $x_i$:
   (a) Verify the commitment to each secret share $x_i^j$. Namely, if for any $j \in C_{\mathsf{WLR}}$ it holds that $x_i^j \neq \mathsf{Rec}(\mathsf{crs}_i, \mathsf{decom}_i^j, \mathsf{com}_i^j)$, then $g$ terminates and outputs $\perp$.
   (b) Compute $x_i := \bigoplus_{j\in C_{\mathsf{WLR}}} x_i^j$.
2. For each $\ell \in [r]$ and $i^* \in [n_{\mathsf{rand}}]$, reconstruct $b_{\ell,i^*}$:
   (a) Verify the commitment to each secret share $s_{\ell,i^*}^j$. Namely, if for any $j \in C_{\mathsf{WLR}}$ it holds that $s_{\ell,i^*}^j \neq \mathsf{Rec}(\mathsf{crs}_i, d_{\ell,i^*}^j, c_{\ell,i^*}^j)$, then $g$ terminates and outputs $\perp$.
   (b) Compute $s_{\ell,i^*} := \bigoplus_{j\in C_{\mathsf{WLR}}} s_{\ell,i^*}^j$, and parse $s_{\ell,i^*}$ as $(b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*})$.
   (c) Verify this pair against the commitment $\mathsf{com}_{\ell,i^*}$. Namely, if $b_{\ell,i^*} \neq \mathsf{Rec}(\mathsf{crs}_i, \mathsf{decom}_{\ell,i^*}, \mathsf{com}_{\ell,i^*})$, then $g$ terminates and outputs $\perp$.
3. For each $\ell \in [r]$, take $b_\ell := \bigoplus_{i^*\in C_{\mathsf{rand}}} b_{\ell,i^*}$ as the $\ell$ bit of evaluation randomness.
4. Evaluate $f$ on inputs $x_1, \ldots, x_n$ using randomness $\mathsf{rand} := (b_1, \ldots, b_r)$. Namely, compute $\mathsf{answer} = f(x_1, \ldots, x_n; \mathsf{rand})$.

Output: To all parties: $\mathsf{answer}$.

**Figure 4.4:** The modified deterministic function $g$ to be computed via the underlying WLR-MPC protocol (for deterministic functions) in order to evaluate the randomized functionality $f$.

Same as the previous, except that the experiment ends in abort if either committee $C_{\mathsf{WLR}} \subset M$ or $C_{\mathsf{rand}} \subset M$ is composed of completely malicious parties.

**Lemma 4.4.23.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_0$ in Hybrid 0, it holds for the same adversary $\mathcal{A}_1 = \mathcal{A}_0$ in Hybrid 1 that*

$$\Delta\left[\mathsf{HYB}_0\left(\mathcal{A}_0, 1^k, z, \{x_i\}_{i=1}^n\right), \mathsf{HYB}_1\left(\mathcal{A}_1, 1^k, z, \{x_i\}_{i=1}^n\right)\right] \leq ne^{-\sqrt{n}/2},$$

*where $\delta[\cdot,\cdot]$ denotes the statistical distance between distributions.*

*Proof.* By the properties of the Feige committee election protocol (see Corollary 4.2.8), the probability of ending in abort in Hybrid 1 but not in Hybrid 0 is bounded by $ne^{-\sqrt{n}/2}$. □

**Hybrid 2. (Replace $\Pi_g^{\mathsf{det}}$ with ideal functionality).**

Same as the previous, except that the underlying WLR-MPC protocol execution $\Pi_g^{\mathsf{det}}$ for *deterministic* functions is replaced by the corresponding (leaky) ideal functionality $F_g$.

**Lemma 4.4.24.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_1$ in Hybrid 1, there exists a PPT adversary $\mathcal{A}_2$ in Hybrid 2 such that*

$$\mathsf{HYB}_1\left(\mathcal{A}_1, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_2\left(\mathcal{A}_2, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Fix any PPT adversary $\mathcal{A}_1$ in Hybrid 1. Consider the adversary $\mathcal{A}_2$ in Hybrid 2 that does the following. $\mathcal{A}_2$ simulates the actions of $\mathcal{A}_1$ exactly up until the execution of the underlying WLR-MPC protocol $\Pi_g^{\mathsf{det}}$. Let $\mathsf{view}$ denote the view of $\mathcal{A}_1$ up to this point. $\mathcal{A}_2$ runs the WLR-MPC simulator $\mathcal{S}_{\mathsf{WLR}}^{\mathsf{det}}$ with auxiliary information $\mathsf{view}$ in order to simulate the execution of $\Pi_g^{\mathsf{det}}$ in the corresponding leaky ideal world. At the conclusion of the simulation of $\Pi_g^{\mathsf{det}}$, $\mathcal{A}_2$ returns to simulating the actions of $\mathcal{A}_1$ exactly until the end of the experiment.

Indistinguishability of the simulated output follows directly by the security of the underlying WLR-MPC protocol for deterministic functions. □

**Hybrid 3. (Replace $\Pi_{i^*}^{\mathsf{rand}}$ executions with ideal functionalities).**

Same as before, except that each of the executions of the weakly leakage-resilient fully simulatable coin tossing protocol $\Pi_{i^*}^{\mathsf{rand}}$ is replaced by the corresponding ideal functionality $F_{\mathsf{com},i^*}^{\mathsf{rand},+L}$ (as defined in Remark 4.4.13).

**Lemma 4.4.25.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_2$ in Hybrid 2, there exists a PPT adversary $\mathcal{A}_3$ in Hybrid 3 such that*

$$\mathsf{HYB}_2\left(\mathcal{A}_2, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_3\left(\mathcal{A}_3, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Fix any PPT adversary $\mathcal{A}_2$ in Hybrid 2. We can think of $\mathcal{A}_2$ as composed of a collection of sub-algorithms $(\mathcal{A}_2^1, \mathcal{A}_2^{\mathsf{coin}_1}, \mathcal{A}_2^2, \mathcal{A}_2^{\mathsf{coin}_2}, \ldots)$ corresponding to the actions of $\mathcal{A}_2$ during and between each instance of the coin tossing protocol $\Pi_{i^*}^{\mathsf{rand}}$. We define the output of each $\mathcal{A}_2^i$ (and $\mathcal{A}_2^{\mathsf{coin}_i}$) to be the view of $\mathcal{A}_2$ up to that point. Each sub-adversary $\mathcal{A}_2^{\mathsf{coin}_i}$ receives as auxiliary input $z^{\mathsf{coin}_i}$ the output of the previous sub-adversary $\mathcal{A}_2^i$. By the security of the weakly leakage-resilient fully simulatable coin tossing protocol $\Pi_{i^*}^{\mathsf{rand}}$, for each such $\mathcal{A}_2^{\mathsf{coin}_i}$, there exists a simulator $\mathcal{S}_{\mathsf{coin}}^i$ who, given the same auxiliary information $z^{\mathsf{coin}_i}$ and access to the corresponding ideal functionality $F_{\mathsf{com},i^*}^{\mathsf{rand},+L}$ (which outputs a random bit and decommitment information to $P_{i^*}$, outputs a commitment to the bit to all participating parties, and answers a single leakage query), simulates the view of $\mathcal{A}_2$ during the $i$th coin tossing protocol execution. Loosely speaking, $\mathcal{A}_3$ will simulate $\mathcal{A}_2$ by piecing all these simulations together.

Consider the following adversary $\mathcal{A}_3$ in Hybrid 3. Between executions of the coin tossing protocol (i.e., corresponding to each sub-algorithm $\mathcal{A}_2^i$), the adversary $\mathcal{A}_3$ simulates the actions of $\mathcal{A}_2$ exactly. Each time $\mathcal{A}_2$ enters into the $i$th execution of the coin tossing protocol, $\mathcal{A}_3$ instead runs the simulator $\mathcal{S}_{\mathsf{coin}}^i$ with auxiliary input $z^{\mathsf{coin}_i} = \mathsf{view}(\mathcal{A}_2)$, which is the entire simulated view of $\mathcal{A}_2$ up to that point.

Indistinguishability of the simulated output follows directly by the security of the weakly leakage-resilient fully simulatable coin tossing protocol, together with a standard hybrid argument.

□

**Hybrid 4. (Simulate commitments for honest parties).**

Same as the previous hybrid, with the following two changes:

- For each honest party, the CRS for the equivocal commitment scheme is *simulated*, and the corresponding trapdoor is given to the adversary. That is, for each honest party $P_i$, the value $\mathsf{crs}_i$ is sampled as $(\mathsf{crs}_i, \mathsf{trap}_i) \leftarrow \mathsf{Sim_{eq}}^{\mathsf{crs}}(1^k)$.
- Honest parties generate *simulated* commitments. Namely, each time an honest party $P_i$ would previously commit to a value $y$ by sampling randomness $r$, computing $(\mathsf{com}, \mathsf{decom}) = \mathsf{Com}(\mathsf{crs}_i, y; r)$, and updating his state as $\mathsf{state}_i \leftarrow \mathsf{state}_i \cup \{r\}$, he now does the following. First, he generates a simulated commitment $(\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1) \leftarrow \mathsf{Sim_{eq}}^{\mathsf{com}}(\mathsf{crs}_i, \mathsf{trap}_i)$ and outputs $\mathsf{eqcom}$. Then, he updates his secret state as $\mathsf{state}_i \leftarrow \mathsf{state}_i \cup \{\mathsf{eqdec}^y\}$ with the choice of randomness corresponding to the *correct* value $y$.

**Lemma 4.4.26.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_3$ in Hybrid 3, then for the same adversary $\mathcal{A}_4 = \mathcal{A}_3$ in Hybrid 4, it holds that*

$$\mathsf{HYB}_3\left(\mathcal{A}_3, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_4\left(\mathcal{A}_4, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Indistinguishability holds directly by the indistinguishability of simulated commitments in the equivocal commitment scheme (Definition 4.4.4), together with a standard hybrid argument.

□

**Hybrid 5. (Extract committed values on behalf of corrupt parties).**

In this hybrid, two changes take place. First, the CRS of *corrupted* parties for the extractable equivocal commitment scheme is now sampled using the *extractable* CRS generation algorithm $\mathsf{crsGen}_E(1^k)$. Second, parties directly give their inputs $x_i$ to one large ideal functionality $F'_f$ and then erase their secret state. The functionality $F'_f$ samples a random $r$-bit string $R = (b_1, \dots, b_r)$, secret shares each bit $R$ as $(b_{\ell,1}, \dots, b_{\ell,n_{\mathsf{rand}}}) \leftarrow \mathsf{Share}(b_\ell)$ among the parties in $C_{\mathsf{rand}}$ via the xor secret sharing scheme, generates a commitment to each share $b_{\ell,i^*}$, gives the commitment to all parties in $C_{\mathsf{rand}}$, and gives the share $b_{\ell,i^*}$ together with the corresponding decommitment information to party $P_i^* \in C_{\mathsf{rand}}$. Then, $F'_f$ evaluates the randomized function $f$ on inputs $(x_1, \dots, x_n)$ with randomness $R$. The functionality $F'_f$ is formally defined below.

Input: Each party $P_i$ submits his input $x_i$.

Compute: First generate values for randomness $R$:

1. Sample a random $r$-bit string $R \leftarrow \{0,1\}^r$. Denote $R = (b_1, \dots, b_r)$.
2. For each bit $b_\ell$ of $R$, secret share $(b_{\ell,1}, \dots, b_{\ell,n_{\mathsf{rand}}}) \leftarrow \mathsf{Share}(b_\ell)$ using the $n_{\mathsf{rand}}$-out-of-$n_{\mathsf{rand}}$ xor secret sharing scheme.
3. For each secret share $b_{\ell,i^*}$, generate a commitment as follows. If $P_{i^*}$ is a corrupted party, honestly generate a commitment as $(\mathsf{com}_{\ell,i^*}, \mathsf{decom}_{\ell,i^*}) \leftarrow \mathsf{Com}(\mathsf{crs}_{i^*}, b_{\ell,i^*})$. If $P_{i^*}$ is an honest party, then generate a *simulated* commitment as

$$(\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1) \leftarrow \mathsf{Sim_{eq}}^{\mathsf{com}}(\mathsf{crs}_{i^*}, \mathsf{trap}_{i^*}),$$

   and define $\mathsf{decom}_{\ell,i^*} = \mathsf{eqdec}^{b_{\ell,i^*}}$.

Evaluate the function $f$: Compute $\mathsf{answer} = f(x_1, \dots, x_n; R)$.

Output: To all parties: $\mathsf{answer}$.
To each party $P_{i^*} \in C_{\mathsf{rand}}$: $(b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*})_{\ell \in [r]}$, $\{\mathsf{com}_{\ell,i}\}_{\ell \in [r], i \in C_{\mathsf{rand}}}$.

**Lemma 4.4.27.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_4$ in Hybrid 4, there exists a PPT adversary $\mathcal{A}_5$ in Hybrid 5 such that*

$$\mathsf{HYB}_4\left(\mathcal{A}_4, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{s}{\cong} \mathsf{HYB}_5\left(\mathcal{A}_5, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Fix any PPT adversary $\mathcal{A}_4$ in Hybrid 4. Consider the following adversary $\mathcal{A}_5$ in Hybrid 5.

Adversary $\mathcal{A}_5$:

1. Simulate CRS generation.
   (a) For each honest party $P_i$, simulate an *equivocable* CRS: $(\mathsf{crs}_i, \mathsf{trap}_i) \leftarrow \mathsf{Sim_{eq}}^{\mathsf{crs}}(1^k)$.
   (b) For each corrupted party $P_i$, sample an *extractable* CRS: $(\mathsf{crs}_i, \mathsf{trap}_i) \leftarrow \mathsf{crsGen}_E(1^k)$.

2. Simulate honest parties committing to inputs.

   Select one honest party in the committee $C_{\mathsf{WLR}}$ (note that by Hybrid 1 such a party necessarily exists). Without loss of generality, denote this party by $P_1$. For each honest party $P_i \in [n]$, simulate $P_i$ secret sharing his input as follows:

   (a) Sample *all but one* secret share $(x_i^2, \dots, x_i^{n_{\mathsf{WLR}}})$ of the input $x_i$ at random, and give them to parties $P_2, \dots, P_{n_{\mathsf{WLR}}}$ in $C_{\mathsf{WLR}}$. Sample a (simulated) commitment to each $x_i^j$, send it to all parties in $C_{\mathsf{WLR}}$, and update the secret state of $P_i$ as before.

   (b) For the special honest party $P_1$, sample a simulated commitment

   $$(\mathsf{eqcom}, \mathsf{eqdec}^0, \mathsf{eqdec}^1) \leftarrow \mathsf{Sim_{eq}}^{\mathsf{com}}(\mathsf{crs}_i, \mathsf{trap}_i)$$

   that allegedly commits to the final secret share of $x_i$. Send $\mathsf{eqcom}$ to all parties in $C_{\mathsf{WLR}}$ and update the simulated secret state of $P_1$ as $\mathsf{state}_1 \leftarrow \mathsf{state}_1 \cup \{\mathsf{eqdec}^{y_i}\}$, where $y_i$ is defined to be equal to $y_i := x_i \oplus \bigoplus_{i'=2}^{n_{\mathsf{WLR}}} x_i^{i'}$. Note that $\mathcal{A}_4$ does not explicitly know the value of $y_i$ since he does not know the secret input $x_i$. However, the simulated state of party $P_1$ remains an efficiently computable function of the secret inputs $\{x_i\}_{i \notin M}$.

3. Extract inputs of corrupted parties.

   For each corrupted party $P_i \in [n]$, do the following:

   (a) Let $(x_i^j, \mathsf{decom}_i^j, \{\mathsf{com}_i^{j'}\}_{j' \in [n_{\mathsf{WLR}}]})$ denote the values received by each honest party $P_j \in C_{\mathsf{WLR}}$ from the (corrupted) party $P_i$. If any received decommitment information is invalid (i.e., if for any $j \in C_{\mathsf{WLR}}$ it holds that $x_i^j \neq \mathsf{Rec}(\mathsf{crs}_i, \mathsf{com}_i^j, \mathsf{decom}_i^j)$), then set $x_j = \perp$.

   (b) Otherwise, extract the secret shares of $P_i$'s input $x_i$ using the extractor trapdoor generated in Step 1. That is, for each $j \in C_{\mathsf{WLR}}$, extract $y_i^j \leftarrow E(\mathsf{crs}_i, \mathsf{com}_i^j, \mathsf{trap}_i)$, and let $x_i' := \bigoplus_{j \in C_{\mathsf{WLR}}} y_i^j$.

4. Erase all intermediate values from the secret states of honest parties $P_i$.

5. Submit the collection of extracted inputs $\{x_i'\}_{i \in M}$ to the ideal functionality $F_f'$ on behalf of corrupted parties. Receive back $\mathsf{answer}$ and the random bits $b_{\ell,i^*}$ and commitment information given to each corrupt party $P_{i^*}$: $(b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*})_{\ell \in [r]}$, $\{\mathsf{com}_{\ell,i}\}_{\ell \in [r], i \in C_{\mathsf{rand}}}$.

6. Simulate randomness generation procedures.

   For each value of $\ell = 1, \dots, r$ and $i^* = 1, \dots, n_{\mathsf{rand}}$, simulate the $(\ell, i^*)$th random bit generation phase as follows.

   - If $P_{i^*}$ is a corrupt party (in which case we know $(b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*})$ from $F_f'$):

     (a) Simulate the output of the $(\ell, i^*)$th evocation of the ideal functionality $F_{\mathsf{com},i^*}^{\mathsf{rand},+L}$ by sending $(b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*})$ to party $P_{i^*}$, and sending $\mathsf{com}_{\ell,i^*}$ to all parties in $C_{\mathsf{rand}}$.

     (b) Simulate the actions of $\mathcal{A}_4$ exactly until the next coin tossing ideal functionality execution.

   - If $P_{i^*}$ is an honest party:

(a) Simulate the output of the $(\ell, i^*)$th evocation of the ideal functionality $F^{\mathsf{rand},+L}_{\mathsf{com},i^*}$ by sending $\mathsf{com}_{\ell,i^*}$ to all parties in $C_{\mathsf{rand}}$. Note that the simulating adversary $\mathcal{A}_5$ does *not* know the correct bit $b_{\ell,i^*}$ or corresponding decommitment information $\mathsf{decom}_{\ell,i^*}$; but, any leakage queries on these values can be answered by forwarding the query to the ideal functionality $F'_f$.

(b) Simulate $P_{i^*}$ secret sharing $(b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*})$ identically to the simulation of the input secret sharing phase in Step 2 above. Namely, do the following:

i. Sample *all but one* secret share $(s^2_{\ell,i^*}, \ldots, s^{n_{\mathsf{WLR}}}_{\ell,i^*})$ of the (unknown) secret information $(b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*})$ at random, and give them to parties $P_2, \ldots, P_{n_{\mathsf{WLR}}}$ in $C_{\mathsf{WLR}}$. Sample a (simulated) commitment to each $s^j_{\ell,i^*}$, send it to all parties in $C_{\mathsf{WLR}}$, and update the secret state of $P_i$ as before.

ii. For the special honest party $P_1$, sample a simulated commitment

$$(\mathsf{eqcom}^{(1)}_{\ell,i^*}, \mathsf{eqdec}^0_{\ell,i^*}, \mathsf{eqdec}^1_{\ell,i^*}) \leftarrow \mathsf{Sim_{eq}}^{\mathsf{com}}(\mathsf{crs}_{i^*}, \mathsf{trap}_{i^*})$$

that allegedly commits to the final secret share of $(b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*})$. Send $\mathsf{eqcom}^{(1)}_{\ell,i^*}$ to all parties in $C_{\mathsf{WLR}}$ and update the simulated secret state of $P_1$ as $\mathsf{state}_1 \leftarrow \mathsf{state}_1 \cup \{\mathsf{eqdec}^{y_{\ell,i^*}}_{\ell,i^*}\}$, where $y_{\ell,i^*}$ is defined to be equal to $y_{\ell,i^*} := (b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*}) \oplus \bigoplus_{j'=2}^{n_{\mathsf{WLR}}} s^{j'}_{\ell,i^*}$. Note that $\mathcal{A}_4$ does not explicitly know the value of $y_{\ell,i^*}$ since he does not know the secret values $(b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*})$. However, the simulated state of party $P_1$ remains an efficiently computable function of these secret values.

7. Simulate the execution of $F_g$.

The Hybrid 4 adversary $\mathcal{A}_4$ will submit a collection of input shares (plus commitment information) and secret shares of random bits (plus commitment information) on behalf of the corrupt parties to the alleged Hybrid 4 ideal functionality $F_g$. The Hybrid 5 adversary $\mathcal{A}_5$ will accept these values and simulate $F_g$ as follows.

(a) Check the correctness and consistency of all accepted values and commitment information. Namely, if any received commitment does not agree with the corresponding commitments held by the (simulated) honest parties, or if any supplied decommitment information is not valid for the corresponding submitted committed value, then output $\perp$ to all corrupt parties as the simulated output of $F_g$.

(b) Otherwise, output the value answer that was received by the Hybrid 5 ideal functionality $F'_f$ in Step 5.

8. Simulate the actions of $\mathcal{A}_4$ exactly for the remainder of the experiment.

9. Simulating leakage queries: At any point that $\mathcal{A}_4$ submits a leakage query $L$ on the secret states of honest parties, the simulating adversary $\mathcal{A}_5$ submits the leakage query $L'$ to his corresponding Hybrid 5 leakage oracle (leaking on $x_1, \ldots, x_n, \{b_{\ell,i^*}\}_{\ell,i^*}$) that first reconstructs the simulated states $\{\mathsf{state}_i\}_{i \notin M}$ of honest parties as a function of the secret information $x_1, \ldots, x_n, \{b_{\ell,i^*}\}_{\ell,i^*}$, and then applies $L$ to the reconstructed states.

Indistinguishability of the simulated output follows by the extraction property of the extractable equivocal commitment scheme (see Definition 4.4.8), together with the (perfect) secrecy and reconstruction properties of the xor secret sharing scheme.

□

**Hybrid 6. (Ideal world).**

The ideal-world experiment. Namely, parties directly submit their inputs to the ideal functionality $F_f$ that samples a random $r$-bit string $R$, outputs the evaluation $f(x_1, \dots, x_n; R)$, and answers leakage queries on the (joint) secret values $(x_1, \dots, x_n, R)$.

**Lemma 4.4.28.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_5$ in Hybrid 5, then for the same adversary $\mathcal{A}_6 = \mathcal{A}_5$ in Hybrid 6, it holds that*

$$\mathsf{HYB}_5\left(\mathcal{A}_5, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_6\left(\mathcal{A}_6, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Note that the difference between the current hybrid and the previous is that the ideal functionality outputs less information. Namely, in Hybrid 5, the ideal functionality $F'_f$ accepted inputs $x_i$ and output the desired evaluation answer *in addition to* a collection of random secret shares of the random bits used, and corresponding commitment and decommitment information. In Hybrid 6, the ideal functionality $F_f$ accepts inputs $x_i$ and responds *only* with the desired evaluation answer. Thus, simulating the output of Hybrid 5 amounts to properly simulating these additional output values.

Fix any PPT adversary $\mathcal{A}_5$ in Hybrid 5. Consider the following adversary $\mathcal{A}_6$ in the ideal world, Hybrid 6.

Adversary $\mathcal{A}_6$:

1. Simulate CRS generation.
   (a) For each honest party $P_i$, simulate an *equivocable* CRS: $(\mathsf{crs}_i, \mathsf{trap}_i) \leftarrow \mathsf{Sim_{eq}}^{\mathsf{crs}}(1^k)$.
   (b) For each corrupted party $P_i$, sample an *extractable* CRS: $(\mathsf{crs}_i, \mathsf{trap}_i) \leftarrow \mathsf{crsGen}_E(1^k)$.
2. Receive inputs $x'_j$ from each corrupted party $P_j$ (supposedly submitted to the Hybrid 5 ideal functionality $F'_f$). Forward these inputs to the Hybrid 6 ideal functionality $F_f$. Receive back the desired (random) evaluation answer, and forward it to $\mathcal{A}_5$ as part of the output of $F'_f$.
3. Simulate additional $F'_f$ outputs.
   For each $\ell = 1, \dots, r$, simulate as follows. For simplicity of notation, assume that party $P_1$ is an honest party within the committee $C_{\mathsf{rand}}$ (note that at least one honest party is guaranteed to exist in $C_{\mathsf{rand}}$ by Hybrid 1).
   (a) Sample *all but one* share $(b_{\ell,2}, \dots, b_{\ell,n_{\mathsf{rand}}})$ of the final secret random bit $b_\ell$ (generated by $F_f$ and used in evaluating $f$) at random. For each such bit $b_{\ell,i^*}$, generate a commitment (and decommitment information) honestly as was done by $F'_f$ in Hybrid 5,

and output accordingly. (Namely, output $(b_{\ell,i^*}, \mathsf{decom}_{\ell,i^*})$ to party $P_{i^*} \in C_{\mathsf{rand}}$, and output $\mathsf{com}_{\ell,i^*}$ to all parties in $C_{\mathsf{rand}}$.

(b) For the remaining honest party $P_1$, sample a *simulated* commitment

$$(\mathsf{eqcom}_{\ell,1}, \mathsf{eqdec}^0_{\ell,1}, \mathsf{eqdec}^1_{\ell,1}) \leftarrow \mathsf{Sim_{eq}}^{\mathsf{com}}(\mathsf{crs}_1, \mathsf{trap}_1)$$

that allegedly commits to the final secret share of the secret bit $b_\ell$. Send $\mathsf{eqcom}_{\ell,1}$ to all parties in $C_{\mathsf{rand}}$ and update the simulated secret state of $P_1$ as $\mathsf{state}_1 \leftarrow \mathsf{state}_1 \cup \{\mathsf{eqdec}^{y_\ell}_{\ell,1}\}$, where $y_\ell$ is defined to be equal to $y_\ell := b_\ell \oplus \bigoplus_{i^*=2}^{n_{\mathsf{rand}}} b_{\ell,i^*}$. Note that $\mathcal{A}_6$ does not know the secret random value $b_\ell$; however, the simulated state of party $P_1$ can be efficiently computed given this value.

4. Simulate the actions of $\mathcal{A}_5$ exactly for the remainder of the experiment.
5. Simulating leakage queries: At any point that $\mathcal{A}_5$ submits a leakage query $L$ on the secret states of honest parties, the simulating adversary $\mathcal{A}_6$ submits the leakage query $L'$ to his corresponding Hybrid 6 leakage oracle (leaking on $x_1, \dots, x_n, R$) that first reconstructs the simulated states $\{\mathsf{state}_i\}_{i \notin M}$ of honest parties as a function of the secret information $x_1, \dots, x_n, R$, and then applies $L$ to the reconstructed states.

Indistinguishability of the simulated output follows directly by the secrecy property of the xor secret sharing scheme.

□

This concludes the proof of Theorem 4.4.22.

□

## 4.5 Leakage-Resilient MPC Protocol Construction

In this section, we construct a leakage-resilient multi-function MPC protocol, as defined in Section 4.3. Our construction uses the following ingredients:

1. $(\mathcal{C}, \mathsf{Update})$: a $\lambda$-LDS secure circuit compiler, as in Theorem 4.2.14.
Recall for a circuit $C$, the compiler $\mathcal{C} : C \mapsto (\mathsf{Sub}_1, ..., \mathsf{Sub}_m)$ yields a collection of modules whose sequential execution evaluates $C$, and which are secure in the LDS model (see Section 4.2.5 for details).

2. Elect: a public-coin protocol for electing $m$ disjoint committees, each of size approximately $k$, as in Lemma 4.2.9.

3. $(\mathsf{crsGen}, \mathsf{Com}, \mathsf{Rec}, \mathsf{Sim_{eq}} = (\mathsf{Sim_{eq}}^{\mathsf{crs}}, \mathsf{Sim_{eq}}^{\mathsf{com}}))$: a crs-based equivocal commitment scheme, as in Lemma 4.4.6.

4. $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$: a fully homomorphic public-key encryption (FHE) scheme that is certifiable with respect to an efficiently testable set $R \subseteq \{0,1\}^{\mathsf{poly}(k)}$, as described in Section 5.2.3.

5. $(\mathsf{Gen}_{\mathsf{nizk}}, \mathsf{P}, \mathsf{V}, \mathcal{S}_{\mathsf{nizk}} = (\mathcal{S}^{\mathsf{crs}}_{\mathsf{nizk}}, \mathcal{S}^{\mathsf{proof}}_{\mathsf{nizk}}))$: a non-interactive zero-knowledge (NIZK) proof of knowledge (as in Lemma 4.2.3) for the NP language

$$L = \{(\mathsf{pk}, \hat{x}) : \exists\ (x, r) \text{ s.t. } r \in R,\ \hat{x} = \mathsf{Enc}_{\mathsf{pk}}(x; r)\}, \tag{4.1}$$

where $R \subseteq \{0,1\}^{\mathsf{poly}(k)}$ is the set for which the FHE scheme is certifiable.

6. $\mathsf{MPC}(F)$: a standard multiparty computation protocol for evaluating a function $F$, with no leakage resilience guarantees, such as [GMW87].

7. $(\mathsf{Gen}_{\mathsf{WLR}}, \mathsf{MPC}_{\mathsf{WLR}}(F))$: a $\lambda$-*weakly* leakage-resilient multiparty computation (WLR-MPC) protocol for evaluating a *randomized* function $F$ in the common random string model, as given by Theorem 4.4.22.

**Theorem 4.5.1.** *Fix any constants $\epsilon, \delta > 0$. Then, assuming the existence of the ingredients 1 - 7 listed above (where the* LDS *circuit compiler and WLR-MPC protocol are secure with leakage parameter $\lambda$), there exists a $\lambda$-leakage-resilient multi-function evaluation MPC protocol* $\Pi = (\Pi_{\mathsf{Pre}}, \Pi_{\mathsf{input}}, \Pi_{\mathsf{Update}})$ *for $n \geq k^\delta$ parties, tolerating $t = (1-\epsilon)n$ corrupted parties.*

**Remark.** The reason we need the number of parties to be polynomially related to the security parameter is two-fold. First, in the preprocessing phase, the protocol $\Pi_{\mathsf{Pre}}$ elects committees $\mathcal{E}_1, \ldots, \mathcal{E}_m$, and security of the protocol relies on the fact that these committees are disjoint and each committee contains at least one honest party. Thus, if $n$ is a constant, then the resulting security guarantee is that the advantage of any PPT distinguisher in the security game is bounded (from below) by a constant. More generally, the advantage is $\geq 2^{-\epsilon n}$ (see Lemma 4.2.9). This is also the reason we require that at least a constant fraction of the parties are honest.[15]

The second reason we need the number of parties to be large is that the number of disjoint committees $\mathcal{E}_1, \ldots, \mathcal{E}_m$ we need to elect is large. The reason is that the number of committees is exactly the number of modules generated by the LDS compiler, when applied to the decryption circuit $\mathsf{Dec}_{\mathsf{sk}}$ of the underlying FHE scheme. Since the only LDS compiler we know (that does not use secure hardware) requires $m = O(|\mathsf{Dec}_{\mathsf{sk}}|)$, the number of modules must be at least the security parameter of the underlying FHE scheme (which we can set to be $k^\delta$).

We now present the protocol $\Pi = (\Pi_{\mathsf{Pre}}, \Pi_{\mathsf{input}}, \Pi_{\mathsf{Online}})$, where $\Pi_{\mathsf{Online}} = (\Pi_{\mathsf{Comp}}, \Pi_{\mathsf{Update}})$. At a high level, $\Pi$ is defined as follows:

**Preprocessing phase $\Pi_{\mathsf{Pre}}$:** In the preprocessing phase, the parties run a (standard) MPC to collectively generate a key pair $(\mathsf{pk}, \mathsf{sk})$ for the FHE scheme, and to secret share $\mathsf{sk}$ in such a way that (a) learning the shares of corrupted parties, and leakage on each remaining share, does not damage the security of the FHE, but (b) collectively, the shares can be used to evaluate the decryption circuit in a leaky environment. More specifically, shares are generated by running the LDS compiler on the decryption circuit $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$ (with $\mathsf{sk}$ hardwired) to obtain a sequence of modules $\mathsf{Sub}_1, \ldots, \mathsf{Sub}_m$; the parties elect corresponding (disjoint) committees

[15]Again, this requirement can be relaxed if all we want is the distinguishing probability to be bounded by $k^{-\log k}$, as opposed to bounded by $2^{-k^{O(1)}}$.

$\mathcal{E}_1, ..., \mathcal{E}_m$, and secret share each $\mathsf{Sub}_j$ among parties in $\mathcal{E}_j$, using the simple xor secret sharing scheme. To ensure that parties provide the correct secret shares of the $\mathsf{Sub}_j$'s in future computations, within the MPC the parties collectively generate and publish commitments to each correct share.

In addition, the preprocessing phase is used to generate crs setup information for subsidiary tools used throughout the protocol. This is also done via a (standard) MPC.

(Note that the preprocessing procedure is independent of parties' secret inputs and functions to be evaluated.)

**Input phase** $\Pi_{\mathsf{Input}}$**:** Each time a party $P_i$ wishes to submit a new secret input $x_i$, she computes and publishes an encryption $\hat{x}_i$ of $x_i$ under the FHE scheme (specifically, under the public key pk for the FHE that was generated during the preprocessing phase). To ensure that malicious parties do not send malformed ciphertexts, which could ruin the correctness of homomorphic evaluation later down the line (and potentially damage security), each party accompanies her published ciphertext $\hat{x}_i$ with a NIZK proof of knowledge that the ciphertext is properly formed.

**Online phase** $\Pi_{\mathsf{Online}}$**:** The online phase consists of two parts: the *computation phase*, in which parties collectively evaluate a queried function $f$ on all inputs, and the *update phase*, in which parties collectively refresh their secret states.

**Computation phase** $\Pi_{\mathsf{Comp}}$**:** Each time the adversary requests the evaluation of a function $f$ on all parties' inputs, two steps take place. First, each party (individually) homomorphically evaluates the function $f$ on the *encrypted* vector of inputs $\hat{x} = (\hat{x}_1, ..., \hat{x}_n)$. Note that the result, $\hat{y}_f$, is an encryption of the desired value $f(\vec{x})$. Next, the parties jointly decrypt, using their shares of sk from the preprocessing phase. Namely, the parties execute the sequence of modules $\mathsf{Sub}_1, ..., \mathsf{Sub}_m$ obtained by the LDS compiler applied to $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$, where the input to the first module $\mathsf{Sub}_1$ is $\hat{y}_f$. To emulate the execution of each module $\mathsf{Sub}_j$, the parties of committee $\mathcal{E}_j$ run a WLR-MPC protocol among themselves. Within the WLR-MPC, the parties of $\mathcal{E}_j$ combine their secret shares $\mathsf{Sub}_{j,i}$ (checking first to make sure each party's share agrees with the corresponding published commitment) and execute the computation dictated by $\mathsf{Sub}_j$. Communication between modules is performed by having *all* parties of committee $\mathcal{E}_j$ send the appropriate message to *all* parties of the next committee, $\mathcal{E}_{j+1}$. The output of the final module, $\mathsf{Sub}_m$, is the evaluation $f(\vec{x})$.

**Update phase** $\Pi_{\mathsf{Update}}$**:** Each time the adversary requests that parties update their secret states, the parties execute the update procedure of the LDS compiler, where each module computation is performed via a WLR-MPC among the parties of the corresponding committee, as above. The only difference here is that the secret state $\mathsf{Sub}_j$ of each module is also changing. Thus, during each execution of a module $\mathsf{Sub}_j$, the corresponding committee must also generate fresh secret shares for its parties, and new commitment and decommitment information for each share. To provide the required correctness and secrecy guarantees, this process takes place as part of the committee's WLR-MPC execution.

The formal description of $\Pi_{\mathsf{Pre}}$ appears in Figure 4.5. The formal description of $\Pi_{\mathsf{Input}}$ appears in Figure 4.6. The formal description of $\Pi_{\mathsf{Comp}}$ appears in Figure 4.7, and the formal description of $\Pi_{\mathsf{Update}}$ appears in Figure 4.8.

**Remark 4.5.2.** Throughout the protocol description (as well as throughout the proof), we define abort to be the action of broadcasting the message "abort" to all parties. At any point in which a party receives an "abort" message, he runs abort and exits the protocol.

**Remark 4.5.3.** Recall that we assume (for simplicity) that all functions $f$ to be evaluated output the same value to all parties. If one wishes to consider functions $f$ that output a *secret* value to a single party $P_i$, then $\Pi_{\mathsf{Comp}}$ can be modified in the following way. In the WLR-MPC execution of the final module $\mathsf{Sub}_m$, rather than outputting $\mathsf{input}_{m+1} = \mathsf{Sub}_m(\mathsf{input}_m)$ to all parties in $\mathcal{E}_m$, instead generate secret shares of $\mathsf{input}_{m+1}$ for each party in $\mathcal{E}_m$ (using the simple xor secret sharing scheme), along with corresponding commitment and decommitment information. Each party of $\mathcal{E}_m$ then sends his secret share of $\mathsf{input}_{m+1}$, together with corresponding decommitment information, to the receiving party $P_i$. Clearly, once $P_i$ learns enough information to reconstruct the final secret output $\mathsf{input}_{m+1}$, one cannot hope to protect it from being leaked on by the adversary; however, this modification will guarantee that the adversary learns *only* leakage information on this output.

## 4.6 Proof of Security

*Proof.* Let $\mathcal{A}$ be any real-world PPT adversary for $\Pi$. Denote by $M \subset \mathcal{P}$ the set of parties corrupted by $\mathcal{A}$.

We construct an adversary $\mathcal{S}$ in the ideal world who simulates the real-world view of $\mathcal{A}$ by simulating the honest parties in the real world experiment. We do so by a sequence of intermediate steps, where we show how to simulate these values given less and less information, eventually given only the function evaluations $f(x_1, ..., x_n)$, as in the ideal-world experiment. More explicitly, we consider the following sequence of hybrid experiments. We note that all ideal functionalities in the hybrid experiments are implicitly *with abort*: i.e., the ideal functionality first outputs to only the adversary, who decides whether outputs are also delivered to honest parties, or whether the protocol ends in abort.

**Hybrid 0.** The real world: i.e., the adversary interacts with honest parties in the real-world experiment running $\Pi$.

**Hybrid 1.** The same as the real-world experiment, except that if any of the committees $\mathcal{E}_1, ..., \mathcal{E}_m$ elected during the preprocessing phase satisfies $|\mathcal{E}_j| \leq k'$ (where $k'$ is the security parameter for the FHE scheme), or $\frac{|\mathcal{E}_j \cap M|}{|\mathcal{E}_j|} \geq (1-\epsilon) + \eta$ (for a fixed constant $\eta > 0$) the experiment immediately concludes with output fail. We assume for simplicity of notation that, if the experiment does not fail, the first party of each committee $\mathcal{E}_j$ is honest.

**Hybrid 2.** The same as Hybrid 1, except instead of collectively generating the CRS values (for the equivocal commitment scheme, the WLR-MPC, and the NIZK proof system) via an MPC protocol during the preprocessing phase, we assume a setup model where these values

**Preprocessing Phase:** No inputs. No leakage allowed.

1. The parties elect $m$ disjoint committees $\mathcal{E}_j$ of size approximately $k'$ by running $\mathsf{Elect}$. Here, $m = \mathsf{poly}(k')$ is the number of modules produced by the $\mathsf{LDS}$ compiler, when run on the decryption circuit $\mathsf{Dec}_{\mathsf{sk}}$ of the FHE scheme with security parameter $k' = k^{O(1)}$. We take $k'$ so that $m \cdot k' \leq n$

2. All parties engage in an execution of the (standard) MPC protocol $\mathsf{MPC}(F_{\mathsf{crs}})$ to compute the (randomized) functionality $F_{\mathsf{crs}}$ described as follows. Functionality $F_{\mathsf{crs}}$ does not take any inputs and computes the following: (a) a CRS $\mathsf{crs}_{\mathsf{WLR}} \leftarrow \mathsf{Gen}_{\mathsf{WLR}}(1^k)$ for the weakly leakage-resilient MPC protocol $(\mathsf{Gen}_{\mathsf{WLR}}, \mathsf{MPC}_{\mathsf{WLR}}(F))$, (b) a CRS $\mathsf{crs}^i_{\mathsf{eq}} \leftarrow \mathsf{crsGen}(1^k)$ for each party $P_i$ for the equivocal commitment scheme $(\mathsf{crsGen}, \mathsf{Com}, \mathsf{Rec}, \mathsf{Sim}_{\mathsf{eq}})$, and (c) a CRS $\mathsf{crs}^i_{\mathsf{nizk}} \leftarrow \mathsf{Gen}_{\mathsf{nizk}}(1^k)$ for each party $P_i$ for the NIZK proof of knowledge system $(\mathsf{Gen}_{\mathsf{nizk}}, \mathsf{P}, \mathsf{V}, \mathcal{S}_{\mathsf{nizk}})$. Denote by $\mathsf{crs}$ the tuple $(\{\mathsf{crs}^i_{\mathsf{eq}}, \mathsf{crs}^i_{\mathsf{nizk}}\}_{i=1}^n, \mathsf{crs}_{\mathsf{WLR}})$.

3. All parties engage in an execution of the (standard) MPC protocol $\mathsf{MPC}(F_{\mathcal{E}_1,\ldots,\mathcal{E}_m,\mathsf{crs}})$ to collectively compute the randomized functionality $F_{\mathcal{E}_1,\ldots,\mathcal{E}_m,\mathsf{crs}}$ (that does not take any inputs) defined as follows:

   The (randomized) function:

   (a) Generate a key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^{k'})$ for the FHE scheme.

   (b) Evaluate the $\mathsf{LDS}$ circuit transformation on the decryption circuit for $\mathsf{sk}$:

   $$(\mathsf{Sub}_1, \ldots, \mathsf{Sub}_m) \leftarrow \mathcal{C}(\mathsf{Dec}_{\mathsf{sk}}).$$

   (We abuse notation and denote by $\mathsf{Sub}_j$ both the computation of the submodule and the secret state corresponding to the submodule.)

   (c) For each $j \in [m]$, secret share $\mathsf{Sub}_j = \mathsf{Sub}_{j,1} \oplus \cdots \oplus \mathsf{Sub}_{j,|\mathcal{E}_j|}$ among the parties in the $j$'th committee, $\mathcal{E}_j$, using the xor secret sharing scheme.

   (d) For each share $\mathsf{Sub}_{j,i}$ generated in the previous step, compute a commitment $(c_{j,i}, d_{j,i}) \leftarrow \mathsf{Com}(\mathsf{crs}^{\alpha}_{\mathsf{eq}}, \mathsf{Sub}_{j,i})$, where $P_\alpha$ is the $i$'th party in $\mathcal{E}_j$ (i.e., the party that receives the share $\mathsf{Sub}_{j,i}$).

   Output: The outputs are as follows.

   | | |
   |---|---|
   | All parties: | $\mathsf{pk}$, $\{c_{j,i}\}_{j\in[m],i\in[\lvert\mathcal{E}_j\rvert]}$ |
   | Party $i$ of $\mathcal{E}_j$: | $\mathsf{Sub}_{j,i}$, $d_{j,i}$ |

4. Each party erases all intermediate values of the MPC executions.

(Note that Steps 2 and 3 can be combined into a single multi-party computation execution, but have been split into two separate executions for ease of explanation and proof).

**Figure 4.5:** Protocol $\Pi_{\mathsf{Pre}}$: Preprocessing phase.

**Input Phase:** Party $P_i$ wishes to submit a new private input, $x_i$. No leakage allowed.
Public inputs: $\mathsf{pk}$, $\{\mathsf{crs}^i_{\mathsf{nizk}}\}_{i=1}^n$.
Private input: $x_i$, held by party $P_i$.

Party $P_i$ performs the following steps:

1. Sample a value $r_i \leftarrow R \subseteq \{0,1\}^{\mathsf{poly}(k)}$ via rejection sampling. Recall the FHE scheme is certifiable with respect to the set $R \subseteq \{0,1\}^{\mathsf{poly}(k)}$ (see Definition 5.2.4).
2. Encrypt $\hat{x}_i = \mathsf{Enc}_{\mathsf{pk}}(x_i; r_i)$.
3. Compute a NIZK proof of knowledge that $(\mathsf{pk}, \hat{x}_i) \in L$ using witness $(x_i, r_i)$ and CRS $\mathsf{crs}^i_{\mathsf{nizk}}$. (See Equation (4.1) above for the definition of $L$). That is, $\pi_i \leftarrow \mathsf{P}(\mathsf{crs}^i_{\mathsf{nizk}}, (\mathsf{pk}, \hat{x}_i), (x_i, r_i))$.
4. Send the pair $(\hat{x}_i, \pi_i)$ to all parties.
   (It suffices to send it to parties in $\mathcal{E}_1$.)
5. Erase initial input $x_i$, together with all intermediate values of the input phase.

**Figure 4.6:** Protocol $\Pi_{\mathsf{input}}$: Input phase.

are (honestly) generated beforehand, and all parties run with these CRS values as shared common knowledge. We denote this ideal functionality by $\mathsf{crs}$.

*Ideal functionalities in Hybrid 2:* $\mathsf{crs}$.

**Hybrid 3.** The same as Hybrid 2, except that some of the CRS values are generated using the simulation algorithms. More specifically,

- For the first party in each committee, its crs for the equivocal commitment scheme is generated using the simulator; i.e., for each such party $P_i$, $(\mathsf{crs}^i_{\mathsf{eq}}, \mathsf{trap}^i) \leftarrow \mathsf{Sim_{eq}}^{\mathsf{crs}}(1^k)$.
- For each malicious party $P_\ell$, we generate its crs for the NIZK proof of knowledge using the simulator, by computing $(\mathsf{crs}^\ell_{\mathsf{nizk}}, \mathsf{trap}^\ell) \leftarrow \mathcal{S}^{\mathsf{crs}}_{\mathsf{nizk}}(1^k)$.
- The crs for the WLR-MPC protocol is simulated by computing $(\mathsf{crs_{WLR}}, \mathsf{trap_{WLR}}) \leftarrow \mathcal{S}^{\mathsf{crs}}_{\mathsf{WLR}}(1^k)$.

The remaining crs values are generated honestly, as before. We denote this new ideal functionality by $\mathsf{crsSim}$.

*Ideal functionalities in Hybrid 3:* $\mathsf{crsSim}$.

**Hybrid 4.** The same as Hybrid 3, except that the second MPC in the preprocessing phase (which generates a key pair for the FHE scheme, runs the $\mathsf{LDS}$ transformation, etc) is replaced by the corresponding ideal (randomized) functionality $F_{\mathsf{Pre}}$. Note that $F_{\mathsf{Pre}}$ takes no inputs.

Overall, this hybrid is the same as the real world, except that the preprocessing phase consists only of the execution of $\mathsf{Elect}$ and one-time oracle access to $\mathsf{crsSim}$ and $F_{\mathsf{Pre}}$.

*Ideal functionalities in Hybrid 4:* $\mathsf{crsSim}, F_{\mathsf{Pre}}$.

**Computation Phase:**
Public inputs: $f$, $\mathsf{pk}$, $\hat{x} = (\mathsf{Enc}_{\mathsf{pk}}(x_1), ..., \mathsf{Enc}_{\mathsf{pk}}(x_n))$, $\mathsf{crs} = (\{\mathsf{crs}^i_{\mathsf{eq}}, \mathsf{crs}^i_{\mathsf{nizk}}\}_{i=1}^n, \mathsf{crs}_{\mathsf{WLR}})$, $\mathcal{E}_1, ..., \mathcal{E}_m$, $\{c_{j,i}\}_{j\in[m], i\in[|\mathcal{E}_j|]}$.
Private inputs: $(\mathsf{Sub}_{j,i}, d_{j,i})$, held by party $i$ of $\mathcal{E}_j$.

1. All parties homomorphically evaluate $f$ on the encrypted input vector: $\hat{y} = \mathsf{Eval}_{\mathsf{pk}}(\hat{x}, f)$.
   (It suffices that only parties in $\mathcal{E}_1$ compute $\hat{y}$.)
2. The parties execute the Decryption Cascade with $\mathsf{input}_1 = \hat{y}$.

Decryption Cascade:

1. For $j = 1, ..., m$:
   (a) The parties in $\mathcal{E}_j$ engage in an execution of the $\lambda$-weakly leakage-resilient MPC protocol $\mathsf{MPC}_{\mathsf{WLR}}(F_j)$ using CRS $\mathsf{crs}_{\mathsf{WLR}}$ to compute the (randomized) functionality $F_j$ defined as follows:
   Input: $(\mathsf{Sub}_{j,i}, d_{j,i}, \mathsf{input}_j)$, held by party $i$ of $\mathcal{E}_j$.
   The function $F_j$:
      i. If any of the $\mathsf{input}_j$'s are inconsistent, or $\mathsf{Sub}_{j,i} \neq \mathsf{Rec}(\mathsf{crs}^{\alpha}_{\mathsf{eq}}, c_{j,i}, d_{j,i})$ for any $i$, where $P_\alpha$ is the $i$'th party in committee $\mathcal{E}_j$ (i.e., if any party's share does not agree with the corresponding published commitment), then $\mathsf{abort}$.
      ii. Otherwise, let $\mathsf{Sub}_j = \bigoplus_{i=1}^{|\mathcal{E}_j|} \mathsf{Sub}_{j,i}$.
      iii. Evaluate the $j$'th module on $\mathsf{input}_j$: that is, $\mathsf{input}_{j+1} := \mathsf{Sub}_j(\mathsf{input}_j)$.
   Output: All parties learn $\mathsf{input}_{j+1}$.
   At the conclusion of the WLR-MPC execution, each party in $\mathcal{E}_j$ erases all intermediate values generated during the WLR-MPC, keeping only $(\mathsf{Sub}_{j,i}, d_{j,i})$.
   (b) Each party in $\mathcal{E}_j$ sends the value of $\mathsf{input}_{j+1}$ to all parties in $\mathcal{E}_{j+1}$ (where $\mathcal{E}_{m+1} := \mathcal{P}$).
   (c) If any party in $\mathcal{E}_{j+1}$ receives disagreeing values of $\mathsf{input}_{j+1}$ from parties in $\mathcal{E}_j$, then $\mathsf{abort}$.
2. Output the desired evaluation $f(x)$ to be the value $\mathsf{input}_{m+1}$.

**Figure 4.7:** Protocol $\Pi_{\mathsf{Comp}}$: Compute phase.

**Update Phase:**
Public inputs: $\mathcal{E}_1, ..., \mathcal{E}_m$, $\mathsf{crs} = (\{\mathsf{crs}^i_{\mathsf{eq}}, \mathsf{crs}^i_{\mathsf{nizk}}\}_{i=1}^n, \mathsf{crs}_{\mathsf{WLR}})$, $\{c_{j,i}\}_{j\in[m], i\in[|\mathcal{E}_j|]}$.
Private inputs: $(\mathsf{Sub}_{j,i}, d_{j,i})$, held by party $i$ of $\mathcal{E}_j$.

All parties run the $\mathsf{Update}$ protocol of the LDS compiler, as follows.

1. Each time the parties in committee $\mathcal{E}_j$, who are simulating submodule $\mathsf{Sub}_j$, receive a message $\mathsf{msg}_j$ from the parties in committee $\mathcal{E}_{j-1}$, who are simulating submodule $\mathsf{Sub}_{j-1}$, they compute the function $G$ that would have been computed by $\mathsf{Sub}_j$ upon receiving the message $\mathsf{input}_j$ when running the $\mathsf{Update}$ protocol. (The parties in committee $\mathcal{E}_1$ start with $\mathsf{msg}_1 \triangleq \perp$).

   The computation of $G$ is done by running an execution of the $\lambda$-weakly leakage-resilient MPC protocol using $\mathsf{crs}_{\mathsf{WLR}}$ to collectively execute the following (randomized) function:

   Input: $(\mathsf{Sub}_{j,i}, d_{j,i}, \mathsf{msg}_j)$, held by party $i$ of $\mathcal{E}_j$,
   $\mathsf{crs}$, $\{c_{j,i}\}_{i\in[|\mathcal{E}_j|]}$, held by all parties.

   The (randomized) function:

   (a) If any of the $\mathsf{msg}_j$'s are inconsistent, or $\mathsf{Sub}_{j,i} \neq \mathsf{Rec}(\mathsf{crs}^\alpha_{\mathsf{eq}}, c_{j,i}, d_{j,i})$ for any $i$, where $P_\alpha$ is the $i$'th party in committee $\mathcal{E}_j$ (i.e., if any party's share does not agree with the corresponding published commitment), then $\mathsf{abort}$. Otherwise, let $\mathsf{Sub}_j = \bigoplus_{i=1}^{|\mathcal{E}_j|} \mathsf{Sub}_{j,i}$.

   (b) Evaluate $(\mathsf{Sub}'_j, \mathsf{msg}_{j+1}) \leftarrow G(\mathsf{Sub}_j, \mathsf{msg}_j)$. Here, $\mathsf{Sub}'_j$ denotes an updated version of the submodule information, and $\mathsf{msg}_{j+1}$ denotes the message to be sent to submodule $j+1$ as dictated by $\mathsf{Update}$.

   (c) Secret share the new value $\mathsf{Sub}'_j = \mathsf{Sub}'_{j,1} \oplus \cdots \oplus \mathsf{Sub}'_{j,|\mathcal{E}_j|}$ into $|\mathcal{E}_j|$ shares using the xor secret sharing scheme.

   (d) For each share $\mathsf{Sub}'_{j,i}$ generated in the previous step, compute a new commitment $(c'_{j,i}, d'_{j,i}) \leftarrow \mathsf{Com}(\mathsf{crs}^\alpha_{\mathsf{eq}}, \mathsf{Sub}'_{j,i})$, where $P_\alpha$ is the $i$'th party in committee $\mathcal{E}_j$.

   Output: The outputs are as follows.

   All parties: $\mathsf{msg}_{j+1}$, $\{c'_{j,i}\}_{i\in[|\mathcal{E}_j|]}$
   Party $i$ of $\mathcal{E}_j$: $\mathsf{Sub}'_{j,i}$, $d'_{j,i}$

   At the conclusion of the WLR-MPC execution, each party in $\mathcal{E}_j$ erases all intermediate values generated during the WLR-MPC, keeping only $(\mathsf{Sub}_{j,i}, d_{j,i})$.

2. *All* the parties of $\mathcal{E}_j$ send $\mathsf{msg}_{j+1}$ to *all* parties in $\mathcal{E}_{j+1}$.

3. Each party in $\mathcal{E}_j$ broadcasts all new commitments $\{c'_{j,i}\}_{i\in[|\mathcal{E}_j|]}$. If any disagreeing values are sent by parties in $\mathcal{E}_j$, then $\mathsf{abort}$.

At the conclusion of the update phase, each party erases their initial input together with all intermediate values of the update phase.

**Figure 4.8:** Protocol $\Pi_{\mathsf{Update}}$: Update phase.

**Hybrid 5.** The same as Hybrid 4, except each underlying weakly leakage-resilient MPC execution in the decryption cascade is replaced with the ideal functionality $F_j$ that accepts inputs from all parties in $\mathcal{E}_j$, samples uniform randomness, and replies with the evaluation of $F_j$ on these inputs and randomness (as described in Figure 4.7). Similarly, each WLR-MPC execution in the update phase is replaced with the ideal functionality $G_j$ that accepts inputs from parties, samples randomness, and replies with the evaluation of $G_j$ on these inputs and randomness (as described in Figure 4.8).

The adversary no longer makes leakage queries of the form $\mathsf{Leak}(i, L)$, as he did in all previous hybrids. Instead, leakage queries are of the form $\mathsf{Leak}(L)$, and are made directly to the ideal functionalities $\{F_j\}, \{G_j\}$. The corresponding ideal functionality evaluates the queried function $L$ on the collection of received inputs from parties and sampled randomness. As before, leakage time periods span from the beginning of one $\mathsf{Update}$ procedure to the end of the next, and the adversary may make no more than $\lambda$ leakage queries in any time period.

*Ideal functionalities in Hybrid 5:* $\mathsf{crsSim}, F_{\mathsf{Pre}}, \{F_j\}, \{G_j\}$.

**Hybrid 6.** Same as Hybrid 5, except that the ideal functionality $F_{\mathsf{Pre}}$ is replaced by a slightly modified functionality $F'_{\mathsf{Pre}}$. Loosely speaking, $F'_{\mathsf{Pre}}$ is the same as $F_{\mathsf{Pre}}$, except that for the first party in each committee (which is assumed to be honest), $F'_{\mathsf{Pre}}$ generates a *simulated* commitment to the party's secret share.

Explicitly, $F'_{\mathsf{Pre}}$ has a trapdoor $\mathsf{trap}^j$, for the first party of each committee $\mathcal{E}_j$, hardwired into it. Just as $F_{\mathsf{Pre}}$, the functionality $F'_{\mathsf{Pre}}$ takes no inputs; it samples a key pair for the FHE scheme, evaluates the LDS transformation of the circuit $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$, and generates secret shares $\mathsf{Sub}_{j,i}$ for each of the resulting secret modules. Further, $F'_{\mathsf{Pre}}$ honestly generates a commitment $(c_{j,i}, d_{j,i}) \leftarrow \mathsf{Com}(1^k, \mathsf{Sub}_{j,i})$ to $\mathsf{Sub}_{j,i}$ as usual for the secret share of all *but the first* party in each committee. For the *first* party in each committee (which is assumed to be honest), $F'_{\mathsf{Pre}}$ generates a *simulated* commitment $(\tilde{c}_{j,1}, \tilde{d}^0_{j,1}, \tilde{d}^1_{j,1}) \leftarrow {\mathsf{Sim}_{\mathsf{eq}}}^{\mathsf{com}}(\mathsf{crs}^j_{\mathsf{eq}}, \mathsf{trap}^j)$, and sets $d_{j,1} = \tilde{d}_{j,1}^{\mathsf{Sub}_{j,1}}$.

*Ideal functionalities in Hybrid 6:* $\mathsf{crsSim}, F'_{\mathsf{Pre}}, \{F_j\}, \{G_j\}$.

**Hybrid 7.** Same as Hybrid 6, except that the ideal functionalities $\{G_j\}$ are modified in the same fashion as the step above. Namely, we replace each $G_j$ with a new ideal functionality $G'_j$ with the following differences. $G'_j$ has a trapdoor $\mathsf{trap}^j$, for the first party of each committee $\mathcal{E}_j$, hardwired into it. $G'_j$ accepts the same inputs as $G_j$, and carries out the same computation as $G_j$, with the following exception: For the first party in each committee, instead of honestly generating a commitment to the secret share $\mathsf{Sub}_{j,1}$, the functionality $G'_j$ generates a *simulated* commitment $(\tilde{c}_{j,1}, \tilde{d}^0_{j,1}, \tilde{d}^1_{j,1}) \leftarrow {\mathsf{Sim}_{\mathsf{eq}}}^{\mathsf{com}}(\mathsf{crs}^j_{\mathsf{eq}}, \mathsf{trap}^j)$, and sets $d_{j,1} = \tilde{d}_{j,1}^{\mathsf{Sub}_{j,1}}$. (Note that the ideal functionalities $\{F_j\}$ in the decryption cascade do not generate new secret shares and thus do not need to be modified in this fashion).

*Ideal functionalities in Hybrid 7:* $\mathsf{crsSim}, F'_{\mathsf{Pre}}, \{F_j\}, \{G'_j\}$.

**Hybrid 8.** Similar to Hybrid 7, except that all secret shares are eliminated, and committees interact directly with the $m$ modules $(\mathsf{Sub}_1, ..., \mathsf{Sub}_m)$. More specifically, the following changes are made:

- The ideal functionality crsSim is replaced by a slightly modified functionality crsSim$'$, which executes exactly as crsSim, but in addition sends to the adversary all trapdoors for simulated equivocal commitment crs values (for the first party in each committee).
- The ideal functionality $F'_{\mathsf{Pre}}$ is replaced by a simple ideal functionality $F_{\mathsf{pk}}$ that takes no inputs, generates a key pair (pk, sk) for the FHE scheme, and publishes pk.
- The sequence of ideal functionalities $\{F_j\}, \{G'_j\}$, as introduced in the previous steps, are replaced by the corresponding interactions with the $m$ modules $(\mathsf{Sub}_1, ..., \mathsf{Sub}_m)$ generated by the LDS compiler:

$$(\mathsf{Sub}_1, ..., \mathsf{Sub}_m) \leftarrow \mathcal{C}(\mathsf{Dec}_{\mathsf{sk}}(\cdot)).$$

  Namely,

  - The decryption cascade takes place as follows. For each $j = 1, ..., m$, beginning with $\mathcal{E}_1$ and $\mathsf{input}_1 = \hat{y}$, all parties in committee $\mathcal{E}_j$ send $\mathsf{input}_j$ to the corresponding module $\mathsf{Sub}_j$. If all $\mathsf{input}_j$'s are consistent, they receive back $\mathsf{input}_{j+1} \leftarrow \mathsf{Sub}_j(\mathsf{input}_j)$. The parties of $\mathcal{E}_j$ then send $\mathsf{input}_{j+1}$ to all parties in the next committee $\mathcal{E}_{j+1}$, who (if the received values are consistent) repeat the same process. At the conclusion of the decryption cascade, the parties of the final committee $\mathcal{E}_m$ broadcast the resulting value $\mathsf{input}_{m+1}$ (which is supposedly $f(\vec{x})$) to all parties.
  - The update procedure is similar to the decryption cascade. The modules execute the LDS update procedure, interacting with each other via the committees $\mathcal{E}_1, ..., \mathcal{E}_m$.

Instead of making leakage queries to the ideal functionalities $\{F_j\}, \{G_j\}$, the adversary now makes queries of the form $\mathsf{Leak}(j, L)$, and receives the evaluation of $L$ on the secret state of the $j$th module, $\mathsf{Sub}_j$. As before, leakage time periods span from the beginning of one Update procedure to the end of the next, and the adversary may make no more than $\lambda$ leakage queries in any time period.

*Ideal functionalities in Hybrid 8:* crsSim$'$, $F_{\mathsf{pk}}$, $\{\mathsf{Sub}_j\}$.

**Hybrid 9.** Same as Hybrid 8, except that all modules $\mathsf{Sub}_j$ are removed. Instead, parties interact with an ideal decryption functionality $\mathsf{Dec}_{\mathsf{sk}}$, as described below.

In the preprocessing phase, the parties execute Elect, and are given pk and crs values (where some of the crs values are generated with trapdoors, as described in Hybrid 3). The input phase takes place as usual. In the online phase, for each function $f$ that is queried by the adversary, the parties homomorphically compute the corresponding ciphertext $\hat{y} = \mathsf{Eval}_{\mathsf{pk}}(\hat{x}, f)$. All parties in the first committee, $\mathcal{E}_1$, send $\hat{y}$ to the ideal decryption functionality $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$ with abort. If all received $\hat{y}$'s are consistent, the ideal functionality responds by sending the resulting decryption $\mathsf{Dec}_{\mathsf{sk}}(\hat{y})$ to the adversary, where sk is the decryption key that was generated by $F_{\mathsf{pk}}$. If the adversary allows, $\mathsf{Dec}_{\mathsf{sk}}(\hat{y})$ is also sent to all honest parties; otherwise, the experiment concludes in abort. The update phase no longer takes place. No leakage queries are allowed at any point of the experiment.

*Ideal functionalities in Hybrid 9:* crsSim$'$, $F_{\mathsf{pk}}$, $\mathsf{Dec}_{\mathsf{sk}}$.

**Hybrid 10.** Differs from Hybrid 9 in the following ways:

- The ideal functionalities $\mathsf{crsSim}'$, $F_{\mathsf{Pre}}$, and the execution of Elect, are removed from the preprocessing phase.
- The input phase no longer takes place.
- The ideal decryption functionality $\mathsf{Dec}_{\mathsf{sk}}$ is replaced by the ideal-world functionality Evaluate, which takes input $x_i$ from each party and evaluates functions $f$ queried by the adversary on the set of all parties' inputs $\vec{x}$, as defined in Section 5.3.
- In addition, the adversary is given as auxiliary input

$$z' := \left(\mathsf{pk}, \{\hat{x}_i, \mathsf{crs}^i_{\mathsf{nizk}}, \pi_i\}_{i \notin M}\right),$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$, and for each honest party $P_i$, the triple $(\mathsf{crs}^i_{\mathsf{nizk}}, \hat{x}_i, \pi_i)$ is computed using the *real* input $x_i$ of $P_i$. That is, the values in the triple are computed by $\mathsf{crs}^i_{\mathsf{nizk}} \leftarrow \mathsf{Gen}_{\mathsf{nizk}}(1^k)$; $r_i \leftarrow R$; $\hat{x}_i = \mathsf{Enc}_{\mathsf{pk}}(x_i; r_i)$; $\pi_i \leftarrow \mathsf{P}(\mathsf{crs}^i_{\mathsf{nizk}}, (\hat{x}_i, \mathsf{pk})(x_i, r_i))$.

Overall, Hybrid 11 is the following.

Parties begin by submitting their inputs to the ideal functionality Evaluate. More specifically, each *honest* party $P_i$ submits his input $x_i$. The adversary is given the corresponding auxiliary input $z'$, computed as a function of the honest parties' inputs $\{x_i\}_{i \notin M}$. Upon receiving $z'$, the adversary submits the inputs of *malicious* parties to Evaluate.

The preprocessing and input phases no longer take place. During the online phase, for each function $f$ that is queried by the adversary, Evaluate responds by sending the adversary the evaluation of $f$ on the set of all submitted inputs $(x_1, ..., x_n)$. If the adversary allows, the evaluation is also sent to all honest parties; otherwise, the experiment concludes in abort.

Note that Hybrid 11 is *nearly* the ideal-world experiment. Indeed, the only difference is that the adversary is given the auxiliary input $z'$.

*Ideal functionalities in Hybrid 11:* Evaluate.

**Hybrid 11.** The ideal world: i.e., the adversary *only* receives $f(x_1, ..., x_n)$ for each $f$ selected to be computed. Note that this is the same as Hybrid 11, except that the adversary no longer receives the auxiliary input. (See Section 5.3 for the detailed experiment).

The output of each hybrid experiment consists of the outputs of all parties, where honest parties output in accordance with the dictated protocol, and malicious parties may output any efficiently computable function of the view of the adversary. For every adversary $\mathcal{A}_\ell$ with auxiliary input $z \in \{0,1\}^*$ running in hybrid experiment $\ell$ with initial inputs $\vec{x}$, we denote the output of the corresponding hybrid $\ell$ experiment by

$$\mathsf{HYB}_\ell\left(\mathcal{A}_\ell, 1^k, z, \{x_i\}_{i=1}^n\right).$$

We now go through the hybrids one by one, beginning with Hybrid 0. In each step we show that for any adversary $\mathcal{A}_\ell$ running in Hybrid $\ell$, there exists an adversary $\mathcal{A}_{\ell+1}$ running in Hybrid

$(\ell+1)$ such that

$$\mathsf{HYB}_\ell\left(\mathcal{A}_\ell, 1^k, z, \{x_i\}_{i=1}^n\right) \approx_c \mathsf{HYB}_{\ell+1}\left(\mathcal{A}_{\ell+1}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

Note that once we show this for every step $i = 0, ..., 11$, the theorem will follow, as this will imply that for each adversary $\mathcal{A}$ in the real-world experiment (Hybrid 0), there is an adversary $\mathcal{A}_{19}$ in the ideal-world experiment (Hybrid 19), such that

$$\mathsf{HYB}_0\left(\mathcal{A}, 1^k, z, \{x_i\}_{i=1}^n\right) \approx_c \mathsf{HYB}_{19}\left(\mathcal{A}_{19}, 1^k, z, \{x_i\}_{i=1}^n\right)$$

as desired.

**Step 1: Hybrid 0 to Hybrid 1. (Elect protocol).** Recall that the Hybrid 1 experiment ends in fail if any of the committees is smaller than $(\epsilon - \eta)k'$ for fixed constant $\eta > 0$, or if any committee has fewer than $\epsilon - \eta$ constant fraction of honest parties. By Lemma 4.2.9, the probability of event fail is negligible in the FHE security parameter $k'$, and thus in $k$ (as $k'$ was taken to be $k^\delta$ for a constant $\delta > 0$). Thus, for any adversary $\mathcal{A}$, it holds that $\mathsf{HYB}_0\left(\mathcal{A}, 1^k, z, \{x_i\}_{i=1}^n\right)$ and $\mathsf{HYB}_1\left(\mathcal{A}, 1^k, z, \{x_i\}_{i=1}^n\right)$ are *statistically* indistinguishable.

**Step 2: Hybrid 1 to Hybrid 2. (MPC security).** In this step, the initial MPC protocol in the preprocessing phase, which is used to generate crs values, is replaced by the corresponding ideal functionality. Consider an adversary $\mathcal{A}_1$ in Hybrid 1. We think of $\mathcal{A}_1$ as being composed of three sub-algorithms $(\mathcal{A}_1^{pre}, \mathcal{A}_1^{\mathsf{MPC}}, \mathcal{A}_1^{post})$, corresponding to the actions of $\mathcal{A}_1$ before, during, and after this MPC. Each sub-adversary receives as auxiliary input the complete view of $\mathcal{A}_1$ up to that point in an execution of Hybrid 1, and outputs the corresponding view of $\mathcal{A}_1$ at the conclusion of its execution. We construct a corresponding adversary $\mathcal{A}_2$ in Hybrid 2.

Pre-MPC: Up to the beginning of the MPC, $\mathcal{A}_2$ acts exactly as $\mathcal{A}_1^{pre}$. Note that during this time, the two hybrids are identical. Let $z_{\mathsf{MPC}}$ denote the view of $\mathcal{A}_1$ up to this point.

MPC: At this point, the hybrids diverge. By the security of the underlying (leak-free) MPC protocol, there exists a simulator $\mathcal{S}_{\mathsf{MPC}}$ corresponding to $\mathcal{A}_1^{\mathsf{MPC}}$. $\mathcal{A}_2$ runs $\mathcal{S}_{\mathsf{MPC}}$ with the set of values output by the ideal functionality crs, together with auxiliary input $z_{\mathsf{MPC}}$. (Note that the crs-generation MPC takes no inputs). Let $z_{post}$ denote the output of $\mathcal{S}_{\mathsf{MPC}}$.

Post-MPC: At the conclusion of the crs MPC, $\mathcal{A}_2$ acts exactly as $\mathcal{A}_1^{post}$ with auxiliary input $z_{post}$.

**Claim 4.6.1.** *For any auxiliary input $z$ and set of inputs $\vec{x}$,*

$$\mathsf{HYB}_1\left(\mathcal{A}_1, 1^k, z, \{x_i\}_{i=1}^n\right) \approx_c \mathsf{HYB}_2\left(\mathcal{A}_2, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Suppose, for contradiction, there exists a set of inputs $\vec{x}$, auxiliary input $z$, and a PPT distinguisher $\mathcal{D}$ for which

$$\left|\Pr\left[\mathcal{D}\left(\mathsf{HYB}_1\left(\mathcal{A}_1, 1^k, z, \{x_i\}_{i=1}^n\right)\right) = 1\right] - \Pr\left[\mathcal{D}\left(\mathsf{HYB}_2\left(\mathcal{A}_2, 1^k, z, \{x_i\}_{i=1}^n\right)\right) = 1\right]\right| \geq \epsilon$$

for some noticeable $\epsilon$. We construct an adversary $\mathcal{A}'$, set of inputs $\vec{x}'$, auxiliary input $z'$, and efficient distinguisher $\mathcal{D}'$ that together break the security of the underlying MPC. Take $\mathcal{A}'$ to be the sub-adversary $\mathcal{A}_1^{\mathsf{MPC}}$, and let $\vec{x}' = \emptyset$ (recall the crs-generation MPC does not take inputs). Let $z_{\mathsf{MPC}}$ denote the entire view of $\mathcal{A}_1$ during a real execution of $\Pi$ up to the MPC execution, executed on the above fixed set of inputs $\vec{x}$. Note that $z_{\mathsf{MPC}}$ can be efficiently simulated given the set of all inputs $\vec{x}$ and auxiliary input $z$. Take $z' = z_{\mathsf{MPC}}$.

The distinguisher $\mathcal{D}'$ is given a challenge output $y'$ that is either the output of a real execution of the MPC with adversary $\mathcal{A}'(z')$, or is simulated. Using the given set of all inputs $x$, and the auxiliary input $z_{\mathsf{MPC}}$, together with $y'$ (which corresponds to the collection of crs values, in addition to the entire view of $\mathcal{A}_2$ up to the conclusion of the MPC), $\mathcal{D}'$ simulates the remainder of the protocol $\Pi$. At the conclusion of the simulated experiment, $\mathcal{D}'$ sends the resulting set of all parties' outputs to the distinguisher $\mathcal{D}$. If the challenge output $y'$ was the output of a real execution of the MPC, then this distribution is identical to $\mathsf{HYB}_1\left(\mathcal{A}_1, 1^k, z, \{x_i\}_{i=1}^n\right)$. However, if the challenge output $y'$ was simulated, then the distribution is identical to $\mathsf{HYB}_2\left(\mathcal{A}_2, 1^k, z, \{x_i\}_{i=1}^n\right)$. Thus, $\mathcal{D}'$ distinguishes the two cases with noticeable advantage $\epsilon$.

□

**Step 3: Hybrid 2 to Hybrid 3. (Simulation of crs for Eq-Com, NIZK, WLR-MPC).** In this step, some of the crs values $\{\mathsf{crs}^i_{\mathsf{eq}}\}, \{\mathsf{crs}^i_{\mathsf{nizk}}\}, \mathsf{crs}_{\mathsf{WLR}}$ are sampled using the simulator in the place of the honest generation algorithms. For any adversary $\mathcal{A}_2$ in Hybrid 2, let $\mathcal{A}_3$ be the adversary in Hybrid 3 who acts identically to $\mathcal{A}_2$.

**Claim 4.6.2.** *For any auxiliary input $z$ and set of inputs $\vec{x}$,*

$$\mathsf{HYB}_2\left(\mathcal{A}_2, 1^k, z, \{x_i\}_{i=1}^n\right) \approx_c \mathsf{HYB}_3\left(\mathcal{A}_3, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* To prove this claim it suffices to prove that the sequence of the crs's generated in Hybrid 2 is computationally indistinguishable from the sequence of crs's generated in Hybrid 3. This follows from a standard hybrid argument, together with the fact that a simulated crs is computationally indistinguishable from an honestly generated crs, for the NIZK proof system (see Definition 5.2.1), for the equivocal commitment scheme (see Definition 4.4.4), and for the weakly leakage resilient MPC protocol (see Definition 4.2.17).

□

**Step 4: Hybrid 3 to Hybrid 4. (MPC security).** In this step, the MPC protocol computing $F_{\mathsf{Pre}}$ in the preprocessing phase in Hybrid 3 is replaced by the corresponding ideal functionality $F_{\mathsf{Pre}}$. The proof of this step is almost identical to the proof of Step 2.

Let $\mathcal{A}_3$ be a PPT adversary in Hybrid 3. We think of $\mathcal{A}_3$ as being composed of three sub-algorithms $(\mathcal{A}_3^{pre}, \mathcal{A}_3^{\mathsf{MPC}}, \mathcal{A}_3^{post})$, corresponding to the actions of $\mathcal{A}_3$ before, during, and after the MPC of $\mathbb{F}_{\mathsf{Pre}}$. Each sub-adversary receives as auxiliary input the complete view of $\mathcal{A}_3$ up to that point in an execution of Hybrid 3, and outputs the corresponding view of $\mathcal{A}_3$ at the conclusion of its execution. We construct a corresponding adversary $\mathcal{A}_4$ in Hybrid 4.

Pre-MPC: Up to the beginning of the preprocessing MPC, $\mathcal{A}_4$ acts exactly as $\mathcal{A}_3^{pre}$. Note that during this time the two hybrid experiments are identical. Let $z_{\mathsf{MPC}}$ denote the entire view of $\mathcal{A}_4$ up to this point.

MPC: At this point, the hybrid experiments diverge. At a high level, $\mathcal{A}_4$ will make a call to $F_{\mathsf{Pre}}$ in order to generate the outputs of the MPC with the correct distribution, and then will feed this information to the MPC simulator $\mathcal{S}_{\mathsf{MPC}}$, corresponding to adversary $\mathcal{A}_3^{\mathsf{MPC}}$, to generate the view of $\mathcal{A}_3$ *during* the MPC execution. Let $z_{post}$ denote the output of $\mathcal{S}_{\mathsf{MPC}}$.

Post-MPC: At the conclusion of the MPC for computing $F_{\mathsf{Pre}}$, the two hybrid experiments return to being equivalent. $\mathcal{A}_4$ thus simply acts exactly as $\mathcal{A}_3^{post}$ executed with auxiliary input $z_{post}$.

**Claim 4.6.3.** *For any auxiliary input $z$ and set of inputs $x$,*

$$\mathsf{HYB}_3\left(\mathcal{A}_3, 1^k, z, \{x_i\}_{i=1}^n\right) \approx_c \mathsf{HYB}_4\left(\mathcal{A}_4, 1^k, z, \{x_i\}_{i=1}^n\right).$$

The proof of this claim is essentially identical to the proof of Claim 4.6.1, and is therefore omitted.

**Step 5: Hybrid 4 to Hybrid 5. (WLR-MPC security).** In this step, all the weakly leakage-resilient MPCs (WLR-MPCs) in $\Pi_{\mathsf{Comp}}$ and $\Pi_{\mathsf{Update}}$ are replaced by (leaky) ideal functionalities. Consider an adversary $\mathcal{A}_4$ in Hybrid 4. We can think of $\mathcal{A}_4$ after the preprocessing and input phases as being composed of a collection of sub-algorithms $(\mathcal{A}_4{}^{\mathsf{MPC}_1}, \mathcal{A}_4{}^{\mathsf{MPC}_2}, ...)$ corresponding to each instance of the WLR-MPC protocol. We define the output of each $\mathcal{A}^{\mathsf{MPC}_i}$ to be the view of $\mathcal{A}_4$ up to that point; each $\mathcal{A}^{\mathsf{MPC}_i}$ receives as auxiliary input the output $z^{\mathsf{MPC}_i}$ of the previous sub-adversary, $\mathcal{A}^{\mathsf{MPC}_{i-1}}$. By the security of the $\lambda$-WLR-MPC protocol, for each such $\mathcal{A}_4{}^{\mathsf{MPC}_i}$ there exists a simulator $\mathcal{S}_{\mathsf{WLR}}^{(i)}$ who, given the same auxiliary input $z^{\mathsf{MPC}_i}$ and access to the corresponding WLR-MPC ideal functionality (i.e., the function output together with up to $\lambda$ bits of leakage on the collective inputs), simulates the view of $\mathcal{A}_4{}^{\mathsf{MPC}_i}$ during the $i$th MPC execution. Loosely speaking, $\mathcal{A}_5$ will simulate $\mathcal{A}_4$ by piecing all these simulations together.

Consider the following adversary $\mathcal{A}_5$ in Hybrid 5. In the preprocessing and input phase, $\mathcal{A}_5$ acts precisely as $\mathcal{A}_4$. Once the online phase begins, $\mathcal{A}_5$ simulates $\mathcal{A}_4$ as follows. Each time $\mathcal{A}_4$ requests to compute a function $f$, $\mathcal{A}_5$ does the same. Each time $\mathcal{A}_4$ enters in an execution of the weakly leakage-resilient MPC protocol $\mathsf{MPC}_i$, $\mathcal{A}_5$ instead runs the simulator $\mathcal{S}_{\mathsf{WLR}}^{(i)}$ with auxiliary input $z^{\mathsf{MPC}_i} = \mathsf{view}(\mathcal{A}_4)$, which is the entire simulated view of $\mathcal{A}_4$ up to that point. Note that $\mathcal{S}_{\mathsf{WLR}}^{(i)}$ accesses the corresponding ideal functionality, which supplies the function output and answers up to $\lambda$ leakage queries.

**Claim 4.6.4.** *For any auxiliary input $z$ and set of inputs $\vec{x}$,*

$$\mathsf{HYB}_4\left(\mathcal{A}_4, 1^k, z, \{x_i\}_{i=1}^n\right) \approx_c \mathsf{HYB}_5\left(\mathcal{A}_5, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* First, recall that security of the WLR-MPC protocol for randomized functionalities running among $\ell$ parties holds with probability $1 - \mathsf{negl}(k) - \ell e^{-\sqrt{\ell}/2}$, as long as a constant fraction of the $\ell$ parties is honest. In our case, each WLR-MPC protocols is run among an elected committee $\mathcal{E}_j$

(corresponding to a module $\mathsf{Sub}_j$ of the LDS-compiled circuit), which is guaranteed by Hybrid 1 to have at least $\ell \geq k' = k^\delta$ parties, and a constant fraction of honest parties. Thus, security of each individual WLR-MPC execution is guaranteed with probability $1 - \mathsf{negl}(k)$.

Now, suppose for contradiction that there is some choice of $z$ and $\vec{x}$ for which there exists a PPT algorithm $\mathcal{D}$ who distinguishes between these distributions with noticeable advantage:

$$\left|\Pr\left[\mathcal{D}\left(\mathsf{HYB}_4\left(\mathcal{A}_4, 1^k, z, \{x_i\}_{i=1}^n\right)\right) = 1\right] - \Pr\left[\mathcal{D}\left(\mathsf{HYB}_5\left(\mathcal{A}_5, 1^k, z, \{x_i\}_{i=1}^n\right)\right) = 1\right]\right| \geq \epsilon.$$

We will use $\mathcal{D}$ to break the weak leakage resilience of one of the underlying WLR-MPC protocols. This is done via a standard hybrid argument.

Since $\mathcal{A}_4$ is PPT, there exists some polynomial bound $Q = Q(k)$ such that the number of WLR-MPC protocols in any execution of $\mathcal{A}_4$ is bounded by $Q$. Consider a sequence of $Q$ intermediate experiments $\mathbb{E}^{(i)}$ between Hybrid 4 and Hybrid 5, in which the first $i$ MPC executions in the protocol are run as normal and the remaining $(Q - i)$ are replaced by the ideal functionality. We can similarly define adversaries $\mathcal{A}^{(i)}$ in each intermediate experiment, which act as $\mathcal{A}_4$ up through the $i$th MPC execution, and then act as $\mathcal{A}_5$ for the remainder of the experiment. Since $\mathbb{E}^{(Q)}$ is Hybrid 4 and $\mathbb{E}^{(0)}$ is Hybrid 5, there must exist some $i$ for which

$$\left|\Pr\left[\mathcal{D}\left(\overset{(i-1)}{\mathbb{E}}(\mathcal{A}^{(i-1)})\right) = 1\right] - \Pr\left[\mathcal{D}\left(\overset{(i)}{\mathbb{E}}(\mathcal{A}^{(i)})\right) = 1\right]\right| \geq \frac{\epsilon}{Q}.$$

We now construct $\mathcal{A}', z', \vec{x}', \mathcal{D}'$ that together break the security of the $i$th WLR-MPC protocol, $\mathsf{MPC}_i$. Take $\mathcal{A}'$ to be the $i$th MPC sub-adversary, $\mathcal{A}_4{}^{\mathsf{MPC}_i}$. Let $z'$ denote the entire view of $\mathcal{A}_4$ during a real execution of $\Pi$ up to $\mathsf{MPC}_i$, executing on the above fixed set of inputs $\vec{x}$ and auxiliary input $z$. Note that $z'$ can be efficiently simulated given $\vec{x}$ and $z$. Let $\vec{x}'$ be the set of inputs of all parties to $\mathsf{MPC}_i$, as dictated by this partial execution.

The distinguisher $\mathcal{D}'$ is given a challenge output $y'$ that is either the output of a real WLR-MPC execution of $\mathsf{MPC}_i$ with adversary $\mathcal{A}'(z')$ on inputs $\vec{x}'$, or is simulated. At a high level, $\mathcal{D}'$ will now simulate the remainder of the Hybrid 5 execution as dictated by $\mathcal{A}_5$, and feed the corresponding outputs to the distinguisher $\mathcal{D}$. We first argue that $\mathcal{D}'$ is able to simulate the remainder of this execution.

Recall that $\mathsf{MPC}_i$ is only run among parties within a single committee (say, $\mathcal{E}_j$), and the internal state of honest parties $\notin \mathcal{E}_j$ does not change during this period. Now, $\mathcal{D}'$ knows the following values: (a) the view of all parties *up to the start* of $\mathsf{MPC}_i$ (including their inputs to $\mathsf{MPC}_i$), since $\mathcal{D}'$ simulated this entire portion of the protocol on the given inputs $x$ and $z$; (b) the view of *honest parties* $\notin \mathcal{E}_j$ at the conclusion of $\mathsf{MPC}_i$, since their internal state did not change as a course of $\mathsf{MPC}_i$; (c) the view of the *adversary* at the conclusion of $\mathsf{MPC}_i$, since this is contained in the output $y'$ (by the definition of $\mathcal{A}_4^{\mathsf{MPC}_i}$); and (d) the *output* (but not the whole view) of *honest parties* $\in \mathcal{E}_j$ in $\mathsf{MPC}_i$, since this is part of $y'$. Thus, $\mathcal{D}'$ knows everything except the intermediate values of honest parties $\in \mathcal{E}_j$ *during* $\mathsf{MPC}_i$. But, such intermediate values are not needed to simulate the remainder of the execution of Hybrid 5, since honest parties *erase* all intermediate values at the conclusion of each WLR-MPC execution, as dictated by $\Pi$. Thus, $\mathcal{D}'$ can indeed complete this simulation.

At the conclusion of the experiment simulation, $\mathcal{D}'$ sends the resulting set of all parties' outputs out to the distinguisher $\mathcal{D}$. If the challenge output $y'$ was the output of a real execution of $\mathsf{MPC}_i$, then the distribution out is identical to the output $\mathbb{E}^{(i)}(\mathcal{A}^{(i)})$. However, if the challenge output $y'$ was simulated, then the distribution out is identical to $\mathbb{E}^{(i-1)}(\mathcal{A}^{(i-1)})$. Thus, $\mathcal{D}'$ distinguishes the two cases with nonnegligible advantage, $\epsilon/Q$.

□

**Step 6: Hybrid 5 to Hybrid 6. (Equivocation of EqCom).** In this step, the ideal functionality $F_{\mathsf{Pre}}$ is modified so that the commitment to the secret share $\mathsf{Sub}_{j,1}$ for the first party in each committee is generated using the simulation algorithm of the equivocal commitment scheme. Let $\mathcal{A}_5$ be an adversary in Hybrid 5. Define $\mathcal{A}_6$ to be the adversary in Hybrid 6 who acts exactly as $\mathcal{A}_5$.

**Claim 4.6.5.** *For any auxiliary input $z$ and set of inputs $\vec{x}$,*

$$\mathsf{HYB}_5\left(\mathcal{A}_5, 1^k, z, \{x_i\}_{i=1}^n\right) \approx_c \mathsf{HYB}_6\left(\mathcal{A}_6, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Suppose there exists a set of inputs $\vec{x}$, auxiliary input $z$, and ppt distinguisher $\mathcal{D}$ for which the claim does not hold; namely, there exists a non-negligible $\epsilon = \epsilon(k)$ such that

$$\left|\Pr\left[\mathcal{D}\left(\mathsf{HYB}_5\left(\mathcal{A}_5, 1^k, z, \{x_i\}_{i=1}^n\right)\right) = 1\right] - \Pr\left[\mathcal{D}\left(\mathsf{HYB}_6\left(\mathcal{A}_6, 1^k, z, \{x_i\}_{i=1}^n\right)\right) = 1\right]\right| \geq \epsilon.$$

Then there exists a ppt distinguisher $\mathcal{D}'$ that has the values $1^k, z, \{x_i\}_{i=1}^n$ hardwired into it, such that

$$\begin{aligned} &\left|\Pr\left[\mathcal{D}'\left(sk, pk, \{\mathsf{crs}_{\mathsf{eq}}^j\}_{j\in[m]}, \{\mathsf{Sub}_{j,1}, c_{j,1}, d_{j,1}\}_{j\in[m]}\right) = 1\right] - \right. \\ &\left. \Pr\left[\mathcal{D}'\left(sk, pk, \{\mathsf{crs}_{\mathsf{eq}}^j\}_{j\in[m]}, \{\mathsf{Sub}_{j,1}, \tilde{c}_{j,1}, \tilde{d}_{j,1}\}_{j\in[m]}\right) = 1\right]\right| \geq \epsilon. \end{aligned}$$

The distinguisher $\mathcal{D}'$, given his input and hardwired values, can simulate the entire hybrid as follows. He thinks of $\mathsf{crs}_{\mathsf{eq}}^j$ as the CRS of the equivocal commitment for the first party in committee $j$, and simulates all the other crs's. He simulates all parties running Elect to obtain committees $\mathcal{E}_1, \ldots, \mathcal{E}_m$. Then he simulates the ideal functionality $F_{\mathsf{Pre}}$ or $F'_{\mathsf{Pre}}$, depending on whether the commitments/decommitment pairs, that he is given as inputs, are simulated or not. This is done as follows.

He generates $\mathsf{Sub}_1, \ldots, \mathsf{Sub}_m \leftarrow \mathcal{C}(\mathsf{Dec}_{\mathsf{sk}})$; for each committee $\mathcal{E}_j$ he secret shares $\mathsf{Sub}_j$ among the committee members by choosing at random $\mathsf{Sub}_{j,2}, \ldots, \mathsf{Sub}_{j,|\mathcal{E}_j|}$ such that $\mathsf{Sub}_j = \mathsf{Sub}_{j,1} \oplus \ldots \oplus \mathsf{Sub}_{j,|\mathcal{E}_j|}$. Then he computes a commitment/decommitment pair $(c_{j,i}, d_{j,i}) \leftarrow \mathsf{com}(\mathsf{Sub}_{j,i})$ to all the shares except $\mathsf{Sub}_{j,1}$ (for which he uses his input). He simulates the ideal functionality giving $\mathsf{pk}, \{c_{j,i}\}_{j\in[m], i\in[|\mathcal{E}_j|]}$ to all parties, and giving $\mathsf{Sub}_{j,i}, d_{j,i}$ to the $i$'th party of $\mathcal{E}_j$. He simulates the rest of the hybrid by simply simulating all parties, and simulating the ideal functionalities $\{F_j\}$ and $\{G_j\}$.

Note that if the commitment/decommitment pairs given to $\mathcal{D}'$ are generated using the underlying commitment algorithm com, then $\mathcal{D}'$ simulates Hybrid 5 perfectly. On the other hand, if the commitment/decommitment pairs were generated using the corresponding simulation algorithms $\mathcal{S}_{\mathsf{eq}}$ then $\mathcal{D}'$ perfectly simulates Hybrid 6.

Finally, $\mathcal{D}'$ runs $\mathcal{D}$ on the simulated hybrid, and thus distinguishes between the two possible inputs with non-negligible probability $\geq \epsilon$. It remains to show that the existence of such a distinguisher $\mathcal{D}'$ contradicts the simulation property of the underlying equivocal commitment scheme. This follows immediately from Claim 4.2.5, together with a standard hybrid argument. $\square$

**Step 7: Hybrid 6 to Hybrid 7. (Equivocation of EqCom).** In this step, the ideal functionalities $\{G_j\}$ are replaced with $\{G'_j\}$. The proof of this step is very similar to the proof of Step 6. Let $\mathcal{A}_6$ be an adversary in Hybrid 6. Define $\mathcal{A}_7$ to be the adversary in Hybrid 7 who acts exactly as $\mathcal{A}_6$.

**Claim 4.6.6.** *For any auxiliary input $z$ and set of inputs $\vec{x}$,*

$$\mathsf{HYB}_6\left(\mathcal{A}_6, 1^k, z, \{x_i\}_{i=1}^n\right) \approx_c \mathsf{HYB}_7\left(\mathcal{A}_7, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Suppose there exists a set of inputs $\vec{x}$, auxiliary input $z$, and ppt distinguisher $\mathcal{D}$ for which the claim does not hold; namely, there exists a non-negligible $\epsilon = \epsilon(k)$ such that

$$\left|\Pr\left[\mathcal{D}\left(\mathsf{HYB}_6\left(\mathcal{A}_6, 1^k, z, \{x_i\}_{i=1}^n\right)\right) = 1\right] - \Pr\left[\mathcal{D}\left(\mathsf{HYB}_7\left(\mathcal{A}_7, 1^k, z, \{x_i\}_{i=1}^n\right)\right) = 1\right]\right| \geq \epsilon.$$

Let $Q = \mathsf{poly}(k)$ be an upper bound on *total* the number of bit commitments that each party receives from the ideal functionalities $\{G_j\}$. Similarly to Step 6, we prove that there exists a ppt distinguisher $\mathcal{D}'$ that has the values $1^k, z, \{x_i\}_{i=1}^n$ hardwired into it, such that

$$\begin{aligned}&\left|\Pr\left[\mathcal{D}'\left(\{\mathsf{crs}_{\mathsf{eq}}^j\}_{j\in[m]}, \{c_{j,b,i}, d_{j,b,i}\}_{j\in[m],b\in\{0,1\},i\in[Q]}\right) = 1\right] - \right.\\ &\left.\Pr\left[\mathcal{D}'\left(\{\mathsf{crs}_{\mathsf{eq}}^j\}_{j\in[m]}, \{\tilde{c}_{j,b,i}, \tilde{d}_{j,b,i}\}_{j\in[m],b\in\{0,1\},i\in[Q]}\right) = 1\right]\right| \geq \epsilon,\end{aligned}$$

where each pair $(c_{j,b,i}, d_{j,b,i}) \leftarrow \mathsf{com}(\mathsf{crs}_{\mathsf{eq}}^j, b)$ is a (fresh) commitment/decommitment pair to the bit $b$ with respect to $\mathsf{crs}_{\mathsf{eq}}^j$, and where each pair $(\tilde{c}_{j,b,i}, \tilde{d}_{j,b,i})$ is a (fresh) simulated commitment/decommitment pair to the bit $b$ computed using $(\mathsf{crs}_{\mathsf{eq}}^j, \mathsf{trap}^j)$.

Exactly as was done in Step 6, the distinguisher $\mathcal{D}'$ uses his inputs to simulate the entire hybrid. Note that $\mathcal{D}'$ has the inputs of all parties, and thus can simulate the entire hybrid, including the ideal functionalities, on his own. Indeed that is what he does, except that he uses each $\mathsf{crs}_{\mathsf{eq}}^j$ (which is part of his input) as the CRS of the equivocal commitment for the first party in committee $j$. In addition, when simulating the functionalities $\{G_j\}$ (or $\{G'_j\}$), whenever the functionality needs to generate a commitment/decommitment pair of a bit a bit $b$, the distinguisher $\mathcal{D}'$ uses one of his input commitment/decommitment pair instead of generating it on his own. Other than that, he simulates the rest of the hybrid on his own.

Note that if the commitment/decommitment pairs given to $\mathcal{D}'$ are generated using the underlying commitment algorithm com, then $\mathcal{D}'$ simulates Hybrid 6 perfectly. On the other hand, if the commitment/decommitment pairs were generated using the corresponding simulation algorithms $\mathcal{S}_{\mathsf{eq}}$ then $\mathcal{D}'$ perfectly simulates Hybrid 7.

$\mathcal{D}'$ runs $\mathcal{D}$ on the simulated hybrid, and thus distinguishes between the two possible inputs with non-negligible probability $\geq \epsilon$. It remains to show that the existence of such a distinguisher $\mathcal{D}'$ contradicts the simulation property of the underlying equivocal commitment scheme. This follows immediately from Claim 4.2.5, together with a standard hybrid argument. $\square$

**Step 8: Hybrid 7 to Hybrid 8. (Binding of Com).** In this step, the ideal functionalities $\mathsf{crsSim}, F'_{\mathsf{Pre}}, \{F_j\}, \{G'_j\}$ are replaced by $\mathsf{crsSim}', F_{\mathsf{pk}}$ and interaction with the submodules $\{\mathsf{Sub}_j\}$. Let $\mathcal{A}_7$ be an adversary in Hybrid 7; we construct a corresponding adversary $\mathcal{A}_8$ in Hybrid 8. Loosely speaking, $\mathcal{A}_8$ simulates $\mathcal{A}_7$ as follows. At each point when an ideal functionality in Hybrid 7 would generate and distribute secret shares of some module $\mathsf{Sub}_j$, $\mathcal{A}_8$ generates *random* shares for *all but the first* party in each committee, and generates a *simulated* commitment for the first party in each committee. This allows $\mathcal{A}_8$ to simulate leakage on *every* party's secret state $(\mathsf{Sub}_{j,i}, d_{j,i})$ directly from leakage on the corresponding module $\mathsf{Sub}_j$. Adversary $\mathcal{A}_8$ simulates the remaining outputs of the ideal functionalities $\{F_j\}$ and $\{G'_j\}$ essentially by forwarding the outputs from the corresponding modules. More explicitly, $\mathcal{A}_8$ does the following.

Preprocessing phase: First, $\mathcal{A}_8$ simulates the output of $F'_{\mathsf{Pre}}$ by outputting $\mathsf{pk}$ received from $F_{\mathsf{pk}}$, and running the following secret sharing procedure:

Secret sharing procedure: For *all but the first* party in each committee, sample a secret share $\mathsf{Sub}_{j,i}$ at random. Honestly compute a commitment $(c_{j,i}, d_{j,i}) \leftarrow \mathsf{Com}(\mathsf{crs}^{j,i}_{\mathsf{eq}}, \mathsf{Sub}_{j,i})$.[16] For the *first* party in each committee $\mathcal{E}_j$, generate a *simulated* commitment $(c_{j,1}, d^0_{j,1}, d^1_{j,1}) \leftarrow \mathsf{Sim}_{\mathsf{eq}}(\mathsf{crs}^{j,i}_{\mathsf{eq}}, \mathsf{trap}^{j,i})$, using the corresponding trapdoor received from the $\mathsf{crsSim}'$ functionality. Send all generated commitments $\{c_{j,i}\}$ (including the simulated ones) to all parties.[17] For each malicious party $i$ of committee $j$, send the corresponding secret share $\mathsf{Sub}_{j,i}$ and decommitment information $d_{j,i}$ to $\mathcal{A}_7$.

Simulate $\mathsf{Elect}$ as $\mathcal{A}_7$.

Input phase: Act exactly as $\mathcal{A}_7$ to obtain $\hat{x} = (\hat{x}_1, ..., \hat{x}_n)$.

Online phase:

Computation procedure: Begin by acting as $\mathcal{A}_7$. Each time $\mathcal{A}_7$ requests the evaluation of a function $f$, $\mathcal{A}_8$ makes the same request in Hybrid 8 and proceeds as follows. Set $\mathsf{correct}_1 = \mathsf{Eval}_{\mathsf{pk}}(\hat{x}, f)$, and run the following for $j = 1, ..., m$:

- For the current value of $j$, when $\mathcal{A}_7$ queries the ideal functionality $F_j$ by submitting an input $(\mathsf{input}_j, \mathsf{Sub}_{j,i}, d_{j,i})$ for each malicious party in $\mathcal{E}_j$, $\mathcal{A}_8$ does the following. If any $\mathsf{input}_j \neq \mathsf{correct}_j$, or if any $\mathsf{Sub}_{j,i} \neq \mathsf{Rec}(\mathsf{crs}^{j,i}_{\mathsf{eq}}, c_{j,i}, d_{j,i})$, then abort. Otherwise, set $\mathsf{correct}_{j+1} \leftarrow \mathsf{Sub}_j(\mathsf{correct}_j)$, and simulate the output of the ideal functionality $F_j$ as $\mathsf{correct}_{j+1}$. Send $\mathsf{correct}_{j+1}$ to $\mathcal{A}_7$, and increment $j \leftarrow j + 1$.

Update procedure: Similar to the computation procedure, each time $\mathcal{A}_7$ requests that honest parties update their secret state, begin with $\mathsf{correct}_1 = \emptyset$. Each time $\mathcal{A}_7$ queries the appropriate ideal functionality $G_\ell$ by submitting input $(\mathsf{msg}_j, \mathsf{Sub}_{j,i}, d_{j,i})$ for each malicious party in the corresponding committee $\mathcal{E}_j$, do the following. If any $\mathsf{msg}_j \neq \mathsf{correct}_j$, or if any

[16]For simplicity of notation, in this section we refer to a party $P_\ell$ who is the $i$th party of committee $j$ as party $(j, i)$, and denote the party's corresponding crs values as $\mathsf{crs}_{j,i}$.

[17]Technically, to simulate the ideal functionality $F'_{\mathsf{Pre}}$ *with abort*, $\mathcal{A}_8$ first sends these values only to $\mathcal{A}_7$. If $\mathcal{A}_7$ allows, the values are also sent to honest parties; otherwise, the honest parties receive abort.

$\mathsf{Sub}_{j,i} \neq \mathsf{Rec}(\mathsf{crs}^{j,i}_{\mathsf{eq}}, c_{j,i}, d_{j,i})$, then abort. Otherwise, set $\mathsf{correct}_{j+1} \leftarrow \mathsf{Sub}_j(\mathsf{correct}_j)$, and simulate the output of $G_\ell$ as $\mathsf{correct}_{j+1}$. Send $\mathsf{correct}_{j+1}$ to $\mathcal{A}_7$, run the secret sharing procedure from above, and increment $j \leftarrow j+1$.

Leakage queries: At any point when $\mathcal{A}_7$ makes a leakage query $\mathsf{Leak}((j,i),L)$, if fewer than $\lambda$ such queries have been made during the current leakage time period, do the following.

- If $i \neq 1$, then simply send the evaluation $L(\mathsf{Sub}_{j,i}, d_{j,i})$ to $\mathcal{A}_7$.
- If $i = 1$, then $\mathcal{A}_8$ queries the following modified leakage function $L'$ to module $\mathsf{Sub}_j$

$$L'(\mathsf{Sub}_j) := \left( \begin{array}{c} L\left(\mathsf{Sub}_{j,1}, d_{j,1}^{\mathsf{Sub}_{j,1}}\right), \text{ where} \\ \mathsf{Sub}_{j,1} := \mathsf{Sub}_j \oplus \bigoplus_{i=2}^{|\mathcal{E}_j|} \mathsf{Sub}_{j,i} \end{array} \right),$$

and forwards the response to $\mathcal{A}_7$.

**Claim 4.6.7.** *For any auxiliary input $z$ and set of inputs $\vec{x}$,*

$$\mathsf{HYB}_7\left(\mathcal{A}_7, 1^k, z, \{x_i\}_{i=1}^n\right) \approx_s \mathsf{HYB}_8\left(\mathcal{A}_8, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Note that $\mathcal{A}_8$ simulates the preprocessing and input phase perfectly. We now consider the online phase. Namely, we show the $\mathcal{A}_8$ correctly simulates the output of the $F_j$'s, the output of the $G'_j$'s, and the responses to leakage queries.

$F_j$: Recall that, on input $(\mathsf{input}_j, \mathsf{Sub}_{j,i}, d_{j,i})$ from all parties in committee $\mathcal{E}_j$, the ideal functionality $F_j$ computes the following function:

$$F_j : \begin{cases} (\bigoplus_i \mathsf{Sub}_{j,i})\,(\mathsf{input}_j) & \text{If } \mathsf{Sub}_{j,i} = \mathsf{Rec}(\mathsf{crs}^{j,i}_{\mathsf{eq}}, c_{j,i}, d_{j,i})\ \forall i \in [|\mathcal{E}_j|] \\ & \quad \text{and all the } \mathsf{input}_j \text{ agree} \\ \mathsf{abort} & \text{Else} \end{cases}.$$

In the simulation, $\mathcal{A}_8$ only receives inputs to $F_j$ from *malicious* parties. He decides whether the simulated output of $F_j$ should be abort by checking (a) whether $\mathsf{Sub}_{j,i} = \mathsf{Rec}(\mathsf{crs}^{j,i}_{\mathsf{eq}}, c_{j,i}, d_{j,i})$ for *malicious* parties (instead of all parties), and (b) whether $\mathsf{input}_j = \mathsf{correct}_j$ for these parties (instead of checking that $\mathsf{input}_j$ is consistent among *all* parties). Note that in Hybrid 7, the condition $\mathsf{Sub}_{j,i} = \mathsf{Rec}(\mathsf{crs}, c_{j,i}, d_{j,i})$ will always hold for honest parties in $\mathcal{E}_j$, since they always give their (correct) received secret shares and decommitment values. Further, the value of $\mathsf{input}_j$ submitted by honest parties is exactly the value of $\mathsf{correct}_j$ maintained by $\mathcal{A}_8$. Thus, $\mathcal{A}_8$ simulates output abort in exactly the same cases that $F_j$ aborts in Hybrid 7.

We now argue that, if $F_j$ does *not* abort, then with overwhelming probability its output will be $\mathsf{Sub}_j(\mathsf{correct}_j)$. For each malicious party $(j,i)$, let $(\mathsf{Sub}^{\mathsf{Sim}}_{j,i}, d^{\mathsf{Sim}}_{j,i})$ denote the values generated by $\mathcal{A}_8$ for party $(j,i)$, and $(\mathsf{Sub}_{j,i}, d_{j,i})$ denote the corresponding values submitted by $\mathcal{A}_7$ to the ideal functionality $F_j$. By the binding property of the commitment scheme, for each malicious party $(j,i)$,

$$\Pr\left[\mathsf{Sub}_{j,i} = \mathsf{Sub}^{\mathsf{Sim}}_{j,i} \mid \mathsf{Sub}_{j,i} = \mathsf{Rec}(\mathsf{crs}^{j,i}_{\mathsf{eq}}, c_{j,i}, d_{j,i})\right] = 1 - \mathsf{negl}(k).$$

That is, $\mathcal{A}_7$ cannot open the commitment $c_{j,i}$ to a new value. Thus, if $F_j$ does not abort, then with overwhelming probability *all* parties in committee $\mathcal{E}_j$ sent their correct secret share, in which case $\bigoplus_i \mathsf{Sub}_{j,i} = \mathsf{Sub}_j$ and the output of $F_j$ is precisely $\mathsf{Sub}_j(\mathsf{correct}_j)$.

$G'_j$: Combining the arguments above (i.e., for the secret sharing procedure and the simulation of the $F_j$), it follows that, with overwhelming probability, $\mathcal{A}_8$ correctly simulates the output of each ideal functionality $G'_j$.

*Leakage*: It remains to show that the responses to leakage queries are simulated with the correct distribution. First, note that $\mathcal{A}_7$ makes no more than $\lambda$ leakage queries from the beginning of any Update procedure to the end of the next, and thus $\mathcal{A}_8$ makes $\leq \lambda$ leakage queries to any one module $\mathsf{Sub}_j$ during this time period, as required in Hybrid 8. Now, in Hybrid 7, the secret state of each honest party $(j, i)$ consists only of $\mathsf{Sub}_{j,i}$ and $d_{j,i}$. (In particular, since the WLR-MPC executions are already removed, parties no longer generate additional secret randomness as part of the protocol). For all but the first party in each committee, the simulated leakage clearly has the correct distribution, since $\mathcal{A}_8$ knows $\mathsf{Sub}_{j,i}, d_{j,i}$ in their entirety. For the first party in each committee, the value of $\mathsf{Sub}_{j,1}$ in Hybrid 7 is distributed precisely as $\mathsf{Sub}_{j,1} = \mathsf{Sub}_j \oplus (\bigoplus_{i \geq 2} \mathsf{Sub}_{j,i})$, and the value of $d_{j,1}$ generated by $F'_{\mathsf{Pre}}$ or $G'_j$ is sampled via the equivocal commitment simulator, and selecting the correct decommitment value. Thus, the simulated leakage in Hybrid 8 has exactly the same distribution as the real leakage in Hybrid 7.

□

**Step 9: Hybrid 8 to Hybrid 9. (LDS security).** Consider an adversary $\mathcal{A}_8$ in Hybrid 8. In this step, we simulate all interactions of $\mathcal{A}_8$ with the modules $\mathsf{Sub}_1, ..., \mathsf{Sub}_m$ (including leakage queries) given access only to a decryption oracle $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$. This is possible by the LDS security of the circuit compiler $(\mathcal{C}, \mathsf{Update})$.

We begin by considering an intermediate experiment, $\mathbb{E}'_8$, which is identical to Hybrid 8, except that the adversary communicates *directly* with the modules $\mathsf{Sub}_j$ instead of via committees. That is, at any point during the online phase, the adversary may query any module $\mathsf{Sub}_j$ on any value $\mathsf{input}_j$, and receive back the (randomized) output $\mathsf{Sub}_j(\mathsf{input}_j)$. Leakage queries on each $\mathsf{Sub}_j$ are exactly as in Hybrid 8. Note that the online portion of $\mathbb{E}'_8$ corresponds directly to LDS access to the modules, as described in Section 4.2.5.

Consider the following adversary $\mathcal{A}'_8$ in $\mathbb{E}'_8$. In the preprocessing and input phases, $\mathcal{A}'_8$ acts precisely as $\mathcal{A}_8$. In the online phase, $\mathcal{A}'_8$ simulates the execution of $\mathcal{A}_8$, but makes only those queries to modules that are "correct" (i.e., those consistent with the messages of honest parties in the corresponding committee), and ignores the rest. More formally, $\mathcal{A}'_8$ does the following.

- Each time $\mathcal{A}_8$ requests a function $f$ to be evaluated, $\mathcal{A}'_8$ requests the same function in $\mathbb{E}'_8$. He then homomorphically evaluates $\hat{y} = \mathsf{Eval}_{\mathsf{pk}}(\hat{x}, f)$, and sets $\mathsf{correct} = (\mathsf{Sub}_1, \hat{y})$.
- Each time $\mathcal{A}_8$ requests that the honest parties update their secret state, $\mathcal{A}'_8$ sets $\mathsf{correct} = (\mathsf{Sub}_1, \emptyset)$.[18]
- Each time $\mathcal{A}_8$ queries a module $\mathsf{Sub}_j$ by sending a value $\mathsf{input}_j$ from each malicious party in committee $\mathcal{E}_j$, $\mathcal{A}'_8$ does the following. Suppose the current value of correct is $(\mathsf{Sub}_{\mathsf{correct}}, \mathsf{input}_{\mathsf{correct}})$.

[18]We assume, without loss of generality, that all computation and update procedures are disjoint.

If $\mathsf{Sub}_j \neq \mathsf{Sub}_{\mathsf{correct}}$, or if $\mathsf{input}_j \neq \mathsf{input}_{\mathsf{correct}}$, then do nothing. Otherwise, send query $\mathsf{input}_j$ to $\mathsf{Sub}_j$, receive $\mathsf{input}_{j+1} \leftarrow \mathsf{Sub}_j(\mathsf{input}_j)$, and set correct $= (\mathsf{Sub}_{j+1}, \mathsf{input}_{j+1})$.

- Each time $\mathcal{A}_8$ makes a leakage query $\mathsf{Leak}(j, L)$, $\mathcal{A}'_8$ makes the same query in $\mathbb{E}'_8$.

**Claim 4.6.8.** *For any auxiliary input $z$ and set of inputs $\vec{x}$,*

$$\mathsf{HYB}_8\left(\mathcal{A}_8, 1^k, z, \{x_i\}_{i=1}^n\right) \approx_s \mathop{\mathbb{E}}_8^{\prime}\left(\mathcal{A}'_8, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Recall that in Hybrid 8, each ideal functionality $\mathsf{Sub}_j$ only responds if *all* parties in the corresponding committee $\mathcal{E}_j$ submit the same value $\mathsf{input}_j$. Since each committee has at least one honest party, the only module queries that will be answered are precisely the "correct" ones: i.e., those consistent with messages of honest parties in $\mathcal{E}_j$. Thus, any query to $\mathsf{Sub}_j$ made by $\mathcal{A}_8$ that is ignored by $\mathcal{A}'_8$ will, in fact, be ignored by $\mathsf{Sub}_j$ in Hybrid 8 as well. □

We now construct an adversary $\mathcal{A}_9$ in Hybrid 9 that simulates the adversary $\mathcal{A}'_8$ in $\mathbb{E}'_8$, such that the outputs of all parties in the two experiments are computationally indistinguishable. We can think of $\mathcal{A}'_8$ as being composed of two sub-adversaries $(\mathcal{A}'^{\mathsf{Pre}}_8, \mathcal{A}'^{\mathsf{Online}}_8)$, corresponding to the actions of $\mathcal{A}'_8$ in the preprocessing/input phase, and the online phase. The second sub-adversary, $\mathcal{A}'^{\mathsf{Online}}_8$, receives as auxiliary input the entire view of $\mathcal{A}'_8$ up to that point, and acts as an LDS adversary to the modules. By the $\lambda$-LDS security of the circuit compiler, there exists simulator $\mathcal{S}_{\mathsf{LDS}}$ corresponding to $\mathcal{A}'^{\mathsf{Online}}_8$.

We define the adversary $\mathcal{A}_9$ in Hybrid 9 as follows. In the preprocessing and input phases, $\mathcal{A}_9$ acts precisely as $\mathcal{A}'^{\mathsf{Pre}}_8$. Denote the view of $\mathcal{A}_9$ up to this point as $z'$. In the online phase, $\mathcal{A}_9$ runs the LDS simulator $\mathcal{S}_{\mathsf{LDS}}$ corresponding to $\mathcal{A}'^{\mathsf{Online}}_8$, with the auxiliary input $z'$. Recall that $\mathcal{S}_{\mathsf{LDS}}(z')$ requires oracle access to the decryption circuit $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$ (the circuit that was compiled). $\mathcal{A}_9$ simulates access to this oracle using access to his own ideal functionality, $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$. Note, however, that $\mathcal{A}_9$ does not have direct access to $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$, as the ideal functionality only responds to queries that are consistently requested by *all* parties in $\mathcal{E}_1$.

Each time $\mathcal{A}'^{\mathsf{Online}}_8$ requests a function $f$ to be evaluated, $\mathcal{A}_9$ requests the same function in Hybrid 9, and performs the following steps: (a) homomophically evaluate $\hat{y} = \mathsf{Eval}_{\mathsf{pk}}(\hat{x}, f)$, (b) submit $\hat{y}$ to the ideal decryption functionality $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$, from each malicious party in $\mathcal{E}_1$, and (c) send the received value $y$ to $\mathcal{S}_{\mathsf{LDS}}$. At the conclusion of the experiment, $\mathcal{A}_9$ outputs the corresponding output of $\mathcal{S}_{\mathsf{LDS}}$.

**Claim 4.6.9.** *For any auxiliary input $z$ and set of inputs $\vec{x}$,*

$$\mathop{\mathbb{E}}_8^{\prime}\left(\mathcal{A}'_8, 1^k, z, \{x_i\}_{i=1}^n\right) \approx_c \mathsf{HYB}_9\left(\mathcal{A}_9, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* The preprocessing and input phases are simulated perfectly. We now consider the online phase. Note that $\mathcal{A}'^{\mathsf{Online}}_8$ is precisely a $\lambda$-LDS adversary. (Recall, in particular, that in Hybrid 8, the adversary is limited to a maximum of $\lambda$ leakage queries during each time period.) By the $\lambda$-LDS security of the circuit compiler, it thus remains to show that $\mathcal{A}_9$ correctly simulates the decryption oracle for $\mathcal{S}_{\mathsf{LDS}}$, on each $\mathsf{input}_1$ that $\mathcal{A}'^{\mathsf{Online}}_8$ queries to $\mathsf{Sub}_1$. By construction, the only queries that $\mathcal{A}'^{\mathsf{Online}}_8$ makes to $\mathsf{Sub}_1$ are the homomorphically evaluated values $\hat{y}_f = \mathsf{Eval}_{\mathsf{pk}}(\hat{x}, f)$ for each function $f$ that $\mathcal{A}_8'^{\mathsf{Online}}$ requests to be evaluated. But, $\mathcal{A}_9$ sends exactly these decryptions $\mathsf{Dec}_{\mathsf{sk}}(\hat{y}_f)$ to $\mathcal{S}_{\mathsf{LDS}}$. Thus, the decryption oracle is perfectly simulated, and the claim holds.

□

**Step 11: Hybrid 9 to Hybrid 11. (Soundness/PoK of NIZK, certifiability of FHE)** In this step, $\mathsf{Elect}, F_{\mathsf{pk}}$, and $\mathsf{crsSim}$ are removed from the preprocessing phase, and the decryption oracle $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$ is replaced by the ideal-world functionality $\mathsf{Evaluate}$, which takes as input $x_i$ from each party $P_i$, and evaluates each of the adversary's queried functions $f$ on all inputs $(x_1, ..., x_n)$. In addition, the adversary is given auxiliary input $z'$ that depends on the honest parties' inputs (see description of Hybrid 11).

Let $\mathcal{A}_9$ be an adversary in Hybrid 9. We construct a corresponding adversary $\mathcal{A}_{11}$ in Hybrid 11 that simulates $\mathcal{A}_9$ as follows. At a high level, $\mathcal{A}_{11}$ will (a) simulate the preprocessing phase on his own, (b) simulate the honest party input phase by simply forwarding the information from his auxiliary input $z'$ to $\mathcal{A}_9$, (c) extract the inputs of malicious parties from their NIZK proofs of knowledge, and send them to $\mathsf{Evaluate}$, and (d) for each function $f$ queried by $\mathcal{A}_9$, make the same query $f$ to $\mathsf{Evaluate}$, and use this to simulate $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$.

More explicitly, $\mathcal{A}_{11}$ does the following.

Preprocessing phase: Take $\mathsf{pk}$ to be the corresponding value in the auxiliary input $z'$. Generate all $\mathsf{crs}$ values for the equivocal commitment scheme and WLR-MPC protocol as described in Hybrid 3. For the NIZK, generate $\mathsf{crs}$ values as follows: for each *malicious* party $P_\ell$, compute $(\mathsf{crs}^\ell_{\mathsf{nizk}}, \mathsf{trap}^\ell) \leftarrow \mathcal{S}^{\mathsf{crs}}_{\mathsf{nizk}}(1^k)$; for each *honest* party $P_i$, take $\mathsf{crs}^i_{\mathsf{nizk}}$ to be the corresponding value from the auxiliary input $z'$. Send $\mathsf{pk}$ and all $\mathsf{crs}$ values to $\mathcal{A}_9$. Honestly simulate the actions of honest parties in $\mathsf{Elect}$.

Honest party input phase: For each honest party $P_i$, forward the corresponding input pair $(\hat{x}_i, \pi_i)$ contained in $z'$ to $\mathcal{A}_9$.

Malicious party input phase: After $\mathcal{A}_9$ receives $(\hat{x}_i, \pi_i)$ for all honest parties $i \notin M$, it will respond with pairs $(\hat{x}_\ell, \pi_\ell)$ for the malicious parties $\ell \in M$. For each such $\ell$, $\mathcal{A}_{11}$ does the following. If the supplied proof does not verify (ie, if $1 \neq \mathsf{V}(\mathsf{crs}^\ell_{\mathsf{nizk}}, (\hat{x}_\ell, \mathsf{pk}), \pi_\ell)$), then $\mathsf{abort}$. Otherwise, extract the input $x_\ell$ of malicious party $P_\ell$ from the NIZK proof of knowledge by computing $(x_\ell, r_\ell) = E(\mathsf{crs}^\ell_{\mathsf{nizk}}, \mathsf{trap}^\ell, (\hat{x}_\ell, \mathsf{pk}), \pi_\ell)$. For each $\ell \in M$, send $x_\ell$ to the ideal-world functionality $\mathsf{Evaluate}$.

Online phase: $\mathcal{A}_{11}$ acts exactly as $\mathcal{A}_9$, while simulating oracle access to $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$ as follows. Each time $\mathcal{A}_9$ requests the computation of a function $f$, $\mathcal{A}_{11}$ makes the same request $f$ to $\mathsf{Evaluate}$ and receives $f(\vec{x})$. In the case that all malicious parties in $\mathcal{E}_1$ send the correct value $\hat{y} = \mathsf{Eval}_{\mathsf{pk}}(\hat{x}, f)$ to the decryption oracle, $\mathcal{A}_{11}$ simulates the response by sending $f(\vec{x})$.

**Claim 4.6.10.** *For any auxiliary input $z$ and set of inputs $\vec{x}$,*

$$\mathsf{HYB}_9\left(\mathcal{A}_9, 1^k, z, \{x_i\}_{i=1}^n\right) \approx_s \mathsf{HYB}_{11}\left(\mathcal{A}_{11}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Note that all values in the preprocessing and input phase in Hybrid 11 are generated with the exact same distribution as in Hybrid 9.

Recall that in Hybrid 9, the decryption oracle *only* responds to queries that are consistently requested by *all* parties in committee $\mathcal{E}_1$. Since $\mathcal{E}_1$ contains at least one honest party, such queries will be limited to the homomorphically evaluated ciphertexts $\mathsf{Eval}_{\mathsf{pk}}(\hat{x}, f)$ corresponding to the adversary's queried functions $f$. In Hybrid 11, the responses to these queries are simulated using $f(x)$, where $\{x_\ell\}_{\ell \in M}$ are the values extracted from the input pairs $(\hat{x}_\ell, \pi_\ell)$ of the corresponding malicious parties. Thus, it remains only to show that, for each queried function $f$,

$$\Pr[\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{pk}}((\hat{x}_1, ..., \hat{x}_\ell), f)) = f(x_1, ..., x_n)] = 1 - \mathsf{negl}(k).$$

By the soundness of the NIZK, it must hold for any choice of inputs $x$, and for every $\ell$,

$$\Pr\left[(\hat{x}_\ell, \mathsf{pk}) \in L \mid \mathsf{V}(\mathsf{crs}^{\ell}_{\mathsf{nizk}}, (\hat{x}_\ell, \mathsf{pk}), \pi_\ell) = 1\right] = 1 - \mathsf{negl}(k).$$

Boringly, this is the case, since otherwise this adversary $\mathcal{A}_9$ can directly be used in the security game for adaptive soundness, simply by embedding the challenge crs as $\mathsf{crs}^{\ell}_{\mathsf{nizk}}$ and simulating the rest of the experiment honestly using the corresponding inputs $x$.

Recall $(\hat{x}_\ell, \mathsf{pk}) \in L$ implies $\exists (x_\ell, r_\ell)$ for which $r_\ell \in R$ and $\hat{x}_\ell = \mathsf{Enc}_{\mathsf{pk}}(x_\ell; r_\ell)$. Further, for each honest party $i \notin M$, we have that $\hat{x}_i = \mathsf{Enc}_{\mathsf{pk}}(x_i; r_i)$ for some $r_i \in R$. Thus, by the certifiability of the FHE scheme with respect to $R$ (see Definition 5.2.4), it follows that

$$\Pr\left[\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{pk}}((\hat{x}_1, ..., \hat{x}_n), f)) = f(x_1, ..., x_n)\right] = 1 - \mathsf{negl}(k),$$

as desired.

□

**Step 19: Hybrid 11 to Hybrid 19. (Security of FHE, ZK of NIZK)** In the final step, the auxiliary input $z'$ is removed. Let $\mathcal{A}_{11}$ be an adversary in Hybrid 11; we construct a corresponding adversary $\mathcal{A}_{19}$ in the ideal world, Hybrid 19. At a high level, $\mathcal{A}_{19}$ simply simulates the auxiliary input $z'$, and then simulates the actions of $\mathcal{A}_{11}$ with this simulated $z'_{\mathsf{Sim}}$.

More explicitly, $\mathcal{A}_{19}$ does the following. First, generate a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$. Then, for each honest party $P_i$, perform the following steps. Generate $\mathsf{crs}^{i}_{\mathsf{nizk}} \leftarrow \mathsf{Gen}_{\mathsf{nizk}}(1^k)$; sample $r_i \leftarrow R$ and encrypt $\hat{0}_i = \mathsf{Enc}_{\mathsf{pk}}(0; r_i)$; and (honestly) generate a proof $\pi_i \leftarrow \mathsf{P}(\mathsf{crs}^{i}_{\mathsf{nizk}}, (\hat{0}_i, \mathsf{pk}), (0, r_i))$. Let $z'_{\mathsf{Sim}}$ denote the tuple of values $(\mathsf{pk}, \{\mathsf{crs}^{i}_{\mathsf{nizk}}, \hat{0}_i, \pi_i\}_{i \notin M})$.

**Claim 4.6.11.** *For any auxiliary input $z$ and set of inputs $\vec{x}$,*

$$\mathsf{HYB}_{11}\left(\mathcal{A}_{11}, 1^k, z, \{x_i\}_{i=1}^{n}\right) \approx_c \mathsf{HYB}_{19}\left(\mathcal{A}_{19}, 1^k, z, \{x_i\}_{i=1}^{n}\right).$$

*Proof.* It suffices to show that the simulated auxiliary input $z'_{\mathsf{Sim}}$ is computationally indistinguishable from $z'$. Recall that the auxiliary input $z'$ is composed of a public key pk and triples $(\mathsf{crs}^{i}_{\mathsf{nizk}}, \hat{x}_i, \pi_i)$ for each honest party $P_i$. By a standard hybrid argument, it is sufficient to prove that indistinguishability holds for a *single* such triple together with the $\mathsf{pk}$. That is, for any value of $x$, we need to show the following distributions are computationally indistinguishable:

$$\begin{pmatrix} \mathsf{pk}, \mathsf{crs}_{\mathsf{NIZK}}, \\ c = \mathsf{Enc}_{\mathsf{pk}}(x;r), \\ \pi \leftarrow \mathsf{P}(\mathsf{crs}_{\mathsf{NIZK}}, (c, \mathsf{pk}), (x, r)) \end{pmatrix} \approx_c \begin{pmatrix} \mathsf{pk}, \mathsf{crs}_{\mathsf{NIZK}}, \\ c' = \mathsf{Enc}_{\mathsf{pk}}(0;r), \\ \pi \leftarrow \mathsf{P}(\mathsf{crs}_{\mathsf{NIZK}}, (c', \mathsf{pk}), (0, r)) \end{pmatrix},$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$, $\mathsf{crs}_{\mathsf{NIZK}} \leftarrow \mathsf{crsGen}(1^k)$, and $r \leftarrow R \subseteq \{0,1\}^{\mathsf{poly}(k)}$.

To do so, we rely on the security of the FHE and the NIZK. Namely, let $(\mathcal{S}^{\mathsf{crs}}_{\mathsf{nizk}}, \mathcal{S}^{\mathsf{proof}}_{\mathsf{nizk}})$ be the corresponding simulators for the NIZK. Then we have

$$\begin{array}{llll} & (\ \mathsf{pk}, & \mathsf{crs}_{\mathsf{NIZK}} \leftarrow \mathsf{crsGen}_{\mathsf{NIZK}}(1^k), & c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x;r), & \pi \leftarrow \mathsf{P}(\mathsf{crs}_{\mathsf{NIZK}}, (c, \mathsf{pk}), (x, r))\ ) \\ \approx_c & (\ \mathsf{pk}, & \mathsf{crs}'_{\mathsf{NIZK}} : (\mathsf{crs}'_{\mathsf{NIZK}}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}_{\mathsf{nizk}}(1^k), & c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x;r), & \pi' \leftarrow \mathcal{S}^{\mathsf{proof}}_{\mathsf{nizk}}(\mathsf{crs}'_{\mathsf{NIZK}}, \mathsf{trap}, (c, \mathsf{pk}))\ ) \\ \approx_c & (\ \mathsf{pk}, & \mathsf{crs}'_{\mathsf{NIZK}} : (\mathsf{crs}'_{\mathsf{NIZK}}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}_{\mathsf{nizk}}(1^k), & c' \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0;r), & \pi' \leftarrow \mathcal{S}^{\mathsf{proof}}_{\mathsf{nizk}}(\mathsf{crs}'_{\mathsf{NIZK}}, \mathsf{trap}, (c', \mathsf{pk}))\ ) \\ \approx_c & (\ \mathsf{pk}, & \mathsf{crs}_{\mathsf{NIZK}} \leftarrow \mathsf{crsGen}_{\mathsf{NIZK}}(1^k), & c' \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0;r), & \pi'' \leftarrow \mathsf{P}(\mathsf{crs}_{\mathsf{NIZK}}, (c', \mathsf{pk}), (0, r))\ ), \end{array}$$

where the first and third indistinguishabilities follow by the zero knowledge property of the NIZK, and the second indistinguishability holds by the semantic security of the FHE scheme.

□

□

# Chapter 5

# Communication Locality in Secure Multi-party Computation

In this work, we devise multi-party computation protocols for general secure function evaluation with the property that *each party is only required to communicate with a small number of dynamically chosen parties.* More explicitly, starting with $n$ parties connected via a complete and synchronous network, our protocol requires each party to send messages to (and process messages from) at most $\mathsf{polylog}(n)$ other parties using $\mathsf{polylog}(n)$ rounds. It achieves secure computation of any polynomial-time computable randomized function $f$ under cryptographic assumptions, and tolerates up to $(\frac{1}{3} - \epsilon) \cdot n$ statically scheduled Byzantine faults.

We then focus on the particularly interesting setting in which the function to be computed is a *sublinear algorithm*: An evaluation of $f$ depends on the inputs of at most $q = o(n)$ of the parties, where the identity of these parties can be chosen randomly and possibly adaptively. Typically, $q = \mathsf{polylog}(n)$. While the sublinear query complexity of $f$ makes it possible in principle to dramatically reduce the *communication complexity* of our general protocol, the challenge is to achieve this while maintaining security: in particular, while keeping the *identities* of the selected inputs completely hidden. We solve this challenge, and we provide a protocol for securely computing such sublinear $f$ that runs in $\mathsf{polylog}(n) + O(q)$ rounds, has each party communicating with at most $q \cdot \mathsf{polylog}(n)$ other parties, and supports *message sizes* $\mathsf{polylog}(n) \cdot (\ell + n)$, where $\ell$ is the parties' input size.

Our optimized protocols rely on a multi-signature scheme, fully homomorphic encryption (FHE), and simulation-sound adaptive NIZK arguments. However, we remark that multi-signatures and FHE are used to obtain our bounds on message size and round complexity. Assuming only standard digital signatures and public-key encryption, one can still obtain the property that each party only communicates with $\mathsf{polylog}(n)$ other parties. We emphasize that the scheduling of faults can depend on the initial PKI setup of digital signatures and the NIZK parameters.

We begin in Section 5.1 by giving an overview of our solution. In Section 5.2, we define some preliminary tools to be used in our construction. Section 5.3 contains our formal security definition. In Section 5.4, we present our protocol for the case that $f$ is a *sublinear algorithm.* Then, in Section 5.5, we describe the modified protocol for achieving communication locality for *general* functions $f$.

## 5.1 Overview of Our Solution

We first describe how to achieve our second result, for the case when $f$ is a sublinear algorithm. This setting requires additional techniques in order to attain the communication complexity gains. After this, we describe the appropriate modifications required to maintain $\mathsf{polylog}(n)$ communication locality for general functions $f$.

There are three main technical components to our second result. The first is to set up a committee structure constituted of a *supreme committee* $C$ and $n$ *input committees* $C_1, ..C_n$. These committees will all be of size $\mathsf{polylog}(n)$ and with high probability have a 2/3 majority of honest players. The committee $C_i$ will (to begin with) hold shares of the input $x_i$ whereas the role of the supreme committee will essentially be to govern the running of the protocol. A major challenge is to ensure that all players in the network know the identity of players in all the committees. The starting point to address this challenge is to utilize the communication efficient *almost everywhere* leader election protocol of [KSSV06]. We remark that [KSSV06] achieves better total communication complexity of $\mathsf{polylog}(n)$ bits and offers unconditional results, but only achieves an almost-everywhere agreement: there may be a $o(1)$ fraction of honest players which will not reach agreement and in our context will not know the make up of the committees. The main idea to remedy this situation is to add an iterated certification procedure using multi-signatures to the protocol of [KSSV06], while keeping the complexity of only $\mathsf{polylog}(n)$ messages sent and processed by any honest party. In the process however we move from unconditional to computational security and our message sizes grow as they will be signed by multi-signatures which are only security parameter size but include the identities of the signers – this is cause for the increased size of messages.

The second component is to implement a randomly chosen secret reshuffling $\rho$ of players inputs within the complexity restrictions we have alloted. At the end of the shuffling committee $C_{\rho(i)}$ will hold the input of committee $C_i$. Informally, this will address the major privacy issue in executing a sub-linear algorithm in a distributed setting which is to ensure that the adversary does not learn which of the $n$ input is used by the algorithm. We implement the shuffling via distributed evaluation of a switching network with very good mixing properties under random switching, all under central coordination by the supreme committee. We assume that a fixed switching network over $n$ wires is given, with depth $d = \mathsf{polylog}(n)$, and is known to everyone.

The third component, once the inputs will be thus permuted, is to actually run the execution of the sub-linear algorithm. For lack of space let us illustrate how this is done for the sub class of non-adaptive sub-linear algorithms. This is a class of algorithms which proceed in two steps:

- First, a random subset $I$ of size $q$ of the indices $1, ..., n$ is selected.
- Second, an arbitrary polynomial time algorithm is computed on inputs $x_j$ for $j \in I$.

To run an execution of such an algorithm, the supreme committee: first selects a random and secret $q = poly(\log n)$ size subset $I$ of the input; and second runs a SFE protocol on the set of inputs in $\rho(I)$ with the assistance of players in committees $C_j$ for $j \in \rho(I)$. In the adaptive case, one essentially assumes queries are asked in sequence, and executes in a similar way the sublinear algorithm query after query, contacting committe $\rho(i)$ for each query $i$, instead of parallelizing the computation for all inputs from $I$. The price to pay is an additive factor $q$ in the number of rounds of the protocol. However, note that in the common case $q = \mathsf{polylog}(n)$, this does not affect the overall asymptotic complexity.

Now, consider now the case when $f$ is a general polynomial-time function, whose evaluation may depend on a large number of its inputs. In this case, we can skip the aforementioned shuffling procedure, and instead simply have *each party $P_i$* send his (encrypted) input up to the supreme committee $C$ to run the evaluation of $f$. That is, each $P_i$ gives an encryption of his input to the members of his input committee $C_i$, and each party in $C_i$ sends the ciphertext up to $C$ via the communication tree. Then, the members of the supreme committee $C$ (who collectively have the ability to decrypt ciphertexts) are able to evaluate the functionality $f$ directly via a standard MPC.

## 5.2 Preliminaries

We first recall the definitions of standard basic tools such as pseudorandom functions and generators, NIZK arguments of knowledge, and fully homomorphic encryption. (Note that our definition of FHE correctness is slightly non-standard, albeit achieved by the most efficient recent schemes.) We briefly review the definition of the less common notion of a multisignature scheme used within our protocol, followed by a review of security of multi-party protocols. We then move to reviewing some important results about shuffling and switching networks, as well as our notation for sublinear algorithms.

### 5.2.1 Pseudorandom functions and generators.

Recall that a family of functions $\mathbb{F} = \{F_s\}_{s \in S}$, indexed by elements of a set $S$, and where $F_s : D \to R$ for all $s$, is a *pseudorandom function (PRF) family* [GGM86] if for a randomly chosen $s$, the following and all PPT $\mathcal{A}$, the distinguishing advantage $\Pr_{s \leftarrow S}[\mathcal{A}^{f_s(\cdot)} = 1] - \Pr_{f \leftarrow (D \to R)}[\mathcal{A}^{\rho(\cdot)} = 1]$ is negligible, where $(D \to R)$ denotes the set of *all* functions from $D$ to $R$. A weaker notion is that for a *pseudorandom generator* (PRG), a length expanding function $\mathsf{prg} : \{0,1\}^k \to \{0,1\}^n$ (for $n > k$) such that $\mathsf{prg}(U_k)$ and $U_n$ are computationally indistinguishable, where $U_\ell$ is a uniformly distributed $\ell$-bit string.

### 5.2.2 Non-Interactive Zero Knowledge.

We are going to need some basic tools from non-interactive zero-knowledge. Let us start with the definition of a NIZK proof system with simulation-soundness, following various definitions in the literature [FLS90, BFM88, BSMP91, Sah99, SCO+01].

**Definition 5.2.1.** [FLS90, BFM88, BSMP91] A tuple $\Pi = (\mathsf{Gen}, \mathsf{P}, \mathsf{V}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{proof}}))$ is an *efficient adaptive NIZK argument system* for a language $L \in \mathsf{NP}$ with witness relation $\mathcal{R}$ if $\mathsf{Gen}, \mathsf{P}, \mathsf{V}, \mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{proof}}$ are all PPT algorithms, and there exists a negligible function $\mu$ such that for all $k$ the following three requirements hold.

- **Completeness**: For all $x, w$ such that $\mathcal{R}(x, w) = 1$, and for all strings $\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)$, it holds that $\mathsf{V}(\mathsf{crs}, x, \mathsf{P}(\mathsf{crs}, x, w)) = 1$.

- **Unbounded Adaptive Simulation Soundness**[Sah99, SCO$^+$01]: For all ppt adversaries $\mathcal{A}$, it holds that $\Pr[\mathsf{Exp}_{\mathcal{A}}(k)] \leq \mu(k)$, where the experiment $\mathsf{Exp}_{\mathcal{A}}(k)$ is defined by:

$$\begin{aligned}
&(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k)\\
&(x, \pi) \leftarrow \mathcal{A}^{\mathcal{S}'(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs})\\
&\text{Let } Q \text{ be the list of proofs given by oracle } \mathcal{S}' \text{ above}\\
&\text{Return } \mathsf{true} \text{ iff } (\pi \notin Q) \wedge (x \notin L) \wedge (\mathsf{V}(\mathsf{crs}, x, \pi) = 1),
\end{aligned}$$

where $\mathcal{S}'(\mathsf{crs}, \mathsf{trap}, x, w) := \mathcal{S}^{\mathsf{proof}}(\mathsf{crs}, \mathsf{trap}, x)$.

- **Adaptive Zero-Knowledge**: For all ppt adversaries $\mathcal{A}$, it holds that $\left|\Pr[\mathsf{Exp}_{\mathcal{A}}(k) = 1] - \Pr[\mathsf{Exp}^{\mathcal{S}}_{\mathcal{A}}(k) = 1]\right| \leq \mu(k)$, where the experiment $\mathsf{Exp}_{\mathcal{A}}(k)$ is defined by:

$$\begin{aligned}
&\mathsf{crs} \leftarrow \mathsf{Gen}(1^k)\\
&\text{Return } \mathcal{A}^{\mathsf{P}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs})
\end{aligned}$$

and the experiment $\mathsf{Exp}^{\mathcal{S}}_{\mathcal{A}}(k)$ is defined by:

$$\begin{aligned}
&(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k)\\
&\text{Return } \mathcal{A}^{\mathcal{S}'(\mathsf{crs},\mathsf{trap},\cdot,\cdot)}(\mathsf{crs}),
\end{aligned}$$

where $\mathcal{S}'(\mathsf{crs}, \mathsf{trap}, x, w) := \mathcal{S}^{\mathsf{proof}}(\mathsf{crs}, \mathsf{trap}, x)$.

**Theorem 5.2.2.** *[SCO$^+$01] There exists an unbounded simulation-sound, adaptive NIZK proof system for any* NP *language $L$, based on one-way functions, with proof length* $\mathsf{poly}(|x|, |w|)$, *where $x$ is the statement and $w$ is the witness.*

### 5.2.3 Fully Homomorphic Encryption.

A fully homomorphic public-key encryption scheme (FHE) consists of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$. The first three are the standard key generation, encryption and decryption algorithms of a public key scheme. The additional algorithm $\mathsf{Eval}$ is a deterministic polynomial-time algorithm that takes as input a public key $\mathsf{pk}$, a ciphertext $\hat{x} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x)$ and a circuit $C$, and outputs a new ciphertext $c = \mathsf{Eval}_{\mathsf{pk}}(\hat{x}, C)$ such that $\mathsf{Dec}_{\mathsf{sk}}(c) = C(x)$, where $\mathsf{sk}$ is the secret key corresponding to the public key $\mathsf{pk}$. It is required that the size of $c$ depends polynomially on the security parameter and the length of the output $C(x)$, but is otherwise independent of the size of the circuit $C$.

Several such FHE schemes have been constructed, starting with the seminal work of Gentry [Gen09]. Recently, new schemes were presented by Brakerski, Gentry and Vaikuntanathan [BV11, BGV11] that achieve greater efficiency and are based on the Learning with Errors (LWE) assumption. We note that in these schemes, the size of the public key depends linearly on the depth of the functions being evaluated. However, this issue can be avoided altogether if we assume that the schemes of [BV11, BGV11] are circular secure.

For our construction, we need an FHE scheme with the following additional property, which we refer to as *certifiability*. Loosely speaking, an FHE scheme is said to be certifiable, if there is an efficient algorithm that takes as input a random string $r$ and tests whether it is "good" to use $r$ as randomness in the encryption algorithm $\mathsf{Enc}$. More precisely, a certifiable FHE scheme is associated with a set $R$, which consists of all the "good" random strings, such that (1) a random string is in $R$ with overwhelming probability; and (2) the $\mathsf{Eval}$ algorithm and the decryption algorithm $\mathsf{Dec}$ are correct on ciphertexts derived from those using randomness from $R$ to encrypt.

We now formally define these notions.

**Definition 5.2.3.** For a given subset $R \subseteq \{0,1\}^{\mathsf{poly}(k)}$ of possible randomness values, we (recursively) define the class of *$R$-evolved* ciphertexts with respect to a public key $\mathsf{pk}$, evaluation key $\mathsf{evk}$, to include all ciphertexts $c$ of the form:

- $c = \mathsf{Enc}_{\mathsf{pk}}(m; r)$ for some $m$ in the valid message space and randomness $r \in R$, and
- $c = \mathsf{Eval}(\{c_i\}_{i \in I}, f)$ for some $\mathsf{poly}(k)$-size collection of $R$-evolved ciphertexts $\{c_i\}_{i \in I}$ and some poly-size circuit $f$.

**Definition 5.2.4.** A FHE scheme is said to be *certifiable* if there exists a subset $R \subseteq \{0,1\}^{\mathsf{poly}(k)}$ of possible randomness values for which the following hold.

1. $\Pr[r \in R] = 1 - \mathsf{negl}(k)$, where the probability is over uniformly sampled $r \leftarrow \{0,1\}^{\mathsf{poly}(k)}$.
2. There exists an efficient algorithm $\mathcal{A}_R$ such that $\mathcal{A}_R(r) = 1$ for $r \in R$ and 0 otherwise.
3. We have

$$\Pr_{\mathsf{pk},\mathsf{sk}}\left[\begin{array}{c} \forall\ R\text{-evolved ciphertexts } c_1, \ldots, c_n, \\ \forall \text{ poly-size circuits } f : \{0,1\}^n \to \{0,1\} \\ \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{pk}}(f, c_1, \ldots, c_n)) = f(b_1, \ldots, b_n), \\ \text{where } b_i = \mathsf{Dec}_{\mathsf{sk}}(c_i) \end{array}\right] = 1 - \mathsf{negl}(k).$$

We say that a ciphertext $c$ is *evaluation-enabled* if it is $R$-evolved for this special set $R$.

We note that this certifiability property holds, for example, for the schemes of [BV11, BGV11], assuming both LWE and circular security. For the readers who are familiar with these constructions, the set of "good" randomness $R$ corresponds to encrypting with sufficiently "small noise."

### 5.2.4 Multisignatures

In a multisignature scheme, a single short object—the *multisignature*—can take the place of $n$ signatures by $n$ signers, all on the same message.[1] The first formal treatment of multisignatures was given by Micali, Ohta, and Reyzin [MOR01]. We consider a variant of the Micali-Ohta-Reyzin model due to Boldyreva [Bol03], as presented in [LOS+06]. In this model, the adversary is given a single challenge verification key $\mathsf{vk}$, and a signing oracle for that key. His goal is to output a forged multisignature $\sigma^*$ on a message $m^*$ under keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell$, where at least one of these keys is a challenge verification key (wlog, $\mathsf{vk}_1$). For the forgery to be nontrivial, the adversary must not have queried the signing oracle at $m^*$.

[1]Note that multisignatures are a special case of *aggregate* signatures [BGLS03], which in contrast allow combining signatures from $n$ different parties on $n$ *different* messages.

For simplicity, we present a slightly weaker version of the security definition achieved by [LOS+06], which suffices for our application.[2]

**Definition 5.2.5.** A *multisignature* scheme is a tuple of algorithms

$\mathsf{Gen}(1^k)$**:** Key generation algorithm. Outputs a secret signing key $\mathsf{sk}$ together with corresponding public verification key $\mathsf{vk}$.

$\mathsf{Sign}(\mathsf{sk}, m)$**:** Standard signing algorithm, with respect to message $m$ and single signing key $\mathsf{sk}$.

$\mathsf{V}(\mathsf{vk}, \sigma)$**:** Standard signature scheme verification, with respect to a single verification key $\mathsf{vk}$.

$\mathsf{Combine}(\{\mathsf{vk}_i, \sigma_i\}_{i=1}^{\ell}, m)$**:** Takes as input a collection of signatures (or multisignatures) and outputs a combined multisignature, with respect to the union of verification keys.

$\mathsf{MultiVerify}(\{\mathsf{vk}_i\}_{i=1}^{\ell}, m, \sigma)$**:** Verifies multisignature $\sigma$ with respect to the collection of verification keys $\{\mathsf{vk}_i\}_{i=1}^{\ell}$. Outputs 0 or 1.

that satisfies the following properties:

**Correctness:** The standard signature correctness requirement must hold for $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{V})$. In addition, for any message $m$, any collection of honestly generated signatures $\{\sigma_i \leftarrow \mathsf{Sign}_{\mathsf{sk}_i}(m)\}_{i \in I}$ on $m$ (for $I \subset [n]$), the combined multisignature formed by $\bar{\sigma} \leftarrow \mathsf{Combine}(\{\mathsf{vk}_i, \sigma_i\}_{i \in I}, m)$ will properly verify with overwhelming probability: $\Pr[1 \leftarrow \mathsf{MultiVerify}(\{\mathsf{vk}_i\}_{i \in I}, m, \bar{\sigma})] \geq 1 - \mathsf{negl}(k)$.

**Unforgeability:** For any PPT adversary $\mathcal{A}$, the probability that the challenger outputs 1 when intreracting with $\mathcal{A}$ in the following game is negligible in the security parameter $k$:

Setup. The challenger samples $n$ pulic key-secret key pairs, $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^k)$ for each $i \in [n]$, and gives $\mathcal{A}$ all verification keys $\{\mathsf{vk}_i\}_{i \in [n]}$. $\mathcal{A}$ selects a proper subset $M \subset [n]$ (corresponding to parties to corrupt) and receives the corresponding set of secret signing keys $\{\mathsf{sk}_i\}_{i \in M}$.

Signing queries. $\mathcal{A}$ may make polynomially many adaptive signature queries, of the form $(m, \mathsf{vk}_i)$. For each such query, the challenger responds with a signature $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}_i}(m)$ on message $m$ with respect to the corresponding signing key $\mathsf{sk}_i$.

Output. $A$ outputs a triple $(\bar{\sigma}^*, m^*, \{\mathsf{vk}_i\}_{i \in S})$, where $\bar{\sigma}^*$ is an alleged forgery multisignature on message $m^*$ with respect to a subset of verification keys $S \subset [n]$. The challenger outputs 1 if at least one of the provided verification keys $\mathsf{vk}_i$ corresponds to a challenge (honest party) key, the message $m^*$ was not queried to the signature oracle with this verification key $\mathsf{vk}_i$, and the provided forgery $\sigma^*$ is a valid multisignature: i.e., $1 \leftarrow \mathsf{MultiVerify}(\{\mathsf{vk}_i\}_{i \in S}, m^*, \sigma^*)$.

[2]The security game in [LOS+06] also allows the adversary the power to choose verification keys on behalf of corrupted parties, as long as he also provides certification that the keys were properly generated.

The following theorem follows from a combination of the (standard) signature scheme of Waters [Wat05] together with a transformation from this scheme to a multisignature scheme due to Lu et. al. [LOS$^+$06].

**Theorem 5.2.6.** *[Wat05, LOS$^+$06] There exists a secure multisignature scheme with signature size* $\mathsf{poly}(k)$ *(independent of message length and number of potential signers), based on the Bilinear Computational Diffie-Hellman assumption.*

### 5.2.5 Multi-party protocols: Model and Security Definitions

We consider the setting of $n$ parties $\mathcal{P} = \{P_1, ..., P_n\}$ within a synchronous network who wish to jointly compute any PPT function $f$ over their private inputs. We allow up to $\leq (\frac{1}{3} - \epsilon)n$ statically chosen Byzantine (malicious) faults, for any constant $\epsilon > 0$, and a rushing adversary. We assume that every pair of parties has the ability to initiate direct communication via a point-to-point private, authenticated channel. (However, we remark that in our protocol, each (honest) party will only ever send or process information along subset of only $\mathsf{polylog}(n)$ such channels.) We assume the existence of a public-key infrastructure, but allow the adversary's choice of corruptions to be made as a function of this public information.

The notion of security we consider is the standard simulation-based definition of secure multi-party computation (MPC) i.e., to formally define security, we turn to the real/ideal paradigm. A full description of the ideal- and real-world experiments, together with our definition of security, are contained in Section 5.3.

**General multi-party computation.** The following theorem is well known and will be use throughout this paper. Let $C$ be a circuit with $n$ inputs, and let $F_C$ the functionality that computes the circuit.

**Theorem 5.2.7.** *[BGW88] For any $t < n/3$, there exists a protocol that securely computes the functionality $F_C$ functionality, with* perfect *security. The protocol proceeds in $|C|$ rounds, and each party sends* $\mathsf{poly}(n)$ *messages of size* $O(\mathsf{poly}(k, n))$ *each.*

**Verifiable Secret Sharing.** A *secret sharing* scheme, a notion introduced by Shamir [Sha79], is a protocol that allows a dealer who holds a secret input $s$, to share his secret among $n$ parties. The guarantee is that even if $t$ of the parties are malicious, they gain no information about the secret $s$. A *verifiable secret sharing* (VSS) scheme, introduced by Chor et al. [CGMA85], is a secret sharing scheme with the additional guarantee that after the sharing phase, a dishonest dealer is either rejected, or is committed to a single secret $s$, that the honest parties can later reconstruct. Further, if the dealer is honest, then the original secret will be reconstructed, even if dishonest parties do not provide their correct shares.

For concreteness, we consider a class of VSS constructions that takes advantage of reconstruction and secrecy properties of low-degree polynomials [Sha79, MS81]. Security of such a VSS protocol $\mathsf{Share}$ can be formalized as emulating the following ideal functionality.

**Definition 5.2.8.** The ideal polynomial VSS functionality $F^t_{\mathsf{VSS}}$ for parties $P_D, P_1, ..., P_n$ with distinguished dealer $P_D$, is defined as:

$$F_{\mathsf{VSS}}(q(x), (\emptyset, ..., \emptyset)) = \begin{cases} (\emptyset, (q(\alpha_1), ..., q(\alpha_n))) & \text{if } \deg(q) \leq t \\ (\emptyset, (\perp, ..., \perp)) & \text{else} \end{cases}.$$

The party can also run a *reconstruction protocol* Rec such that if honest parties input the correct shares output by the above functionality to them, then they recover the right value. The following result is well known.

**Theorem 5.2.9.** *[BGW88, AL11] For any $t < n/3$, there exists a constant-round protocol* Share *that securely computes the $F^t_{\mathsf{VSS}}$ functionality, with* perfect *security. Each party sends* $\mathsf{poly}(n)$ *messages of size $O(l \log l)$, where $l = \max\{|x|, n\}$.*

Also, we will be interested in the case where the dealer $D$ can be any of the $n$ players, and he sends shares to a subset $P'$ of the $n$ players of size $n'$ (e.g., $n' = \mathsf{polylog}(n)$), and we may not necessarily have $D \in P'$. The above functionality can be extended to this case naturally, and it is a folklore result that the protocols given by the above theorem also remain secure in this case as long as less than a fraction $1/3$ of the parties in $P'$ are corrupted.

**Broadcast.** Another important functionality we need to implement is broadcast. This can also be seen as an example of a MPC implementing the following functionality $F_{\mathsf{BC}}$ for parties $P_D, P_1, ..., P_n$ with distinguished dealer $P_D$, is defined as $F_{\mathsf{BC}}(m, (\emptyset, ..., \emptyset)) = \big(\emptyset, (m, \ldots, m)\big)$, where $m$ is the message to be broadcast.

**Theorem 5.2.10.** *[FM88] For any $t < n/3$, there exists a constant-round protocol that securely computes the $F_{\mathsf{BC}}$ functionality, with* perfect *security. Each party sends* $\mathsf{poly}(n)$ *messages of size $O(|m|)$ each.*

### 5.2.6 Random Switching Networks and Random Permutations.

Our protocol will employ what we call an *$n$-wire switching network*, which consists of a sequence of *layers*, each layer in turn consisting of one or more swapping gates operating which decide to swap the values of two wires depending on a bit. Formally, given $\vec{x} = (x_1, \ldots, x_n)$ (which we assume to be integers wlog), a swap operation $\mathsf{swap}(i, j, \vec{x}, b)$ returns $\vec{x}'$ where $\vec{x} = \vec{x}'$ if $b = 0$, and otherwise, if $b = 1$, $x'_i = x_j$ and $x'_j = x_i$, whereas $x'_k = x_k$ for all $k \neq i, j$. Formally, a switching layer is a set $L = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ of pairwise-disjoint pairs of distinct indices of $[n]$. A $d$-depth switching network is a list $SN = (L_1, \ldots, L_d)$ of switching layers. Note that for each assignment of the bits of the gates in $SN$, the network defines a permutation from $[n]$ to $[n]$ by inputting the vector $\vec{x} = (1, 2, \ldots, n)$ to the network. The question we are asking is the following: If we set each bit in each swap gate uniformly and independently at random, how close to uniform does the resulting permutation look like? The following theorem guarantees the existence of a sufficiently shallow switching network giving rise to an almost-uniform random permutation.

**Theorem 5.2.11.** *For all $c > 1$, there exists an efficiently computable $n$-wire switching network of depth $d = O(\mathsf{polylog}(n) \cdot \log^c(k))$ (and size $O(n \cdot d)$) such that the permutation $\widehat{\pi} : [n] \to [n]$*

*implemented by the network when setting swaps randomly and independently has negligible statistical distance (in $k$) from a uniformly distributed random permutation on $[n]$.*

*Proof.* By Theorem 1.11 in [CKLK01], there exists such network $SN$ of depth $d = O(\mathsf{polylog}(n))$ where the statistical distance is of the order $O(1/n)$. Consider now the switching network $SN'$ obtained by cascading $r$ copies of $SN$. Then, when setting switching gates at random, the resulting permutation $\widehat{\pi}$ equals $\widehat{\pi}_1 \circ \cdots \circ \widehat{\pi}_n$, where $\widehat{\pi}_i$ are independent permutations obtained each by setting the gates in $SN$ uniformly at random. With $\pi$ being a random permutation, a well-known property of the statistical distance $\Delta(\cdot,\cdot)$, combined with the fact permutation composition gives a group (see e.g. [MPR07] for a proof) yields

$$\Delta(\widehat{\pi},\pi) \leq 2^{r-1} \cdot \prod_{i=1}^{r} \Delta(\widehat{\pi}_i,\pi) \leq O\left(\left(\frac{2}{n}\right)^r\right) \leq O(2^{r(\log 2 - \log(n))}) \,,$$

which is negligible in $k$ for $r = \log^c(k)$. □

Note that in particular this means that each wire is connected to at most $d = O(\mathsf{polylog}(n) \cdot \log^c(k))$ other wires via a switching gates, as each wire is part of at most one gate per layer.

### 5.2.7 Sublinear algorithms

We briefly discuss some notational conventions when dealing with sublinear algorithms. We consider a model where $n$ inputs $x_1,\ldots,x_n$ are accessible to an algorithm SLA via individual queries for indices $i \in [n]$. In the most general case, a *$Q$-query algorithm* in the $n$-input model is a tuple of (randomized) polynomial time algorithms $\mathsf{SLA} = (\mathsf{SLA.Sel}_1, \mathsf{SLA.Sel}_2, \ldots, \mathsf{SLA.Sel}_Q, \mathsf{SLA.Exec})$. During an execution with inputs $(x_1,\ldots,x_n)$, $\mathsf{SLA.Sel}_1$ takes no input and produces as output a state $\sigma_1$ and a query index $i_1 \in [n]$, and for $j = 2,\ldots,n$, $\mathsf{SLA.Sel}_j$ takes as input a stage $\sigma_{j-1}$, an input $x_{i_{j-1}}$, and outputs a new state $\sigma_j$ and a new query index $i_j$. Finally, SLA.Exec takes as input $\sigma_Q$ and $x_Q$, and produces a final output $y$. We say that SLA is *sublinear* if $Q = o(n)$. For simplicity, we will restrict the presentation of our main result to the case of *non-adaptive* algorithms which consist of only two components $\mathsf{SLA} = (\mathsf{SLA.Sel}, \mathsf{SLA.Exec})$, where SLA.Sel outputs a random subset $I \subseteq [n]$ of indices of inputs to be queries, and the final output is obtain by running SLA.Exec on those inputs $x_i$ for $i \in I$.

Examples of sublinear algorithms, many of them non-adaptive, include algorithms for property testing such as testing sortedness of the inputs, linearity, approximate counting, and numerous graph properties, etc. Surveying this large area and the usefulness of these algorithms goes beyond the scope of this paper, and we refer the reader to available surveys; several pointers can be obtained for example from [Sls].

## 5.3 Simulation-Based Security Definition

The notion of security we consider is the standard simulation-based definition of secure multiparty computation (MPC). To formally define security, we turn to the real/ideal world paradigm.

We begin by describing the ideal world experiment; we then describe the real world experiment; and finally, we present our security definition. Throughout this work, we assume that the functions to be evaluated give the same output to all parties.

**Ideal World** In the ideal world, each party $P_i$ sends her input $x_i$ to a trusted third party, who replies with the desired evaluation $f(x_1, \ldots, x_n)$. We emphasize that for a randomized functionality $f$, the trusted party returns a random evaluation $f(x_1, ..., x_n)$, and no other information about the *randomness* that was used. In particular, in the case that $f$ is a randomized sublinear algorithm, whose evaluation depends only on a small (random) subset of its inputs, no information on which inputs were selected will be revealed.

The ideal world model is formally described below.

**Inputs:** Each party $P_i$ obtains an input $x_i$. The adversary is given auxiliary input $z$. He selects a subset of the parties $M \subset \mathcal{P}$ to corrupt, and is given the inputs $x_\ell$ of each party $P_\ell \in M$.

**Sending inputs to trusted party:** Each honest party $P_i$ sends its input $x_i$ to the ideal functionality. For each corrupted party $P_i \in M$, the adversary may select any value $x'_i$ and send it to the ideal functionality.

**Trusted party computes output:** Let $x'_1, ..., x'_n$ be the inputs that were sent to the trusted party. Then, the trusted party responds to all parties with the message $f(x'_1, ..., x'_n)$.

**Outputs:** Honest parties output the message they obtained from the ideal functionality. Malicious parties may output an arbitrary PPT function of the adversary's view.

The overall output of the ideal-world experiment consists of the outputs of all parties. For any ideal-world adversary $\mathcal{S}$ with auxiliary input $z \in \{0,1\}^*$, any input vector $\vec{x}$, and security parameter $k$, we denote the output of the corresponding ideal-world experiment by

$$\mathsf{IDEAL}_{\mathcal{S},M}\left(1^k, \vec{x}, z, f\right).$$

**Real World** The real world execution begins by the dissemination of the PKI setup information. In response, an adversary $\mathcal{A}$ selects any arbitrary subset of parties $M \subset \mathcal{P}$ of size $|M| < (\frac{1}{3} - \epsilon)|\mathcal{P}|$ to corrupt (for some constant $\epsilon > 0$). The parties then engage in an execution of a real $n$-party protocol $\Pi$, as described below. We assume that honest parties have the ability to toss fresh coins at any point. Throughout the execution of $\Pi$, the adversary $\mathcal{A}$ sends all messages on behalf of the corrupted parties, and may follow an arbitrary polynomial-time strategy. In contrast, the honest parties follow the instructions of $\Pi$. At the conclusion of the protocol execution, honest parties output as directed by $\Pi$. Malicious parties may output an arbitrary PPT function of the adversary's view.

For any adversary $\mathcal{A}$ with auxiliary input $z \in \{0,1\}^*$, any inputs $\{x_i\}_{i=1}^n$, and any security parameter $k$, we denote the output of the MPC protocol $\Pi$ by

$$\mathsf{REAL}^{\Pi}_{\mathcal{A},M}\left(1^k, \vec{x}, z, f\right).$$

Loosely speaking, we say that a protocol $\Pi$ is a secure MPC protocol for computing $f$ if any adversary, who corrupts a subset of parties, and runs the protocol with honest parties on function $f$, gains *no information* about the inputs of the honest parties beyond the function output $f(x_1, ..., x_n)$. We formalize this in the next subsection.

**Security Definition** In what follows, we formally define our model of security; i.e., what it means for a real-world protocol to emulate the desired ideal world.

**Definition 5.3.1** (Secure MPC). A protocol $\Pi$ is said to securely compute a function $f$ if for every PPT adversary $\mathcal{A}$ in the real world, there exists a PPT adversary $\mathcal{S}$ corrupting the same parties in the ideal world such that for every input vector $\vec{x}$, and every auxiliary input $z$, it holds that

$$\mathsf{IDEAL}_{\mathcal{S},M}\left(1^k, \vec{x}, z, f\right) \approx_c \mathsf{REAL}^{\Pi}_{\mathcal{A},M}\left(1^k, \vec{x}, z, f\right).$$

## 5.4 Multiparty Computation of Sublinear Algorithms

This section presents the first main result of this work: a generic compiler for private distributed evaluation of a *sublinear algorithm.* We begin by stating and discussing the result in Section 5.4.1. We then provide a more detailed technical overview of our solution in Section 5.4.2; present the complete protocol in Section 5.4.3; and give the proof of security in Section 5.4.4.

### 5.4.1 Our result

We first present our result for the simpler case of a non-adaptive sublinear algorithm, and then discuss the extension of our result to the adaptive case below. Our protocol compiles a non-adaptive sublinear algorithm $\mathsf{SLA} = (\mathsf{SLA.Sel}, \mathsf{SLA.Exec})$ into a multi-party protocol $\Pi_{\mathsf{SLA}}$ executing the algorithm on the inputs $x_1, \ldots, x_n$ of the $n$ parties. This in particular means that the protocol $\Pi_{\mathsf{SLA}}$ securely computes the functionality $\mathcal{F}_{\mathsf{SLA}}$ which, on inputs $x_1, \ldots, x_n$ first samples a subset $I \subseteq [n]$ according to $\mathsf{SLA.Sel}$, and then runs $\mathsf{SLA.Exec}$ on the $x_i$'s for $i \in I$, and finally outputs the result. Without loss of generality, we assume that $\mathsf{SLA.Sel}$ always makes *exactly* $Q$ distinct queries, and that $Q$ is known. The following is our main theorem, proven below.

**Theorem 5.4.1 (Multi-party evaluation of sublinear algorithms).** *Let* $\mathsf{SLA} = (\mathsf{SLA.Sel}, \mathsf{SLA.Exec})$ *be a non-adaptive sublinear algorithm which retrieves $Q = Q(n) = o(n)$ different inputs. Then, for all constant $\epsilon > 0$, there exists an $n$-party protocol $\Pi_{\mathsf{SLA}}$ that securely computes the functionality $\mathcal{F}_{\mathsf{SLA}}$ tolerating $t < (1/3 - \epsilon)n$ active corruptions, with the following complexities, where $k$ is a security parameter and $|x|$ is the size of the individual inputs held by the parties:*

*(1) The protocol runs in* $\mathsf{poly}(\log n, k)$ *rounds.*

*(2) Each honest party talks with at most $Q \cdot$* $\mathsf{polylog}(n)$ *other parties and sends at most $Q \cdot$* $\mathsf{polylog}(n) + \mathsf{poly}(\log n, k)$ *messages.*

*(3) Each message has size of order* $(|x| + n) \cdot \mathsf{poly}(\log n, k)$.

*(4) The protocol uses a setup consisting of $n \cdot \mathsf{polylog}(n)$ signing keys of size $\mathsf{poly}(k)$, as well as a $\mathsf{poly}(k)$-long additional CRS.*

*The protocol assumes a secure multisignature scheme, a FHE scheme, simulation-sound NIZK arguments, as well as pseudorandom generators.*

Often, as common in the area of security parameters, we think of $n$ as being the security parameter, and one can typically set $k = \mathsf{polylog}(n)$ to make the above result independent of the additional parameter $k$.

Let us briefly compare the complexity of our protocol with the canonical execution of SLA in the traditional non-private setting with a central server; there, one lets the server retrieve the inputs of the subset of players $I \subseteq [n]$ of sublinear size $Q$ (e.g., $Q = \mathsf{polylog}(n)$) output by SLA.Sel, which subsequently locally executes SLA.Exec on these inputs. In particular, the server talks to $Q$ parties, and the overall communication complexity for the server is $Q \cdot |x|$, where often $Q$ is at least $\mathsf{polylog}(n)$. Note that our protocol ensures that the communication for each party is not much larger – in fact, in most cases, it seems reasonable to assume that $|x| = \Omega(n)$.

The above result extends to $q$-query *adaptive* sublinear algorithms. The only asymptotic additional cost are $O(q)$ rounds, otherwise the theorem statement remains unchanged. We sketch at the end of the technical overview how to obtain this result.

### 5.4.2 Technical overview

Ideally, we would like to implement our protocol along the following lines. First, a small committee $C$ consisting of $\mathsf{polylog}(n)$ parties is elected, with the property that at least two thirds of its members are honest. This committee then jointly decides on a random subset of $Q$ players $I$, output by SLA.Sel, from which inputs are obtained. The parties in $C \cup I$ jointly execute a multiparty computation among themselves to produce the output of the sublinear algorithm according to the algorithm SLA.Exec, which is then broadcasted to all parties.

But things will not be as simple. Interestingly, one main challenge is very unique to the setting of sublinear algorithms: An execution of the protocol needs to hide the subset $I$ of players whose input contribute to the output! More precisely, an ideal execution of the sublinear algorithm via the functionality $\mathcal{F}_{\mathsf{SLA}}$ only reveals the output of the sublinear algorithm. Therefore, we need to ensure that the adversary does not learn any additional information about the composition of $I$ from a protocol execution other than what leaked via the final output. Our protocol will indeed hide the set $I$ completely. This will require modifying the above naive approach considerably.

The second challenge is complexity theoretic in nature. Enforcing low complexity of our protocol when implementing the above steps, while realizing our mechanism to hide the subset $I$, will turn out to be a delicate balance act.

In particular, at a high level our protocol will consist of the following components:

**Committee election phase.** The $n$ parties jointly elect a supreme committee $C$, as well as individual committee $C_1, \ldots, C_n$ on which they *all agree*, sending each at most $\mathsf{polylog}(n)$ messages of size each $n \cdot \mathsf{poly}(\log n, \log k)$. All committees have size $\mathsf{polylog}(n)$ and at least a fraction 2/3 of the players in them are honest.

**Commitment phase.** Each party $P_i$ commits to its input so that $C_i$ holds shares of these inputs.

**Shuffling phase.** To hide the access pattern of the algorithm (i.e., which inputs are included in the computation), the committees are going to randomly shuffle the inputs they hold with respect to a random permutation $\rho$. This is going to happen by using a switching network with good shuffling properties, and for each swap gate $(i, j)$ in the switching network by having committees $C_i$ and $C_j$ swapping at random the sharings they hold via a multi-party computation under a random decision taken by the supreme committee $C$. The supreme committee then holds a sharing of $\rho$.

**Evaluation phase.** The parties in the supreme committee $C$ sample a random set $I$ according to $\mathsf{SLA.Sel}$ via MPC and learn $\rho(I)$ only. They are going to then include the parties in all $C_i$ for $i \in \rho(I)$ in a multi-party computation to evaluate the sublinear algorithm on the inputs they hold. (Recall that $C$ holds $\rho$ in shared form.)

**Output phase.** The supreme committee broadcasts the output of the computation to all parties.

In addition, we carefully implement sharings and multi-party computations using fully-homomorphic encryption to improve complexity, making dependency of communication and round complexities linear in the input length $|x|$, rather than polynomial, and independent of the circuit sizes to implement the desired functionalities.

The following paragraphs provide a more detailed account of the techniques used within our protocol, before we turn to a formal description of the protocol.

**Committee election phase.** The backbone behind this first phase is given by the construction of a *communication tree* using a technique of King et al [KSSV06]. Such tree is a sparse communication subnetwork which will ensure both the election of the supreme committee, as well as a basic form of communication between parties and the supreme committee where each party communicates only with $\mathsf{polylog}(n)$ other parties and only $\mathsf{polylog}(n)$ rounds of communication are required. Informally, the protocol setting up the tree assigns (possibly overlapping) subsets of parties of polylogarithmic size to the nodes of a tree with polylogarithmic height and logarithmic degree. The set of players assigned to the root will take the role the supreme committee $C$. Communication from the root to the parties (or the other way round) occurs by communicating messages over paths from the root to the leaves of the tree, with an overall communication cost of $\mathsf{polylog}(n)$ messages per party. To elect the committees $C_1, \ldots, C_n$, we can have the supreme committee agree on the seed $s$ of a PRF family $\mathbb{F} = \{F_s\}_s$ via a coin tossing protocol, where $F_s$ maps elements of $[n]$ to subsets of $[n]$ of size $\mathsf{polylog}(n)$, and send $s$ to all parties. We then let $C_i = F_s(i)$.

However, a closer look reveals that it is only possible for the protocol building the communication tree to enforce that a vast majority of the nodes of the tree are assigned to a set of parties for which a 2/3 majority is honest, but some nodes are unavoidably associated with a too large fraction of corrupted parties. Indeed, some parties may be connected to too many bad nodes and their communication ends up being essentially under adversarial control. As a consequence, the supreme committee is only able to correctly communicate with a $1 - o(1)$ fraction of the (honest) parties. Moreover, individual parties are not capable of determining whether the value they hold is correct or not. We refer to this situation as *almost-everywhere (ae) agreement.*

Our main contribution here is the use of cryptographic techniques to achieve *full agreement* on $C$ and $s$ in this stage, while maintaining polylogarithmic communication complexity, improving on previous work in the information-theoretic setting [KLST11, KS11, DKMS12] which requires higher $O(\sqrt{n} \cdot \mathsf{polylog}(n))$ complexity for agreement. We tackle these two issues in two separate ways.

1. *From ae agreement to ae certified agreement.* We are going to first move to a stage where a large $1 - o(1)$ fraction of the parties learn the value sent by the supreme committee, together with a *proof* that the output is the one sent by the committee, whereas the remaining parties who do not know the output are also aware of this fact. We refer to this scenario as *almost-everywhere certified agreement.* Let us start with the basic idea using traditional signatures (we improve on this below using multisignatures). After having the supreme committee send a value $m$ to all parties with almost-everywhere agreement, each party $P_i$ receiving a value $m_i$ is going to *sign* $m_i$ with his own signing key, producing a signature $\sigma_i$. Then, $P_i$ sends $(m_i, \sigma_i)$ up the tree to the supreme committee, and each member is going to collect at least $n/2$ signatures on $\sigma_i$ on some message $m$. Note that this will always be possible, as a fraction $1 - o(1) > n/2$ of the honest players is going to receive the message $m_i = m$, and send a valid signature up the tree. Moreover, the adversary will need to forge signatures for honest parties in order to produce a valid certificate for a message which was not broadcast by the supreme committee.

2. *From ae certified agreement to full agreement.* We finally describe a transformation from ae certified agreement to full agreement. If a committee wants to broadcast $m$ to all parties, the committee additionally generates a seed $s$ for a and broadcasts $(m, s)$ in a certified way using the above transformations. Each party receiving $(m, s)$ with a valid certificate $\pi$ forwards $(m, s, \pi)$ to all parties in $F_s(i)$. Whenever a party receives $(m, s, \pi)$ with a valid certificate, it stops and outputs $m$. Note that no party sends more than $\mathsf{polylog}(n)$ additional messages in this transformation. Moreover, it is not hard to see that with very high probabilities every honest party is going to be in at least one of the $F_s(i)$ for a party $i$ which receives $(m, s)$ correctly with a certificate by the pseudorandomness of $\mathbb{F}$. Note in particular that the same seed $s$ can be used over multiple executions of the broadcast from the committee to the parties, and can be used directly to generate the committees $C_1, \ldots, C_n$.

While we do guarantee that every party *sends* at most $\mathsf{polylog}(n)$ messages, a problem of the above approach is the high complexity of processing incoming messages due to dishonest parties flooding honest parties by sending too many messages. Namely, the $t = \Theta(n)$ corrupted parties can always each send $(m, s)$ with an invalid certificate to some honest party $P_i$, who needs to verify all signatures in the certificate to confirm that these messages are not valid. We propose a solution based on multisignatures that alleviates this problem by making certificates only consist of an individual *aggregate* signature (instead of of $\Theta(n)$), as well as of a description of the subset of parties whose signatures have been aggregated. The main idea is to have all parties initially sign the value they receive from the supreme committee with their own signing keys. However, when sending their values up the tree, parties assigned to inner nodes of the tree are going to aggregate valid signatures on the message which was previously sent down the tree, and keep track of which signatures have contributed.

**Commitment phase.** Our instantiations of multi-party computations are going to be based on fully-homomorphic encryption. To this end, we want parties in $C_i$ to store an encryption of $\mathsf{Enc}(\mathsf{pk}, x_i)$ that we want to be committing. The public key $\mathsf{pk}$ is generated by the supreme committee, who holds secret shares of the matching secret key $\mathsf{sk}$, and sent to all parties with the methods outlined above. A player $i$ is committed to the value $x_i$ if the honest parties in $C_i$ all hold the *same* ciphertext encrypting $x_i$. This presents some challenges which we address and solve as follows:

1. First, a malicious party $P_i$ must not be able to broadcast an invalid ciphertext to the members of the committee $C_i$. This is prevented by appending a simulation-sound NIZK argument $\pi$ to the ciphertext $c$ that there exists a message $x$ and randomness $r$ such that $\mathsf{Enc}(\mathsf{pk}, x; r) = c$.

2. Second, for a security proof to be possible, it is well known that not only the encryption needs to be hiding and binding, but a simulator needs to be able to have some way to extract the corresponding plaintext from a valid ciphertext-proof pair $(c, \pi)$. A major issue here is that the simulated setup is independent of the corrupted set in our model. This prevents the use of NIZK arguments of knowledge. Moreover, we can expect the FHE encryption to be secure against chosen plaintext attacks only. We will solve this by means of *double encryption*, following Sahai's construction [Sah99] of a CCA-secure encryption scheme from a CPA-secure one. Namely, we provide an additional encryption $c_2$ of $x$ under a different public-key (for which no one needs to hold the secret key), together with an additional NIZK argument that $c_1$ and $c_2$ encrypt the same message. The ciphertext $c_2$ will not be necessary at any later point in time and serves only the purpose of verifying commitment validity (and permitting extraction in the proof).

3. Third, a final problem we have to face is due to rushing adversaries and to possibility to maul commitments in view of the use of the same public key $\mathsf{pk}$ for all commitments. This can be prevented in a black-box way by letting every party $P_i$ first in parallel VSS its commitment to the parties in $C_i$, and then in a second phase let every committee $C_i$ reconstruct the corresponding commitment. If the VSS protocol is perfectly secure, this ensures input-independence.

One challenge is how to ensure that ciphertext sizes and the associated NIZK proof length are all of the order $|x| \cdot \mathsf{poly}(k)$, instead of $\mathsf{poly}(|x|, k)$. We are going to achieve this by encrypting messages bit-by-bit using a bit-FHE scheme, whose ciphertexts are hence of length $\mathsf{poly}(k)$. The NIZK proof is obtained by sequentially concatenating individual proofs (each of length $\mathsf{poly}(k)$) for the encryptions of individual bits.

**Shuffling phase.** The major privacy issue in executing a sub-linear algorithm in a distributed setting is that we do not want the adversary to learn which parties have contributed with their inputs to the protocol, and which ones did, more than what the algorithm's output may itself reveal. Ideally, we would like parties to randomly permute their inputs in a random (yet oblivious) fashion, so that at the end of such a protocol each party $P_i$ holds the input of player $P_{\pi(i)}$ for a random permutation $\pi$, but such that the adversary has no information about the choice of $\pi$ and for which player $\pi(i)$ he holds input for. At the same time, the committee jointly holds information about the permutation $\pi$ in a shared way. However, this seems impossible to achieve: A disrupting adversary may always refuse to hold inputs for other players. However, we can now exploit the fact that the inputs are held by *committees* $C_1, \ldots, C_n$ containing a majority of honest players.

The actual shuffling is implemented via distributed evaluation of a switching network $SN$, under central coordination by the supreme committee. We assume that a switching network over $n$ wires is given, with depth $d = \mathsf{polylog}(n)$, and is known to everyone, and with the property given by Theorem 5.2.11, i.e., it implements a nearly uniform permutation on $[n]$ under random switching. For each swap gate $(i, j)$ in the network, the committee members jointly produce an encryption $\hat{b}_{i,j}$ of a random bit $b_{i,j}$, indicating whether the inputs $x_i$ and $x_j$ are to be swapped or not when evaluating the corresponding swapping gate. This is achieved by broadcasting $\hat{b}_{i,j}$ to all parties in $C_i$ and $C_j$. At this point, each party in $C_i$ broadcasts his copy of $\widehat{x}_i$ to all parties in $C_j$, and each party in $C_j$ does the same with $\hat{x}_j$ to all parties in $C_i$. (Each party then, given ciphertexts from the other committee, will choose the most frequent one as the right one.) Then, each party in $C_i$ (or $C_j$) will update his encryption $\hat{x}_i$ to be an encryption of $\mathsf{Dec}(\mathsf{sk}, \hat{x}_j)$ or $\mathsf{Dec}(\mathsf{sk}, \hat{x}_i)$, depending on the value of $\hat{b}_{i,j}$, using homomorphic evaluation. We note that the operation can be executed in parallel for all gates on the same layer, hence the swapping requires $d$ rounds.

**Evaluation phase.** Once the parties' inputs have been (obliviously) shuffled, we are ready to run the sublinear algorithm. The execution is controlled by the supreme committee $C$. First, the members of $C$ will run an MPC to randomly select the subset of inputs $I \subset [n]$ to be used by the algorithm. The output of the MPC will be the set of *permuted* indices $\sigma(I) := \{\sigma(i) : i \in I\}$. The corresponding committees $\{C_j : j \in \sigma(I)\}$ are invited to join in a second MPC. Each member of $C_j$ enters the MPC with input equal to his currently held encrypted secret share (of some unknown input $x_i$, for which $j = \sigma(i)$). Each member of $C$ enters the MPC with input equal to his share of the secret decryption key $\mathsf{sk}$. Collectively, the members of $C \cup (\bigcup_{j \in \sigma(I)} C_j)$ run an MPC which (1) recombines the shares of $\mathsf{sk}$, (2) decrypts the secret shares held by each $C_j$, (3) reconstructs each of the relevant inputs $x_i$, $i \in I$, from the corresponding set of secret shares, (4) executes the sublinear algorithm on the reconstructed inputs, and (5) outputs *only* the output value dictated by the sublinear algorithm (e.g., for many algorithms, this will simply be YES/NO).

The main challenge is making the complexity of this stage such that only $\mathsf{poly}(\log n, \log k)$ rounds are executed, and only messages of size $|x| \cdot \mathsf{poly}(\log k, \log n)$ are going to be exchanged. This will be achieved by performing most of the computations locally via FHE by the parties in the supreme committee, and by generating the randomness used by $\mathsf{SLA.Sel}$ and $\mathsf{SLA.Exec}$ by first agree on a $\mathsf{poly}(k)$-short seed of a PRG via coin-tossing, and then subsequently using as the actual randomness the PRG output.

**Extension: Adaptive algorithms.** Here, given $\mathsf{SLA} = (\mathsf{SLA.Sel}_1, \ldots, \mathsf{SLA.Sel}_q, \mathsf{SLA.Exec})$, we just need to modify the evaluation phase such that an MPC is run for each next-query $\mathsf{SLA.Sel}_j$ to obtain $\rho(i_j)$, the permuted index of the next query. We need to guarantee that queries are distinct without loss of generality, which is easy to enforce. Note that the number of rounds unavoidably increases: Namely, we need $O(q)$ additional rounds to obtain inputs from the committees $C_{\rho(i_j)}$ one by one. Otherwise, the proof and the protocol are quite similar, and we postpone a more detailed description to the final version of this paper.

### 5.4.3 The protocol $\Pi_{\mathsf{SLA}}$

In this section, we describe in detail our main protocol $\Pi_{\mathsf{SLA}}$.

**Cryptographic tools.** We start with a description of the cryptographic tools, some of which are formally defined in Section 5.2:

- We use $\mathsf{FHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$: a fully homomorphic public-key encryption (FHE) scheme that is certifiable with respect to an efficiently testable set $R \subseteq \{0,1\}^{\mathsf{poly}(k)}$, as described in Section 5.2.3, and which takes inputs of length $|x|$ with ciphertext length $|x|\cdot\mathsf{poly}(k)$.
- We use $\mathsf{Enc}^{\mathsf{CPA}} = (\mathsf{Gen}^{\mathsf{CPA}}, \mathsf{Enc}^{\mathsf{CPA}}, \mathsf{Dec}^{\mathsf{CPA}})$: a standard semantic secure ("chosen plaintext attack" CPA-secure) public-key encryption scheme which takes inputs of length $|x|$ with ciphertext length $|x| \cdot \mathsf{poly}(k)$.
- $\mathsf{NIZK} = (\mathsf{Gen}, \mathsf{P}, \mathsf{V}, \mathcal{S} = (\mathcal{S}^{\mathsf{crs}}, \mathcal{S}^{\mathsf{proof}}))$: a simulation-sound non-interactive zero-knowledge argument system for the NP language

  $$\mathcal{L} = \{(\mathsf{pk}, \hat{m}) : \exists (m, r) \text{ s.t. } r \in R,\ \hat{m} = \mathsf{Enc}_{\mathsf{pk}}(m; r)\},$$

  where $R \subseteq \{0,1\}^{\mathsf{poly}(k)}$ is the set of randomness for which the FHE scheme is certifiable.

  We also make use of a second instantiation of the NIZK argument system, denoted $(\mathsf{Gen}', \mathsf{P}', \mathsf{V}', \mathcal{S}' = (\mathcal{S}'^{\mathsf{crs}}, \mathcal{S}'^{\mathsf{proof}}))$, for the NP language

  $$\mathcal{L}^{\mathsf{same}} = \left\{(\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, c_1, c_2^{\mathsf{CPA}}) : \exists (m, r_1, r_2) \text{ s.t. } c_1 = \mathsf{Enc}_{\mathsf{pk}}(m; r_1) \text{ and } c_2^{\mathsf{CPA}} = \mathsf{Enc}^{\mathsf{CPA}}_{\mathsf{pk}^{\mathsf{CPA}}}(m; r_2)\right\}.$$

  For our application, multi-bit messages $m$ will be encrypted bit-by-bit, and to correspond to a *vector* of $|m|$ individual NIZK proofs. This is done to ensure the total proof size remains $|m| \cdot \mathsf{poly}(k)$. To simplify notation, however, we suppress the vector notation and denote the corresponding collection of proofs $\pi_1, ..., \pi_{|m|}$ simply by $\pi$.
- A multisignature scheme $\mathsf{MultiSig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{V}, \mathsf{Combine}, \mathsf{MultiVerify})$ with signature size $\mathsf{poly}(k)$ and signing messages of arbitrary length. (The later requirement can be obtained by signing the hash of the message to be signed using a CRHF.)
- A verifiable secret sharing protocol $\mathsf{VSS}$ which can be run on any subset of $n' \leq n$ parties, with complexity $\mathsf{poly}(n')$ and which achieves *perfect security* when less than one third of the $n'$ parties are corrupted. An example is the VSS protocol from BGW [BGW88].
- A multi-party computation protocol $\mathsf{MPC}$ to evaluate a circuit $\mathcal{C}$ which can be run on any subset of $n' \leq n$ parties, with complexity $|\mathcal{C}| \cdot \mathsf{poly}(n')$ in $|\mathcal{C}|$ rounds and which achieves *perfect security* when less than one third of the $n'$ parties are corrupted. An example is the MPC protocol from BGW [BGW88]. In particular, we may need some isolated components or special cases of such a protocol, all achieving perfect security:[3]

[3]We note that perfect security is never necessary (our protocol being based on computational assumptions), but will allow us an easier reasoning when executing many instances of these protocols concurrently rather than dealing with more complex concurrently-secure multi-party protocols.

1. A verifiable secret sharing protocol VSS which allows any of the $n$ players to act as a dealer and to distribute shares to any subset of $n' \leq n$ parties (with the dealer possibly being external to the $n'$ parties), with message and communication complexity $\mathsf{poly}(n')$. Moreover, an associated reconstruction protocols allows later robust reconstruction of the shared value. The protocol tolerates $t < n'/3$ of the parties (and possibly the dealer) being corrupted.
2. A broadcast protocol BC which can be initiated by any of the $n$ parties to broadcast a value to a subset of $n' \leq n$ parties (with the dealer possibly being external to the $n'$ parties), with message and communication complexity $\mathsf{poly}(n')$. The protocol tolerates $t < n'/3$ of the parties (and possibly the dealer) being corrupted.
3. A coin tossing protocol CoinToss which be run by any subset of $n'$ players of $n' \leq n$ parties to agree on a random uniform value of length polynomial in $k$, with message and communication complexity $\mathsf{poly}(k, n')$, and which tolerates $t' < n'/3$ of these parties being corrupted.

- A pseudorandom function family $\mathbb{F} = \{F_s : [n] \to S(n, \log^2(n))\}_{s \in \{0,1\}^k}$, where $S(n, n')$ denotes the set of subsets of size $n'$ of $[n]$. Moreover, we also need a pseudorandom generator $\mathsf{prg} : \{0,1\}^k \to \{0,1\}^{\ell_{\max}}$, where $\ell_{\max}$ is an upper bound on the randomness needed by SLA.Sel and SLA.Exec.

- A switching network $SN$ with $n$ wires and depth $d = \mathsf{polylog}(n) \cdot \log^2(k)$ given by Theorem 5.2.11, ensuring nearly-uniform shuffling.

**Building a communication tree.** We first review the protocol by King et al. [KSSV06], which is jointly run by the $n$ parties to create a "communication tree" which acts as the backbone of our protocol. A communication tree consists of a tree $T = (V, E)$ and an assignment $S$ associating with each tree node $\mathsf{v} \in V$ a subset $S(\mathsf{v}) \subseteq [n]$ of the $n$ parties. Note that for any two nodes $\mathsf{v}$ and $\mathsf{v}'$, the subsets $S(\mathsf{v})$ and $S(\mathsf{v}')$ may well *not* be disjoint. A *protocol to create a communication tree* $T = (V, E)$ has all parties starting with no input, and at the end of the protocol each party $P_i$ holds a function $S_i : V \to \mathcal{P}([n])$, where $\mathcal{P}([n])$ is the set of subsets of $[n]$, such that at the end of the execution there exists an assignment $S : V \to \mathcal{P}([n])$ with the following properties:

1. For all parties $P_i$ and $\mathsf{v} \in V$, if $i \in S(\mathsf{v})$, then $S_i(\mathsf{v}) = S(\mathsf{v})$ – if $S$ assigns a party $P_i$ to some node $\mathsf{v}$, then $P_i$ actually knows the parties assigned to $\mathsf{v}$.

2. The value of $S(\mathsf{l})$ for all leaves $\mathsf{l} \in V$ is known a priori to everyone and does not depend on the execution of the protocol. In particular $S_i(\mathsf{l}) = S(\mathsf{l})$ for all $i \in [n]$.

Moreover, for the assignment $S$, given a set $A \subseteq [n]$ of corrupted parties (where $|A| \leq (\frac{1}{3} - \epsilon)n$), we say that a node $\mathsf{v} \in V$ is *good* if at least a 2/3-fraction of the parties in $S(\mathsf{v})$ are honest, and it is *bad* otherwise. We say that a node has a *good path to the root* if all nodes on the path from the node to the root are good. We say that a party $P_i$ is *lucky* if a majority of the leaves $\mathsf{l}$ he is assigned to are good, and he is *unlucky* otherwise. We are going to use a protocol BuildTree satisfying the following theroem – which is a stronger version of the original theorem from [KSSV06] and was given in [KS11].

**Theorem 5.4.2.** *[KSSV06] Suppose there are $n$ parties, of which $t < n/3$ are corrupted. Then, there is an algorithm* BuildTree *operating in* $\mathsf{polylog}(n)$ *rounds, and in which each good processor sends and processes* $\mathsf{polylog}(n)$ *bits, that creates a communication tree with the following properties with overwhelming probability:*

1. *The tree has height $\ell^* \in O(\log n / \log\log n)$. Each node $\mathsf{v}$ from level $\ell > 0$ has $\log n$ nodes from layer $\ell - 1$ as its children.*
2. *Each leaf node of the tree is assigned a set of $\log^5 n$ parties.*
3. *Each internal node of the tree is assigned a set of $\log^3 n$ parties.*
4. *Each party is assigned to $O(\log^4 n)$ nodes at each level.*
5. *All but a $3/\log n$ fraction of the leaf nodes have a good path up to the root node.*
6. *Suppose there is a good path from $\mathsf{v} \in V$ to the root, and let $\mathsf{v}'$ be the parent of $\mathsf{v}$. Then all honest parties in $S(\mathsf{v})$ know all parties in $S(\mathsf{v}')$, and all honest parties in $S(\mathsf{v}')$ know all parties in $S(\mathsf{v})$.*

An important observation is that the fact that $1 - o(1)$ fraction of the leaves are on good paths to the root implies that a $1 - o(1)$ fraction of the parties is lucky, too: This can be proven as follows. Namely, assume this is not true, i.e., there exists a set $X \subseteq [n]$ of $\zeta \cdot n$ parties, where $\zeta = \frac{1}{\log^6(n)}$, for which a majority of the leaves they are contained in has no good path up to the root. Note that the set $Z$ of leaves that contain parties in $X$ has size $|Z| \geq |X|/\log^5(n)$, since every leaf contains $\log^5 n$ parties. However, we know that at least half of these have no good path to the root, i.e., there exists at least $|Z|/2 \geq |X|/(2 \cdot \log^5(n))$ leaves with no good paths to the root. However, as there are $\Theta(n/\log^{10}(n))$ leaves in the communication three, this means that a fraction larger than $3/\log(n)$ leaves has no good path to the root, contradicting the property enforced by the communication tree.

**Sending a message down the tree.** The following protocol allows parties assigned to the root of the communication tree to send messages to the lucky players by exploiting the communication tree built by BuildTree. By inspection, it is not hard to verify that its complexity is such that within $\mathsf{polylog}(n)$ rounds, each party sends $\mathsf{polylog}(n)$ messages of size at most $|m|$, which is the length of the inputs by the parties assigned to the root by $S$.

**Protocol** SendDownTree**:**
Each party $P_i$ holds an input $m_i$, and each party $P_j$ has a final output $m'_j$, as well as $\{m'_{\mathsf{v},i}\}$ for all $\mathsf{v} \in V$.

1. Each party $P_i$ who believes to be in the root sends $m_i$ to all parties he believes to be in the children nodes on level $\ell^* - 1$
2. For each level $\ell = \ell^* - 1, ..., 1$ down the tree, do the following in parallel for each vertex $\mathsf{v}$ on level $\ell$ and on each party $P_j$ who believes to be assigned to $\mathsf{v}$:
   (a) Let $m$ be the message received by party $P_j$ (in level $\ell$) by a *majority* of the parties assigned to the parent node of $\mathsf{v}$ in level $\ell + 1$.
   (b) $P_j$ sets $m'_{\mathsf{v},j} = m$
   (c) If $\ell \geq 2$, $P_j$ sends $m$ to all parties he believes to be assigned to the children nodes of $\mathsf{v}$ on level $\ell - 1$.
3. For each party $P_i$, define the output $m'_i = \text{majority}\{m'_{\mathsf{l},i} \ : \ \mathsf{l} \in V \text{ is a leaf}\}$ (And let $m'_i = \perp$ if no well-defined majority exists.)

The following lemma states that Protocol SendDownTree satisfies our requirement, and its proof follows by easy induction using the fact that every node on a good path to the root will have a majority of parties sending the same message to parties assigned to the children nodes, and parties assigned to a node $\mathsf{v}$ on a good path know exactly the composition of the set of parties assigned to the parent node $\mathsf{v}'$ on the good path (and conversely, parties assigned to $\mathsf{v}'$ know all parties assigned to $\mathsf{v}$).

**Lemma 5.4.3.** *If more than* $\log^3(n)/2$ *honest players assigned by* $S$ *to the root start protocol* SendDownTree *with the same input* $m$*, then for every node* $\mathsf{v}$ *with a good path to the root, every honest player* $P_i$ *assigned to* $\mathsf{v}$ *has* $m_{\mathsf{v},i} = m$*. In particular, all lucky players output* $m$*.*

**Sending messages up the communication tree and certificates.** The goal is now to design a protocol SendCertUpTree ensuring that if all honest parties associated with the root have executed SendDownTree with the same input $m$, then each party associated with the root terminates SendCertUpTree outputting a valid *certificate* for $m$, i.e., a proof which certifies that this is the value the honest parties in the root have agreed upon. As motivated in the high-level overview, we will build such a proof using *multi-signatures.* To this end, let MultiSig $=$ (Gen, Sign, V, Combine, MultiVerify) be a multi-signature scheme as defined in Section 5.2. We assume that the setup contains $n \cdot \mathsf{polylog}(n)$ independent verification keys for the scheme, where each party $P_i$ and each leaf $\mathsf{l}$ associated with $P_i$ corresponds to a verification key $\mathsf{vk}_{\mathsf{l},i}$ – the corresponding signing key $\mathsf{sk}_{\mathsf{l},i}$ being held by $P_i$.

**Definition 5.4.4 (Certificates).** A valid *certificate for a message* $m$ consists of a pair $(\vec{1}_S, \sigma)$, where $\vec{1}_S$ is the characteristic vector of a set $S$ consisting of pairs $(\mathsf{l}, i)$, where $\mathsf{l}$ is a leaf of $T$ and $i \in [n]$, and $\sigma$ is a signature for MultiSig, such that: (1) Pairs corresponding to at least $n/2$ distinct indices $i$ are contained in $S$. (2) The signature $\sigma$ is such that $\mathsf{MultiVerify}(\{\mathsf{vk}_{\mathsf{l},i}\}_{(\mathsf{l},i)\in S}, m, \sigma)$ outputs 1.

A formal description of the protocol follows in the Figure SendCertUpTree.

**Protocol** SendCertUpTree:
Each party $P_i$ holds an input $m_{\mathsf{v},i}$ for each node $\mathsf{v} \in V$, and holds a signing key $\mathsf{sk}_{\mathsf{l},i}$ for each leaf node $\mathsf{l} \in L$ he has been assigned to. For $S \subseteq L \times [n]$, $\bar{1}_S$ denotes the binary vector $\vec{x}$ with $x_{(\mathsf{l},i)} = 1$ iff $(\mathsf{l}, i) \in S$, and $x_{(\mathsf{l},i)} = 0$ else.

1. For all leaves $\mathsf{l}$ of the communication tree do the following in parallel: All parties $P_i$ who are associated with $\mathsf{l}$ compute $\sigma_{\mathsf{l},i} \leftarrow \mathsf{Sign}_{\mathsf{sk}_{\mathsf{l},i}}(m_{\mathsf{l},i})$, and send $(m_i, \sigma_{\mathsf{l},i}, \bar{1}_{\{(\mathsf{l},i)\}})$ to all parties $P_j$ assigned to the parent node of $\mathsf{l}$.

2. For each level $\ell = 2, ..., \ell^* - 1$ of the communication tree, do the following in parallel for each node $\mathsf{v}$ on level $\ell$:

   (a) For each player $P_j$ who believes to be assigned to $\mathsf{v}$, let $\{(m_t, \bar{\sigma}_t, \bar{1}_{S_t})\}_{t \in I}$ be the collection of messages received by party $P_j$ in level $\ell$ by parties in the children nodes of $\mathsf{v}$ in level $\ell - 1$. Delete from the list all triples for which $m_t \neq m_{\mathsf{v},i}$.

   (b) For each remaining pair $(\bar{\sigma}_t, \bar{1}_{S_t})$ that appeared with $m_t = m_{\mathsf{v},j}$, party $P_j$ verifies the multisignature $\bar{\sigma}_t$ with respect to the collection of verification keys as indicated by $\bar{1}_{S_t}$:

   $$\mathsf{verify}_t \leftarrow \mathsf{MultiVerify}(\{\mathsf{vk}_{(\mathsf{l}',i')}\}_{(\mathsf{l}',i') \in S_t}, m_{\mathsf{v},j}, \bar{\sigma}_t).$$

   Party $P_j$ deletes from the list all messages for which $\mathsf{verify}_t = 0$.

   (c) Party $P_j$ combines all verified multisignatures into a single multisignature, as follows. Let $T$ be the indices of the remaining verified multisignatures, and let $S = \bigcup_{t \in T} S_t$ be the set of all leaf-party pairs whose signature appears in exactly one of these multisignatures. Then $P_j$ computes

   $$\bar{\sigma}' \leftarrow \mathsf{Combine}(\{\mathsf{vk}_{\mathsf{l}',i'}, \sigma_{\mathsf{l}',i'}\}_{(\mathsf{l}',i') \in S}, m_{\mathsf{v},j}).$$

   (d) If $\ell < \ell^*$, $P_j$ broadcasts the triple $(m_{\mathsf{v},j}, \bar{\sigma}', \bar{1}_S)$ to all parties assigned to the parent node in level $\ell + 1$. Otherwise, $P_j$ outputs $(m_{\mathsf{v},j}, \bar{\sigma}', \bar{1}_S)$.

By inspection, we observe that the protocol proceeds in $\ell^* = \mathsf{polylog}(n)$ rounds. Since signatures have size $\mathsf{poly}(k)$, each party sends $\mathsf{polylog}(n)$ messages of size each $n \cdot \mathsf{polylog}(n) + \mathsf{poly}(k) + |m|$, where $|m|$ is the maximal length of the inputs of the parties. We are going to prove the following lemma about the combination of protocols SendDownTree and SendCertUpTree.

**Lemma 5.4.5.** *Assume that $n$ parties execute* SendDownTree *following by* SendCertUpTree. *Then, if all honest parties assigned to the root of the tree agree on their input $m$, at the end of* SendCertUpTree *they all output the same valid certificate for the message $m$.*

*Proof.* (Sketch) Clearly, honest players in the committee can only output a valid certificate for $m$. It is easy to verify that sufficiently many signatures are going to be aggregated on the way up, since every lucky party (a $1 - o(1)$ fraction of the $n$ parties) is going to forward at least one signature on $m$ to be aggregated and which will be forwarded to the players associated with the root. □

**Setting up committees.** With the tools at hand from the previous paragraphs, combined with the available coin tossing protocol, we can now easily obtain a protocol which has all parties achieve certified almost everywhere agreement on the supreme committee $C$ and on committees $C_1, \dots, C_n$, where $C_i = f_s(i)$ for a random element from a PRF family $\mathbb{F} = \{f_s\}$.

***Certified* Almost Everywhere Election:**
Election of supreme committee $C$ and individual committees $C_1, \dots, C_n$

1. Run protocol BuildTree for setting up the communication tree $T = (V, E)$.
2. The parties run protocol SendDownTree where each party $P_i$ who believes to be in the root has input $C^{(i)} = S_i(\mathsf{v}_{root})$, where $\mathsf{v}_{root}$ is the root of $T$.
3. The parties run protocol SendCertUpTree and all honest parties associated with the root agree on the output $(C, \vec{1}_S, \sigma)$ where $C = S(\mathsf{v}_{root})$.
4. The parties in the supreme committee $C$ run a coin tossing protocol amongst themselves to generate a random seed $s$ for the pseudorandom function family $\mathbb{F}$. They run protocol SendDownTree with input $s$.
5. The parties run protocol SendCertUpTree with the outputs of the previous step, and the parties in the supreme committee output $(s, \vec{1}_{S'}, \sigma')$
6. Parties in $C$ send $(C, s, \vec{1}_S, \vec{1}_{S'}, \sigma, \sigma')$ to the lucky parties using protocol SendDownTree.
7. If $P_i$ receives a valid $(C, s, \vec{1}_S, \vec{1}_{S'}, \sigma, \sigma')$ stores the composition of $C$ and sets $C_i = f_s(i)$ for all $i \in [n]$. If signature verification fails, then output $\perp$.

By inspection, the protocol achieves has round complexity $\mathsf{polylog}(n)$, and each party sends at most $\mathsf{poly}(\log(n), k) + n \cdot \mathsf{polylog}(n)$ bits. The following lemma is straightforward given previous Lemmas 5.4.5 and 5.4.3.

**Lemma 5.4.6 (Almost everywhere certified agreement).** *If the $n$ parties run the protocol* ***Certified* Almost Everywhere Election,** *a fraction $1 - o(1)$ of the honest parties terminate by outputting $C$ equal to the set of parties assigned to the root, and committees $C_1, \dots, C_n$ such that $C_i = f_s(i)$ for a random seed $s$ and all $i \in [n]$.*

The last step now make sure that we can go from almost everywhere certified agreement to everywhere agreement.

**Protocol Certified Almost Everywhere to Certified Everywhere**
Assumed: a.e. agreement on $C, \{C_i\}_{i\in[n]}$, with certification

1. Each party $P_i$ who possesses a valid $(C, s, \vec{1}_S, \vec{1}_{S'}, \sigma, \sigma') \neq \perp$ sends his triple to each party in his personal committee $C_i$: that is, to all parties $P_j$ for which $j \in f_s(i)$.
2. Each party $P_j$ who does *not* currently possess a valid $(C, s, \vec{1}_S, \vec{1}_{S'}, \sigma, \sigma') \neq \perp$ listens for incoming messages. Each received message that is not properly certified is ignored. Party $P_i$ adopts the first properly certified received $(C, s)$ as his choice for $C$ as well as the seed to compute $C_i = f_s(i)$.

By using the pseudorandomness of $\mathbb{F}$, the following lemma finally shows that $C, C_1, \ldots, C_n$.

**Lemma 5.4.7.** *If an execution of Protocol* **Certified Almost Everywhere to Certified Everywhere** *follows the execution of Protocol* ***Certified*** **Almost Everywhere Election**, *then all honest parties terminate with the same outputs $C, C_1, \ldots, C_n$, where all committees contain at least a fraction $2/3$ of honest players, except with negligible probability.*

*Proof.* Note that protocol ***Certified*** **Almost Everywhere Election** will have the lucky honest players terminate with a certified common output $(C, s)$, and forward the output with a certificate to all members of their personal committees. Note that for sufficiently large $n$, there are at least $n/2$ lucky players. Let $i_1, \ldots, i_{n/2}$ be the indices of $n/2$ lucky players. The probability that some $i$ does not appear in any of $C_{i_1}, \ldots, C_{i_{n/2}}$ is upper bounded by

$$\Pr[i \notin C_{i_1}, \ldots, C_{i_{n/2}}] \leq \left(1 - \frac{1}{n}\right)^{\log^2(n)\cdot n/2} + \nu \leq e^{-\Omega(\log^2(n))} + \nu\ ,$$

where $\nu$ is some negligible additive term due to the choice of the $C_i$'s being only pseudorandom. Consequently, $\Pr[\exists i : i \notin C_{i_1}, \ldots, C_{i_{n/2}}]$ is also negligible by a union bound, which implies agreement. The fact that $C_1, \ldots, C_n$ contains a $2/3$ fraction of honest players, except with negligible probability, follows easily from their pseudorandomness, as well as a simple Chernoff bound combined with the fact that $(2/3 + \epsilon)n$ parties are honest. The fact that $C$ has also a $2/3$ fraction of honest players follows from the fact that the root is a good node in the communication tree. □

The following lemma follows directly from the previous lemmas.

**Lemma 5.4.8.** *If a majority of parties in the supreme committee $C$ initiate* ComBroadcast$(m)$ *on the* same *message $m$, then at the conclusion of the protocol execution, all parties $P_i \in [n]$ receive $m$, together with certification of correctness.*

After the committee infrastructure $C, \{C_i\}$ has been built and agreed upon, the parties are able to proceed to the second main phase of the protocol, which we now describe.

**Committee Broadcast Protocol:** $\mathsf{ComBroadcast}(m)$
Inputs: Each party holds a (common) PRF seed $s$, which defines the individual committees $C_j := f_s(j)$.

1. Each party $P_i \in C$ in the committee initiates $\mathsf{SendDownTree}(m)$ to send the message $m$ down the communication tree.
2. Every party $P_j$ initiates $\mathsf{SendCertUpTree}(m)$, where $m$ is the message received by $P_j$ at the conclusion of the previous step.
3. Each party $P_i \in C$ in the committee initiates $\mathsf{SendDownTree}((m, \bar{\sigma}, \bar{1}_S))$, where $(\bar{\sigma}, \bar{1}_S)$ is a certification of $m$ learned at the conclusion of the previous step.
4. Every party $P_j$ holds some alleged version $(m_j, \bar{\sigma}_j, 1_{S_j})$. $P_j$ sends the triple $(m_j, \bar{\sigma}_j, \bar{1}_{S_j})$ to every party in the set defined by $f_s(j)$ (i.e., the committee $C_j$).
5. Every party $P_i$ processes only messages received by parties $P_j$ for which $i \in f_s(j)$. From these messages $(m_j, \bar{\sigma}_j, \bar{1}_{S_j})$, $P_i$ takes his own output $(m_i, \bar{\sigma}_i, \bar{1}_{S_i})$ to be the triple that appears most frequently.

**Setting up the encryption schemes.** The first step is to generate key information for the two different public-key encryption schemes. This key sampling is done by the supreme committee $C$ via an MPC execution, as described in Figure 5.1. The parties in $C$ will all learn the public keys $\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}$ for the two schemes, and will each be given a *secret share* of the decryption key $\mathsf{sk}$ for the FHE scheme. Looking ahead, this will allow the committee members to *collectively* decrypt ciphertexts, but prevents any subset of malicious parties from breaking the security of the FHE scheme. The members of $C$ then communicate the public keys $\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}$ to *all* parties, by using the $\mathsf{ComBroadcast}$ protocol.

**Committing to inputs.** In this stage, each party $P_i$ to commits his secret input $x_i$ to the members of his personal committee $C_i$. This is done by VSSing a collection of information among the parties in $C_i$:

- An encryption of the input $x_i$ under the FHE scheme,
- A NIZK proof that this encryption was performed with good randomness (guaranteeing that correctness of homomorphic evaluation will hold),
- A second encryption of $x_i$, under a standard CPA-secure scheme, and
- A NIZK proof that the two provided ciphertexts encrypt the same message.

In addition, during this phase the parties in $C_i$ collectively generate an encryption of the index $i$ under the FHE. This will be used to keep track of which input is currently stored by the committee (and will be updated when the committees swap inputs in the following section).

**Encryption Key Generation:**
The parties of $C$ run a MPC evaluating the following (randomized) functionality:

Input: $\emptyset$

Compute:

1. Sample a key pair $(\mathsf{pk}^{\mathsf{CPA}}, \mathsf{sk}^{\mathsf{CPA}}) \leftarrow \mathsf{Gen}^{\mathsf{CPA}}(1^k)$ for the standard CPA-secure encryption scheme.
2. Sample a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$ for the FHE encryption scheme
3. Secret share the FHE secret key $(\mathsf{sk}_1, ..., \mathsf{sk}_{|C|}) \leftarrow \mathsf{Share}(\mathsf{sk})$.

Output: All parties (in $C$): $\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}$. Party $P_i \in C$: $\mathsf{sk}_i$.

**Figure 5.1:** Initial key generation procedure run by the supreme committee $C$ after it is elected. The public keys $(\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}})$ will be communicated to all parties $\mathcal{P}$ via the communication tree.

The input commitment protocol is run in parallel between each party $P_i$ and his committee $C_i$. In particular, *all* parties $P_i$ execute the sharing phase of the VSS simultaneously, so that no execution enters the reconstruction phase until the sharing phase is completed for all parties. At the conclusion of the procedure for party $P_i$, all parties in committee $C_i$ hold an FHE encryption of $P_i$'s index $i$ and input $x_i$ (or a default ciphertext if $P_i$ did not follow the protocol faithfully). From this point forward, the committee $C_i$ will take the role of storing and supplying $P_i$'s input whenever necessary.

A complete description of the input commitment procedure is given in Figure 5.2.

**Generating the input shuffle permutation.** Next, the shuffle generation is executed by parties in the supreme committee $C$ to generate an encrypted, (nearly) random permutation on $[n]$, which will be used to implement and oblivious shuffle of the inputs held by the committees $C_i$. The permutation is represented in the form of a sequence of swapping bits, corresponding to a fixed switching network $SN$ (see Section 5.2.6). Each swapping bit is generated by $C$ via a standard coin tossing procedure: by sampling and VSSing random bits, and then each taking the exclusive-or of the reconstructed values. However, in our case, this procedure is performed *homomorphically*, under the layer of the fully homomorphic encryption. A formal description of the shuffle generation protocol is given in Figure 5.3.

The resulting swap-bit ciphertexts are then communicated from $C$ to *all* the parties in the network, via the protocol $\mathsf{ComBroadcast}$.

**Shuffling inputs.** Once the the encrypted permutation is generated and communicated to all parties, the committees $C_i$ *obliviously* implement the dictated data shuffle. This is done by pairs of committees $C_i, C_j$ iteratively executing the pairwise committee swap protocol, described in Figure 5.4.

The swap protocol takes place for each swap bit in the switching network $SN$. Each layer of the

**Input Commitment:**
Input: Each $P_i$: secret input $x_i$

1. Performed by $P_i$:
   (a) Sample good encryption randomness $r_i \leftarrow R$ via rejection sampling.
   (b) Encrypt input $x_i$ under the FHE scheme, using $r_i$: $\hat{x}_i = \mathsf{Enc}_{\mathsf{pk}}(x_i; r_i)$.
   (c) Generate a proof of correctness $\pi_i \leftarrow \mathsf{P}(\mathsf{crs}, (\mathsf{pk}, \hat{x}_i), (x_i, r_i))$.
   (d) Sample a *second* encryption of the input $x_i$, under the standard CPA-secure encryption scheme: $\hat{x}_i^{\mathsf{CPA}} \leftarrow \mathsf{Enc}^{\mathsf{CPA}}_{\mathsf{pk}^{\mathsf{CPA}}}(x_i; r_i^{\mathsf{CPA}})$.
   (e) Generate a proof (with respect to language $\mathcal{L}^{\mathsf{same}}$) that the two ciphertexts are consistent:
   $$\pi_i^{\mathsf{CPA}} \leftarrow \mathsf{P}'(\mathsf{crs}^{\mathsf{same}}, (\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \hat{x}_i, \hat{x}_i^{\mathsf{CPA}}), (x_i, r_i^{\mathsf{CPA}})).$$
   (f) Execute VSS sharing phase as dealer, to share input $(\hat{x}_i, \pi_i, \hat{x}_i^{\mathsf{CPA}}, \pi_i^{\mathsf{CPA}})$ among the parties in $C_i$.

2. Performed by each party $P_j \in C_i$:
   (a) Execute the reconstruction phase of the VSS, together with the other parties in $C_i$, to jointly reconstruct the value shared by $P_i$ in Step (f) above.
   Let $(\hat{x}_i, \pi_i, \hat{x}_i^{\mathsf{CPA}}, \pi_i^{\mathsf{CPA}})$ be the value reconstructed by party $P_j$.
   (b) If the proofs $\pi_i, \pi_i^{\mathsf{CPA}}$ properly verify; i.e., if
   $$1 = \mathsf{V}(\mathsf{crs}, (\mathsf{pk}, \hat{x}_j), \pi_j) \quad \text{and}$$
   $$1 = \mathsf{V}'(\mathsf{crs}^{\mathsf{same}}, (\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \hat{x}_j, \hat{x}_j^{\mathsf{CPA}}), \pi_j^{\mathsf{CPA}})),$$
   then party $P_j$ sets his belief of $P_i$'s encrypted input (denoted $\hat{x}_i^j$) to be the received ciphertext $\hat{x}_i$.
   (c) Otherwise, if either proof does *not* properly verify, party $P_j$ initializes $\hat{x}_i^j$ to a default public cipertext value $\hat{0}$, encrypting 0.
   (d) $P_j$ executes an MPC together with the other parties of $C_i$ to compute a (good) encryption of the index $i$. That is, they evaluate the (randomized) functionality $\mathcal{F}_{\mathsf{Enc}(i)}$ that takes no inputs, samples $r \leftarrow R$ via rejection sampling, encrypts $\hat{i} = \mathsf{Enc}_{\mathsf{pk}}(i; r)$, and outputs $\hat{i}$ to all parties in $C_i$.

**Figure 5.2:** The Input Commitment procedure, in which each party $P_i$ commits his secret input $x_i$ to his personal committee, $C_i$.

**Shuffle Generation:**
Let $[q] \equiv [d] \times [n/2]$ index the total number of swaps in the switching network.

1. Each party $P_i \in C$ samples $q$ random bits and generates pairs of encryptions, together with proofs of correctness, for each. That is, for each $\ell \in [q]$,
   (a) Sample $b^i_\ell \leftarrow \{0,1\}$, $r_\ell \leftarrow R$ (via rejection sampling).
   (b) Compute $\hat{b}^i_\ell = \mathsf{Enc}_{\mathsf{pk}}(b^i_\ell; r_\ell)$ and proof $\pi^i_\ell \leftarrow \mathsf{P}(\mathsf{crs}, (\mathsf{pk}, \hat{b}^i_\ell)(b^i_\ell, r_\ell))$ that $\hat{b}^i_\ell$ is good.
   (c) Sample a *second* encryption $(\hat{b}^i_\ell)^{\mathsf{CPA}} = \mathsf{Enc}^{\mathsf{CPA}}_{\mathsf{pk}^{\mathsf{CPA}}}(b^i_\ell; r^{\mathsf{CPA}}_\ell)$, and prove that the two ciphertexts are consistent:
   $$(\pi_\ell)^{\mathsf{CPA}} \leftarrow \mathsf{P}'\left(\mathsf{crs}^{\mathsf{same}}, (\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \hat{b}^i_\ell, (\hat{b}^i_\ell)^{\mathsf{CPA}}), (b^i_\ell, r^i_\ell, (r^i_\ell)^{\mathsf{CPA}})\right)$$
2. In parallel, each party $P_i \in C$ acts as dealer in a VSS, to secret share the collection of ciphertext-proof pairs
   $$\left\{(\hat{b}^i_\ell, \pi^i_\ell), \left((\hat{b}^i_\ell)^{\mathsf{CPA}}, (\pi^i_\ell)^{\mathsf{CPA}}\right)\right\}_{\ell \in [q]}$$
   to all other parties in $C$.
3. Each party $P_i \in C$ executes the reconstruction phase of the VSS, together with the other parties in $C$, to jointly reconstruct all values shared in the previous step.
4. For each $j \in C$ let $\{(\hat{b}^j_\ell, \pi^j_\ell), ((\hat{b}^j_\ell)^{\mathsf{CPA}}, (\pi^j_\ell)^{\mathsf{CPA}})\}_{\ell \in [q]}$ denote the ciphertext-proof pairs reconstructed by a party $P_i$ *from* party $P_j$. Locally, $P_i$ performs the following steps for each bit position $\ell \in [q]$:
   (a) Initialize $\mathsf{good}_\ell = \emptyset$.
   (b) For each $j \in C$, if party $P_j$'s proofs verify (i.e., if $1 = \mathsf{V}(\mathsf{crs}, (\mathsf{pk}, \hat{b}^j_\ell), \pi^j_\ell)$, and $1 = \mathsf{V}'(\mathsf{crs}^{\mathsf{same}}, (\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \hat{x}_j, \hat{x}^{\mathsf{CPA}}_j), \pi^{\mathsf{CPA}}_j)))$, then add $\mathsf{good}_\ell \leftarrow \mathsf{good}_\ell \cup \{j\}$.
   (c) Take $\hat{B}_\ell = \mathsf{Eval}(\{\hat{b}^j_\ell\}_{j \in \mathsf{good}_\ell}, \bigoplus)$ to be the homomorphically evaluated xor of all good ciphertexts.

**Figure 5.3:** Shuffle generation procedure, executed by parties in the supreme committee $C$ to generate an *encrypted* random permutation on $[n]$, in the form of a sequence of swapping bits, corresponding to a fixed switching network $SN$.

switching network defines a collection of disjoint pairs of the indices in $[n]$, and the corresponding pairs of committees $D$, $E$ each run the swapping procedure in parallel. The swapping procedure is simply composed of a homomorphic evaluation of the swap-or-not function dictated by the corresponding encrypted swap bit $\hat{b}$ generated by $C$ during the Shuffle Generation procedure (and then communicated to all parties). Each party in committee $D$ (respectively, $E$) enters the swap protocol with the encrypted input of some party $P_i$ and corresponding encrypted index $i$. The party exits either with a ciphertext of the same value, or with a ciphertext corresponding to the input of a different party $P_j$ that was held by the other committee $E$ (resp., $E$). At each stage of the homomorphic evaluation in which a party receives a collection of ciphertexts from the other committee, he takes only the ciphertext that was sent by a majority of parties in the committee (to weed out a potential minority of malicious ciphertexts).

After the swapping procedure is performed for every swap bit in the switching network, the input values that are held by the committees $C_i$ will be permuted as dictated by the secret permutation.

**Selecting the (permuted) query indices.** Once the committees have shuffled their stored inputs as per the selected permutation, the supreme committee $C$ runs a protocol to select a subset of parties $I$ whose inputs will be used in the sublinear algorithm evaluation. The output of the selection computation will be the *permuted* set $\rho(I)$, indicating the committees who currently hold the relevant parties' inputs $\{x_i\}_{i \in I}$.

This procedure takes place in three steps (see Figure 5.5). First, the parties of $C$ collectively generate an encryption of randomness seed, by executing an MPC. Then, each party homomorphically evaluates a pseudorandom generator on seed, and uses this as the randomness with which to homomorphically evaluate the query selection algorithm SLA.Sel (represented as a circuit). Finally, the committee members collectively decrypt this evaluated ciphertext via an MPC, taking as additional input the parties' secret shares $\mathsf{sk}_i$ of the FHE decryption key.

**Evaluating the sublinear algorithm** As the final step, the Evaluation protocol is run by the supreme committee $C$ to evaluate the sublinear algorithm on the collection of queried inputs (see Figure 5.6).

The evaluation protocol is executed once the parties in $C$ receive the input and index ciphertexts from a majority of parties in each of the queried committees. The members of $C$ first collectively generate an encryption of a random value seed that will be used as the seed to a pseudrorandom generator, and then utilized as the randomness for the sublinear algorithm execution. After this ciphertext $\widehat{\mathsf{seed}}'$ is generated, the parties of $C$ homomorphically evaluate the algorithm SLA.Exec (in the form of a circuit) on the set of encrypted inputs received from the queried committees. They then collectively decrypt the output by running an MPC that takes as input the evaluated ciphertext, together with their secret shares $\mathsf{sk}_i$ of the FHE decryption key.

Directly after the evaluation protocol, the committee members communicate the resulting output to all parties, and the protocol concludes.

**Committee Swap Protocol:**
For pair of committees $D$ and $E$. Inputs:
Each party in $D$: swap bit ciphertext $\hat{b}$, current (index, input) ciphertexts $(\hat{p}^i, \hat{x}_p^i)$
Each party in $E$: swap bit ciphertext $\hat{b}$, current (index, input) ciphertexts $(\hat{q}^j, \hat{x}_q^j)$

1. Each party $P_i \in D$ broadcasts his pair of ciphertexts $(\hat{p}^i, \hat{x}_p^i)$ to all parties in $D \cup E$.
   Each party $P_j \in E$ broadcasts his pair of ciphertexts $(\hat{q}^j, \hat{x}_q^j)$ to all parties in $D \cup E$.
2. Locally, each party performs the following steps.
   (a) Let $\hat{p} = \mathsf{maj}_{i \in D}\{\hat{p}^i\}$ and $\hat{q} = \mathsf{maj}_{j \in E}\{\hat{q}^j\}$ be the most frequent *index* ciphertexts.
   (b) Let $\hat{x}_p = \mathsf{maj}_{i \in D}\{\hat{x}_p^i\}$ and $\hat{x}_q = \mathsf{maj}_{j \in E}\{\hat{x}_q^j\}$ be the most frequent *input* ciphertexts.
   (c) Homomorphically evaluate (locally) the swap functionality on the resulting index ciphertexts $\hat{p}, \hat{q}$ and input ciphertexts $\hat{x}_p, \hat{x}_q$, as dictated by the ciphertext swap bit $\hat{b}$, as follows.
   For each value $a = 0, 1$, define the swap functionality $\mathsf{swap}_a$ as:
   $$\mathsf{swap}_a(y, z, b) = \begin{cases} y & \text{if } a \oplus b = 0 \\ z & \text{if } a \oplus b = 1 \end{cases}.$$
   Then each party $P_i \in D$ reassigns
   $$\hat{p}^i = \mathsf{Eval}\left((\hat{p}, \hat{q}, \hat{b}), \mathsf{swap}_0\right), \quad \hat{x}_p^i = \mathsf{Eval}\left((\hat{x}_p, \hat{x}_q, \hat{b}), \mathsf{swap}_0\right),$$
   and each party $P_j \in E$ reassigns
   $$\hat{q}^j = \mathsf{Eval}\left((\hat{p}, \hat{q}, \hat{b}), \mathsf{swap}_1\right), \quad \hat{x}_q^j = \mathsf{Eval}\left((\hat{x}_p, \hat{x}_q, \hat{b}), \mathsf{swap}_1\right).$$

**Figure 5.4:** Swapping procedure executed by pairs of committees $D, E$ for each swap bit in the switching network $SN$.

**Input Selection Protocol**
Recall that $Q$ is a bound on the total number of input queries made by the sublinear algorithm.

1. The parties of $C$ run an MPC to jointly compute an encryption of a PRG seed for generating the randomness to be used in the sublinear algorithm execution. That is, they evaluate the following (randomized) functionality $F^{\mathsf{seed}}_{\mathsf{Enc}}$, on input pk:

   Compute:

   (a) Sample random seed $\leftarrow \{0,1\}^k$.

   (b) Sample "good" encryption randomness $r \leftarrow R$, via rejection sampling.

   (c) Encrypt $\widehat{\mathsf{seed}} = \mathsf{Enc}_{\mathsf{pk}}(\mathsf{seed}; r)$.

   Output: To all parties (in $C$): encrypted seed, $\widehat{\mathsf{seed}}$.

2. Each party in $C$ uses FHE evaluation to obtain an encryption $\hat{I}$ of the set $I$ of the indices output by SLA.Sel using randomness obtained by applying a PRG prg with suitable output length to the encrypted seed seed. In other words, we homomorphically evaluate

   $$\hat{I} = \mathsf{Eval}(\widehat{\mathsf{seed}}, \mathsf{SLA.Sel}(\mathsf{prg}(\cdot))) \ ,$$

   where SLA.Sel takes as input the randomness.

3. Locally, each party $P_i \in C$ homomorphically evaluates the (encrypted) permutation $\rho$ on the (encrypted) indices $p \in I$. That is, for each of the $Q$ encrypted indices $\hat{p}$ held by $P_i$, update $\hat{p}$ as follows. For each swapping pair $(i_s^\ell, j_s^\ell)$ in the switching network $SN$ (beginning with level $\ell = 1$, and proceeding to $\ell = 2, ..., d$), homomorphically evaluate the corresponding swap

   $$\hat{p} = \mathsf{Eval}\left((\hat{p}, \hat{b}_s^\ell), \mathsf{swap}_{i_s^\ell, j_s^\ell}\right) ,$$

   where $\mathsf{swap}'_{i,j}(p,b)$ is the swap function that performs the involution $(i,j)$ on $p$ iff $b=1$:

   $$\mathsf{swap}'_{i,j}(p,b) = \begin{cases} p & \text{if } b = 0 \vee p \notin \{i,j\} \\ i \oplus j \oplus p & \text{if } b = 1 \wedge p \in \{i,j\} \end{cases} .$$

   Denote the final updated indices held by party $P_i$ as $\hat{K}'_i = \{\hat{p}'_i\}$

4. The parties in $C$ run an MPC to collectively decrypt the resulting permuted set of indices (taking the values held by a majority). Formally, they evaluate the following functionality:

   Inputs: Each party $P_i$: secret share $\mathsf{sk}_i$ for the FHE, set of encrypted permuted indices $\hat{K}'_i = \{\hat{p}'_i\}$

   Compute:

   (a) Let $\hat{K}' = \mathsf{maj}_{i \in C} \hat{K}'_i$ be the most frequent set of ciphertexts input by parties $P_i \in C$.

   (b) Reconstruct the FHE decryption key from the input secret shares: $\mathsf{sk} \leftarrow \mathsf{Rec}(\{\mathsf{sk}_i\}_{i \in C})$.

   (c) For each of the $Q$ indices $\hat{p}' \in \hat{K}'$, decrypt $q := \mathsf{Dec}_{\mathsf{sk}}(\hat{p}')$.

   Output: To all parties (in $C$): The set of $Q$ decrypted indices $K := \{q\}$ (allegedly $K = \rho(I)$)

**Figure 5.5:** Executed by primary committee $C$ to select the indices that will be accessed in the sublinear algorithm. The selected indices are output in *permuted* form, as dictated by the secret encrypted permutation $\hat{\rho}$.

**Evaluation Protocol**
Inputs:

1. The parties of $C$ run an MPC to jointly compute an encryption of a PRG seed for generating the randomness to be used in the sublinear algorithm execution. That is, they evaluate the (randomized) functionality $F^{\mathsf{seed}}_{\mathsf{Enc}}$, as described in the input selection protocol (see Figure 5.5).

   Output: To all parties in $C$: encrypted seed $\widehat{\mathsf{seed}}'$.

2. Each party $P_i \in C$ locally evaluates the sublinear algorithm, SLA.Exec, homomorphically on the set of his received ciphertexts $(\{\hat{p}^j_\ell\}_{j\in C_\ell}, \{\hat{x}^j_{p_\ell}\}_{j\in C_\ell})_{\ell\in\rho(I)}$ and using randomness derived from the encrypted seed $\widehat{\mathsf{seed}}'$, as follows.

   For each queried input $\ell \in \rho(I)$, party $P_i$ first takes a majority vote for the most frequently occurring index and input ciphertexts sent by parties in $C_\ell$:

   $$\hat{P}_\ell = \mathsf{maj}_{j\in C_\ell}\{\hat{p}^j_\ell\}, \qquad \hat{X}_\ell = \mathsf{maj}_{j\in C_\ell}\{\hat{x}^j_{p_\ell}\}.$$

   Then, party $P_i$ homomorphically evaluates SLA.Exec on the resulting ciphertexts:

   $$\hat{Z} \leftarrow \mathsf{Eval}\left((\{\hat{P}_\ell\}_{\ell\in\rho(I)}, \{\hat{X}_\ell\}_{\ell\in\rho(I)}, \widehat{\mathsf{seed}}'), \mathsf{execute}\right),$$

   where $\mathsf{execute}(\{i_\ell\}, \{x_\ell\}, \mathsf{rand})$ is the function that evaluates SLA.Exec with randomness $\mathsf{rand} = \mathsf{prg}(\mathsf{seed}')$, taking each $x_\ell$ as the requested input of index $i_\ell$.

3. The parties of $C$ run an MPC to collectively decrypt the resulting ciphertext computed by each party $P_i \in C$ (taking the value held by the majority of parties of $C$ as the final output). Explicitly, they evaluate the functionality $F_{\mathsf{Eval}}$:

   Inputs: Each party $P_i \in C$: secret share $\mathsf{sk}_i$ for FHE, evaluated ciphertext $\hat{Z}_i$ from the previous step.

   Compute:

   (a) Reconstruct the FHE decryption key from the input secret shares: $\mathsf{sk} \leftarrow \mathsf{Rec}(\{\mathsf{sk}_i\}_{i\in C})$.
   (b) Take $\hat{Z} = \mathsf{maj}_{i\in C}\{\hat{Z}_i\}$ to be the most frequently occurring input ciphertext.
   (c) Decrypt the ciphertext $\hat{Z}$: let $\mathsf{answer} = \mathsf{Dec}_{\mathsf{sk}}(\hat{Z})$.

   Output: To all parties (in $C$): answer.

**Figure 5.6:** The evaluation protocol, in which the parties in the supreme committee $C$ evaluate the sublinear algorithm on the set of queried inputs.

**Overall Protocol**

1. Execute *Certified* Almost Everywhere committee election for primary committee $C$ and individual committees $\{C_i\}_{i\in[n]}$.

   Outcome: $(1-o(1))$ fraction of honest parties agree on good committees $C, \{C_i\}_{i\in[n]}$ *and* hold a certificate of correctness.

2. Execute Certified a.e. to Certified everywhere transformation protocol to agree on $C, \{C_i\}_{i\in[n]}$.

   Outcome: *all* honest parties agree on good committees $C, \{C_i\}_{i\in[n]}$ and hold a certificate of correctness.

3. Parties in the primary committee $C$ execute the Encryption Key Generation procedure, as described in Figure 5.1, and broadcast the output public keys $\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}$ to all parties via $\mathsf{ComBroadcast}(\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}})$.

   Outcome: All parties learn $\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}$.

4. Each party $P_i$ commits to his input to the parties in his personal committee $C_i$, as in Figure 5.2.

   Outcome: Every member of $C_i$ holds an encryption of $x_i$ (and $i$).

5. Parties in primary committee $C$ execute the Shuffle Generation procedure (see Figure 5.3), and broadcast the resulting encrypted permutation $\hat{\rho} = \{(\hat{B}^1_\ell, ..., \hat{B}^{n/2}_\ell)\}_{\ell\in[d]}$ to all parties via $\mathsf{ComBroadcast}(\hat{\rho})$.

   Outcome: All parties learn the encrypted random permutation $\hat{\rho}$, expressed as a sequence of encrypted swap bits in the switching network $SN$ (see Section 5.2.6).

6. The committees $C_i$ obliviously shuffle their stored input values, as follows.

   For each layer $L_1, ..., L_d$ in the sorting network $SN$,

   - Let $L_\ell = ((i_1, j_1), ..., (i_{n/2}, j_{n/2}))$ be the swapping pairs in the current layer $\ell$.
   - In *parallel*, the corresponding pairs of parties $(C_{i_1}, C_{j_1}), ..., (C_{i_{n/2}}, C_{j_{n/2}})$ perform the pairwise swap protocol detailed in Figure 5.4. Each pair $(C_{i_s}, C_{j_s})$ uses the (encrypted) swap bit $\hat{B}^s_\ell$ from Step 3 (wrt level $\ell$) to dictate whether or not to swap.

   Outcome: each party in committee $C_i$ holds encryptions of index $\rho(i)$ and input $x_{\rho(i)}$.

7. Parties in primary committee $C$ select which inputs will be used in the sublinear algorithm computation, as described in Figure 5.5 (revealing only the *permuted* indices).

   Outcome: the members of $C$ each agree on a (permuted) set of input indices $\rho(I) \subset [n]$.

8. Every party $P_i$ in primary committee $C$ sends a message "Please send encrypted input $\ell$" to every party $P_j$ in $C_\ell$ for which $\ell \in \rho(I)$ (note that some parties $P_j$ may receive multiple such messages, if contained within more than one such committee $C_\ell$).

9. Each party $P_j \in C_\ell$ who receives consistent messages "Please send encrypted input $\ell$" from a *majority* of the parties in $C$, broadcasts his corresponding pair of ciphertexts $(\hat{p}^j_\ell, \hat{x}^j_{p_\ell})$ to all parties in $C$. (Recall that these values allegedly correspond to an encryption of the party index $p = \rho^{-1}(\ell)$ and corresponding input $x_p$ held by the committee $C_\ell = C_{\rho(p)}$ after the $\rho$-permutation shuffle).

10. The parties of $C$ evaluate the second portion of the sublinear algorithm, $\mathsf{SLA.Exec}$, as described in Figure 5.6. Denote the resulting output as $\mathsf{answer}$.

11. Each party $P_i \in C$ broadcasts the resulting output $\mathsf{answer}$ to all parties via $\mathsf{ComBroadcast}(\mathsf{answer})$.

### 5.4.4 Security Proof of Theorem 5.4.1

*Proof.* Let $\mathcal{A}$ be any real-world PPT adversary for $\Pi$. Denote by $M \subset \mathcal{P}$ the set of parties corrupted by $\mathcal{A}$. We first construct an adversary $\mathcal{S}$ – called the *simulator* – in the ideal world who simulates the real-world view of $\mathcal{A}$ by simulating the honest parties in the ideal-world experiment. Later, we then proceed by proving that the output of $\mathcal{S}$ within the ideal world is indistinguishable from the output of the real-world experiment.

Description of the simulator $\mathcal{S}$:

Simulate PKI information:

1. Sample signature key pairs $(\mathsf{vk}_{\mathsf{l},i}, \mathsf{sk}_{\mathsf{l},i}) \leftarrow \mathsf{Gen}(1^k)$ for each party $P_i \in [n]$ and each leaf $\mathsf{l}$ of the communication tree $P_i$ is assigned to.
2. Generate *simulated* CRS values for the two instantiations of the simulation-sound NIZK proof system:
$$(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k), \quad (\mathsf{crs}^{\mathsf{same}}, \mathsf{trap}^{\mathsf{same}}) \leftarrow \mathcal{S}'^{\mathsf{crs}}(1^k).$$
3. Send the collection of verification keys $\{\mathsf{vk}_{\mathsf{l},i}\}$ and $\mathsf{crs}, \mathsf{crs}^{\mathsf{same}}$ to $\mathcal{A}$.
4. The adversary $\mathcal{A}$ responds with a set $M \subset [n]$ of parties to corrupt. For each party $P_j$ with $j \in M$, send $\mathcal{A}$ the corresponding secret signing keys $\mathsf{sk}_{\mathsf{l},j}$ for each leaf $\mathsf{l}$ containing $P_j$.

Simulate committee setup:

1. Honestly simulate actions of honest parties during certified almost everywhere committee election protocol (independent of honest parties' inputs). In particular, send corresponding messages to $\mathcal{A}$. Moreover, $\mathcal{A}$'s messages sent from corrupted parties are sent to the simulated honest parties.
2. Honestly simulate actions of honest parties during (certified a.e. to certified everywhere) transformation protocol (independent of honest parties' inputs)
3. If committees $C, \{C_i\}_{i\in[n]}$ are not good (at least 2/3 honest), or not every honest party agrees on their composition, then abort the simulation.

Simulate encryption key generation procedure:

1. Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$ for the FHE scheme.
2. Sample $(\mathsf{pk}^{\mathsf{CPA}}, \mathsf{sk}^{\mathsf{CPA}}) \leftarrow \mathsf{Gen}^{\mathsf{CPA}}(1^k)$ for the standard CPA-secure encryption scheme.
3. For each corrupted party $P_j \in C \cap M$ in the committee $C$, sample a (bogus) secret share $\mathsf{sk}_j \leftarrow \{0,1\}^{\mathsf{poly}(k)}$ at *random.*
4. Run the MPC simulator, fed with output values $(\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \{\mathsf{sk}_j\}_{j\in C\cap M})$, to simulate the MPC interactions among the parties of $C$ in generating these values.

5. Honestly simulate the actions of parties in the execution of $\mathsf{ComBroadcast}(\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}})$, including initiating an independent execution on behalf of each honest party $P_i \in C$.

Simulate input commitment process:

1. For each *honest* party $P_i$:
   (a) Construct simulated message to VSS
      i. Sample a (bogus) encryption $\hat{x}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$ for the FHE.
      ii. Sample a (bogus) encryption $\hat{x}_i^{\mathsf{CPA}} \leftarrow \mathsf{Enc}^{\mathsf{CPA}}_{\mathsf{pk}^{\mathsf{CPA}}}(0)$ for the standard CPA-secure encryption scheme.
      iii. Generate a *simulated* NIZK proof for the FHE ciphertext:
$$\pi_i \leftarrow \mathcal{S}^{\mathsf{proof}}(\mathsf{crs}, (\mathsf{pk}, \hat{x}_i), \mathsf{trap}),$$
      using the simulation trapdoor trap generated during the PKI simulation.
      iv. Generate a *simulated* NIZK proof that the ciphertexts are consistent: $\pi_i^{\mathsf{CPA}} \leftarrow \mathcal{S}'^{\mathsf{proof}}(\mathsf{crs}^{\mathsf{same}}, (\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \hat{x}_i, \hat{x}_i^{\mathsf{CPA}}), \mathsf{trap}^{\mathsf{same}})$, using the trapdoor $\mathsf{trap}^{\mathsf{same}}$.
   (b) Simulate the VSS sharing phase
      i. Sample *random* secret shares $\mathsf{input}_i^j$ for each corrupt party $P_j \in C_i$.
      ii. Sample the remaining $|C_i \setminus M|$ shares $\{\mathsf{input}_i^j\}_{j \in C_i \setminus M}$ randomly, subject to the constraint that $\mathsf{Rec}(\{\mathsf{input}_i^j\}_{j \in C_i}) = (\hat{x}_i, \pi_i, \hat{x}_i^{\mathsf{CPA}}, \pi_i^{\mathsf{CPA}})$; ie, the shares reconstruct to the simulated tuple of values generated above.
      iii. Run the VSS protocol simulator $\mathcal{S}^{\mathsf{VSS}}$ on the collection of output secret shares $\{\mathsf{input}_i^j\}_{j \in C_i}$ to simulate the interaction between $P_i$ and the members of $C_i$ during the VSS sharing procedure.
   (c) Simulate the VSS reconstruction phase honestly, with respect to message $(\hat{x}_i, \pi_i, \hat{x}_i^{\mathsf{CPA}}, \pi_i^{\mathsf{CPA}})$.
   (d) Simulate the index-encryption MPC execution:
      i. Sample encryption randomness $r \leftarrow R$ and encrypt $\hat{\imath} = \mathsf{Enc}_{\mathsf{pk}}(i; r)$.
      ii. Run the MPC simulator on input $\hat{\imath}$ to simulate the interactions among the members of committee $C_i$ in the MPC to generate this ciphertext.
   (e) For each honest party $P_j \in C_i$, set $P_j$'s beliefs for $P_i$'s values to $\hat{x}_i^j \leftarrow \hat{x}_i$ and $\hat{\imath}^j \leftarrow \hat{\imath}$.
2. For each *malicious* party $P_i$:
   Note that in this step, in addition to simulating the actions of honest parties, the simulator will extract the inputs $x_i'$ of each corrupted party $P_i$ and submit them to the ideal functionality.
   (a) During the VSS sharing phase:
      i. Simulate the actions of honest parties honestly.
      ii. For each honest party $P_j \in C_i$, let $\mathsf{input}_i^j$ be the secret share received by $P_j$.
      iii. Reconstruct $P_i$'s message from the collection of honest party shares:
$$(\hat{x}_j, \pi_j, \hat{x}_j^{\mathsf{CPA}}, \pi_j^{\mathsf{CPA}}) \leftarrow \mathsf{Rec}(\{\mathsf{input}_i^j\}_{j \in C_i \setminus M}).$$

iv. If either of the received proofs $\pi_i, \pi_i^{\mathsf{CPA}}$ does not properly verify, or if the reconstruction process yields $\perp$, then set $x'_i = 0$.

v. Otherwise, extract $P_i$'s input by decrypting the standard CPA ciphertext $\hat{x}_i^{\mathsf{CPA}}$. That is, take $x'_i = \mathsf{Dec}^{\mathsf{CPA}}_{\mathsf{sk}^{\mathsf{CPA}}}(\hat{x}_i^{\mathsf{CPA}})$, where $\mathsf{sk}^{\mathsf{CPA}}$ was sampled during the simulation of the key generation procedure.

(b) Call to ideal functionality:
Submit the collection of all retrieved input values $\{x'_j\}_{j\in M}$ as inputs to the ideal functionality $\mathcal{F}_{\mathsf{ideal}}$, on behalf of corrupted parties. Denote by $\mathsf{answer}$ the response from $\mathcal{F}_{\mathsf{ideal}}$.
Recall that this value corresponds to a random output of the sublinear algorithm, evaluated on the collection of honest party inputs together with corrupt party inputs $\{x'_j\}_{j\in M}$.

Simulate shuffle generation procedure:

1. Generate message values. For each honest party $P_i$ and each swapping pair $\ell \in [q]$ in the switching network $SN$,
   (a) Sample a (bogus) ciphertext $\hat{b}_\ell \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$.
   (b) Sample a (bogus) ciphertext $\hat{b}_\ell^{\mathsf{CPA}} \leftarrow \mathsf{Enc}^{\mathsf{CPA}}_{\mathsf{pk}^{\mathsf{CPA}}}(0)$.
   (c) Generate a simulated proof for the FHE ciphertext: $\pi_\ell \leftarrow \mathcal{S}^{\mathsf{proof}}(\mathsf{crs}, (\mathsf{pk}, \hat{b}_\ell), \mathsf{trap})$.
   (d) Generate a simulated proof that the ciphertexts are consistent:
   $\pi_\ell^{\mathsf{CPA}} \leftarrow \mathcal{S}'^{\mathsf{proof}}(\mathsf{crs}^{\mathsf{same}}, (\mathsf{pk}, \hat{b}_\ell, \mathsf{pk}^{\mathsf{CPA}}, \hat{b}_\ell^{\mathsf{CPA}}), \mathsf{trap}^{\mathsf{same}})$.
2. Simulate the VSS sharing phase
   (a) For each honest party $P_i$, sample *random* secret shares $\mathsf{bits}_i^j$ for each malicious party $P_j$, and sample the remaining shares $\mathsf{bits}_i^j$ honestly, subject to the constraint that $\mathsf{Rec}(\{\mathsf{bits}_i^j\}_{j\in C}) = (\hat{b}_\ell, \pi_\ell, \hat{b}_\ell^{\mathsf{CPA}}, \pi_\ell^{\mathsf{CPA}})$.
   (b) Run the VSS simulator $\mathcal{S}^{\mathsf{VSS}}$, on this set of shares, to simulate the interaction during the VSS sharing phase.
3. Honestly simulate the actions of honest parties for the remainder of the shuffle generation procedure: namely, in the sharing phase for malicious parties, in the reconstruction phase all parties, and the local homomorphic evaluations on the resulting ciphertexts.
4. Honestly simulate the action of honest parties in the executions of $\mathsf{ComBroadcast}(\hat{\rho} := \{\hat{B}_\ell\}_{\ell\in[q]})$ on the ciphertexts $\hat{B}_\ell$ resulting from honest homomorphic evaluation. This includes initiating an execution of $\mathsf{ComBroadcast}(\hat{\rho})$ on behalf of each honest party $P_i \in C$.

Simulate shuffling process

*Honestly* simulate the actions of honest parties, given the collection of initial ciphertexts $\{\hat{x}_i\}_{i\in[n]}$, $\{\hat{i}\}_{i\in[n]}$, and $\{B^\ell\}_{\ell\in[q]}$ generated as a result of the simulation of the input commitment phase and shuffle generation phase.

Simulate input query selection protocol

1. Sample a (bogus) encryption $\widehat{\mathsf{seed}} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$.
2. Run the MPC simulator on input $\widehat{\mathsf{seed}}$ to simulate the interactions between members of $C$ during the MPC to generate this output value.
3. Honestly simulate the dictated homomorphic evaluations on ciphertexts on behalf of the honest parties.
4. Sample a *random* subset $J \subset [n]$ of size $Q$ (allegedly equal to $\rho(I)$).
5. Run the MPC simulator on input $J$ to simulate the interaction between members of $C$ in the corresponding MPC execution.

Simulate sublinear algorithm execution

1. On behalf of each honest party $P_i$ in the committee $C$, send the message "Please send encrypted input $\ell$" to every party $P_j$ in $C_\ell$ for which $\ell \in K$ (for the set $J$ chosen by the simulator in the previous step).
2. For each honest party $P_j \in C_\ell$ who receives messages "Please send encrypted input $\ell$" from a majority of parties in $C$ (in particular, for which the honest parties in $C$ send this message), honestly simulate $P_j$ broadcasting his pair of ciphertexts $(\hat{p}^j_\ell, \hat{x}^j_{p_\ell})$ to all parties in $C$ (where these ciphertexts are the values resulting from the shuffling process above).
3. Sample a (bogus) ciphertext $\widehat{\mathsf{seed}}' \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$.
4. Run the MPC simulator on input $\widehat{\mathsf{seed}}'$ to simulate the interaction between parties in $C$ during the corresponding MPC execution.
5. Honestly simulate the dictated homomorphic evaluations on ciphertexts on behalf of the honest parties.
6. Run the MPC simulator $\mathsf{Sim}_{\mathsf{MPC}}$ to simulate the final MPC execution, feeding the simulator with the final output value $\mathsf{answer}$ that was received by the ideal functionality $\mathcal{F}_{\mathsf{ideal}}$.
7. Honestly simulate the actions of honest parties in the final execution of $\mathsf{ComBroadcast}(\mathsf{answer})$.

To prove the indistinguishability of the output of the simulator and the output of the real-world experiment, we consider a sequence of hybrid experiments in which the real-world interaction is replaced one piece at a time by the simulated version. The output of each hybrid experiment consists of the outputs of all parties, where honest parties output in accordance with the dictated protocol, and malicious parties may output any efficiently computable function of the view of the adversary. For every adversary $\mathcal{A}_\ell$ with auxiliary input $z \in \{0,1\}^*$ running in hybrid experiment $\ell$ with initial inputs $\vec{x}$, we denote the output of the corresponding hybrid $\ell$ experiment by

$$\mathsf{HYB}_\ell\left(\mathcal{A}_\ell, 1^k, z, \{x_i\}_{i=1}^n\right).$$

For each pair of hybrids $\ell$ and $\ell+1$, we show that for any adversary $\mathcal{A}_\ell$ running in Hybrid $\ell$, there exists an adversary $\mathcal{A}_{\ell+1}$ running in Hybrid $(\ell+1)$ such that

$$\mathsf{HYB}_\ell\left(\mathcal{A}_\ell, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_{\ell+1}\left(\mathcal{A}_{\ell+1}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

Note that once we show this for every step $i = 0, ..., 19$, the theorem will follow, as this will imply that for each adversary $\mathcal{A}$ in the real-world experiment (Hybrid 0), there is an adversary $\mathcal{A}_{19}$ in the ideal-world experiment (Hybrid 19), such that

$$\mathsf{HYB}_0\left(\mathcal{A}_0, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_{19}\left(\mathcal{A}_{19}, 1^k, z, \{x_i\}_{i=1}^n\right),$$

as desired.

We now formally describe the sequence of hybrid experiments.

**Hybrid 0.** The real world: i.e., the adversary interacts with honest parties in the real-world experiment running $\Pi$.

**Hybrid 1. (Abort if committee elections fail).**

Same as the previous hybrid, except that the experiment ends in abort if at the conclusion of the Certified Almost Everywhere Committee Election and Certified A.E. to Certified Everywhere transformation protocols, any of the following conditions *fails* to hold:

- All honest parties agree on the committees $C, \{C_i\}_{i\in[n]}$. That is, for every pair of honest parties $P_i, P_j$, it holds that $C^{(i)} = C^{(j)}$ and $C_\ell^{(i)} = C_\ell^{(j)}$ $\forall \ell \in [n]$.
- Every honest party possesses valid certification of each committee $C, \{C_i\}_{i\in[n]}$.
- All elected committees are "good": there exists a constant $\delta > 0$ for which $|C \cap M| \leq (\frac{1}{3} - \delta)|C|$, and $|C_i \cap M| \leq (\frac{1}{3} - \delta)|C_i|$ $\forall i \in [n]$.

In addition, the experiment ends in abort at any point that a majority of the supreme committee members ($C$) initiate an execution of ComBroadcast on the same message $m$ but there exists an honest party who does not receive the correct message $m$.

**Lemma 5.4.9.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_0$ in Hybrid 0, there exists a PPT adversary $\mathcal{A}_1$ in Hybrid 1 such that*

$$\mathsf{HYB}_0\left(\mathcal{A}_0, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{s}{\cong} \mathsf{HYB}_1\left(\mathcal{A}_1, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* It follows by Lemmas 5.4.3-5.4.7 that the probability of aborting is negligible in $k$.

□

**Hybrid 2. (Replace MPC executions with ideal functionalities).**

This hybrid is similar to Hybrid 1, but with each execution of the underlying MPC protocol replaced by its corresponding ideal functionality. Explicitly, the experiment now contains the following ideal functionalities:

- $F_{(\mathsf{pk},\mathsf{sk})}$: In the place of the FHE key generation MPC (defined in Figure 5.1).
- $F_{\hat{i}}$: In the place of each index encryption MPC run by the committees $C_i$ during the input commitment stage (defined in Figure 5.2).
- $F^{\mathsf{seed}}_{\mathsf{Enc}}$: In the place of the (encrypted) PRG seed generation MPC run by $C$ during the input selection procedure (defined in Figure 5.5).
- $F^{\mathsf{queries}}_{\mathsf{Dec}}$: In the place of the decryption MPC (for the encrypted set of permuted query indices) run by $C$ during the input selection procedure (defined in Figure 5.5).
- $F^{\mathsf{seed}'}_{\mathsf{Enc}}$: In the place of the (encrypted) PRG seed generation MPC run by $C$ during the evaluation stage (defined in Figure 5.5, used in Figure 5.6).
- $F^{\mathsf{ans}}_{\mathsf{Dec}}$: In the place of the decryption MPC (for the encrypted final answer) run by members of $C$ in the evaluation stage (defined in Figure 5.6).

Note that the ciphertexts $\{\hat{p}_i\}$, $\widehat{\mathsf{seed}}$, $\widehat{\mathsf{seed}}'$ generated by the ideal functionalities $F_{\hat{i}}$ and $F^{\mathsf{seed}}_{\mathsf{Enc}}, F^{\mathsf{seed}'}_{\mathsf{Enc}}$ are necessarily evaluation-enabled (as per Definition 5.2.3), since these functionalities sample encryption randomness from the "good" set $R$.

Define $\mathsf{seed} = \mathsf{Dec}_{\mathsf{sk}}(\widehat{\mathsf{seed}})$ and $\mathsf{seed}' = \mathsf{Dec}_{\mathsf{sk}}(\widehat{\mathsf{seed}}')$.
Similarly, for each $i \in [n]$, define $p_i = \mathsf{Dec}_{\mathsf{sk}}(\hat{p}_i)$. (Note that by the specification of $F_{\hat{i}}$ and the certifiabiliy of the FHE scheme with respect to $R$, it holds that $p_i = i$).

Ideal functionalities: $\mathbf{F}_{(\mathsf{pk},\mathsf{sk})}, \mathbf{F}_{\hat{i}}, \mathbf{F}^{\mathsf{seed}}_{\mathsf{Enc}}, \mathbf{F}^{\mathsf{queries}}_{\mathsf{Dec}}, \mathbf{F}^{\mathsf{seed}'}_{\mathsf{Enc}}, \mathbf{F}^{\mathsf{ans}}_{\mathsf{Dec}}$.

**Lemma 5.4.10.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_1$ in Hybrid 1, there exists a PPT adversary $\mathcal{A}_2$ in Hybrid 2 such that*

$$\mathsf{HYB}_1\left(\mathcal{A}_1, 1^k, z, \{x_i\}_{i=1}^n\right) \equiv \mathsf{HYB}_2\left(\mathcal{A}_2, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* We define the adversary $\mathcal{A}_2$ as a sequence of sub-algorithms $\mathcal{A}_2 = (\mathcal{A}_2^1, \mathcal{A}_2^{\mathsf{MPC1}}, \mathcal{A}_2^2, \mathcal{A}_2^{\mathsf{MPC2}}, \ldots)$ corresponding to separating the actions of $\mathcal{A}_2$ between and during each of the MPC executions. We define the output of each sub-adversary $\mathcal{A}_2^i$ (or $\mathcal{A}_2^{\mathsf{MPC}i}$) to be the view of $\mathcal{A}_2$ up to that point: each sub-adversary receives as auxiliary input the output $z^i$ (or $z^{\mathsf{MPC}i}$) of the previous sub-adversary. By the (perfect) security of the underlying MPC protocol, for each MPC execution there exists a simulator $\mathcal{S}^{\mathsf{MPC}i}$ who, given the same auxiliary input $z^{\mathsf{MPC}i}$ and access to the corresponding ideal functionality (i.e., $F_{(\mathsf{pk},\mathsf{sk})}$, $F_{\hat{i}}$, $F^{\mathsf{seed}}_{\mathsf{Enc}}$,...) simulates the view of $\mathcal{A}_2^{\mathsf{MPC}i}$ during that MPC execution. Each intermediate sub-adversary $\mathcal{A}_2^i$ simulates the actions of $\mathcal{A}_1$ exactly in between the MPC executions, given current view $z^i$. The complete Hybrid 2 adversary $\mathcal{A}_2$ will simulate $\mathcal{A}_1$ by piecing all these simulations together.

Indistinguishability of the simulated output follows directly by the perfect security of the underlying MPC protocol, together with a standard hybrid argument. We emphasize that this argument holds in the present *concurrent* setting due to the *perfect* security of the MPC protocol. $\square$

**Hybrid 3. (Replace VSS with ideal functionality).**

Similar to Hybrid 2, except that the VSS sharing protocol is replaced by the associated ideal functionality $F_{\mathsf{VSS}}$ (as described in Definition 5.2.8). This affects the protocol in two places.

- The input commitment phase for each party $P_i$ now takes place as follows:
  1. Party $P_i$ generates $r_i \leftarrow R$, $\hat{x}_i = \mathsf{Enc}_{\mathsf{pk}}(x; r_i)$, $\pi_i \leftarrow \mathsf{P}(\mathsf{crs}(\mathsf{pk}, \hat{x}_i), (x_i, r_i))$, $r_i^{\mathsf{CPA}} \leftarrow R$, $\hat{x}_i^{\mathsf{CPA}} = \mathsf{Enc}^{\mathsf{CPA}}_{\mathsf{pk}^{\mathsf{CPA}}}(x; r_i^{\mathsf{CPA}})$, and $\pi^{\mathsf{CPA}}$ as before.
  2. Instead of $P_i$ running as dealer in the VSS to share the values $(\hat{x}_i, \pi_i, \hat{x}_i^{\mathsf{CPA}}.\pi_i^{\mathsf{CPA}})$, party $P_i$ now samples a random polynomial $f(x)$ for which $f(0) = (\hat{x}_i, \pi_i, \hat{x}_i^{\mathsf{CPA}}.\pi_i^{\mathsf{CPA}})$ and executes the ideal functionality $F_{\mathsf{VSS}}$ on input $f(x)$.
  3. The parties $P_j \in C_i$ run the VSS reconstruction procedure, as before, on the shares received from the ideal functionality. The remainder of the input commitment phase continues as in the previous hybrid.
- Similarly, in the shuffle generation phase, for each swap index $\ell \in [q]$ of the switching network, the following takes place:
  1. Each party $P_i \in C$ generates $b_\ell^i, r_\ell^i, \hat{b}_\ell^i, \pi_\ell^i, (r_\ell^i)^{\mathsf{CPA}}, (\hat{b}_\ell^i)^{\mathsf{CPA}}, (\pi_\ell^i)^{\mathsf{CPA}}$ as before
  2. Instead of $P_i$ running as dealer in the VSS to share the values $(\hat{b}_\ell^i, \pi_\ell^i, (\hat{b}_\ell^i)^{\mathsf{CPA}}, (\pi_\ell^i)^{\mathsf{CPA}})$, $P_i$ now samples a random polynomial $g(x)$ for which $g(0) = (\hat{b}_\ell^i, \pi_\ell^i, (\hat{b}_\ell^i)^{\mathsf{CPA}}, (\pi_\ell^i)^{\mathsf{CPA}})$ and executes the ideal functionality $F_{\mathsf{VSS}}$ on input $g(x)$.
  3. The parties $P_j \in C$ run the VSS reconstruction procedure, as before, on the shares received from the ideal functionality. The remainder of the shuffle generation phase continues as in the previous hybrid.

Ideal functionalities: $F_{(\mathsf{pk},\mathsf{sk})}, F_i^{}, \mathbf{F}_{\mathsf{VSS}}, F^{\mathsf{seed}}_{\mathsf{Enc}}, F^{\mathsf{queries}}_{\mathsf{Dec}}, F^{\mathsf{seed}'}_{\mathsf{Enc}}, F^{\mathsf{ans}}_{\mathsf{Dec}}$.

**Lemma 5.4.11.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_2$ in Hybrid 2, there exists a PPT adversary $\mathcal{A}_3$ in Hybrid 3 such that*

$$\mathsf{HYB}_2\left(\mathcal{A}_2, 1^k, z, \{x_i\}_{i=1}^n\right) \equiv \mathsf{HYB}_3\left(\mathcal{A}_3, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* As in the proof of the previous lemma, we consider an adversary $\mathcal{A}_3$ composed of multiple sub-adversaries, corresponding to the actions of $\mathcal{A}_2$ between and during each VSS execution. By the simulation-based security of the underlying VSS protocol, there exists a simulator $\mathcal{S}^{\mathsf{VSS}}$ who, given the current view of $\mathcal{A}_2$ and access to the ideal functionality $F_{\mathsf{VSS}}$, simulates the view of $\mathcal{A}_3$ during the corresponding VSS execution. The complete Hybrid 3 adversary $\mathcal{A}_3$ will simulate the output of $\mathcal{A}_2$ by piecing all these simulations together (and simulating $\mathcal{A}_2$'s actions exactly in between VSS executions).

Indistinguishability of output distributions follows directly by the *perfect* simulation security of the underlying VSS protocol, together with a standard hybrid argument. We again emphasize that *concurrent* security is satisfied, as the security of the VSS protocol is *perfect*. □

**Hybrid 4. (Simulated NIZK CRS, simulated proofs for honest parties).**

Same as the previous hybrid, with three changes:

- The CRS values for the two instantiations of the NIZK argument system are replaced by an ideal functionality $F_{\mathsf{crs}}$ that samples *simulated* CRS values, as $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathcal{S}^{\mathsf{crs}}(1^k)$ and $(\mathsf{crs}^{\mathsf{CPA}}, \mathsf{trap}^{\mathsf{CPA}}) \leftarrow \mathcal{S}'^{\mathsf{crs}}(1^k)$, and outputs $\mathsf{crs}, \mathsf{crs}^{\mathsf{CPA}}$ to all parties.
- Honest parties no longer honestly generate proofs $\pi$ of statements as instructed by the protocol. Instead, they submit the relevant instance $x$ (i.e., $(\mathsf{pk}, \hat{y})$ or $(\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \hat{y}, \hat{y}^{\mathsf{CPA}})$) to prove validity of some ciphertext $\hat{y}$, or consistency of two ciphertexts) to an ideal functionality $F_{\mathsf{NIZK}}$, who responds with a *simulated* proof $\pi' \leftarrow \mathcal{S}^{\mathsf{proof}}(\mathsf{crs}, \mathsf{trap}, x)$ (or $\pi' \leftarrow \mathcal{S}^{\mathsf{proof}}(\mathsf{crs}^{\mathsf{same}}, \mathsf{trap}^{\mathsf{CPA}}, x)$, in the second case); the honest party then uses the resulting simulated proof $\pi'$ in the place of $\pi$. Explicitly,
  - In the Shuffle Generation phase, when generating encryptions $\hat{b}_\ell, \hat{b}_\ell^{\mathsf{CPA}}$ of random bits, the honest parties no longer produce accompanying proofs in the honest fashion; rather, they call the ideal functionality $F_{\mathsf{NIZK}}$ on input $(\mathsf{pk}, \hat{b}_\ell)$ and $(\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \hat{b}_\ell, \hat{b}_\ell^{\mathsf{CPA}})$, and use the resulting proofs $\pi_\ell, \pi_\ell^{\mathsf{CPA}}$.
  - Similarly, in the Input Commitment stage, honest parties $P_i$ generate the ciphertexts $\hat{x}_i, \hat{x}_i^{\mathsf{CPA}}$, then call the ideal functionality $F_{\mathsf{NIZK}}$ on input $(\mathsf{pk}, \hat{x}_i)$, $(\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \hat{x}_i, \hat{x}_i^{\mathsf{CPA}})$, and use the resulting simulated proofs.

Ideal functionalities: $\mathbf{F}_{\mathsf{crs}}, F_{(\mathsf{pk},\mathsf{sk})}, F_{\hat{i}}, \mathbf{F}_{\mathsf{NIZK}}, F_{\mathsf{VSS}}, F_{\mathsf{Enc}}^{\mathsf{seed}}, F_{\mathsf{Dec}}^{\mathsf{queries}}, F_{\mathsf{Enc}}^{\mathsf{seed}'}, F_{\mathsf{Dec}}^{\mathsf{ans}}$.

**Lemma 5.4.12.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_3$ in Hybrid 3, there exists a PPT adversary $\mathcal{A}_4$ in Hybrid 4 such that*

$$\mathsf{HYB}_3\left(\mathcal{A}_3, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_4\left(\mathcal{A}_4, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* For any PPT adversary $\mathcal{A}_3$ in Hybrid 3, we consider the identical adversary $\mathcal{A}_4$ in Hybrid 4. Computational indistinguishability of the corresponding experiment outputs follows directly from the adaptive zero knowledge property of the simulation-sound NIZK argument system (see Definition 5.2.1). □

**Hybrid 5. (Abort if $\mathcal{A}$ proves false statement).**

Same as the previous hybrid, except that the experiment ends in fail in the Input Commitment stage if any corrupted party $P_i$ produces a tuple $(\hat{x}_i, \pi_i, \hat{x}_i^{\mathsf{CPA}}, \pi_i^{\mathsf{CPA}})$ for which the proofs $\pi_i, \pi_i^{\mathsf{CPA}}$ verify correctly: i.e.,

$$1 = \mathsf{V}(\mathsf{crs}, (\mathsf{pk}, \hat{x}_i), \pi_i) \text{ and } 1 = \mathsf{V}'(\mathsf{crs}^{\mathsf{same}}, (\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \hat{x}_i, \hat{x}_i^{\mathsf{CPA}}), \pi_i^{\mathsf{CPA}}),$$

but either $(\mathsf{pk}, \hat{x}_i) \notin \mathcal{L}$ or $(\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \hat{x}_i, \hat{x}_i^{\mathsf{CPA}}) \notin \mathcal{L}^{\mathsf{same}}$.

Similarly, the experiment ends in fail in the Shuffle Generation stage if in Step 3 the adversary broadcasts a tuple $(\hat{b}_\ell, \pi_\ell, \hat{b}_\ell^{\mathsf{CPA}}, \pi_\ell^{\mathsf{CPA}})$ for which the proofs $\pi_\ell, \pi_\ell^{\mathsf{CPA}}$ verify correctly but $(\mathsf{pk}, \hat{b}_\ell) \notin \mathcal{L}$ or $(\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \hat{b}_\ell, \hat{b}_\ell^{\mathsf{CPA}}) \notin \mathcal{L}^{\mathsf{same}}$.

Ideal functionalities: $F_{\mathsf{crs}}, F_{(\mathsf{pk},\mathsf{sk})}, F_{\hat{i}}, F_{\mathsf{NIZK}}, F_{\mathsf{VSS}}, F_{\mathsf{Enc}}^{\mathsf{seed}}, F_{\mathsf{Dec}}^{\mathsf{queries}}, F_{\mathsf{Enc}}^{\mathsf{seed}'}, F_{\mathsf{Dec}}^{\mathsf{ans}}$.

**Lemma 5.4.13.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_4$ in Hybrid 4, there exists a PPT adversary $\mathcal{A}_5$ in Hybrid 5 such that*

$$\mathsf{HYB}_4\left(\mathcal{A}_4, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{s}{\cong} \mathsf{HYB}_5\left(\mathcal{A}_5, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* For any PPT adversary $\mathcal{A}_4$ in Hybrid 4, we consider the identical adversary $\mathcal{A}_5$ in Hybrid 5. By the unbounded adaptive simulation soundness of the NIZK argument system (see Definition 5.2.1), it follows that the probability of ending in abort in Hybrid 5 cannot exceed the probability of ending in abort in Hybrid 4 by more than a negligible amount; the lemma follows. □

**Hybrid 6. (Random secret shares of sk for corrupted parties).**

Identical to the previous hybrid, except that the ideal functionality $F_{(\mathsf{pk},\mathsf{sk})}$ is replaced by a slightly modified functionality $F'_{(\mathsf{pk},\mathsf{sk})}$ that samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$ as before, but samples secret shares of sk by first selecting the shares $\{\mathsf{sk}_j\}_{j\in C\cap M}$ for malicious parties *uniformly* at random, and then sampling a random collection of remaining shares $\{\mathsf{sk}_i\}_{i\in C\notin M}$ for honest parties subject to $\mathsf{Rec}(\{\mathsf{sk}_i\}_{i\in C}) = \mathsf{sk}$. $F_{(\mathsf{pk},\mathsf{sk})}$ outputs $(\mathsf{pk}, \mathsf{sk}_i)$ to each party $P_i \in C$.

Note that the adversary now has *no* information about sk (beyond collective access to two ideal decryption functionalities, which only decrypt values submitted by a committee majority).

Ideal functionalities: $F_{\mathsf{crs}}, \mathbf{F}'_{(\mathsf{pk},\mathsf{sk})}, F_i^{\cdot}, F_{\mathsf{NIZK}}, F_{\mathsf{VSS}}, F_{\mathsf{Enc}}^{\mathsf{seed}}, F_{\mathsf{Dec}}^{\mathsf{queries}}, F_{\mathsf{Enc}}^{\mathsf{seed}'}, F_{\mathsf{Dec}}^{\mathsf{ans}}$.

**Lemma 5.4.14.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_5$ in Hybrid 5, there exists a PPT adversary $\mathcal{A}_6$ in Hybrid 6 such that*

$$\mathsf{HYB}_5\left(\mathcal{A}_5, 1^k, z, \{x_i\}_{i=1}^n\right) \equiv \mathsf{HYB}_6\left(\mathcal{A}_6, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* The experiments are identical. This follows by the perfect secrecy property of the $\lceil\frac{2}{3}|C|\rceil$-out-of-$|C|$ secret sharing scheme (Share, Rec), together with the fact that the committee $C$ is guaranteed to be composed of at least $\frac{2}{3}$ fraction honest parties (by Hybrid 1). □

**Hybrid 7. (Abort if any input/index CT is invalid).**

Abort if at the conclusion of the Input Commitment stage there exists $i \in [n]$ for which either of the following properties *fails* to hold:

- All honest parties $P_j \in C_i$ agree on the *same* ciphertexts $\hat{x}_i$ and $\hat{i}$ for party $P_i$'s input and index.
- Each such ciphertext $\hat{x}_i, \hat{i}$ is evaluation-enabled, as per Definition 5.2.3.
- It holds that $\mathsf{Dec}_{\mathsf{sk}}(\hat{i}) = i$.

Define $x_i' := \mathsf{Dec}_{\mathsf{sk}}(\hat{x}_i)$, where $\hat{x}_i$ is the ciphertext agreed upon by honest parties.

Ideal functionalities: $F_{\mathsf{crs}}, F'_{(\mathsf{pk},\mathsf{sk})}, F_i^{\cdot}, F_{\mathsf{NIZK}}, F_{\mathsf{VSS}}, F_{\mathsf{Enc}}^{\mathsf{seed}}, F_{\mathsf{Dec}}^{\mathsf{queries}}, F_{\mathsf{Enc}}^{\mathsf{seed}'}, F_{\mathsf{Dec}}^{\mathsf{ans}}$.

**Lemma 5.4.15.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_6$ in Hybrid 6, there exists a PPT adversary $\mathcal{A}_7$ in Hybrid 7 such that*

$$\mathsf{HYB}_6\left(\mathcal{A}_6, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{s}{\cong} \mathsf{HYB}_7\left(\mathcal{A}_7, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* We consider the same adversary $\mathcal{A}_7 = \mathcal{A}_6$ and argue that the probability of the experiment ending in abort due to the added constraints is negligible in $k$.

Recall that the $i$th input commitment stage currently takes place in the following steps: (1) Party $P_i$ generates $(\hat{x}_i, \pi_i)$, where for honest $P_i$ the proof $\pi_i$ is simulated and provided by the ideal functionality $F_{\mathsf{NIZK}}$, and for malicious $P_i$ the experiment aborts if $\pi_i$ verifies but the statement is false, (2) Party $P_i$ samples a random secret sharing polynomial $f(\cdot)$ with $f(0) = (\hat{x}_i, \pi_i)$ and submits it to the ideal VSS functionality $F_{\mathsf{VSS}}$, (3) The parties of $C_i$ run the reconstruction procedure on the shares $\{f(j)\}_{j \in C_i}$, where $f(j)$ is the value received by party $P_j$ from $F_{\mathsf{VSS}}$, (5) Each party $P_j$ sets his local version of the encrypted input $i$ to be the reconstructed value $\hat{x}_i$ if $\pi_i$ is valid, and to the default ciphertext $\hat{0}$ otherwise, and (6) All parties in $C_i$ receive a good encryption $\hat{i}$ of the index $i$ from the ideal functionality $F_{\hat{i}}$.

By the definition of $F_{\hat{i}}$ and the certifiability of the FHE scheme with respect to the set of randomness $R$, we immediately have that $\hat{i}$ is evaluation-enabled and $\mathsf{Dec}_{\mathsf{sk}}(\hat{i}) = i$. Since $F_{\hat{i}}$ delivers the same ciphertext to all parties, all honest parties agree on $\hat{i}$.

By the reconstruction property of the polynomial sharing from the VSS scheme, *all* honest parties will reconstruct and agree on the original pair $(\hat{x}_i, \pi_i)$. For honest $P_i$, who follow the prescribed protocol, the ciphertext $\hat{x}_i$ will be evaluation-enabled. For malicious $P_i$, one of two cases will occur. If the corresponding proof $\pi_i$ verifies, then it must hold that the ciphertext $\hat{x}_i$ is evaluation-enabled (otherwise the experiment aborts, by Hybrid 5); if $\pi_i$ does not verify, then the honest parties $P_j \in C_i$ will all set $\hat{x}_i$ to be the default ciphertext $\hat{0}$, which is evaluation-enabled by construction.

□

**Hybrid 8. (Abort if permutation CTs are bad).**

Abort if at the conclusion of the Shuffle Generation stage (and after executing $\mathsf{ComBroadcast}$), any of the following *fails* to hold:

- *All* honest parties $P_i \in [n]$ agree on the *same* permutation ciphertext values $\{\hat{B}_\ell\}_{\ell \in [q]}$
- Each such ciphertext $\hat{B}_\ell$ is evaluation-enabled, as per Definition 5.2.3.

Define $B_\ell := \mathsf{Dec}_{\mathsf{sk}}(\hat{B}_\ell)$ for each $\ell \in [q]$.
Denote by $\rho$ the $[n]$-permutation defined by implementing the switching network $SN$ with bits $\{B_\ell\}_{\ell \in [q]}$.

Ideal functionalities: $F_{\mathsf{crs}}, F'_{(\mathsf{pk},\mathsf{sk})}, F_{\hat{i}}, F_{\mathsf{NIZK}}, F_{\mathsf{VSS}}, F^{\mathsf{seed}}_{\mathsf{Enc}}, F^{\mathsf{queries}}_{\mathsf{Dec}}, F^{\mathsf{seed}'}_{\mathsf{Enc}}, F^{\mathsf{ans}}_{\mathsf{Dec}}$.

**Lemma 5.4.16.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_7$ in Hybrid 7, there exists a PPT adversary $\mathcal{A}_8$ in Hybrid 8 such that*

$$\mathsf{HYB}_7\left(\mathcal{A}_7, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{s}{\cong} \mathsf{HYB}_8\left(\mathcal{A}_8, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* We consider the same adversary $\mathcal{A}_8 = \mathcal{A}_7$ and argue that the probability of the experiment ending in abort due to the added constraints is negligible in $k$.

Recall that in the shuffle generation procedure, each party $P_j \in C$ encrypts a sequence of bits and VSSes corresponding ciphertex-proof pairs $(\hat{b}_\ell^j, \pi_\ell^j)$ to the parties in $C$; these values are then reconstructed from the secret shares, each party locally verifies the reconstructed proofs, and for each $\ell \in [q]$ the ciphertexts $\{\hat{b}_\ell^j\}_{j\in C}$ with valid proofs are added (xor) homomorphically to yield a final ciphertext $\hat{B}_\ell$.

By an identical argument to that of the previous step (Lemma 5.4.15), it follows that for each $j \in C$, all honest parties $P_i \in C$ agree on the values $\{(\hat{b}_\ell^j, \pi_\ell^j)\}_{\ell\in[q]}$ reconstructed from party $P_j$'s VSS, and further, each ciphertext $\hat{b}_\ell^j$ for which the proof $\pi_\ell^j$ verifies correctly is guaranteed to be evaluation-enabled. We refer to these below as "valid" ciphertexts.

Note that since each honest party $P_i \in C$ agrees on the received ciphertexts and proofs, they also agree on the subset of ciphertexts which are valid. Thus, since homomorphic evaluation is deterministic, they will also all agree on the value of $\hat{B}_\ell := \mathsf{Eval}(\{\hat{b}_\ell^j\}_{\mathsf{valid}}, \bigoplus)$ for each $\ell \in [q]$. Further, since each valid ciphertext $\hat{b}_\ell^j$ is evaluation-enabled, the resulting evaluated ciphertexts $\hat{B}_\ell$ will also be evaluation-enabled.

Finally, recall that a strict majority of parties in the committee $C$ are honest (by Hybrid 1), and thus will initiate $\mathsf{ComBroadcast}(\{\hat{B}_\ell\}_{\ell\in[q]})$ for this same set of ciphertexts. Thus, by Lemma 5.4.8, it holds that at the conclusion of the ComBroadcast execution, *all* parties in $[n]$ will agree on the collection of ciphertexts $\{\hat{B}_\ell\}_{\ell\in[q]}$.

□

**Hybrid 9. (Abort if post-swapped CTs invalid).**

Abort if at the conclusion of the Oblivious Shuffling stage there exists $i \in [n]$ for which any of the three following properties *fails* to hold:

- All honest parties $P_j \in C_i$ agree on the *same* final input and index ciphertexts $\hat{p}_{\mathsf{final}}^i, \hat{x}_{\mathsf{final}}^i$.
- Both ciphertexts $\hat{p}_{\mathsf{final}}^i, \hat{x}_{\mathsf{final}}^i$ are evaluation-enabled, as per Definition 5.2.3.

Define $p_{\mathsf{final}}^i = \mathsf{Dec}_{\mathsf{sk}}(\hat{p}_{\mathsf{final}}^i)$ and $x_{\mathsf{final}}^i = \mathsf{Dec}_{\mathsf{sk}}(\hat{x}_{\mathsf{final}}^i)$.

- It holds that
$$p_{\mathsf{final}}^i = \rho^{-1}(i), \quad x_{\mathsf{final}}^i = x'_{\rho^{-1}(i)},$$
where $x'_j$ is defined for each $j$ by party $P_j$'s committed ciphertext $\hat{x}_j$ as in Hybrid 7, and $\rho$ is the permutation defined from the homomorphically added ciphertexts $\{\hat{B}_\ell\}_{\ell\in[q]}$, as defined in Hybrid 8.

Ideal functionalities: $F_{\mathsf{crs}}, F'_{(\mathsf{pk},\mathsf{sk})}, F_{\hat{i}}, F_{\mathsf{NIZK}}, F_{\mathsf{VSS}}, F^{\mathsf{seed}}_{\mathsf{Enc}}, F^{\mathsf{queries}}_{\mathsf{Dec}}, F^{\mathsf{seed}'}_{\mathsf{Enc}}, F^{\mathsf{ans}}_{\mathsf{Dec}}$.

**Lemma 5.4.17.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_8$ in Hybrid 8, there exists a PPT adversary $\mathcal{A}_9$ in Hybrid 9 such that*

$$\mathsf{HYB}_8\left(\mathcal{A}_8, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{s}{\cong} \mathsf{HYB}_9\left(\mathcal{A}_9, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* We prove the probability that the experiment ends in abort due to the added constraints is negligible in $k$, and thus the lemma holds for the same adversary $\mathcal{A}_9 = \mathcal{A}_8$.

This claim is proved inductively, over the iterated executions of the Committee Swap protocol. As a result of Hybrids 7 and 8, it holds that all original input and index ciphertexts $\{\hat{x}_i, \hat{i}\}_{i\in[n]}$, and all permutation ciphertexts $\{\hat{B}_\ell\}_{\ell\in[q]}$ are all evaluation-enabled, and are agreed upon by the relevant honest parties (i.e., honest parties in $C_i$ for the $i$th input/index ciphertexts, and *all* honest parties for the permutation ciphertexts).

In each swap evaluation $\ell$, the parties select the most frequently occurring index and input ciphertext pairs, which will be the ciphertexts held (and agreed upon) by honest parties, who form a majority of each committee. Thus, the homomorphically evaluated swapped ciphertext values will also be agreed upon by all honest parties in the relevant pair of committees, they will be evaluation-enabled, and they will correctly decrypt to the original values in *swapped order* iff $B_\ell = 1$.

At the conclusion of all swaps $\ell = 1, ..., q$, it thus holds that the honest parties in each $C_i$ agree on the same final ciphertexts $\hat{p}^i_{\mathsf{final}}, \hat{x}^i_{\mathsf{final}}$, which are evaluation-enabled. Further, by the correctness of homomorphic evaluation, the original values $(i, x'_i)$ for each $i$ are now stored within the ciphertexts $\hat{p}^j_{\mathsf{final}}, \hat{x}^j_{\mathsf{final}}$ for which $\rho(i) = j$: or, equivalently, $\mathsf{Dec}_{\mathsf{sk}}(\hat{p}^i_{\mathsf{final}}) = \rho^{-1}(i)$ and $\mathsf{Dec}_{\mathsf{sk}}(\hat{x}^i_{\mathsf{final}}) = x'_{\rho^{-1}(i)}$.

□

**Hybrid 10. (Replace Input Selection stage with ideal functionality $F^{\mathsf{seed}}_{\rho(I)}$).**

Recall that in the previous hybrid, the Input Selection stage took place by each party in $C$ doing the following: (1) submitting pk to an ideal functionality $F^{\mathsf{seed}}_{\mathsf{Enc}}$, and receiving back an encrypted PRG seed; (2) locally performing homomorphic evaluations on this ciphertext (to select and permute a set of query indices, using randomness prg(seed), all under the encryption layer); and then (3) submitting the resulting evaluated ciphertexts to a second ideal functionality $F^{\mathsf{sel}}_{\mathsf{Dec}}$, who decrypts ciphertexts submitted by a majority of parties in $C$.

In this hybrid, these 3 steps are replaced by the following *single* ideal functionality $F^{\mathsf{seed}}_{\rho(I)}$:

Input: pk

Compute:

1. Sample seed $\leftarrow \{0,1\}^k$ and $r \leftarrow R$. Encrypt $\widehat{\mathsf{seed}} = \mathsf{Enc}_{\mathsf{sk}}(\mathsf{seed}; r)$.
2. Sample query indices $I = \mathsf{SLA.Sel}(\mathsf{prg}(\mathsf{seed}))$, using randomness prg(seed).

3. For each of the $Q$ selected indices $p \in I$, apply the permutation $\rho$, as defined in Hybrid 8.

Output: To all parties (in $C$): $\widehat{\mathsf{seed}}$, and the set of $Q$ indices $\rho(I)$.

Note that this essentially amounts to replacing the homomorphic evaluation on ciphertexts by an ideal functionality that computes the same functions on the unencrypted plaintexts.

Ideal functionalities: $F_{\mathsf{crs}}, F'_{(\mathsf{pk},\mathsf{sk})}, F_{\hat{i}}, F_{\mathsf{NIZK}}, F_{\mathsf{VSS}}, \mathbf{F}^{\mathsf{seed}}_{\rho(\mathbf{I})}, F^{\mathsf{seed}'}_{\mathsf{Enc}}, F^{\mathsf{ans}}_{\mathsf{Dec}}$.

**Lemma 5.4.18.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_9$ in Hybrid 9, there exists a PPT adversary $\mathcal{A}_{10}$ in Hybrid 10 such that*

$$\mathsf{HYB}_9\left(\mathcal{A}_9, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{s}{\cong} \mathsf{HYB}_{10}\left(\mathcal{A}_{10}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Consider any PPT adversary $\mathcal{A}_9$ in Hybrid 9. Consider the adversary $\mathcal{A}_{10}$ in Hybrid 10 who performs the following steps:

The adversary $\mathcal{A}_{10}$:

1. Until the start of the input selection stage, simulate the actions of $\mathcal{A}_9$ exactly.
2. Submit pk received from $F'_{(\mathsf{pk},\mathsf{sk})}$ to the Hybrid 10 functionality $F^{\mathsf{seed}}_{\rho(I)}$. Denote the response as $\widehat{\mathsf{seed}}, \rho(I)$.
3. Send $\widehat{\mathsf{seed}}$ to $\mathcal{A}_9$, as the simulated response from the Hybrid 9 functionality $F^{\mathsf{seed}}_{\mathsf{Enc}}$.
4. Simulate the actions of $\mathcal{A}_9$ up to the point where $\mathcal{A}_9$ submits his values to the Hybrid 9 functionality $F^{\mathsf{sel}}_{\mathsf{Dec}}$. Send $\rho(I)$ to $\mathcal{A}_9$ as the simulated response.
5. Continue to exactly simulate the actions of $\mathcal{A}_9$ for the remainder of the protocol.

In order to show that $\mathcal{A}_{10}$ correctly simulates the output distribution of $\mathcal{A}_9$ in Hybrid 9, it suffices to show that the output $\rho(I)$ from $F^{\mathsf{seed}}_{\rho(I)}$ has the same distribution as the output of $F^{\mathsf{sel}}_{\mathsf{Dec}}$ in the Hybrid 9 experiment. Namely, we must show that with overwhelming probability, the ciphertext submitted to the decryption functionality $F^{\mathsf{sel}}_{\mathsf{Dec}}$ in Hybrid 9 by a *majority* of parties in $C$ (in particular, by the honest parties in $C$) necessarily decrypts to the value $\rho(I)$ that is computed by $F^{\mathsf{seed}}_{\rho(I)}$. This will follow by the correctness of homomorphic evaluation of the FHE scheme.

More formally, note that the ideal functionality $F^{\mathsf{seed}}_{\mathsf{Enc}}$ in Hybrid 9 produces and distributes an evaluation-enabled ciphertext $\widehat{\mathsf{seed}}$. This implies that the homomorphically evaluated ciphertext $\hat{I}$ of input indices to be queried (as defined in Figure 5.5) is also evaluation-enabled, and with overwhelming probability it satisfies $\mathsf{Dec}_{\mathsf{sk}}(\hat{I}) = I := \mathsf{SLA.Sel}(\mathsf{prg}(\mathsf{seed}))$ as expected, where $\mathsf{seed} := \mathsf{Dec}_{\mathsf{sk}}(\widehat{\mathsf{seed}})$.

Recall from Hybrid 8 that all honest parties in $C$ agree on the permutation bit ciphertexts $\{\hat{B}_\ell\}_{\ell\in[q]}$, and each of the ciphertexts is evaluation-enabled. Thus, all honest parties in $C$ will compute the same homomorphic evaluation $\hat{K}$ for the permuted set of query indices (as defined in Figure 5.5), which in turn will be evaluation-enabled. And, by the correctness of homomorphic evaluation on evaluation-enabled ciphertexts, $\hat{K}$ will (with overwhelming probability) satisfy $\mathsf{Dec}_{\mathsf{sk}}(\hat{K}) = \rho(I)$, since $\rho$ is exactly the permutation given by evaluating the swaps $\{B_\ell := \mathsf{Dec}_{\mathsf{sk}}(\hat{B}_\ell)\}_{\ell\in[q]}$.

This means that in Hybrid 9, when all honest parties in $C$ submit the ciphertext $\hat{K}$ to the decryption functionality $F^{\mathsf{sel}}_{\mathsf{Dec}}$, it will respond exactly with $\rho(I)$, as computed by $F^{\mathsf{seed}}_{\rho(I)}$ in Hybrid 10. The lemma follows.

□

**Hybrid 11. (Replace Evaluation stage with ideal functionality $F^{\mathsf{seed}}_{\mathsf{eval}}$).**

Similar to the previous hybrid, except that the homomorphic evaluation on ciphertexts during the evaluation stage is absorbed into the adjacent ideal functionalities. The evaluation stage now takes place as follows.

After the conclusion of the Input Selection stage, the parties in $C$ send "Please send encrypted input $\ell$" to the parties $P_j \in C_\ell$ in the relevant queried committees, and the queried parties $P_j$ broadcast their post-shuffle pair of (index, input) ciphertexts $(\hat{p}^j_\ell, \hat{x}^j_{p_\ell})$ to $C$, as before. However, the members of $C$ no longer proceed by executing the ideal functionality $F^{\mathsf{seed}}_{\mathsf{Enc}}$ to receive $\widehat{\mathsf{seed}}'$, performing homomorphic evaluations on the ciphertexts, and submitting the resulting values to the ideal functionality $F^{\mathsf{ans}}_{\mathsf{Dec}}$. Instead, the members of $C$ now directly forward the received ciphertext pairs (from parties $P_j \in C_\ell$) to a *new* ideal functionality $F^{\mathsf{seed}}_{\mathsf{eval}}$, defined as follows:

Input: Each party $P_i \in C$: $\mathsf{sk}_i$, $(\{\hat{p}^j_\ell\}_{j\in C_\ell}, \{\hat{x}^j_{p_\ell}\}_{j\in C_\ell})_{\ell\in K}$.

Compute:

1. For each $\ell$, take $\hat{p}_\ell = \mathsf{maj}_{j\in C_\ell}\{\hat{p}^j_\ell\}$ and $\hat{x}_{p_\ell} = \mathsf{maj}_{j\in C_\ell}\{\hat{x}^j_{p_\ell}\}$.
2. Reconstruct the FHE decryption key from the input secret shares: $\mathsf{sk} \leftarrow \mathsf{Rec}(\{\mathsf{sk}_i\}_{i\in C})$.
3. For each $\ell$, decrypt $p_\ell = \mathsf{Dec}_{\mathsf{sk}}(\hat{p}_\ell)$ and $x_{p_\ell} = \mathsf{Dec}_{\mathsf{sk}}(\hat{x}_{p_\ell})$.
4. Sample a PRG seed $\mathsf{seed}' \leftarrow \{0,1\}^k$ and encryption randomness $r \leftarrow R$. Encrypt $\widehat{\mathsf{seed}}' = \mathsf{Enc}_{\mathsf{pk}}(\mathsf{seed}', r)$.
5. Evaluate the sublinear algorithm:

$$\mathsf{answer} = \mathsf{SLA.Exec}\left((p_\ell, x_{p_\ell})_{\ell\in K}; \mathsf{prg}(\mathsf{seed}')\right),$$

taking each $x_{p_\ell}$ as the requested input of index $p_\ell$, and using randomness $\mathsf{prg}(\mathsf{seed}')$.

Output: To all parties (in $C$): $\widehat{\mathsf{seed}}'$, $\mathsf{answer}$

Ideal functionalities: $F_{\mathsf{crs}}, F'_{(\mathsf{pk},\mathsf{sk})}, F_{\hat{i}}, F_{\mathsf{NIZK}}, F_{\mathsf{VSS}}, F^{\mathsf{seed}}_{\rho(I)}, \mathbf{F}^{\mathsf{seed}}_{\mathsf{eval}}$.

**Lemma 5.4.19.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_{10}$ in Hybrid 10, there exists a PPT adversary $\mathcal{A}_{11}$ in Hybrid 11 such that*

$$\mathsf{HYB}_{10}\left(\mathcal{A}_{10}, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{s}{\cong} \mathsf{HYB}_{11}\left(\mathcal{A}_{11}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Similar to the previous step, for any PPT Hybrid 10 adversary $\mathcal{A}_{10}$, we define the adversary $\mathcal{A}_{11}$ in Hybrid 11 who simulates the actions of $\mathcal{A}_{10}$ up to the start of the evaluation phase, submits $\mathsf{pk}$ to his functionality $F_{\mathsf{eval}}^{\mathsf{seed}}$ and receives back $\widehat{\mathsf{seed}}'$, $\mathsf{answer}$, and then simulates the responses of the two separate functionalities $F_{\mathsf{Enc}}^{\mathsf{seed}'}, F_{\mathsf{Dec}}^{\mathsf{ans}}$ by responding with $\widehat{\mathsf{seed}}'$ for the first and $\mathsf{answer}$ for the second. The argument for indistinguishability is identical to that of the previous lemma, and holds by the correctness of homomorphic evaluation for evaluation-enabled ciphertexts.

□

**Hybrid 12. (Replace Committee Shuffle, Input Selection, and Evaluation stages with single ideal functionality $\mathsf{Eval\text{-}leak}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I))$).**

In this hybrid, after parties commit to their input/index ciphertexts, and the supreme committee generates and agrees upon the permutation ciphertexts, *all* parties submit the ciphertexts they hold to one large ideal functionality. This functionality evaluates the sublinear algorithm on the inputs defined by the ciphertexts (and randomness generated via a PRG) and outputs the final evaluation $\mathsf{answer}$ together with additional "leaked" information: FHE encryptions $\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}'$ of the PRG seeds that were used to generate the randomness used in the execution, and the permuted index set $\rho(I)$ where $I$ is the set of inputs used in the evaluation, and $\rho$ is the permutation defined by the ciphertexts input to the functionality.

In essence, this hybrid replaces all the homomorphic evaluations of the committee shuffle phase, and the $F_{\rho(I)}^{\mathsf{seed}}, F_{\mathsf{Eval}}^{\mathsf{seed}}$ functionalities, with a single functionality that correctly performs all these steps, and outputs the same collection of values ($\mathsf{answer}, \widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}'$, and $\rho(I)$).

More formally, Hybrid 12 proceeds as follows.

1. All parties receive a CRS from the ideal functionality $F_{\mathsf{crs}}$ and execute the committee election protocol (aborting if the committees are not "good" as defined in Hybrid 1). The ideal functionality $F'_{(\mathsf{pk},\mathsf{sk})}$ outputs public keys $\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}$ for the FHE and standard CPA-secure encryption schemes to all parties, and secret shares $\mathsf{sk}_i$ of the corresponding FHE secret key to the members of the supreme committee (where the shares $\mathsf{sk}_i$ given to corrupt parties are completely random).
2. In the input commitment phase, each party $P_i$ generates an encryption-and-proof tuple $(\hat{x}_i, \pi_i, \hat{x}_i^{\mathsf{CPA}}, \pi_i^{\mathsf{CPA}})$ as in the previous hybrids and VSSes this value to his committee $C_i$, via the ideal functionality $F_{\mathsf{VSS}}$. (Recall that for honest parties, the proofs are simulated and generated by the ideal functionality $F_{\mathsf{NIZK}}$). The value is then reconstructed by the members of $C_i$ (and those ciphertexts with invalid proofs are replaced by the default ciphertext $\hat{0}$). In addition, each party in $C_i$ receives an encryption $\hat{i}$ of the index $i$ from the ideal functionality $F_{\hat{i}}$.
3. Next, each party $P_i$ in the supreme committee $C$ samples random bits $\{b_\ell^i\}_{\ell\in[q]}$ (one for each swap gate in the switching network), generates an encryption-and-proof tuple $(\hat{b}_\ell^i, \pi_\ell^i, (\hat{b}_\ell^i)^{\mathsf{CPA}}, (\pi_\ell^i)^{\mathsf{CPA}})$ as in the previous hybrids, and VSSes this tuple to the committee $C$ via $F_{\mathsf{VSS}}$. The value is then reconstructed by the members of $C$.

4. At this point, all parties submit their reconstructed ciphertexts $\hat{x}_i^j$ (for $P_j \in C_i$) and/or $\{\hat{B}_\ell^j\}_{\ell\in[q]}$ (for $P_j \in C$) to an ideal functionality $\mathsf{Eval\text{-}leak}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I))$ described below, and receive the described outputs.

Functionality $\mathsf{Eval\text{-}leak}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I))$:

Input: Each party $P_j \in C_i$: $\{\hat{x}_i^j\}$ (for each $i \in [n]$).
Each party $P_j \in C$: $\{\hat{B}_\ell^j\}_{\ell\in[q]}$.

Compute:

1. Take the majority votes: for each $i \in [n]$, take $\hat{x}_i := \mathsf{maj}_{j\in C_i}\{\hat{x}_i^j\}$, for each $\ell \in [q]$, take $\hat{B}_\ell := \mathsf{maj}_{j\in C}\{\hat{B}_\ell^j\}$.
2. Decrypt these ciphertexts: $x_i := \mathsf{Dec}_{\mathsf{sk}}(\hat{x}_i)$ and $B_\ell := \mathsf{Dec}_{\mathsf{sk}}(\hat{B}_\ell)$. (Note that the secret key $\mathsf{sk}$ is hard-coded into the ideal functionality).
3. Sample $\mathsf{seed}, \mathsf{seed}' \leftarrow \{0,1\}^k$ and $r, r' \leftarrow R$ (recall $R$ is the set of "good" randomness for the FHE scheme).
4. Compute $I = \mathsf{SLA.Sel}(\mathsf{prg}(\mathsf{seed}))$.
5. Compute $\mathsf{answer} = \mathsf{SLA.Exec}(\{x_i\}_{i\in I}; \mathsf{prg}(\mathsf{seed}'))$.
6. Compute the permuted indices $\rho(I)$, where $\rho$ is the permutation defined by the swap bits $\{B_\ell\}_{\ell\in[q]}$ in the switching network $SN$.
7. Encrypt $\widehat{\mathsf{seed}} = \mathsf{Enc}_{\mathsf{pk}}(\mathsf{seed}; r)$ and $\widehat{\mathsf{seed}}' = \mathsf{Enc}_{\mathsf{pk}}(\mathsf{seed}'; r')$.

Output: To all parties *in* $C$: $\mathsf{answer}, \widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I)$.

Ideal functionalities: $F_{\mathsf{crs}}, F'_{(\mathsf{pk},\mathsf{sk})}, F_{\hat{i}}, F_{\mathsf{NIZK}}, F_{\mathsf{VSS}}, \mathsf{Eval\text{-}leak}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I))$.

**Lemma 5.4.20.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_{11}$ in Hybrid 11, there exists a PPT adversary $\mathcal{A}_{12}$ in Hybrid 12 such that*

$$\mathsf{HYB}_{11}\left(\mathcal{A}_{11}, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_{12}\left(\mathcal{A}_{12}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Note that all replaced steps (namely, the homomorphic evaluations on public ciphertexts during the committee shuffle phase, and the communication of public values while interfacing between Hybrid 11 ideal functionalities in the input selection and evaluation phases) are completely based on non-secret values, and can be simulated honestly.

Similar to the previous hybrids, indistinguishability of the two hybrid experiments holds since a majority of parties in each committee are honest (so that at each state, a majority of parties will agree on the same honestly evaluated ciphertexts), that each of the starting ciphertexts is evaluation-enabled (as proved in the previous hybrids), and that correctness of homomorphic evaluation holds for evaluation-enabled ciphertexts. Further, since a majority of parties in the supreme committee are honest, then by the correctness property of the secret sharing scheme, the value reconstructed from parties' secret shares $\mathsf{sk}_i$ of the FHE secret key (as used by the ideal functionalities to decrypt in Hybrid 11) is equal to the correct secret key $\mathsf{sk}$ (as used to decrypt in the present functionality).

□

**Hybrid 13. (Honest parties replace redundant CPA ciphertexts with $\mathsf{Enc}_{\mathsf{pk}^{\mathsf{CPA}}}(0)$).**

Same as the previous hybrid, except that during the input commitment phase, honest parties no longer generate a second encryption of their input under the standard CPA-secure encryption scheme. Similarly, in the shuffle generation phase, honest parties no longer generate a second encryption of their bits $b_\ell$. Instead, in each place they simply provide a fresh encryption $\mathsf{Enc}_{\mathsf{pk}^{\mathsf{CPA}}}(0)$ of the value 0.

Ideal functionalities: $F_{\mathsf{crs}}, F'_{(\mathsf{pk},\mathsf{sk})}, F^i_i, F_{\mathsf{NIZK}}, F_{\mathsf{VSS}}, \mathsf{Eval\text{-}leak}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I))$.

**Lemma 5.4.21.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_{12}$ in Hybrid 12, there exists a PPT adversary $\mathcal{A}_{13}$ in Hybrid 13 such that*

$$\mathsf{HYB}_{12}\left(\mathcal{A}_{12}, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_{13}\left(\mathcal{A}_{13}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* For any PPT adversary $\mathcal{A}_{12}$ in Hybrid 12, consider the adversary $\mathcal{A}_{13}$ in Hybrid 13 who generates *simulated* CPA ciphertexts $\hat{x}_i^{\mathsf{CPA}} \leftarrow \mathsf{Enc}^{\mathsf{CPA}}_{\mathsf{pk}^{\mathsf{CPA}}}(0)$ and $(\hat{b}^i_\ell)^{\mathsf{CPA}} \leftarrow \mathsf{Enc}^{\mathsf{CPA}}_{\mathsf{pk}^{\mathsf{CPA}}}(0)$ on behalf of each honest party $P_i$ (and $\ell \in [q]$), and otherwise simulates the actions of $\mathcal{A}_{12}$ exactly.

The indistinguishability of the simulated output holds directly by the semantic (CPA) security of the underlying encryption scheme $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$, together with a standard hybrid argument. Indeed, since the remainder of the hybrid experiment execution is independent of the secret key $\mathsf{sk}^{\mathsf{CPA}}$, any successful distinguisher between experiment outputs can be used to distinguish between encryptions of 0 and encryptions of the corresponding correct value (which is known by the reduction).

□

**Hybrid 14. (Parties submit only *plaintext* inputs $x_i$ and $b_\ell$ to ideal functionality).**

Two changes take place from the previous hybrid.

First, the ideal functionality $F'_{(\mathsf{pk},\mathsf{sk})}$ is replaced by a modified version $F^{\mathsf{FHE}}_{\mathsf{pk}}$, which no longer samples a public key $\mathsf{pk}^{\mathsf{CPA}}$ for the standard CPA-secure encryption scheme, and which no longer outputs shares of the secret key of the FHE scheme. Instead, the functionality samples a key pair $(\mathsf{pk},\mathsf{sk})$ for the FHE scheme and outputs *only* $\mathsf{pk}$ to all parties.

Second, the input and shuffle generation phases no longer take place, and instead the ideal functionality $\mathsf{Eval\text{-}leak}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I))$ from the previous hybrid is replaced by a modified version $\mathsf{Eval\text{-}leak}^{\mathsf{PT}}_{\mathsf{pk}}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I), \{\hat{x}_i\}_{i\in[n]}, \{\hat{b}^i_\ell\}_{i\in C, \ell\in[q]})$, which directly accepts *plaintext* values $x_i, \{b^i_\ell\}$. Specifially:

- Input phase: In the previous hybrid, each party $P_i$ generated a ciphertext-and-proof tuple $(\hat{x}_i, \pi_i, \hat{x}_i^{\mathsf{CPA}}, \pi_i^{\mathsf{CPA}})$ and used the VSS functionality $F_{\mathsf{VSS}}$ to share the tuple among the parties in his input committee $C_i$. The parties of $C_i$ then exchanged shares, reconstructed the ciphertext-and-proof tuple, and each submitted the ciphertext $\hat{x}_i$ to the Hybrid 13 ideal functionality. Now, instead, each party $P_i$ simply submits his input $x_i$ directly to the new ideal functionality in Hybrid 14.
- Shuffle generation phase: In the previous hybrid, each party $P_i \in C$ in the supreme committee sampled random bits $\{b_\ell^i\}_{\ell\in[q]}$ for each swap gate in the switching network $SN$, then for each generated a ciphertext-and-proof tuple $(\hat{b}_\ell^i, \pi_\ell^i, (\hat{b}_\ell^i)^{\mathsf{CPA}}, (\pi_\ell^i)^{\mathsf{CPA}})$, and used the VSS functionality $F_{\mathsf{VSS}}$ to share the tuple among the parties in $C$. The parties of $C$ then exchanged shares, reconstructed all the ciphertext-and-proof tuples, and each submitted the collection of ciphertexts $\{\hat{b}_\ell^i\}_{i\in C,\ell\in[q]}$ to the Hybrid 13 ideal functionality. Now, instead, each party $P_i \in C$ simply samples bits $\{b_\ell^i\}_{\ell\in[q]}$ and submits them *directly* to the new ideal functionality in Hybrid 14.

Note that the ideal functionalities $F_{\mathsf{crs}}, F_{\hat{i}}, F_{\mathsf{NIZK}}$, and $F_{\mathsf{VSS}}$ from Hybrid 13 are no longer present in Hybrid 14.

The new ideal functionality $\mathsf{Eval\text{-}leak}_{\mathsf{pk}}^{\mathsf{PT}}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I), \{\hat{x}_i\}_{i\in[n]}, \{\hat{b}_\ell^i\}_{i\in C,\ell\in[q]})$ is hardcoded with the FHE public key $\mathsf{pk}$. Upon input $\{x_i\}_{i\in[n]}$, it outputs FHE encryptions of each of the received values, $\{\hat{x}_i\}_{i\in[n]}$. Upon input $\{b_\ell^i\}_{i\in C,\ell\in[q]}$, it evaluates the sublinear algorithm on inputs $\{x_i\}_{i\in[n]}$ (using randomness from a PRG), outputs $\mathsf{answer}$, $\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}'$, and $\rho(I)$ as before, and in addition outputs FHE encryptions of each of the received swap bit values, $\{\hat{b}_\ell^i\}_{i\in C,\ell\in[q]}$. Explicitly, the new ideal functionality is defined as follows:

Functionality $\mathsf{Eval\text{-}leak}_{\mathsf{pk}}^{\mathsf{PT}}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I), \{\hat{x}_i\}_{i\in[n]}, \{\hat{b}_\ell^i\}_{i\in C,\ell\in[q]})$:

Input 1: Each party $P_i$: plaintext input value $x_i$.

Compute 1: For each $i \in [n]$, let $\hat{x}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_i)$ (with respect to hardcoded FHE key $\mathsf{pk}$).

Output 1: $\{\hat{x}_i\}_{i\in[n]}$.

Input 2: Each party $P_j \in C$: plaintext bits $\{b_\ell^j\}_{\ell\in[q]}$ for the $q$ swap gates.

Compute 2:

1. Evaluate sublinear algorithm on above inputs $\{x_i\}_{i\in[n]}$ (using PRG randomness):
   (a) Sample $\mathsf{seed}, \mathsf{seed}' \leftarrow \{0,1\}^k$.
   (b) Compute $I = \mathsf{SLA.Sel}(\mathsf{prg}(\mathsf{seed}))$.
   (c) Compute $\mathsf{answer} = \mathsf{SLA.Exec}(\{x_i\}_{i\in I}; \mathsf{prg}(\mathsf{seed}'))$.
2. Compute the additional "leaked" information:
   All encryptions are with respect to the hardcoded FHE key $\mathsf{pk}$.
   (a) Encrypt $\widehat{\mathsf{seed}} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathsf{seed})$ and $\widehat{\mathsf{seed}}' \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathsf{seed}')$.

(b) For each $i \in C, \ell \in [q]$ , encrypt the received bit $b^i_\ell$ (submitted by party $P_i \in C$) as $\hat{b}^i_\ell \leftarrow \mathsf{Enc}_{\mathsf{pk}}(b^i_\ell)$.

(c) For each swap gate $\ell \in [q]$ in the switching network $SN$, take $B_\ell = \bigoplus_{i \in C} b^i_\ell$. Denote by $\rho$ the permutation on $[n]$ induced by swap bits $\{B_\ell\}_{\ell \in [q]}$.

(d) Compute the permuted indices $\rho(I)$.

Output 2: $\mathsf{answer}, \widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I), \{\hat{b}^i_\ell\}_{i \in C, \ell \in [q]}$.

Ideal functionalities: $\mathsf{Eval\text{-}leak}^{\mathsf{PT}}_{\mathsf{pk}}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I), \{\hat{x}_i\}_{i \in [n]}, \{\hat{b}^i_\ell\}_{i \in C, \ell \in [q]})$.

**Lemma 5.4.22.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_{13}$ in Hybrid 13, there exists a PPT adversary $\mathcal{A}_{14}$ in Hybrid 14 such that*

$$\mathsf{HYB}_{13}\left(\mathcal{A}_{13}, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_{14}\left(\mathcal{A}_{14}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* At a high level, the reduction must: (1) simulate the VSS sharing of honest parties' secrets $(x_i, b^i_\ell)$, (2) extract corrupted parties' secrets $x_j, b^j_\ell$ from the information they provide during the VSS sharing phase in order to submit the appropriate values to the Hybrid 14 ideal functionality on behalf of corrupted parties, and (3) given the ciphertexts output by the ideal functionality, simulate the remainder of the VSS reconstruction phases.

We now describe how this is done. Let $\mathcal{A}_{13}$ be an any PPT adversary in Hybrid 13. Consider the following adversary in Hybrid 14.

Adversary $\mathcal{A}_{14}$:

1. Simulate $\mathcal{A}_{13}$ exactly until the point where he expects to receive an output from the Hybrid 13 ideal functionality $F'_{(\mathsf{pk},\mathsf{sk})}$.
2. Simulate $F'_{(\mathsf{pk},\mathsf{sk})}$:
   Receive $\mathsf{pk}$ for the FHE scheme from the Hybrid 14 ideal functionality $F^{\mathsf{FHE}}_{\mathsf{pk}}$. For each corrupted party $P_j$ in the supreme committee $C$, sample a *random* secret share $\mathsf{sk}_j$. In addition, sample a key pair $(\mathsf{pk}^{\mathsf{CPA}}, \mathsf{sk}^{\mathsf{CPA}}) \leftarrow \mathsf{Gen}^{\mathsf{CPA}}(1^k)$ for the standard CPA secure encryption scheme. Send $(\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \{\mathsf{sk}_j\}_{j \in C \cap M})$ to $\mathcal{A}_{13}$ as the collective output he expects from the Hybrid 13 ideal functionality $F'_{(\mathsf{pk},\mathsf{sk})}$. (Recall that $M \subset [n]$ denotes the set of malicious parties).
3. Simulate the actions of $\mathcal{A}_{13}$ exactly up until the $F_{\mathsf{VSS}}$ functionality evaluation in the input commitment phase.
4. Simulate $F_{\mathsf{VSS}}$ in input commitment:
   Recall that the functionality $F_{\mathsf{VSS}}$ among a collection of parties $C'$ receives a polynomial $g$ from a single party (the dealer). If $\deg g \leq \frac{2}{3}|C'|$, then $F_{\mathsf{VSS}}$ outputs the evaluation $g(i)$ to each party $P_i \in C'$; otherwise, it outputs $\perp$ to each party in $C'$.
   In Hybrid 13, each party $P_i$ acts as dealer for the $F_{\mathsf{VSS}}$ functionality to share his ciphertext-and-proof input tuple $(\hat{x}_i, \pi_i, \hat{x}^{\mathsf{CPA}}_i, \pi^{\mathsf{CPA}}_i)$ among his input committee $C_i$. $\mathcal{A}_{14}$ simulates this process within Hybrid 14 as follows.

- When the dealer is an *honest* party $P_i$: For each corrupt party $P_j$ in $C_i$, sample a *random* evaluation value $s_j$, and send the collection $\{s_j\}_{j \in C_i \cap M}$ to $\mathcal{A}_{13}$.
- When the dealer is a *corrupted* party $P_i$: Receive a polynomial $g$ from $\mathcal{A}_{13}$ (which he believes to be submitting to the Hybrid 13 functionality $F_{\mathsf{VSS}}$). If $\deg g > \frac{2}{3}|C_i|$, then output 0 to all parties and set $x'_i = 0$.
  Otherwise, do the following. For each corrupted party $P_j$ in $C_i$, send the polynomial evaluation $g(j)$ to $P_j$. *Extract* the shared tuple, by evaluating the polynomial $g$ at the point 0: i.e., parse $(\hat{x}_i, \pi_i, \hat{x}_i^{\mathsf{CPA}}, \pi_i^{\mathsf{CPA}}) = g(0)$. Verify the accompanying proofs $\pi_i, \pi_i^{\mathsf{CPA}}$. If either proof does not verify, then set $x'_i = 0$. Otherwise, extract the underlying *plaintext* input value $x_i$ by decrypting the standard CPA ciphertext $\hat{x}_i^{\mathsf{CPA}}$. That is, take $x'_i = \mathsf{Dec}^{\mathsf{CPA}}_{\mathsf{sk}^{\mathsf{CPA}}}(\hat{x}_i^{\mathsf{CPA}})$, using the secret key $\mathsf{sk}^{\mathsf{CPA}}$ generated in Step 2.

Note that at the conclusion of all simulated $F_{\mathsf{VSS}}$ executions within the input commitment phase, the Hybrid 14 adversary has extracted an input value $x'_j$ for every corrupted party $P_j$.

5. Submit the set of inputs $\{x'_j\}_{j \in M}$ on behalf of the corrupted parties to the Hybrid 14 ideal functionality $\mathsf{Eval\text{-}leak}^{\mathsf{PT}}_{\mathsf{pk}}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I), \{\hat{x}_i\}_{i \in [n]}, \{\hat{b}^i_\ell\}_{i \in C, \ell \in [q]})$. Receive back ciphertexts of *all* parties inputs, $\{\hat{x}_i\}_{i \in [n]}$ (in particular, providing encryptions of the honest parties' inputs).

6. Simulate VSS reconstruction in input commitment:

   - For each corrupt dealer $P_i$: Simulate the actions of honest parties honestly given the shares provided by the simulated $F_{\mathsf{VSS}}$ functionality.
   - For each honest dealer $P_i$: First, $\mathcal{A}_{14}$ simulates a ciphertext-and-proof tuple on behalf of $P_i$. This is done by including the FHE ciphertext $\hat{x}_i$ from the ideal functionality in the previous step, sampling an encryption of 0 in the standard CPA-secure encryption scheme $\hat{0} \leftarrow \mathsf{Enc}^{\mathsf{CPA}}_{\mathsf{pk}^{\mathsf{CPA}}}(0)$, and generating two simulated proofs $\pi_1 \leftarrow \mathcal{S}^{\mathsf{proof}}(\mathsf{crs}, \mathsf{trap}, (\mathsf{pk}, \hat{x}_i))$ and $\pi_2 \leftarrow \mathcal{S}^{\mathsf{proof}}(\mathsf{crs}^{\mathsf{same}}, \mathsf{trap}^{\mathsf{CPA}}, (\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}, \hat{x}_i, \hat{0}))$. The simulated tuple is $(\hat{x}_i, \pi_1, \hat{0}, \pi_2)$.
     Now, $\mathcal{A}_{14}$ must retroactively simulate secret shares that were allegedly given to honest parties during the VSS sharing step. This amounts to sampling a random polynomial $g$ of appropriate degree $(\frac{2}{3}|C_i| - 1)$ consistent with $g(0) = (\hat{x}_i, \pi_1, \hat{0}, \pi_2)$ and where each evaluation $g(j)$ for corrupt parties $P_j \in C_i$ is consistent with the random secret share given to $P_j$ during the simulation of $F_{\mathsf{VSS}}$ in Step 4. At this point, $\mathcal{A}_{14}$ simulates the actions of honest parties honestly given the shares dictated by the polynomial $g$.

7. Simulate $\mathcal{A}_{13}$ exactly up until the $F_{\mathsf{VSS}}$ functionality evaluations in the shuffle generation phase. At this point, repeat Steps 4-6 with the role of $x_i$ replaced by each secret swap bit contribution $b^i_\ell$ (for $\ell \in [q]$). Note that this includes providing the second set of inputs to the ideal functionality $\mathsf{Eval\text{-}leak}^{\mathsf{PT}}_{\mathsf{pk}}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I), \{\hat{x}_i\}_{i \in [n]}, \{\hat{b}^i_\ell\}_{i \in C, \ell \in [q]})$: namely, the collection of plaintext swap bit contributions $\{b^i_\ell\}_{i \in C \cap M, \ell \in [q]}$ on behalf of the corrupted parties in the supreme committee $C$. In response, $\mathcal{A}_{14}$ receives $\mathsf{answer}, \widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}'$, and

$\rho(I)$ (in addition to the ciphertexts $\{b^i_\ell\}_{i\in C\notin M,\ell\in[q]}$ of the corresponding bits submitted by the honest parties, which are used by $\mathcal{A}_{14}$ to simulate the VSS reconstruction procedure in the shuffle generation phase as above).

8. The Hybrid 13 adversary will submit a collection of ciphertext values to what he believes to be his corresponding ideal functionality within the Hybrid 13 experiment. In response, $\mathcal{A}_{14}$ ignores the submitted ciphertexts, and responds with the outputs $(\mathsf{answer}, \widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I))$.
9. From this point on, simulate the actions of $\mathcal{A}_{13}$ exactly.

We now argue that the output of $\mathcal{A}_{14}$ in Hybrid 14 is indistinguishable from the output of $\mathcal{A}_{13}$ in Hybrid 13.

The output of the functionality $F'_{(\mathsf{pk},\mathsf{sk})}$ is simulated perfectly. By the perfect secrecy property of the polynomial secret sharing scheme, the output of the $F_{sfVSS}$ functionality is simulated perfectly. Similarly, since the Hybrid 14 ideal functionality directly provides correct FHE encryptions of the honest parties' inputs (and so the complete ciphertext-and-proof tuples are simulated perfectly on behalf of honest parties), the VSS reconstruction procedure is simulated perfectly.

It remains to show that $\mathcal{A}_{13}$ correctly extracts the inputs $\{x'_j\}_{j\in M}$ and $\{b^j_\ell\}_{j\in C\cap M,\ell\in[q]}$ of corrupted parties (since the remainder of the execution is identical). Consider the case of one $x'_j$; the argument is identical for all other values. By the soundness of the (argued in Hybrid 5), if the proof $\pi^{\mathsf{CPA}}_j$ within the reconstructed tuple $(\hat{x}_i, \pi_j, \hat{x}^{\mathsf{CPA}}_j, \pi^{\mathsf{CPA}}_j)$ verifies correctly for a particular $j \in M$, then it must hold that the two corresponding ciphertexts $\hat{x}_j$ and $\hat{x}^{\mathsf{CPA}}_j$ encrypt the same message. By decryption correctness of the CPA encryption scheme, with overwhelming probability this message will be the value $x'_j$ that is decrypted by $\mathcal{A}_{14}$ in his simulation, as required.

□

**Hybrid 15. (Ideal functionality samples *random* swap bits $\{B_\ell\}_{\ell\in[q]}$).**

Same as the previous hybrid, except that the ideal functionality

$$\mathsf{Eval\text{-}leak}^{\mathsf{PT}}_{\mathsf{pk}}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I), \{\hat{x}_i\}_{i\in[n]}, \{\hat{b}^i_\ell\}_{i\in C,\ell\in[q]})$$

is replaced by a modified version

$$\mathsf{Eval\text{-}leak}^{B_\ell,\mathsf{PT}}_{\mathsf{pk}}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I), \{\hat{x}_i\}_{i\in[n]}, \{\hat{b}^i_\ell\}_{i\in C,\ell\in[q]})$$

which samples *random* swap bits $B_\ell$ for each swap gate $\ell \in [q]$ in the switching network $SN$ in order to define the permutation $\rho$. It still accepts contribution bits $\{b^j_\ell\}_{j\in C\cap M,\ell\in[q]}$ from corrupt parties as before, and now samples random bits $\{b^i_\ell\}_{i\in C\notin M,\ell\in[q]}$ on behalf of honest parties, subject to $\bigoplus_{i\in C} b^i_\ell = B_\ell$ for every $\ell \in [q]$. The functionality computes and outputs all values as before, including encryptions of these newly sampled bits, $\{\hat{b}^i_\ell\}_{i\in C\notin M,\ell\in[q]}$.

Ideal functionalities: $\mathsf{Eval\text{-}leak}^{B_\ell,\mathsf{PT}}_{\mathsf{pk}}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I), \{\hat{x}_i\}_{i\in[n]}, \{\hat{b}^i_\ell\}_{i\in C,\ell\in[q]})$.

**Lemma 5.4.23.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_{14}$ in Hybrid 14, there exists a PPT adversary $\mathcal{A}_{15}$ in Hybrid 15 such that*

$$\mathsf{HYB}_{14}\left(\mathcal{A}_{14}, 1^k, z, \{x_i\}_{i=1}^n\right) \equiv \mathsf{HYB}_{15}\left(\mathcal{A}_{15}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Since honest parties sample their swap bits $b_\ell^i$ at random, and corrupt parties must submit their corresponding bits $b_\ell^j$ at a point when their view is *independent* of the honest parties' bits, the two output distributions are in fact identical.

□

**Hybrid 16. (Replace output FHE ciphertexts $(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \{\hat{x}_i\}_{i\in[n]}, \{\hat{b}_\ell^i\}_{i\in\mathcal{C},\ell\in[q]})$ with $\mathsf{Enc}(0)$).** Similar to the previous hybrid, except the ideal functionality $F_{\mathsf{pk}}^{\mathsf{FHE}}$ is removed, and the ideal functionality $\mathsf{Eval\text{-}leak}_{\mathsf{pk}}^{\mathsf{PT}}(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \rho(I), \{\hat{x}_i\}_{i\in[n]}, \{\hat{b}_\ell^i\}_{i\in\mathcal{C},\ell\in[q]})$ is replaced by a modified version, $\mathsf{Eval\text{-}leak}(\rho(I))$, which *only* accepts plaintext inputs $x_i$, and no longer leaks encrypted information. This new functionality is defined as follows.

Functionality $\mathsf{Eval\text{-}leak}(\rho(I))$:

Input: Each party $P_i$: plaintext input value $x_i$.

Compute:

1. Sample $\mathsf{seed}, \mathsf{seed}' \leftarrow \{0,1\}^k$.
2. Compute $I = \mathsf{SLA.Sel}(\mathsf{prg}(\mathsf{seed}))$.
3. Compute $\mathsf{answer} = \mathsf{SLA.Exec}(\{x_i\}_{i\in I}; \mathsf{prg}(\mathsf{seed}'))$.
4. Sample random bit $B_\ell \leftarrow \{0,1\}$ for each swap gate $\ell \in [q]$ in the switching network $SN$. Denote by $\rho$ the induced permutation on $[n]$.

Output: $\mathsf{answer}, \rho(I)$

Ideal functionalities: $\mathsf{Eval\text{-}leak}(\rho(I))$.

**Lemma 5.4.24.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_{15}$ in Hybrid 15, there exists a PPT adversary $\mathcal{A}_{16}$ in Hybrid 16 such that*

$$\mathsf{HYB}_{15}\left(\mathcal{A}_{15}, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_{16}\left(\mathcal{A}_{16}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* In this step, the FHE ciphertexts $(\widehat{\mathsf{seed}}, \widehat{\mathsf{seed}}', \{\hat{x}_i\}_{i\in[n]}, \{\hat{b}_\ell^i\}_{i\in\mathcal{C},\ell\in[q]})$ that were previously output by the ideal functionality will be *simulated* by encryptions of 0. More explicitly, let $\mathcal{A}_{15}$ be any PPT adversary in Hybrid 15; we consider the following adversary $\mathcal{A}_{16}$ in Hybrid 16. $\mathcal{A}_{16}$ simulates the actions of $\mathcal{A}_{15}$ up until the call to his Hybrid 15 ideal functionality. At this point, $\mathcal{A}_{16}$ calls his Hybrid 16 ideal functionality on the same inputs and receives as output a permuted index set $\rho(I)$. For each ciphertext that $\mathcal{A}_{15}$ expects to receive in addition, $\mathcal{A}_{16}$ samples a fresh encryption of 0, as $\hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$, and submits the collection of all such ciphertexts together with $\rho(I)$ to $\mathcal{A}_{15}$ as the ideal functionality output. $\mathcal{A}_{16}$ then simulates the actions of $\mathcal{A}_{15}$ exactly for the remainder of the protocol.

The indistinguishability of the simulated output follows directly by the semantic security of the FHE scheme, together with a straightforward hybrid argument. Note that the remainder of the protocol execution can be simulated exactly without knowledge of the secret key $\mathsf{sk}$. Thus, any successful distinguisher can be used to distinguish an encryption of 0 from an encryption of the corresponding (known) other value. □

**Hybrid 17. (Replace $\rho$ with a *random* permutation).**

Same as the previous hybrid, except that the ideal functionality $\mathsf{Eval\text{-}leak}(\rho(I))$ is slightly modified. The previous functionality $\mathsf{Eval\text{-}leak}(\rho(I))$ sampled the permutation $\rho$ by choosing random swap bits $\{B_\ell\}$ for the switching network and then taking $\rho$ to be the induced permutation on $[n]$. In this hybrid, the new functionality $\mathsf{Eval\text{-}leak}(\rho'(I))$ samples a *random* permutation $\rho'$ on $[n]$. Namely, $\mathsf{Eval\text{-}leak}(\rho'(I))$ is as follows:

Functionality $\mathsf{Eval\text{-}leak}(\rho'(I))$:

Input: plaintext input values $\{x_i\}_{i\in[n]}$

Compute:

1. Sample a random permutation $\rho' \leftarrow S_{[n]}$.
2. Sample $\mathsf{seed}, \mathsf{seed}' \leftarrow \{0,1\}^k$.
3. Compute $I = \mathsf{SLA.Sel}(\mathsf{prg}(\mathsf{seed}))$.
4. Compute $\mathsf{answer} = \mathsf{SLA.Exec}(\{x_i\}_{i\in I}; \mathsf{prg}(\mathsf{seed}'))$.
5. Compute the permuted indices $\rho'(I)$.

Output: $\mathsf{answer}, \rho'(I)$

Ideal functionalities: $\mathsf{Eval\text{-}leak}(\rho'(I))$.

**Lemma 5.4.25.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_{16}$ in Hybrid 16, there exists a PPT adversary $\mathcal{A}_{17}$ in Hybrid 17 such that*

$$\mathsf{HYB}_{16}\left(\mathcal{A}_{16}, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{s}{\cong} \mathsf{HYB}_{17}\left(\mathcal{A}_{17}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* By Theorem 5.2.11, the distribution of permutations $\rho$ formed by randomly choosing swap bits $\{B_\ell\}$ for the switching network $SN$ is statistically close to uniform over the space of permutations on $[n]$. Thus, taking the same adversary $\mathcal{A}_{17} = \mathcal{A}_{16}$, the lemma follows. □

**Hybrid 18. (Remove $\rho'(I)$ leakage from ideal functionality).**

Same as the previous hybrid, except the $\rho'(I)$ leakage is removed from the output of the ideal functionality. That is, $\mathsf{Eval\text{-}leak}(\rho'(I))$ is replaced by the following functionality $F^{\mathsf{prg}}_{\mathsf{Eval}}$:

Functionality $F^{\mathsf{prg}}_{\mathsf{Eval}}$:

Input: plaintext input values $\{x_i\}_{i\in[n]}$

Compute:

1. Sample $\mathsf{seed}, \mathsf{seed}' \leftarrow \{0,1\}^k$.
2. Compute $I = \mathsf{SLA.Sel}(\mathsf{prg}(\mathsf{seed}))$.
3. Compute $\mathsf{answer} = \mathsf{SLA.Exec}(\{x_i\}_{i\in I}; \mathsf{prg}(\mathsf{seed}'))$.

Output: answer

Ideal functionalities: $F^{\mathsf{prg}}_{\mathsf{Eval}}$.

**Lemma 5.4.26.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_{17}$ in Hybrid 17, there exists a PPT adversary $\mathcal{A}_{18}$ in Hybrid 18 such that*

$$\mathsf{HYB}_{17}\left(\mathcal{A}_{17}, 1^k, z, \{x_i\}_{i=1}^n\right) \equiv \mathsf{HYB}_{18}\left(\mathcal{A}_{18}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Since the permutation $\rho'$ was random and independent of the rest of the view of the adversary in Hybrid 17 (namely, the value answer), the permuted index set $\rho(I)$ can be perfectly simulated by sampling a *random* subset $J \subset [n]$ of size $Q$.

So, for any PPT adversary $\mathcal{A}_{17}$, define $\mathcal{A}_{18}$ to be the adversary who simulates the actions of $\mathcal{A}_{17}$ given both answer and the simulated set $J$. The experiment outputs will be identical.

$\square$

**Hybrid 19. (Pseudorandomness replaced by true randomness).**

Same as the previous hybrid, except the ideal functionality is modified to use true randomness to evaluate SLA.Sel and SLA.Exec, instead of sampling short random seeds and expanding them using the pseudorandom generator. That is, $F^{\mathsf{prg}}_{\mathsf{Eval}}$ is replaced by the following functionality $F^{\mathsf{rand}}_{\mathsf{Eval}}$:

Functionality $F^{\mathsf{rand}}_{\mathsf{Eval}}$:

Input: plaintext input values $\{x_i\}_{i\in[n]}$

Compute:

1. Compute $I \leftarrow \mathsf{SLA.Sel}()$.
2. Compute $\mathsf{answer} \leftarrow \mathsf{SLA.Exec}(\{x_i\}_{i\in I})$.

Output: answer

Note that this is the ideal world experiment.

**Lemma 5.4.27.** *For any auxiliary input $z$ and set of inputs $\vec{x}$, and for every PPT adversary $\mathcal{A}_{18}$ in Hybrid 18, there exists a PPT adversary $\mathcal{A}_{19}$ in Hybrid 19 such that*

$$\mathsf{HYB}_{18}\left(\mathcal{A}_{18}, 1^k, z, \{x_i\}_{i=1}^n\right) \overset{c}{\cong} \mathsf{HYB}_{19}\left(\mathcal{A}_{19}, 1^k, z, \{x_i\}_{i=1}^n\right).$$

*Proof.* Taking $\mathcal{A}_{19} = \mathcal{A}_{18}$, the lemma holds directly by the security of the pseudorandom generator prg, and a 2-step hybrid argument. Indeed, if there exists some auxiliary input $z$ and set of inputs $\vec{x}$ for which there is a successful distinguisher between experiment outputs, then this distinguisher can directly be used to distinguish a pseudorandom string from a random one, by embedding the challenge as the randomness used to implement the ideal functionality. □

□

## 5.5 General MPC with Communication Locality

In this section, we present the remaining main result of the paper: a protocol for *general* secure function evaluation with polylogarithmic communication locality.

**Theorem 5.5.1 (Multiparty evaluation of general functionalities).** *Let $f$ be any polynomial-time randomized functionality on $n$ inputs. Then, for every constant $\epsilon > 0$, there exists an $n$-party protocol $\Pi_f$ that securely computes $f$ tolerating $t < (1/3 - \epsilon)n$ active corruptions, with the following complexities, where $k$ is a security parameter and $|x|$ is the size of the individual inputs held by the honest players:*

*(1) The protocol runs in $\mathsf{poly}(\log n, k)$ rounds.*

*(2) Each honest party talks with at most $\mathsf{polylog}(n)$ other parties.*

*(3) Each message has size of order $|x| \cdot n \cdot \mathsf{poly}(\log n, k)$.*

*(4) The protocol uses a setup consisting of $n \cdot \mathsf{polylog}(n)$ signing keys of size $\mathsf{poly}(k)$, as well as a $\mathsf{poly}(k)$-long additional CRS.*

*The protocol assumes a secure multisignature scheme, a FHE scheme, simulation-sound NIZK arguments, as well as pseudorandom generators.*

The protocol $\Pi_f$ is actually simpler than the one used for sublinear algorithms, as the main challenge of hiding the subset of queried parties disappears, and we allow larger communication complexity. Indeed, oblivious shuffling of inputs is no longer required. Instead, the protocol takes place as follows (formally described in Figure 5.7).

First, one proceeds as in the previous protocol up to the point where the committees $C_i$ hold the encrypted inputs of the parties. At this point, the parties have set up the communication tree structure, elected a supreme committee $C$ and input committees $\{C_i\}_{i\in[n]}$, and generated and agreed upon a public key pk for the fully homomorphic encryption (FHE) scheme. Each party $P_i$ has also verifiably secret shared his (encrypted) input to committee $C_i$.

Next, each $C_i$ will send the encrypted input value $\hat{x_i}$ up to the supreme committee via the communication tree. That is, each party in $C_i$ signs the value of $\hat{x_i}$ under his secret key, and sends the ciphertext-signature pair $(\hat{x_i}, \sigma)$ to the parties in his parent node; these messages are aggregated and forwarded upward at each intermediate node. This ensures that the overall number of parties each honest party talks to remains $\mathsf{polylog}(n)$. Note that while some individual parties do *not* have a good path up to the supreme committee in the communication tree (and thus cannot send his input up to $C$ directly), with overwhelming probability most parties within each input committee $C_i$ *will* have a good path. Thus, all honest parties in the supreme committee $C$ will successfully obtain an encrypted input $\hat{x_i}$ for each $i \in [n]$ (sent by each corresponding committee $C_i$).

Finally, given the set of all encrypted inputs, the supreme committee is able to evaluate the desired functionality $f$. In order to maintain minimal round and communication complexity, this is done in three stages (otherwise, this can simply be achieved by an MPC among $C$ to decrypt and evaluate $f$). First, the parties of $C$ run an MPC to collectively generated an encrypted random seed for the pseudorandom generator $\mathsf{prg}$. Second, each party in $C$ locally evaluates $f$ homomorphically on the encrypted inputs, using (encrypted) randomness formed by homomorphically evaluating $\mathsf{prg}$ on the encrypted seed. Third, the parties collectively decrypt the resulting output value via a second MPC. This final output is then communicated to all parties using the ComBroadcast protocol.

*Proof Sketch.* We now sketch the proof of Theorem 5.5.1. Note that the communication locality of $\Pi_f$ remains $\mathsf{polylog}(n)$, since each party $P_i$ only communicates with his direct neighbors in the communication tree, the parties in his input committee $C_i$, and the parties of $C_j \cup \{P_j\}$ for each $j$ for which $P_i \in C_j$ (which, by the psuedorandomness of the PRF, totals no greater than $\mathsf{polylog}(n)$ parties with overwhelming probability). The round complexity remains $\mathsf{poly}(\log n, k)$, consisting of: $\mathsf{poly}(\log n, k)$ rounds for executing Steps 1-4 of the previous ($\mathsf{poly}(\log n, k)$-round) protocol, $\mathsf{polylog}(n)$ rounds for sending the encrypted inputs up the communication tree to $C$ in parallel, $\mathsf{poly}(k)$ rounds for $C$ to collectively decrypt the homomorphically evaluated output ciphertext via MPC, and $\mathsf{polylog}(n)$ rounds to execute the ComBroadcast protocol to communicate the final answer to all parties.

The communication complexity of $\Pi_f$ is dominated by Step 2, in which the inputs of all parties are transmitted up the communication tree (all other steps require communication that is only $O\big((|x| + n) \cdot \mathsf{poly}(\log n, k)\big)$). In this step, a party may be required to communicate as much as $n$ ciphertexts (and signatures) in each of the $\mathsf{polylog}(n)$ rounds. This constitutes a communication complexity of $O\big(n \cdot |x| \cdot \mathsf{poly}(k) \cdot \mathsf{polylog}(n)\big)$, as desired.

The proof of security of $\Pi_f$ essentially follows from the proof of Theorem 5.4.1. It remains only to prove the following claim.

**Claim 5.5.2.** *With overwhelming probability, at the conclusion of Step 2 of protocol* $\Pi_f$*, all honest parties in the supreme committee* $C$ *agree on the same collection of* $n$ *ciphertexts* $\{\hat{x_i}\}_{i\in[n]}$*. Further, these ciphertexts are the values that were reconstructed in the VSS procedures from the input commitment stage.*

*Proof.* It suffices to show for each $C_i$ that at least $|C_i|/2$ honest parties have a *good* path up to the root node in the communication tree. Indeed, by the properties of the VSS, all honest parties in

**General Secure Evaluation Protocol**

1. Execute Steps 1-4 of the previous protocol.

   Outcome: all parties agree on committees $C, \{C_i\}_{i \in [n]}$ and public keys $\mathsf{pk}, \mathsf{pk}^{\mathsf{CPA}}$ for the FHE and CPA encryption schemes; every member of $C_i$ holds a ciphertext $\hat{x_i}$ of $P_i$'s input $x_i$.

2. The parties in each committee $C_i$ send their held value $\hat{x_i}$ up to the supreme committee via the communication tree. That is,

   (a) For every $i \in [n]$, every $j$ for which $P_j \in C_i$, and every leaf $\mathsf{l}$ of the communication tree for which $P_j \in \mathsf{l}$, the following is done in parallel:
   Party $P_j$ signs the value of $\hat{x_i}$ as $\sigma_{\mathsf{l},j} \leftarrow \mathsf{Sign}_{\mathsf{sk}_{\mathsf{l},j}}(\hat{x_i})$, and sends the tuple $(i, \hat{x_i}, \sigma_{\mathsf{l},j})$ to all parties assigned to the parent node of $\mathsf{l}$.

   (b) For each level $\ell$ of the communication tree (starting at level $\ell = 2$ and proceeding upward to the final root committee $\ell = \ell^*$), do the following in parallel for each node $\mathsf{v}$ on level $\ell$:
   Each party $P_j$ who believes to be assigned to $\mathsf{v}$ broadcasts the collection of all messages received from the children of $\mathsf{v}$ up to all parties assigned to the parent node in level $\ell + 1$.

3. For each party $P_j$ in the supreme committee $C$, $P_j$ takes the ciphertexts $\{\hat{x_i}\}_{i \in [n]}$ to be the received values $c_i$ that were signed by the majority of parties in committee $C_i$.

4. If $f$ is a randomized functionality, the parties of $C$ run an MPC to jointly compute an encryption of a PRG seed for generating the randomness to be used in the execution of $f$. Explicitly, they evaluate the (randomized) functionality $F_{\mathsf{Enc}}^{\mathsf{seed}}$, as described in Figure 5.2.

   Output: To all parties in $C$: encrypted PRG seed, $\widehat{\mathsf{seed}}$.

5. Each party $P_i \in C$ locally evaluates the functionality $f$ homomorphically on the set of his received ciphertexts $\{\hat{x_i}\}_{i \in [n]}$ and using randomness derived from the encrypted seed $\widehat{\mathsf{seed}}$:

$$\widehat{\mathsf{rand}} \leftarrow \mathsf{Eval}(\widehat{\mathsf{seed}}, \mathsf{prg}),$$
$$\widehat{\mathsf{answer}} \leftarrow \mathsf{Eval}\left((\{\hat{x_i}\}_{i \in [n]}, \widehat{\mathsf{rand}}), f\right),$$

   where $f(\{x_i\}, \mathsf{rand})$ denotes the evaluation of $f$ on inputs $\{x_i\}$ and randomness $\mathsf{rand}$.

6. The parties of $C$ run an MPC to collectively decrypt the resulting ciphertext $\widehat{\mathsf{answer}}$ computed by each party $P_i \in C$ (taking the value held by the majority of parties of $C$ as the final output). Explicitly, they evaluate the functionality $F_{\mathsf{Eval}}$, as defined in Figure 5.6, yielding result $\mathsf{answer}$.

7. Each party $P_j \in C$ broadcasts the resulting output to all parties via $\mathsf{ComBroadcast}(\mathsf{answer})$.

**Figure 5.7:** The protocol $\Pi_f$ for securely evaluating a general function $f$ while maintaining $\mathsf{polylog}(n)$ communication locality.

$C_i$ will agree on the same value $\hat{x}_i$ that they will send up the tree, and for each such party $P_j \in C$ that has a good path up the communication tree, this value $\hat{x}_i$ will be received by all honest parties in $C$ together with a signature on $\hat{x}_i$ from $P_j$

But, this will hold directly by the pseudorandomness of the pseudorandom function $F$ that defines the committees $C_i$. Namely, the completed execution of the communication tree setup defines a subset $S \subset [n]$ of an $o(1)$-fraction of parties such that every party outside of $S$ possesses a good path up to the root node (by Theorem 5.4.2). If $F$ were a truly random function, then for any constant $\eta > 0$ (say, $\eta = 1/10$), the probability that any $F(i)$ defines a set $C_i$ in which more than an $\eta$ fraction of its $\log^2 n$ parties are in $S$ is negligible in $n$, by a straightforward Chernoff bound. Taking a union bound, this holds simultaneously for all $i \in [n]$. Thus, the same must hold for a pseudorandom $F$ with overwhelming probability over a random seed $s$; otherwise, one could construct an efficient distinguisher who first simulates the communication tree setup on his own (mimicking the actions of the adversarial parties according to $\mathcal{A}$), identifies the "disconnected" set $S \subset [n]$ of parties without good paths to the root node, and outputs "pseudorandom" iff the evaluation of the challenge function $F(i)$ on any of the inputs $i \in [n]$ yields a committee with more than $\eta$ fraction in $S$. The claim follows.

$\square$

$\square$

# Bibliography

[ADN$^+$10] Joël Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs. Public-key encryption in the bounded-retrieval model. In *EUROCRYPT*, pages 113–134, 2010.

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[AGH12] Adi Akavia, Shafi Goldwasser, and Carmit Hazay. Distributed public key schemes secure against continual leakage. In *PODC*, pages 155–164, 2012.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[AJL$^+$12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *EUROCRYPT*, pages 483–501, 2012.

[AJLA$^+$12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *EUROCRYPT*, pages 483–501, 2012.

[AK96] Ross Anderson and Markus Kuhn. Tamper resistance: a cautionary note. In *WOEC'96: Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1–11, 1996.

[AL11] Gilad Asharov and Yehuda Lindell. A full proof of the bgw protocol for perfectly-secure multiparty computation. *IACR Cryptology ePrint Archive*, 2011:136, 2011.

[AW07] Ben Adida and Douglas Wikström. How to shuffle in public. In *TCC*, pages 555–574, 2007.

[BCG$^+$11] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In *ASIACRYPT*, pages 722–739, 2011.

[BCH11] Nir Bitansky, Ran Canetti, and Shai Halevi. Leakage tolerant interactive protocols. Cryptology ePrint Archive, Report 2011/204, 2011.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology - EUROCRYPT '97*, pages 37–51, 1997.

[Ben83] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *PODC*, pages 27–30, 1983.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.

[BG10] Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In *CRYPTO*, pages 1–20, 2010.

[BGJ$^+$13] Elette Boyle, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, and Amit Sahai. Secure computation against adaptive auxiliary information. In *CRYPTO*, 2013.

[BGJK12] Elette Boyle, Shafi Goldwasser, Abhishek Jain, and Yael Tauman Kalai. Multiparty computation secure against continual memory leakage. In *STOC*, pages 1235–1254, 2012.

[BGK11] Elette Boyle, Shafi Goldwasser, and Yael Tauman Kalai. Leakage-resilient coin tossing. In *DISC*, 2011.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. ECCC, Report 2011/111, 2011.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *FOCS*, pages 501–510, 2010.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography*, pages 31–46, 2003.

[Bou05] Jean Bourgain. More on the sum-product phenomenon in prime fields and its applications. *International Journal of Number Theory*, 1(1):1–32, 2005.

[Bra84] Gabriel Bracha. An asynchronous $[(n-1)/3]$-resilient consensus protocol. In *PODC*, pages 154–162, 1984.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology - CRYPTO '97*, pages 513–525, 1997.

[BSMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.

[BSW11] Elette Boyle, Gil Segev, and Daniel Wichs. Fully leakage-resilient signatures. In *EUROCRYPT*, 2011.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *FOCS*, 2011.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In Bob Werner, editor, *Proc. 42nd FOCS*, pages 136–147, 2001. Preliminary full version available as Cryptology ePrint Archive Report 2000/067.

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Revised version of [Can01].

[CCD87] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract). In *CRYPTO*, page 462, 1987.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.

[CDMW09] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Improved non-committing encryption with applications to adaptively secure protocols. In *ASIACRYPT*, pages 287–302, 2009.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, pages 19–40, 2001.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multiparty computation. In *STOC*, pages 639–648, 1996.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, pages 383–395, 1985.

[CGO10] Nishanth Chandran, Juan A. Garay, and Rafail Ostrovsky. Improved fault tolerance and secure computation on sparse networks. In *ICALP (2)*, pages 249–260, 2010.

[CGO12] Nishanth Chandran, Juan A. Garay, and Rafail Ostrovsky. Edge fault tolerance on sparse networks. In *ICALP (2)*, pages 452–463, 2012.

[CIO98] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Non-interactive and non-malleable commitment. In *STOC*, pages 141–150, 1998.

[CKLK01] Artur Czumaj, Przemka Kanarek, Krzysztof Lorys, and Miroslaw Kutylowski. Switching networks for generating random permutations, 2001.

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

[CLL$^+$13] Kai-Min Chung, Huijia Lin, Feng-Hao Liu, Rafael Pass, and Hong-Sheng Zhou. Physically-aware composability. Manuscript, 2013.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[DF89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *CRYPTO*, pages 307–315, 1989.

[DGK$^+$10] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC*, pages 361–381, 2010.

[DHLW10a] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[DHLW10b] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*, pages 613–631, 2010.

[DHP11] Ivan Damgard, Carmit Hazay, and Arpita Patra. Leakage resilient two-party computation. Cryptology ePrint Archive, Report 2011/256, 2011.

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *CRYPTO*, pages 501–520, 2006.

[DIK$^+$08] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, pages 241–261, 2008.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT*, pages 445–465, 2010.

[DKL09] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.

[DKMS12] Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Breaking the o(nm) bit barrier: Secure multiparty computation with a static adversary. *CoRR*, abs/1203.0289, 2012.

[DLWW11] Yevgeniy Dodis, Allison Lewko, Brent Waters, and Daniel Wichs. Storing secrets on continually leaky devices. In *FOCS*, 2011.

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *CRYPTO*, pages 432–450, 2000.

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, 2008.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[DP10] Yevgeniy Dodis and Krzysztof Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. In *CRYPTO*, pages 21–40, 2010.

[DSS90] Cynthia Dwork, David B. Shmoys, and Larry J. Stockmeyer. Flipping persuasively in constant time. *SIAM J. Comput.*, 19(3):472–499, 1990.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[Fei99] Uriel Feige. Noncryptographic selection protocols. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, 1999.

[FGK$^+$10] David Mandell Freeman, Oded Goldreich, Eike Kiltz, Alon Rosen, and Gil Segev. More constructions of lossy and correlation-secure trapdoor functions. In *Proceedings of the 13th International Conference on Practice and Theory in Public Key Cryptography*, pages 279–295, 2010.

[FIM$^+$06] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J. Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. *ACM Transactions on Algorithms*, 2(3):435–472, 2006.

[FKPR10] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343–360, 2010.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *FOCS*, pages 308–317, 1990.

[FM85] Paul Feldman and Silvio Micali. Byzantine agreement in constant expected time (and trusting no one). In *FOCS*, pages 267–276, 1985.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *STOC*, pages 148–161, 1988.

[FRR$^+$10] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT*, pages 135–156, 2010.

[FS89] Uriel Feige and Adi Shamir. Zero knowledge proofs of knowledge in two rounds. In *CRYPTO*, pages 526–544, 1989.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GJS11] Sanjam Garg, Abhishek Jain, and Amit Sahai. Leakage-resilient zero knowledge. In *CRYPTO*, pages 297–315, 2011.

[GKK$^+$11] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure computation with sublinear amortized work. *IACR Cryptology ePrint Archive*, 2011:482, 2011.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32, 1989.

[GL02] Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In *DISC*, pages 17–32, 2002.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377, 1982.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *CHES*, pages 251–261, 2001.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. Preliminary version in STOC' 85.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[GMY06] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, 2006.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.

[Gol98] Oded Goldreich. Secure multi-party computation. http://www.wisdom.weizmann.ac.il/ oded/pp.html, 1998.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[GOS06a] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for nizk. In *CRYPTO*, pages 97–111, 2006.

[GOS06b] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *EUROCRYPT*, pages 339–358, 2006.

[GR10] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, pages 59–79, 2010.

[GR12] Shafi Goldwasser and Guy Rothblum. How to compute in the presence of leakage. In *FOCS*, 2012.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, pages 415–432, 2008.

[GSV05] Shafi Goldwasser, Madhu Sudan, and Vinod Vaikuntanathan. Distributed computing with imperfect randomness. In *DISC*, pages 288–302, 2005.

[HIK$^+$11] Iftach Haitner, Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions of protocols for secure computation. *SIAM J. Comput.*, 40(2):225–266, 2011.

[HKKN01] Shai Halevi, Robert Krauthgamer, Eyal Kushilevitz, and Kobbi Nissim. Private approximation of np-hard functions. In *STOC*, pages 550–559, 2001.

[HL11] Shai Halevi and Huijia Lin. After-the-fact leakage in public-key encryption. In *TCC*, pages 107–124, 2011.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO*, pages 132–150, 2011.

[HS09] Nadia Heninger and Hovav Shacham. Reconstructing RSA private keys from random key bits. In *Advances in Cryptology - CRYPTO '09*, pages 1–17, 2009.

[HSH$^+$08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security Symposium*, pages 45–60, 2008.

[HSH+09] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[IW06] Piotr Indyk and David P. Woodruff. Polylogarithmic private approximations and efficient matching. In *TCC*, pages 245–264, 2006.

[JV10] Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO*, pages 41–58, 2010.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99*, pages 388-397, 1999.

[KLR09] Yael Tauman Kalai, Xin Li, and Anup Rao. 2-source extractors under computational assumptions and cryptography with defective randomness. In *FOCS*, pages 617–626, 2009.

[KLRZ08] Yael Tauman Kalai, Xin Li, Anup Rao, and David Zuckerman. Network extractor protocols. In *FOCS*, pages 654–663, 2008.

[KLST11] Valerie King, Steven Lonargan, Jared Saia, and Amitabh Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. In *ICDCN*, pages 203–214, 2011.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.

[KP10] Eike Kiltz and Krzysztof Pietrzak. Leakage resilient elgamal encryption. In *ASIACRYPT*, pages 595–612, 2010.

[KRVZ06] Jesse Kamp, Anup Rao, Salil Vadhan, and David Zuckerman. Deterministic extractors for small-space sources. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 691–700, 2006.

[KS11] Valerie King and Jared Saia. Breaking the $o(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. *J. ACM*, 58(4):18, 2011.

[KSSV06] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *SODA*, pages 990–999, 2006.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.

[LLW11] Allison Lewko, Mark Lewko, and Brent Waters. How to leak on key updates. In *STOC*, 2011.

[LOS$^+$06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT*, pages 465–485, 2006.

[LRW11] Allison Lewko, Yannis Rouselakis, and Brent Waters. Achieving leakage resilience through dual system encryption. In *TCC*, 2011.

[MNS09] Tal Moran, Moni Naor, and Gil Segev. An optimally fair coin toss. In *TCC*, pages 1–18, 2009.

[MOR01] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: extended abstract. In *ACM Conference on Computer and Communications Security*, pages 245–254, 2001.

[MPR07] Ueli M. Maurer, Krzysztof Pietrzak, and Renato Renner. Indistinguishability amplification. In *CRYPTO*, pages 130–149, 2007.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.

[MS81] R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Commun. ACM*, 24:583–584, September 1981.

[MTVY11] Tal Malkin, Isamu Teranishi, Yevgeniy Vahlis, and Moti Yung. Signatures resilient to continual leakage on memory and computation. In *EUROCRYPT*, 2011.

[MY04] Philip D. MacKenzie and Ke Yang. On simulation-sound trapdoor commitments. In *EUROCRYPT*, pages 382–400, 2004.

[NN01] Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *In Proc. of 33rd STOC*, pages 590–599, 2001.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of aes. In *CT-RSA*, pages 1–20, 2006.

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.

[PW11] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. *SIAM J. Comput.*, 40(6):1803–1844, 2011.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, pages 200–210, 2001.

[Rab83] Michael O. Rabin. Randomized byzantine generals. In *FOCS*, pages 403–409, 1983.

[Raz05] Ran Raz. Extractors with weak random seeds. In *STOC*, pages 11–20, 2005.

[Sah99] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proc. 40th FOCS*, pages 543–553, 1999.

[SCO$^+$01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, pages 566–598, 2001.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11), November 1979.

[Sls] Collection of surveys on sublinear algorithms.
`http://people.csail.mit.edu/ronitt/sublinear.html`.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd FOCS*, pages 80–91, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pages 162–167, 1986.