Abstract of "Error Tolerant Cryptography " by Feng-Hao Liu, Ph.D., Brown University, April 2013.

One of the major goals in cryptography is to design protocols that withstand malicious behavior of an adversary. Traditionally, the focus was on a setting where honest users followed their protocol exactly, without fault. But what if an adversary can induce faults, for example a physical attack that changes the state of a user's computation, forcing a user to accept when he should be rejecting; or tries to use a modified secret key? Can any security guarantees still be given when such errors occur? My PhD work studies the implications of various types of errors and develops techniques that protect against them.

I have delved into the following topics for different scenarios of errors: (1) cryptography with imperfect hardware, where the adversary can cause the cryptographic device to leak some secret information and tamper with the device's memory; (2) secure delegation protocols, where a user can delegate some computation to an untrusted server that causes errors.

To highlight some of my results:

- I gave a generic construction to secure *any* cryptographic functionality against continual memory tampering and leakage errors in the *split-state model*. My main tool is to construct a non-malleable code that is also leakage resilient in this model, which resolves one central open problem in the previous work (due to Dziembowski et al. – ICS 10).

- I developed new delegation protocols that allow a user, who only stores a short certificate of his data (potentially very large), to delegate the computation on the data to the cloud, and then verify the outcome in time *sub-linear* in the data size.

In the thesis, I elaborate my work in these two lines, and some potential future directions.

Error Tolerant Cryptography

by

Feng-Hao Liu

Sc.B., National Taiwan University, 2005

Sc.M., Brown University, 2009

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island

April 2013

This dissertation by Feng-Hao Liu is accepted in its present form
by the Department of Computer Science as satisfying the dissertation
requirement for the degree of Doctor of Philosophy.

Date _____ _____
Anna Lysyanskaya, Director

Recommended to the Graduate Council

Date _____ _____
John Savage, Reader
Brown University

Date _____ _____
Paul Valiant, Reader
Brown University

Date _____ _____
Yevgeniy Dodis, Reader
New York University

Approved by the Graduate Council

Date _____ _____
Peter M. Weber
Dean of the Graduate School

Acknowledgements

to be written...

Contents

Chapter 1

Introduction

One of the major goals in cryptography is to design protocols that withstand malicious behavior of an adversary. Traditionally, our focus has been on a setting where honest users followed their protocol exactly, without fault. For example, a user who follows an encryption algorithm correctly should be able to generate a ciphertext that is totally unintelligible to adversaries. This model is simple and elegant, and captures many useful scenarios; thus many important applications have been developed based on the model. A classic example is when two users want to communicate privately at a distance, they can achieve this task through the internet by sending encryptions of their messages, and the security of the encryption algorithm guarantees that any adversary who eavesdrops on the communication channel cannot learn anything meaningful.

As we mentioned, a crucial condition assumed in this model is that the honest party can execute the prescribed protocols without fault. In recent years, however, many scenarios have emerged in which this assumption may not hold. This thesis is devoted to the study of two important scenarios that traditional models fail to capture, because the adversary may induce errors.

- The first scenario considers secure delegation in cloud computing, where a user may store some (potentially very large) data on an untrusted cloud server, and request some computation be preformed over the data. For example, a Gmail user stores all her mails in the Google server and may request some search over all the mails. In this scenario, the cloud may not follow the required computation operations, either deliberately or by random errors. As cloud computing becomes more and more popular (e.g. many companies such as Amazon, Google, Microsoft,

etc. all provide some forms of cloud services), whether a user can enforce authentication of computation from untrusted clouds is an important issue with significant implications.

- The second scenario is tamper and leakage resilience in mobile computing. In the past five years, mobile devices such as smart phones or RFID cards have been widely used in our daily lives. These devices often interact with other devices at a (physically) close distance, and thus, an adversary may launch some physical attacks to interfere with the computation, such as tampering with or getting some partial information of the secret in the device. There has been a line of research studying physical attacks on implementations [Koc96, BS97, AARR02, HSH+08] that dramatically weaken the security guaranteed by traditional schemes. How to obtain secure implementations in the presence of physical attacks is a challenging issue and becomes more and more important as mobile devices are going to be the next generation of computation.

I propose new models that capture security in the presence of new errors induced by a malicious cloud, or physical attacks. In both scenarios, I develop new techniques that achieve stronger security guarantees. Before presenting the formal descriptions of these models, I will present detailed introductions of the two scenarios.

1.1 Part I: Secure Delegation

Delegating computation is a scenario where one party, the *delegator*, wishes to delegate the computation of a function f to another party, the *worker*. The challenge is that the delegator may not trust the worker, and thus it is desirable to have the worker *prove* that the computation was done correctly. Obviously, verifying this proof should be easier than doing the computation.

This concept is also known as *outsourcing* computation and has received a lot of attention in recent years, partly due to the increasing interest in cloud computing, where the goal is to outsource all the computational resources to a (potentially untrusted) *cloud*. There are several reasons why the client (or delegator) may not trust the cloud; for example, the cloud may have an incentive to return incorrect answers. Such an incentive may be a financial one, if the real computation requires a lot of work, whereas computing incorrect answers requires less work and is unlikely to be detected by the client. Moreover, in some cases, the applications outsourced to the cloud may be so critical

that the delegator wishes to rule out accidental faults during the computation. Those incorrect computations for whatever reasons are referred to as "*errors*" as in the title of this thesis. For these reasons, the delegator would like to receive proofs of correctness of the computation.

In order to ensure that the worker (or the cloud) performs the computation correctly, we would like the worker to *prove* this to the delegator, but it is essential that the time it takes to verify the proof is significantly smaller than the time needed to actually run the computation; otherwise the delegator could do the computation herself and there is no point delegating the task. At the same time, the running time of the worker carrying out the proof should also be reasonable, i.e. comparable to the time it takes to do the computation.

The problem of delegating computation has been studied extensively (see Section 1.1.1 for an overview of previous work). However, most previous work on delegation allows the delegator to run in time polynomial in the input size, as long as this runtime is significantly smaller than the time it takes to do the computation. For example, when delegating the computation of a function f that runs in time T and has inputs of size n, typically the desired runtime of the delegator is $\text{poly}(n, \log T)$ and the desired runtime of the worker is $\text{poly}(T)$.

In this work, we want the delegator to run in time that is even smaller than the input size n. Namely, sometimes it can be prohibitively expensive for the delegator to have to read the entire input. At first, this may seem unreasonable. So, let us start by motivating this feature with two specific scenarios to address.

Memory delegation. Suppose that Alice would like to store all her memory in the cloud. The size of her memory may be huge (for example, may include all the emails she ever received). Moreover, suppose she does not trust the cloud. Then, every time she asks the cloud to carry out some computation (for example, compute how many emails she has received from Bob during the last year), she would like the answer to be accompanied by a proof that indeed the computation was done correctly. Note that the input to these delegated functions is her entire memory, which can be huge. Therefore, it is highly undesirable that Alice runs in time that is proportional to this input size. More importantly, Alice does not even hold on to this memory anymore, since she delegated it to the cloud.

Thus, in a memory delegation scheme, a delegator delegates her entire memory to the cloud, and then may ask the could to compute functions of this memory, and expects the answers to be

accompanied by a proof. In order to verify the correctness of these proofs, the delegator must save some short certificate of her memory, say a certificate of size $\text{polylog}(n)$, where n is the memory size. The proofs should be verifiable very efficiently; say, in time $\text{polylog}(n, T)$, where T is the time it takes to compute the function. Moreover, Alice should be able to update her memory efficiently.

Streaming delegation. Suppose that there is some large amount of data that is streaming by, and suppose that a user, Alice, wishes to save this data, so that later on she will be able to compute statistics on this data. However, Alice's memory is limited and she cannot store this data. Instead, she wishes to delegate this to the cloud. Namely, she asks the cloud to store this streaming data for her, and then she asks the cloud to perform computation on this data. As in the case of memory delegation, in order to later verify the correctness of these computations, Alice must save some short certificate of this streaming data. Unlike the setting of memory delegation, this certificate should be computed (and updated) in a streaming manner.

The settings of memory and streaming delegation are quite similar: in both settings Alice asks the cloud to store a huge object (either her memory or the streaming data). There are two main differences between the two: (1) In the setting of streaming delegation, the certificates and updates must be computed in a streaming manner. Thus, in this sense, constructing streaming delegation schemes may be harder than constructing memory delegation schemes. Indeed, our streaming delegation scheme is more complicated than our memory delegation scheme, and proving soundness in the streaming setting is significantly harder than proving soundness in the memory setting. (2) In the setting of streaming delegation, the memory is updated by simply adding elements to it. This is in contrast to the setting of memory delegation, where the memory can be updated in arbitrary ways, depending on the user's needs. However, in the memory setting, we allow the delegator to use the help of the worker when updating her certificate (or secret state), whereas in the streaming setting we require that the delegator updates her certificate on her own. The reason for this discrepancy, is that in the memory setting the delegator may not be able to update her certificate on her own, since she may want to update her memory in involved ways (such as, erase all emails from Bob). On the other hand, in the streaming setting, it seems essential that the delegator updates her certificate on her own, since in this setting the data may be streaming by very quickly, and there may not be enough time for the delegator and worker to interact during each update.

1.1.1 Prior Work

As we mentioned in the previous section, various delegation protocols have been proposed. Some provide delegation protocols that are sound against any cheating worker, whereas others provide delegation protocols that are secure only against computationally bounded cheating workers (i.e., arguments as opposed to proofs. See below for the references.) Some of these protocols are interactive, whereas others are non-interactive. We survey some of the work below, however, we emphasize that in *essentially* all these solutions (except if we allow non-standard assumptions), the delegator runs in time that is (at least) linear in the input size, and thus their results do not apply to our settings of memory delegation or streaming delegation.

Interactive proofs. Briefly speaking, an interactive proof system consists of a (possibly) all-powerful *prover* and a computationally bounded *verifier*. Via some interactions, the prover can convince the verifier of the validity of some statement, and even if the prover is cheating, the verifier will not be convinced (with high probability) if the statement is false. The celebrated IP=PSPACE [1] Theorem [LFKN92, Sha92] yields an interactive proof protocol for any function f computable in polynomial space, with a verifier (delegator) running in polynomial time. Thus, the protocol can be seen as a delegation protocol for languages in PSPACE. However, the complexity of the prover (worker) is only bounded by polynomial space (and hence may be exponential time). This theorem was refined and scaled down in [FL93] to give verifier complexity $\text{poly}(n, s)$ and prover complexity $2^{\text{poly}(s)}$ for functions f computable in time T and space s, on inputs of length n. Note that the prover complexity is still super-polynomial in T, even for computations that run in the smallest possible space, namely $s = O(\log T)$.

The prover complexity was recently improved by Goldwasser et al. [GKR08] to $\text{poly}(T, 2^s)$, which is $\text{poly}(T)$ when $s = O(\log T)$. More generally, Goldwasser et al. [GKR08] give interactive proofs for computations of circuits of small *depth* d. For these circuits, they achieve prover complexity $\text{poly}(T)$ and verifier complexity $\text{poly}(n, d, \log T)$. (This implies that we can delegate space-bounded computations because an algorithm that runs in time T and space s can be converted into a circuit that runs in time $\text{poly}(T, 2^s)$ and depth $d = O(s^2)$.) However, if we do not restrict to computations of small space or depth, then we cannot apply the results of interactive proofs mentioned above. Indeed,

[1] IP is the set of all decision problems that have interactive proof systems. PSPACE is the set of all decision problems that can be solved by a Turing machine using a polynomial amount of space.

any language that has an interactive proof with verifier running time (and hence communication) T_V can be decided in space $\text{poly}(n, T_V)$.

Interactive arguments. Interactive arguments [BCC88] (also known as computationally sound proofs [Mic00]) relax the soundness condition so that it only holds against malicious computationally bounded provers. Namely, instead of requiring that no prover strategy whatsoever can convince the verifier of a false statement, we instead require that no computationally feasible prover strategy can convince the verifier of a false statement. In this model, Kilian [Kil92] and Micali [Mic00] gave constant-round protocols with prover complexity $\text{poly}(T, k)$ and verifier complexity $\text{poly}(n, k, \log T)$ (where k is the security parameter), assuming the existence of collision-resistant hash functions[2] [BG02].

Towards non-interactive solutions. The possibility of efficient non-interactive arguments was suggested by Micali [Mic00], who showed that non-interactive arguments with prover complexity $\text{poly}(T, k)$ and verifier complexity $\text{poly}(n, k, \log T)$ can be constructed in the random oracle model[3] (the oracle is used to eliminate interaction a la Fiat–Shamir [FS86]). Heuristically, one might hope that by instantiating the random oracle with an appropriate family of hash functions, we could obtain a non-interactive solution to delegating computation: first the delegator (or a trusted third party) chooses and publishes a random hash function from the family, and then, the proofs are completely non-interactive (just one message from the prover to the verifier). However, the random oracle heuristic is known to be unsound in general [CGH04] and even in the context of Fiat–Shamir [Bar01, GK03]. Thus, despite extensive efforts, the existence of efficient non-interactive arguments remains a significant open problem in complexity and cryptography.

There has been some recent progress in reducing the number of round of interaction needed. Using a transformation of Kalai and Raz [KR09], the GKR delegation protocol [GKR08] can be converted into a 2-round argument (assuming the existence of single-server private-information retrieval (PIR) schemes[4]). Like the interactive proofs of [GKR08], however, this solution applies only to small-depth computations, as the verifier's complexity grows linearly with the depth.

[2] A collision-resistant hash function guarantees that it is computationally infeasible to find two different inputs that hash to the same value.

[3] In the random oracle model, all parties can access one truly random function via their oracle tapes. This random function is referred to as the random oracle.

[4] A PIR scheme allows a user to retrieve an item from a database server without revealing the item's information (e.g. the index of the item). See Definition 5.

Gennaro, Gentry, and Parno [GGP10], and a followup work of Chung, Kalai, and Vadhan [CKV10], gave a 2-round delegation scheme for arbitrary functions. Yet these constructions need an additional offline phase, where the delegator invests time $\text{poly}(T, k)$ and computes a *secret state* (T is the time it takes to compute the function, and k is the security parameter). In the online phase, the delegator's running time is reduced to $\text{poly}(n, k, \log T)$ for an input of length n, and the worker's complexity is $\text{poly}(T, k)$. Thus, the delegator's large investment in the offline phase can be amortized over many executions of the online phase to delegate the computation of f on many inputs. Their online phase is not completely non-interactive, but rather consists of two rounds, i.e. a message from the delegator followed by a response from the worker. However, in many applications, two rounds will be necessary anyway, as the delegator may need to communicate the input x to the worker.

We remark that one main drawback of that line of work [GGP10, CKV10] is that soundness is only guaranteed as long as the adversarial worker does not learn whether the delegator accepted or rejected the proofs. Very recently, Parno et al. [PRV12] resolved this issue by using a technique of attribute-based encryption (ABE)[5]. In their scheme, the verifier (delegator) does not need to hold any secret information, but only keeps some public certificate instead. Thus, the verifications can be done in public, and their scheme does not suffer from the problem. The main limitation of this approach comes from the constructions of ABE – currently the state of the art of the ABE construction can only give a delegation scheme for bounded polynomial-sized circuits. We also remark that it is not clear how to extend their results to our memory and streaming delegation settings.

In another work, Applebaum, Ishai, and Kushilevitz [AIK10] consider the offline/online setting, but focus on efficient solutions for one-time delegation (i.e., the online phase can only be executed one time). They consider the case when delegation functions are represented as arithmetic circuits.

Streaming interactive proofs. Cormode, Thaler, and Yi [CTY10] considered streaming interactive proofs, which is a strengthening of interactive proofs where the input is given to the verifier in a streaming manner and the verifier is restricted to have sub-linear (ideally, poly-logarithmic) space. They observed that both the GKR protocol [GKR08] and the universal arguments [BG02] can be modified to yield efficient streaming interactive proofs/arguments.

Streaming interactive proofs are closely related to streaming delegation. The main difference is

[5]Briefly speaking, in an attribute encryption ciphertexts are labeled with sets of attributes, and private keys are associated with access structures that control which ciphertexts a user is able to decrypt.

that streaming interactive proofs correspond to *one-time* streaming delegation, where the delegator can only delegate one computation. In our streaming delegation model, the delegator is allowed to delegate as many computations to the worker as she want. Indeed, the GKR protocol and the universal arguments are also the starting points of our construction of streaming delegation schemes. The main effort is to make the scheme *many-time* secure.

Solutions based on non-standard assumptions. A series of concurrent work [GLR11, DFH12, BCCT12] constructed efficient and 2-round delegation protocols for any polynomial-time computable functions, using some extractable hash functions as building blocks; however, their constructions of the extractable hash functions are based on some non-standard (formally non-falsifiable, see the work of Naor [Nao03]) assumptions. Their techniques can be used to construct efficient memory and streaming delegation, yet the security can only be based on non-standard assumptions. Approaches based on standard assumptions are much desirable, and in this thesis we only consider solutions whose security can be proven under standard assumptions.

Delegation of specific functionalities. In this thesis, we only focus on delegations of a large class of functionalities, yet we remark that there have been lots of studies for delegations of specific functionalities. For example, Hohenberger and Lysyanskaya [HL05] studied delegation of modular exponentiations; Papamanthou, Tamassia, and Triandopoulos [PTT11] studied delegation of set operations; Benabbas, Gennaro, and Vahlis [BGV11], and Papamanthou, Shi, and Tamassia [PST13] studied delegation of some operations of polynomials, e.g. polynomial evaluation, computing derivatives, etc. This direction has been extensively studied by the cryptography community since many specific functionalities have important applications in practice.

Other delegation models. There have been several variants of delegation models. Belenkiy et al. [BCE+08] considered a model where parties are modeled as rational players, and given rewards if they do computations correctly. Choi et al. [CKKC13] considered multi-client delegation where multiple clients want to do a joint computation on their inputs through a server. This line of research is outside the scope of this paper, but we highlight this as an interesting open area for future studies.

1.1.2 Our Contributions

Although there have been many delegation protocols studied as mentioned in the above section, we recall that their results (for those based on standard assumptions) cannot be applied to our setting of memory and streaming delegation. The main reason is their delegators' complexity in the context of general functionality delegation– the prior work's delegators run in time that is (at least) linear in the input size, whereas in our settings of memory and streaming delegation, we want the delegator to run in time that is even smaller than the input size n. In the first part of this thesis, we construct both memory delegation and streaming delegation schemes in which the verifications can be done in sub-linear time in the input size, with the help of some short certificate. Let us elaborate on the ideas.

The memory delegation scheme consists of an offline phase, where the delegator D delegates her memory $x \in \{0,1\}^n$ to a worker W. This phase is non-interactive, where the delegator sends a single message, which includes her memory content x to the worker W. The runtime of both the delegator and the worker in the offline phase is $\mathrm{poly}(n)$, where n is the memory size. At the end of this phase, the delegator saves a short certificate σ of her memory, which she will later use when verifying delegation proofs.

The streaming delegation scheme, on the other hand, does not have such an offline phase. In the streaming setting, we consider the scenario where at each time unit t a bit x_t is being streamed. The delegator starts with some secret state (or certificate) σ_0, and at time unit $t+1$ she uses her secret state σ_t and the current bit x_{t+1} being streamed, to update her secret state from σ_t to σ_{t+1}. Moreover, we require the update procedure be efficient.

In both settings, each time the delegator D wants the worker W to compute a function $f(x)$, they run a delegation protocol, which we denote by $\mathsf{Compute}(f)$. The memory delegation scheme has an additional Update protocol, where the delegator D asks the worker W to update her memory and to help her update her secret state σ. The latter can be thought of as a delegation request, and the efficiency guarantees (in term of runtime and communication complexity) are similar to those of the $\mathsf{Compute}$ protocol.

In the streaming setting, the delegator updates her secret state on her own in time $\mathrm{polylog}(N)$, where N is an upper bound on the length of the stream. Namely, the update function, which takes as input a certificate σ_t and a bit x_{t+1}, and outputs a new certificate σ_{t+1}, can be computed in time

polylog(N).

We present two memory and streaming delegation protocols. The first protocol has 2-round message exchanges (i.e, $\mathsf{Compute}(f)$ consists of two rounds of message exchanges, the first sent by the delegator and the second sent by the worker). They are based on the non-interactive version of the delegation protocol of Goldwasser *et al.* [GKR08, KR09], denoted by GKR (though are significantly more complicated than merely running GKR). As in the GKR protocol, the efficiency of the delegator depends linearly on the depth of the circuit being delegated, so such construction is efficient only for delegation of low-depth functions. Our second memory and streaming delegation protocols are interactive (i.e., $\mathsf{Compute}(f)$ consists of four rounds). These schemes are based on a variant of the computational sound proofs of Micali [Mic94], and allow for efficient delegation of all functions in **P**.

In what follows we state our theorems formally. We refer the reader to Section 5.1 for the formal definition of a memory delegation scheme, and to Section 6.1 for the formal definition of a streaming delegation scheme.

Theorem 1 (Memory Delegation) *Assume the existence of a poly-log PIR scheme(as defined in Definition 5), and assume the existence of a family of collision resistant hash functions. Let $\mathcal{F}$ be the class of all $\mathcal{L}$-uniform poly-size boolean circuits*[6]*. Then there exists a 2-round memory delegation scheme* mDel*, for delegating any function $f \in \mathcal{F}$. The delegation scheme,* mDel *has the following properties, for security parameter k.*

- *The scheme has perfect completeness and negligible soundness error.*
- *The delegator and worker are efficient in the offline stage; i.e., both the delegator and the worker run in time* $\mathrm{poly}(k, n)$.
- *The worker is efficient in the online phase. More specifically, it runs in time* $\mathrm{poly}(k, S)$ *during each* $\mathsf{Compute}(f)$ *and* $\mathsf{Update}(f)$ *operation, where S is the size of the $\mathcal{L}$-uniform circuit computing f. The delegator runs in time* $\mathrm{poly}(k, d)$ *during each* $\mathsf{Compute}(f)$ *and* $\mathsf{Update}(f)$ *operation, where d is the depth*[7] *of the $\mathcal{L}$-uniform circuit computing f.*[8]

[6] The class contains all poly-size boolean circuits that can be output by a deterministic Turing machine using a logarithmic space.

[7] We assume that $d \geq \log n$.

[8] Thus, for every constant $c \in \mathbb{N}$, if we restrict the depth of f to be at most k^c, then the delegator is considered efficient.

In particular, assuming the existence of a poly-logarithmic PIR scheme, and assuming the existence of a collision resistant hash family, we obtain a memory delegation scheme for $\mathcal{L}$-uniform **NC** computations[9] ($\mathcal{L}$ stands for logarithmic space, see the footnote for the description of the class), where the delegator D runs in time *poly-logarithmic* in the length of the memory.

Theorem 2 (Streaming Delegation) *Let k be a security parameter, and let N be a parameter (an upper bound on the length of the stream). Let $\mathcal{F}$ be the class of all $\mathcal{L}$-uniform poly-size boolean circuits. Assume the existence of a fully-homomorphic encryption scheme secure against* $\mathrm{poly}(N)$*-size adversaries. Then there exists a streaming delegation scheme $\mathsf{sDel}_{\mathcal{F}}$ for $\mathcal{F}$ with the following properties.*

- $\mathsf{sDel}_{\mathcal{F}}$ *has perfect completeness and negligible soundness error.*
- D *updates her secret state in time* $\mathrm{polylog}(N)$*, per data item.*
- *In the delegation protocol, when delegating a function $f \in \mathcal{F}$ computable by an $\mathcal{L}$-uniform circuit of size S and depth d, the delegator D runs in time* $\mathrm{poly}(k, d, \log N)$*, and the worker W runs in time* $\mathrm{poly}(k, \log N, S)$*.*

In particular, assuming the existence of a fully-homomorphic encryption scheme [10] secure against adversaries of size $\mathrm{poly}(N)$, we obtain a streaming delegation scheme for $\mathcal{L}$-uniform **NC** computations, where the delegator D runs in time *poly-logarithmic* in the length of data stream.

The following two theorems present memory and streaming delegation for all polynomial-time computable functions, at a cost of two extra rounds of message exchanges. The constructions are based on a variant of the computational sound proofs of Micali [Mic94]. We present the technical details in Chapter 7.

Theorem 3 (Interactive Memory Delegation) *Assume the existence of a family of collision resistant hash functions. Then there exists a memory delegation scheme mDel, for delegating any function computable by a polynomial-time Turning machine. The delegation scheme, mDel has the following properties, for security parameter k.*

- *The scheme has perfect completeness and negligible soundness error.*

[9] This is the class of all polynomial-sized logarithmic depth circuits that can be output by a deterministic Turing machine running in logarithmic space.

[10] A fully homomorphic encryption (FHE) scheme allows one to compute a function f (from some function class) and produce a ciphertext $\mathrm{Enc}(f(x))$ given input a ciphtertext $\mathrm{Enc}(x)$. See Section 3.2 for the detailed description.

- *The delegator and worker are efficient in the offline stage; i.e., both the delegator and the worker run in time* $\mathrm{poly}(k, n)$.
- *The worker is efficient in the online phase. More specifically, it runs in time* $\mathrm{poly}(k, T)$ *during each* $\mathsf{Compute}(f)$ *and* $\mathsf{Update}(f)$ *operation, where* T *is an upper-bound on the running time of* f. *The delegator runs in time* $\mathrm{poly}(k, \log T)$ *during each* $\mathsf{Compute}(f)$ *and* $\mathsf{Update}(f)$ *operation.*
- *Both* $\mathsf{Compute}(f)$ *and* $\mathsf{Update}(f)$ *operations consist of* 4 *rounds of message exchanges.*

Theorem 4 (Interactive Streaming Delegation) *Let* k *be a security parameter, and let* N *be a parameter (an upper bound on the length of the stream). Let* $\mathcal{F}$ *be the class of all functions computable by a polynomial-time Turning machine. Assume the existence of a fully-homomorphic encryption scheme secure against* $\mathrm{poly}(N)$*-size adversaries. Then there exists a streaming delegation scheme* $\mathsf{sDel}_{\mathcal{F}}$ *for* $\mathcal{F}$ *with the following properties.*

- $\mathsf{sDel}_{\mathcal{F}}$ *has perfect completeness and negligible soundness error.*
- D *updates her secret state in time* $\mathrm{polylog}(N)$*, per data item.*
- *In the delegation protocol, when delegating a function* $f \in \mathcal{F}$ *computable in time* T*, the delegator* D *runs in time* $\mathrm{poly}(k, \log N, \log T)$*, and the worker* W *runs in time* $\mathrm{poly}(k, \log N, T)$. *The delegation protocol consists of* 4 *rounds of message exchanges.*

We note that in the random oracle (RO) Model [BR97], the delegation scheme of Micali is non-interactive. This yields a non-interactive memory delegation scheme and a non-interactive streaming delegation scheme, for delegating all polynomial-time computable functions, in the RO model.

1.2 Part II: Tamper and Leakage Resilience

The second part of this thesis studies how to achieve secure implementations under tampering and leakage attacks. Briefly speaking, this task can be formulated as various flavors of the following general problem. Suppose that we have a device that implements some cryptographic functionality (for example, a signature scheme or a cryptosystem). Further, suppose that an adversary can, in addition to input/output access to the device, get some side-channel information about its secret

state, potentially on a continual basis; for example, an adversary can measure the power consumption of the device, timing of operations, or even read part of the secret directly [Koc96, HSH+08]. Additionally, suppose that the adversary can, also possibly on a continual basis, somehow alter the secret state of the device through an additional physical attack such as microwaving the device or exposing to heat or EM radiation [BS97, AARR02]. These are also referred to as another type of "*errors*" as the title of this thesis. What can be done about protecting the security of the functionality of the device under such errors?

Unfortunately, strong negative results exist even for highly restricted versions of this general problem. For example, if the device does not have access to randomness, but is subject to arbitrary continual leakage, and so, in each round i, can leak to the adversary just one bit $b_i(s_i)$ for a predicate b_i of the adversary's choice, eventually it will leak its entire secret state. Moreover, even in a very restricted leakage model where the adversary can continually learn a physical bit of the secret state s_i, if the adversary is also allowed to tamper with the device and the device does not have access to randomness, we [LL10] showed that the adversary will eventually learn the entire secret state. Further, even with tampering alone, Gennaro et al. [GLM+04] show that security from arbitrary tampering cannot be achieved unless the device can overwrite its memory; further, they show that security can only be achieved in the common reference string model.

Thus, positive results are only possible for restricted versions of this problem. If we only allow leakage, but not tampering, and access to a source of randomness that the device can use to update itself, devices for signatures and decryption can be secured in this model under appropriate assumptions [BKKV10, DHLAW10, LRW11, LLW11]. Devices that don't have access to randomness after initialization can still be secure in the more restricted bounded-leakage model, introduced by Akavia, Goldwasser, and Vaikuntanathan [AGV09], where the attacker can learn arbitrary information about the secret, as long as the total amount is bounded by some prior parameter [AGV09, NS09, ADW09, KV09].

If only tampering is allowed, Gennaro et al. [GLM+04] gave a construction that secures a device in the model where the manufacturer has a public key and signs the secret key of the device. Dziembowski et al. [DPW10] generalized their solution to the case where the contents of the device is encoded with a non-malleable code; they consider the case where the class of tampering functions is restricted, and construct codes that are non-malleable with respect to these restricted tampering functions. Specifically, they have non-constructive results on existence of non-malleable codes for

broad classes of tampering functions; they construct, in the plain model, a non-malleable code with respect to functions that tamper with individual physical bits; in the random-oracle model, they give a construction for the so-called *split-state* tampering functions, which we will discuss in detail below. Recently, Choi, Kiayias, and Malkin [CKM11] improved the construction (in the plain model) of non-malleable codes that can withstand block-by-block tampering functions for blocks of small (logarithmic) sizes.

Finally, there are positive results for signature and encryption devices when both continual tampering and leakage are possible, and the device has access to a protected source of true randomness [KKS11]. One may be tempted to infer from this positive result that it can be "derandomized" by replacing true randomness with the continuous output of a pseudorandom generator, but this approach is ruled out by Liu and Lysyanskaya [LL10]. Yet, how does a device, while under a physical attack, access true randomness? True randomness is a scarce resource even when a device is not under attack; for example, the GPG implementations of public-key cryptography ask the user to supply random keystrokes whenever true randomness is needed, which leads to non-random bits should a device fall into the adversary's hands.

In the second part of this thesis, we investigate general techniques for protecting cryptographic devices from continual leakage and tampering attacks without requiring access to true randomness after initialization. Since, as we explained above, this is impossible for general classes of leakage and tampering functions, we can only solve this problem for restricted classes of leakage and tampering functions. Which restrictions are reasonable? Suppose that a device is designed such that its memory M is split into two compartments, M_1 and M_2, that are physically separated. For example, a laptop may have more than one hard drive. Then it is reasonable to imagine that the adversary's side channel that leaks information about M_1 does not have access to M_2, and vice versa. Similarly, the adversary's tampering function tampers with M_1 without access to M_2, and vice versa. This is known as the split-state model, and it has been considered before in the context of leakage-only [DP08, DLWW11] and tampering-only [DPW10] attacks.

1.2.1 Prior Work

Indeed the cryptography community has extensively studied security against different classes of physical attacks. Here we give a table summarizing the state of the art in tolerating *continual leakage and tampering attacks*; specific attacks we consider are split-state attacks (abbreviated as

"SS"), attacks on physical bits (abbreviated as "bits"), attacks on small blocks (abbreviated as "blocks"), and attacks by any polynomial-sized circuits (abbreviated as "any"). Also, we consider solutions in the random oracle models (abbreviated as "RO") where all parties have access to an un-tamperable random oracle, the common reference string model (abbreviated as "CRS") where all parties have access to an un-tamperable common reference string sampled by a trusted party, and the plain model where no trusted setup is needed.

Type of leakage	Type of tampering	Local coins	Known results about continual attacks
None	Any	No	Signature and decryption in the CRS model [GLM+04]
Any	None	No	Trivially impossible
Bits	Any	No	Impossible [LL10]
Any	None	Yes	Signature and encryption in the plain model [BKKV10, DHLAW10, LRW11, LLW11]
None	Bits	Yes	All functionalities in the plain model [DPW10]
None	SS	Yes	All functionalities in the RO model [DPW10]
None	Blocks	Yes	All functionalities in the plain model [CKM11]
Any	Any	Yes	Signature and encryption in the CRS model [KKS11]
SS	SS	No	All functionalities in the CRS model [**This thesis**]

We remark that all the results referenced above apply to attacks on the memory of the device, rather than its computation (with one exception). The exception [LLW11] is the work that constructed the first encryption and signature schemes that can leak more than logarithmic number of bits during their update procedure (but cannot be tampered with). Thus, all these works assume computation to be somewhat secure. In this work, for simplicity, we also assume that computation is secure, and remark that there is a line of work on protecting computation from leakage or tampering [ISW03, MR04, IPSW06, DP08, Pie09, DP10, FRR+10, GR10, JV10, FPV11]. This is orthogonal to the study of protecting memory leakage and tampering. In particular, we can combine our work with that of Goldwasser and Rothblum [GR10], or Juma and Vahlis [JV10] to obtain a construction where computation is protected as well; however, this comes at a cost of needing fresh local randomness. All known cryptographic constructions that allow an adversary to issue leakage

queries while the computation is going on rely on fresh local randomness.

We must also stress that the previous positive results on leakage resilient (LR) encryption are weaker than ours. This is because the definition of LR encryption is, of necessity, rather unnatural: once a challenge ciphertext has been created, the adversary can no longer issue leakage queries. Of course, without this restriction, security is unattainable: if the adversary were still allowed to issue a leakage query, it can get leakage of the challenge ciphertext. This means the security can only be guaranteed only when the device stops leaking, which is unnatural in the setting of continual leakage. This important problem was first addressed by Halevi and Lin [HL11] who defined and realized the notion of *after-the-fact* leakage resilience for encryption in the bounded (i.e. one-time) split-state leakage model. Our results are much more general: we secure general functionalities (not just encryption) from tampering as well as leakage, and we attain security under continuous rather than one-time attacks, solving several problems left explicitly open by Halevi and Lin.

Since we consider the split-state model, we can allow the adversary to keep issuing leakage and tampering queries after the challenge ciphertext is generated: we just make sure that any ciphertext cannot be decrypted via split-state leakage functions. In this sense, our results provide stronger guarantees (for LR encryption) than prior work [BKKV10, LRW11, LLW11, KKS11], even if one does not care about trusted local randomness and tamper-resilience.

1.2.2 Our Contributions

We construct a generic compiler that secure *any* cryptographic functionality against tampering and leakage attacks in the split-state model. Let $G(\cdot,\cdot)$ be any deterministic cryptographic functionality that, on input some secret state s and user-provided input x, outputs to the user the value y, and possibly updates its secret state to a new value s'; formally, $(y, s') = G(s, x)$. For example, G can be a stateful pseudorandom generator that, on input an integer m and a seed s, generates $m + |s|$ pseudorandom bits, and lets y be the first m of these bits, and updates its state to be the next $|s|$ bits. A signature scheme and a decryption functionality can also be modeled this way. A participant in an interactive protocol, such as a zero-knowledge proof, or an MPC protocol, can also be modeled as a stateful cryptographic functionality; the initial state s would represent its input and random tape; while the supplied input x would represent a message received by this participant. A construction that secures such a general stateful functionality G against tampering and leakage is therefore the most general possible result. This is what we achieve: our construction works

for any efficient deterministic cryptographic functionality G and secures it against tampering and leakage attacks in the split-state model, without access to any randomness after initialization. Any randomized functionality G can be securely derandomized using a pseudorandom generator whose seed is chosen in the initialization phase; our construction also applies to such a derandomized version of G. Quantitatively, our construction tolerates continual leakage of as many as $(1 - o(1))n$ bits of the secret memory, where n is the size of the secret memory.

Our construction works in the common reference string (CRS) model (depending on the complexity assumptions, this can be weakened to the common random string model); we assume that the adversary cannot alter the CRS. Trusted access to a CRS is not a strong additional assumption. A manufacturer of the device is already trusted to produce a correct device; it is therefore reasonable to also trust the manufacturer to hard-wire a CRS into the device. The CRS itself can potentially be generated in collaboration with other manufacturers, using a secure multi-party protocol.

Our construction makes the following complexity assumptions:

(1) The existence of a public-key cryptosystem that remains semantically secure even when an adversary is given $g(\mathsf{sk})$ for an arbitrary poly-time computable $g : \{0,1\}^{|\mathsf{sk}|} \mapsto \{0,1\}^{|sk|^{\Theta(1)}}$; for example, the decisional Diffie-Hellman (DDH) assumption is sufficient: the cryptosystem due to Naor and Segev [NS09] relies on DDH and is good enough for our purposes; in fact it gives more security than we require. Very recently, Hazay et al. [HLAWW12] showed how to construct a leakage-resilient cryptosystem (encryption scheme) using a regular one (i.e. semantic secure encryption schemes). Their parameters are worse, but their construction does not rely on specific number theoretic assumptions.

(2) The existence of robust non-interactive zero-knowledge proof systems for an appropriate NP language. For example, de Santis et al.'s [DDO$^+$01] construction of robust NIZK for all languages in NP suffices; although a construction for a more specialized language suffices as well.

In Section 11.3 we discuss the complexity assumptions needed here in more detail; we also analyze the efficiency of our construction and show that when instantiated with the NIZK due to Groth [Gro06] and a technique due to Meiklejohn [Mei09], we get efficiency that is compatible with practical use (as opposed to instantiating with NIZK due to de Santis et al., which is only of theoretical interest).

Additional result. Dziembowski et al. [DPW10] only give a random-oracle-based construction of non-malleable codes for the split-state tampering functions; a central open problem from that paper was to construct these codes without relying on the random oracle. We give such a non-malleable code in the CRS model, under the assumptions above. We then use this result as a building block for our main result; but it is of independent interest.

Very recently, Dziembowski et al. [DKO13] constructed non-malleable codes for one bit in the split-state model. Their scheme is information-theoretically secure and leakage resilient, and does not require CRS. The only drawback is that it can only encode one-bit. A follow up work by Aggarwal et al. [ADL13] constructed information theoretically secure non-malleable codes for strings in the split-state model. We also remark that our compiler uses an underlying non-malleable code in a black-box way, and thus any further improvement of the construction will give a direct improvement of our result.

Part I

Secure Delegation – Memory and Streaming Delegation Schemes

Chapter 2

Overview

In what follows we present the high-level overview of our constructions of memory and streaming delegation schemes. For more elaborate overviews, we refer the reader to Section 5.2.1 for an overview of our memory delegation scheme, and to Section 6.2.1 for an overview of our streaming delegation scheme.

2.1 Overview of our Memory Delegation Scheme

The starting point of this work is the observation of Goldwasswer *et al.* [GKR08], that their delegation protocol can be verified *very* efficiently (in time sub-linear in the input size), if the delegator has oracle access to the low-degree extension of the input x (we refer the reader to Section 3.3 for the definition of a low-degree extension). Moreover, as observed by [GKR08], the delegator needs to access this low-degree extension LDE_x at a single point z, which depends only on the random coin tosses of the delegator.

This observation immediately gives rise to a memory delegation scheme with *one-time* soundness: The delegator's secret state will be $(z, \mathrm{LDE}_x(z))$. Then, she will use this secret state in order to verify computation using the GKR protocol. As was argued by Goldwasswer *et al.*, this indeed works if the delegator runs the delegation protocol *once*. However, the soundness crucially relies on the fact that the delegator's secret state is indeed secret, and if the delegator uses this state more than once, then soundness breaks completely.

One idea, following the idea of Gennaro *et al.* [GGP10], is to use a fully homomorphic encryption

(FHE) scheme[1] to encrypt all the communication, in order to hide the secret state. This indeed works if the worker does not learn whether the delegator accepts or rejects his proofs. However, if the worker does learn the verdict of the delegator, then there are known attacks that break soundness.

In the streaming setting, we follow this approach, and we succeed in overcoming this problem, and construct a scheme that is sound even if the worker does learn the verdict of the delegator. We could follow this approach in the memory delegation setting as well. However, for several reasons, we choose to take a different approach. First, the approach above relies on the existence of an FHE scheme, whereas our memory delegation scheme relies on the existence of a poly-logarithmic PIR scheme (see Definition 5), arguably a more reasonable assumption. Second, the approach above results with the delegator having a secret state, whereas in our memory delegation scheme, the state of the delegator is public. Finally, the construction and proof of the memory delegation scheme is simpler.

In our approach, instead of having $(z, \mathrm{LDE}_x(z))$ as the delegator's secret state, the delegator keeps a tree-commitment of the entire LDE_x as her secret state (see Section 3.5 for the definition of a tree-commitment). Namely, she chooses a random hash function h from a collision-resistant hash family, and keeps $(h, T_h(\mathrm{LDE}_x))$ as her state. In addition to giving the worker her memory x, she also gives him the hash function h. We stress that her state is not secret, which makes the proof of security significantly simpler than that in the streaming setting (where the delegator's state is secret).

Very roughly speaking, when the delegator wishes to delegate the computation of a function f, they execute $\mathsf{Compute}(f)$ by simply running the (non-interactive) delegation protocol $\mathrm{GKR}(f)$. Recall that at the end of the GKR protocol the delegator needs to verify the value of $\mathrm{LDE}_x(r)$ for a random r. However, she doesn't have x, since it was delegated to the prover, and all she has is the state $(h, T_h(\mathrm{LDE}_x))$. So, rather than computing the value of $\mathrm{LDE}_x(r)$ on her own, the worker will reveal this value, by sending the augmented path in the Merkle tree corresponding to the leaf r.

Unfortunately the high-level description given above is a gross oversimplification of our actual scheme, and there are several technical issues that complicate matters. We elaborate on these in Section 2.3.

[1] A fully homomorphic encryption (FHE) scheme allows one to compute a function f (from some function class) and produce a ciphertext $\mathrm{Enc}(f(x))$ given input a ciphtertext $\mathrm{Enc}(x)$.

We mention that when the delegator wishes to update her memory from x to $g(x)$, she needs to update her secret state from $(h, T_h(\mathrm{LDE}_x))$ to $(h, T_h(\mathrm{LDE}_{g(x)}))$.[2] However, she cannot perform this operation on her own, since she does not have x. Instead she will delegate this computation to the worker, by requesting a $\mathsf{Compute}(g')$ operation, where $g'(x) = T_h(\mathrm{LDE}_{g(x)})$.

2.2 Overview of our Streaming Delegation Scheme

Our streaming delegation scheme is similar to our memory delegation scheme described above, and the main difference is in the way the certificate is generated and updated, and in the way the worker reveals the value $\mathrm{LDE}_x(r)$.

Generating and updating the certificate. Recall that in the memory delegation scheme, the certificate of the delegator D consists of a tree-commitment to the low-degree extension of her memory x. Namely, her certificate is $(h, T_h(\mathrm{LDE}_x))$, where h is a collision resistant hash function. Note that this certificate cannot be updated in a streaming manner, since any change to x changes the low-degree extension LDE_x almost everywhere.

Instead, in the streaming setting, we replace the tree commitment with an "*algebraic commitment*", which has the property that it can be updated efficiently when new data items arrive. The resulting certificate is a random point in the low-degree extension of the stream x; i.e., $(z, \mathrm{LDE}_x(z))$ for a random point z. This certificate is efficiently updatable, if we assume some upper-bound N on the size of the stream, and we take parameters $\mathbb{H}, \mathbb{F}, m$ of the low-degree extension, such that

$$|\mathbb{H}| = \mathrm{polylog}(N),\ m = \theta\left(\frac{\log N}{\log\log N}\right),\ |\mathbb{F}| = \mathrm{poly}(|\mathbb{H}|) \tag{2.1}$$

(this follows from Proposition 6).

As in the memory delegation scheme, at the end of each delegation protocol, the delegator needs to verify the value of $\mathrm{LDE}_x(r)$ at a random point r. In the memory delegation scheme this was done using a Reveal protocol where the worker reveals the augmented path of the leaf r in the Merkle tree-commitment of LDE_x. In the streaming setting, the Reveal protocol is totally different, since the delegator cannot compute the tree-commitment of LDE_x. Unfortunately, unlike in the memory

[2] Actually, for technical reasons she will need to choose a fresh hash function $h' \leftarrow \mathcal{H}$ during each Update. We discard this technical issue here.

delegation scheme, in the streaming setting constructing a *reusable* and *sound* reveal protocol is highly non-trivial.

The Reveal protocol. Our starting point is a basic reveal protocol Reveal_1 described in Figure 2.1. Note that the soundness of Reveal_1 relies on the secrecy of the certificate σ. Namely, assuming that W does not know the point z, it is not hard to see, by the Schwartz-Zippel Lemma, that an adversarial worker can cheat with probability at most $d/|\mathbb{F}|$, where d is the (total) degree of LDE_x.

Reveal_1 protocol: D stores a *secret* state $\sigma = (z, \text{LDE}_x(z))$, where $x \in \{0,1\}^N$ and z is a random point in $\mathbb{F}^m$, and wants to learn the value of $\text{LDE}_x(s)$ from W.

- D sends to W the line ℓ_{sz} that passes through the points s and z. More specifically, D chooses two random points $\alpha_1, \alpha_2 \leftarrow \mathbb{F}$, and defines $\ell_{s,z}$ to be the line that satisfies $\ell_{s,z}(\alpha_1) = z$ and $\ell_{s,z}(\alpha_2) = s$.
- W returns a univariate polynomial $p : \mathbb{F} \to \mathbb{F}$, which is the polynomial LDE_x restricted to the line $\ell_{s,z}$ (i.e., $p = \text{LDE}_x|_{\ell_{s,z}}$).
- D checks whether $p(\alpha_1) = \text{LDE}_x(z)$, and if so accepts the value $p(\alpha_2) = \text{LDE}_x(s)$. Otherwise, she rejects.

Figure 2.1: Reveal_1 protocol

However, note that the Reveal_1 protocol is not reusable. Suppose that D uses the above reveal protocol to learn the value of LDE_x on two random points $s, s' \in \mathbb{F}^m$. From the two executions, an adversarial worker W^* receives two lines $\ell_{s,z}$ and $\ell_{s',z}$, and can learn the secret point z by taking the intersection of the two lines. Once W^* learns z, W^* can easily cheat by returning any polynomial p^* that agrees with LDE_x only on point z but disagrees on the remaining points.

As observed by Gennaro *et al.* [GGP10], a natural way to protect the secret point z, is to run the above Reveal protocol under a fully-homomorphic encryption (FHE) scheme. Namely, D generates a pair of keys $(\mathsf{pk}, \mathsf{sk})$ for a FHE $(\text{Gen}, \text{Enc}, \text{Dec}, \text{Eval})$, and sends pk and an encrypted line $\hat{\ell}_{s,z} = \text{Enc}_{\mathsf{pk}}(\ell_{s,z})$ to W, who can compute the polynomial $p = \text{LDE}_x|_{\ell_{s,z}}$ homomorphically under the encryption. Indeed, by the semantic security of FHE, an adversarial worker W^* cannot learn any information from D's message $\hat{\ell}_{s,z}$. This indeed makes the protocol reusable provided that W^* does not learn the decision bits of D, as proved in [GGP10, CKV10].

However, since the decision bit of D can potentially contain one bit information about the secret point z, it is not clear that security holds if W^* learns these decision bits. In fact, for both of the

delegation schemes of [GGP10, CKV10], which use FHE to hide the delegator D's secret state, there are known attacks that learn the whole secret state of D bit-by-bit from D's decision bits.

Fortunately, we are able to show that a variant of the Reveal_1 protocol described in Figure 2.2 is reusable even if W^* learns the decision bits of D. The main difference between Reveal_1 and Reveal_2 is that in Reveal_2, the delegator D uses a random *two-dimensional* affine subspace instead of a line, and uses an FHE to mask the entire protocol.

Reveal_2 protocol: D stores a secret state $\sigma = (z, \text{LDE}_x(z))$, where $x \in \{0,1\}^N$ and z is a random point in $\mathbb{F}^m$, and wants to learn the value of $\text{LDE}_x(s)$ from W.

- D does the following.
  1. Generate a pair of keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \text{Gen}(1^k)$ for a fully homomorphic encryption scheme FHE.
  2. Choose a random two-dimensional affine subspace $S_{s,z} \subset \mathbb{F}^m$ that contains the points s and z. More specifically, choose two random points $\alpha_1, \alpha_2 \leftarrow \mathbb{F}^2$ and let $S_{s,z} \subset \mathbb{F}^m$ be a random two-dimensional affine subspace that satisfies $S_{s,z}(\alpha_1) = z$ and $S_{s,z}(\alpha_2) = s$.
  3. Send $\hat{S}_{s,z} \leftarrow \text{Enc}_{\mathsf{pk}}(S_{s,z})$ and pk to W.
- W homomorphically computes the two-variate polynomial $p = \text{LDE}_x|_{S_{s,z}}$ under the FHE (denote the resulting ciphertext $\hat{p}$), and sends $\hat{p}$ to D.
- D decrypts and checks whether $p(\alpha_1) = \text{LDE}_x(z)$, and if so accepts the value $p(\alpha_2) = \text{LDE}_x(s)$.

Figure 2.2: Protocol Reveal_2

We prove that no efficient adversarial W^* can learn useful information about the secret point z from the Reveal_2 protocol. We note that the proof of the above statement is highly non-trivial, and is one of the main technical difficulties in this work. Informally, the proof first uses Lemma 16, which claims that the ciphertext $\hat{S}_{s,z}$ and the decision bit b of D (which depend on the strategy of W^*) do not give too much information about $S_{s,z}$ to W^*. In other words, the random subspace $S_{s,z}$ still has high (pseudo-)entropy from the point of view of W^*. Then it uses an *information-theoretic* argument to argue that a random point z in a sufficiently random (with high entropy) subspace $S_{s,z}$ is *statistically close* to a random point in $\mathbb{F}^m$, which implies that W^* does not learn useful information about z. We refer the reader to Section 4.1 for the techniques developed in order to prove the reusable soundness.

The Field Size. Recall that by Schwartz-Zippel Lemma, an adversarial worker can cheat with probability at most $d/|\mathbb{F}|$, where d is the (total) degree of LDE_x. Recall that in our setting of parameters:

$$|\mathbb{H}| = \mathrm{polylog}(N),\ m = \theta\left(\frac{\log N}{\log\log N}\right),\ |\mathbb{F}| = \mathrm{poly}(|\mathbb{H}|).$$

Thus, a cheating worker can cheat (and more importantly, obtain information about the secret z) with probability $d/|\mathbb{F}| = O(1/\mathrm{polylog}(N))$, which is not low enough.

The idea is to reduce the cheating probability to negligible by simply increasing the field size to be super-polynomial. However, we cannot increase the field size in the GKR protocol, since it will increase the complexity of the worker. Instead, we use an extension field $\tilde{\mathbb{F}}$ of $\mathbb{F}$, of super-polynomial size, only in the certificate and the Reveal protocol, but run the GKR protocols as before. Namely, the secret state is $\sigma = (z, \mathrm{LDE}^{\tilde{\mathbb{F}},\mathbb{H},m}(z))$ where $z \leftarrow \tilde{\mathbb{F}}^m$, The GKR protocol is run exactly as before with the parameters $(\mathbb{H}, \mathbb{F}, m)$.

2.3 Additional Technicalities

The high-level description given above (in Sections 2.1 and 2.2) is a gross oversimplification of our actual schemes, and there are several technical issues that complicate matters.

Recall that in the overview above, we claimed that $\mathsf{Compute}(f)$ merely runs GKR, in addition to a Reveal protocol which helps the delegator verify the GKR protocol.[3] There are several technical reasons why this actually does not work. In what follows, we explain what are the main technical problems with this simple idea, and we give the highlevel idea of how to overcome these problems.

1. The first technicality (the easiest one to deal with), is that the GKR delegation scheme does not have a negligible soundness error. In our setting, especially in the setting of memory delegation, it is very important to have negligible soundness. The reason is that if the soundness is non-negligible, then a cheating worker may cheat in the update procedure of the memory delegation scheme (which is also being delegated). The problem is that if a worker cheats even once in an update procedure, all soundness guarantees are mute from that point on. So, we really need the soundness error to be negligible. In order to reduce the soundness error, we will run the GKR protocol in parallel u times (for any parameter u such that $1/2^u = \mathsf{ngl}(k)$, where

[3] The Reveal protocol in the memory setting is totally different from the Reveal protocol in the streaming setting.

k is the security parameter). We denote the u-fold parallel repetition of GKR by $\mathrm{GKR}^{(u)}$. As a result the worker will need to reveal to u random points in the low-degree extension: $\mathrm{LDE}_x(r_1), \ldots, \mathrm{LDE}_x(r_u)$.

2. The second technical point is more subtle. In the offline stage, when the delegator computes the tree commitment $T_h(\mathrm{LDE}_x)$, she needs to choose the parameters $\mathbb{H}, \mathbb{F}, m$ for the low-degree extension. The typical choice for these parameters is:

$$|\mathbb{H}| = \mathrm{polylog}(n),\ |\mathbb{F}| = \mathrm{poly}(|\mathbb{H}|),\ m = O\left(\frac{\log n}{\log\log n}\right),$$

where $n = |x|$. When delegating the computation of a function f, the worker and delegator run $\mathrm{GKR}^{(u)}(f)$ and need to verify $\mathrm{LDE}_x(r_i) = v_i$ for random points $r_1, \ldots, r_u$. However, here the parameters of the low-degree extension LDE_x depend on the depth d of the circuit computing f. Namely, looking at the parameters given in [GKR08] (see Theorem 8), the parameters of the low-degree extension are

$$|\mathbb{H}'| = \theta(d \cdot \log n),\ m' = \theta\left(\frac{\log n}{\log d}\right),\ |\mathbb{F}'| = \mathrm{poly}(|\mathbb{H}'|).$$

Therefore, the worker cannot simply execute the Reveal protocols of the memory delegation or the streaming delegation. In the memory setting, the tree commitment is w.r.t. parameters $\mathbb{H}, \mathbb{F}, m$ whereas the delegator needs to verify $\mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i) = v_i$. In the streaming setting, the secret state of the delegator is $(z, \mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z))$, as opposed to $(z, \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(z))$, thus the Reveal protocol described in Section 2.2 doesn't work.

We get around this technical problem by delegating the functions $g_{r_i}(x) \triangleq \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i)$. Luckily, these functions can be computed by a poly-size circuit of depth at most $\log^2 n$, assuming the delegated function f is of poly-size (see Proposition 6). We delegate the computation of each of these g_{r_i} using $\mathrm{GKR}^{(u)}$ to ensure negligible soundness. Thus, finally the worker will need to reveal to u^2 points in LDE_x (u points for each g_{r_i}).[4]

3. The final technical difficulty is that all these algorithms need to run in parallel, since we want our final delegation schemes to be non-interactive (i.e., to consist of only two messages).

[4] We note that there are several ways to improve efficiency, such as thinking of $(g_{r_1}, \ldots, g_{r_u})$ as one function. However, for the sake of simplicity of exposition, we focus on the simplest (rather than most efficient) solution.

Typically, there is no problem in running several two-message protocols in parallel [BIN97, CHS05]. However, in our case, the delegator uses a common *secret* input in these protocols. Namely, the delegator uses secret randomness $r_1, \ldots, r_u \in (\mathbb{F}')^{m'}$ in the parallel repetition of the delegation protocol GKR(f) which ends with her needing to verify that $\mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i) = v_i$ for every $i \in [u]$. In addition she uses these same r_i's in the delegation protocols GKR(g_{r_i}). Moreover, at the end of each of the GKR(g_{r_i}) protocols, the delegator needs to verify that $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j}) = w_{i,j}$ for random points $z_{i,1}, \ldots, z_{i,u} \in \mathbb{F}^m$. Finally, they also run a reveal protocol for each $z_{i,j}$, denoted by Reveal($z_{i,j}$).

We note that the protocol GKR(f) (resp. GKR(g)) is not sound if the r_i's (resp. $z_{i,j}$'s) are a priori known to the worker. To ensure that soundness still holds even if we run all these algorithms in parallel, we mask parts of the delegator's message using a PIR scheme or an FHE scheme, and then we claim that the soundness error remains negligible. To this end, we use our parallel composition lemma (Lemma 28), which roughly states that if a set of protocols $\Pi_1, \ldots \Pi_t$ are executed in parallel, and the verifiers use the same common private randomness p in all these protocols, then the soundness remains if the messages of the verifiers hide this common secret randomness p. (We refer the reader to Section 4.2 for details.)

Chapter 3

Preliminaries

In this chapter, we present some cryptographic primitives that we will use for our constructions.

3.1 Computation Private Information Retrieval (PIR)

Definition 5 *Let k be the security parameter and N be the database size. Let Q^{PIR} and D^{PIR} be probabilistic circuits, and let R^{PIR} be a deterministic circuit. We say that* $\mathrm{PIR} = (Q^{\mathrm{PIR}}, D^{\mathrm{PIR}}, R^{\mathrm{PIR}})$ *is a poly-logarithmic private information retrieval scheme if the following conditions are satisfied:*

1. *(Size Restriction:) Q^{PIR} and R^{PIR} are of size $\leq \mathrm{poly}(k, \log N)$, and D^{PIR} is of size $\leq \mathrm{poly}(k, N)$. The output of Q^{PIR} and D^{PIR} is of size $\leq \mathrm{poly}(k, \log N)$.*

2. *(Perfect Correctness:)*[1] *$\forall N, \forall k, \forall$ database $x = (x_1, x_2, \ldots, x_N) \in \{0,1\}^N$, and $\forall i \in [N]$,*

$$\Pr\left[R^{\mathrm{PIR}}(k, N, i, (q, s), a) = x_i | (q, s) \leftarrow Q^{\mathrm{PIR}}(k, N, i), a \leftarrow D^{\mathrm{PIR}}(k, x, q)\right] = 1$$

3. *(User Privacy:) $\forall N, \forall k, \forall i, j \in [N]$, and $\forall$ adversary $\mathcal{A}$ of size at most 2^{k^3},*

$$\left|\Pr[\mathcal{A}(k, N, q) = 1 | (q, s) \leftarrow Q^{\mathrm{PIR}}(k, N, i)] - \Pr[\mathcal{A}(k, N, q) = 1 | (q, s) \leftarrow Q^{\mathrm{PIR}}(k, N, j)]\right| \leq 2^{-k^3}.$$

[1] For simplicity, we only define perfect correctness. However, usually a PIR scheme allows a negligible probability of error.

3.2 Fully Homomorphic Encryption

A public-key encryption scheme $\mathrm{E} = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ is said to be *fully homomorphic* if it is associated with an additional polynomial-time algorithm Eval, that takes as input a public key pk, a ciphertext $\hat{x} = \mathrm{Enc}_{\mathsf{pk}}(x)$ and a circuit C, and outputs, a new ciphertext $c = \mathrm{Eval}_{\mathsf{pk}}(\hat{x}, C)$, such that $\mathrm{Dec}_{\mathsf{sk}}(c) = C(x)$, where sk is the secret key corresponding to the public key pk. It is required that the size of $c = \mathrm{Eval}_{\mathsf{pk}}(\mathrm{Enc}_{\mathsf{pk}}(x), C)$ depends polynomially on the security parameter and the length of $C(x)$, but is otherwise independent of the size of the circuit C. We also require that Eval is deterministic, and the the scheme has perfect correctness (i.e. it always holds that $\mathrm{Dec}_{\mathsf{sk}}(\mathrm{Enc}_{\mathsf{pk}}(x)) = x$ and that $\mathrm{Dec}_{\mathsf{sk}}(\mathrm{Eval}_{\mathsf{pk}}(\mathrm{Enc}_{\mathsf{pk}}(x), C)) = C(x)$). For security, we simply require that E is semantically secure.

In a recent breakthrough, Gentry [Gen09] proposed a fully homomorphic encryption scheme based on ideal lattices. Following this, Dijk, Gentry, Halevi and Vaikuntanathan [vDGHV10] proposed an alternative construction based on the extended GCD assumption. More recently, Brakerski and Vaikuntanathan [BV11], Barkerski, Gentry, and Vaikuntanathan [BGV12], and Brakerski [Bra12] constructed amuch more efficient FHE schemes based on a more well-studied problem – learning with error (LWE) problems [Reg09]. In these schemes, the complexity of the algorithms $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ depends linearly on the *depth* d of the circuit C, where d is an upper bound on the depth of the circuit C that are allowed as inputs to Eval. However, under the additional assumption that these schemes are circular secure (i.e., remain secure even given an encryption of the secret key), the complexity of these algorithms are independent of C.

Our streaming memory delegation scheme relies on the existence of a fully homomorphic scheme. For the sake of simplicity, we assume that the FHE scheme has perfect completeness. We note that the FHE schemes of both [Gen09] and [vDGHV10] indeed have perfect completeness, and the later latticed-based constructions can be tweaked slightly such that they have perfect completeness.

3.3 Low Degree Extension

Let $\mathbb{H}$ be an extension field of $\mathbb{GF}[2]$, and let $\mathbb{F}$ be an extension field of $\mathbb{H}$ (and in particular, an extension field of $\mathbb{GF}[2]$), where $|\mathbb{F}| = \mathrm{poly}(|\mathbb{H}|)$.[2] We always assume that field operations can be

[2]Usually, when doing low degree extensions, $\mathbb{F}$ is taken to be an extension field of $\mathbb{GF}[2]$, and $\mathbb{H}$ is simply a subset of $\mathbb{F}$ (not necessarily a subfield). In this work, following the work of [GKR08], we take $\mathbb{H}$ to be a subfield. However, all that is actually needed is that it is of size 2^ℓ for some $\ell \in \mathbb{N}$.

performed in time that is poly-logarithmic in the field size. Fix an integer $m \in \mathbb{N}$. In what follows, we define the low degree extension of an n-element string $(w_0, w_1, \ldots, w_{n-1}) \in \mathbb{F}^n$ with respect to $\mathbb{F}, \mathbb{H}, m$, where $n \leq |\mathbb{H}|^m$.

Fix $\alpha : \mathbb{H}^m \to \{0, 1, \ldots, |\mathbb{H}|^m - 1\}$ to be any (efficiently computable) one-to-one function. In this paper, we take α to be the lexicographic order of $\mathbb{H}^m$. We can view $(w_0, w_1, \ldots, w_{n-1})$ as a function $W : \mathbb{H}^m \to \mathbb{F}$, where

$$W(z) = \begin{cases} w_{\alpha(z)} & \text{if } \alpha(z) < n, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

A basic fact is that there exists a unique extension of W into a function $\tilde{W} : \mathbb{F}^m \to \mathbb{F}$ (which agrees with W on $\mathbb{H}^m$; i.e., $\tilde{W}|_{\mathbb{H}^m} \equiv W$), such that $\tilde{W}$ is an m-variate polynomial of degree at most $|\mathbb{H}| - 1$ in each variable. Moreover, as is formally stated in the proposition below, the function $\tilde{W}$ can be expressed as

$$\tilde{W}(t_1, \ldots, t_m) = \sum_{i=0}^{n-1} \tilde{\beta}_i(t_1, \ldots, t_m) \cdot w_i,$$

where each $\tilde{\beta}_i : \mathbb{F}^m \to \mathbb{F}$ is an m-variate polynomial, that depends only on the parameters $\mathbb{H}, \mathbb{F}$, and m (and is independent of w), of size $\text{poly}(|\mathbb{H}|, m)$ and degree $|\mathbb{H}| - 1$ in each variable.

The function $\tilde{W}$ is called the *low degree extension* of $w = (w_0, w_1, \ldots, w_{n-1})$ with respect to $\mathbb{F}, \mathbb{H}, m$, and is denoted by $\text{LDE}_w^{\mathbb{F},\mathbb{H},m}$. We omit the index of $\mathbb{F}, \mathbb{H}, m$ when the context is clear. Also, sometimes we use $\tilde{W}$ for simplicity.

Proposition 6 *There exists a Turing machine that takes as input an extension field $\mathbb{H}$ of $\mathbb{GF}[2]$,[3] an extension field $\mathbb{F}$ of $\mathbb{H}$, and integer m. The machine runs in time $\text{poly}(|\mathbb{H}|, m)$ and outputs the unique $2m$-variate polynomial $\tilde{\beta} : \mathbb{F}^m \times \mathbb{F}^m \to \mathbb{F}$ of degree $|\mathbb{H}| - 1$ in each variable (represented as an arithmetic circuit of degree $|\mathbb{H}| - 1$ in each variable), such that for every $w = (w_0, w_1, \ldots, w_{n-1}) \in \mathbb{F}^n$, where $n \leq |\mathbb{H}|^m$, and for every $z \in \mathbb{F}^m$,*

$$\tilde{W}(z) = \sum_{p \in \mathbb{H}^m} \tilde{\beta}(z, p) \cdot W(p),$$

[3] Throughout this work, when we refer to a machine that takes as input a field, we mean that the machine is given a short (poly-logarithmic in the field size) description of the field, that permits field operations to be computed in time that is poly-logarithmic in the field size.

where $W : \mathbb{H}^m \to \mathbb{F}$ *is the function corresponding to* $(w_0, w-1, \ldots, w_{n-1})$ *as defined in Equation* (3.1)*, and* $\tilde{W} : \mathbb{F}^m \to \mathbb{F}$ *is its low degree extension (i.e., the unique extension of* $W : \mathbb{H}^m \to \mathbb{F}$ *of degree at most* $\mathbb{H} - 1$ *in each variable).*

Moreover, $\tilde{\beta}$ *can be evaluated in time* $\mathrm{poly}(|\mathbb{H}|, m, \log|\mathbb{F}|)$*. Namely, there exists a Turing machine that runs in time* $\mathrm{poly}(|\mathbb{H}|, m, \log|\mathbb{F}|)$ *that takes as input parameters* $\mathbb{H}, \mathbb{F}, m$ *(as above), and a pair* $(z, p) \in \mathbb{F}^m \times \mathbb{F}^m$*, and outputs* $\tilde{\beta}(z, p)$*. Furthermore, there exists a circuit for evaluating* $\tilde{\beta}$ *in the above sense with size* $\mathrm{poly}(|\mathbb{H}|, m, \log|\mathbb{F}|)$ *and depth* $\mathrm{poly}(m, \log|\mathbb{F}|)$*.*

Corollary 7 *There exists a Turing machine that takes as input an extension field* $\mathbb{H}$ *of* $\mathbb{GF}[2]$*, an extension field* $\mathbb{F}$ *of* $\mathbb{H}$*, an integer* m*, a sequence* $w = (w_0, w_1, \ldots, w_{n-1}) \in \mathbb{F}^n$ *such that* $n \leq |\mathbb{H}|^m$*, and a coordinate* $z \in \mathbb{F}^m$*. It runs in time* $n \cdot \mathrm{poly}(|\mathbb{H}|, m, \log|\mathbb{F}|)$*, and outputs the value* $\tilde{W}(z)$*, where* $\tilde{W}$ *is the unique low-degree extension of* w *(with respect to* $\mathbb{H}, \mathbb{F}, m$*). Furthermore, there exists a circuit for the same task with size* $n \cdot \mathrm{poly}(|\mathbb{H}|, m, \log|\mathbb{F}|)$ *and depth* $\mathrm{poly}(m, \log|\mathbb{F}|)$*.*

3.4 Delegation Schemes

In recent years, as cloud computing is gaining popularity, there have been many attempts to construct efficient delegation schemes. Loosely speaking, a delegation scheme is a protocol between a delegator D and a worker W, where the delegator asks the worker to do some computation, and prove that he indeed did the computation correctly. Typically, a delegation scheme is with respect to a class of functions $\mathcal{F}$, and the requirement is that on input (f, x) where $f \in \mathcal{F}$ and x is in the domain of f, the worker outputs $f(x)$, along with a proof (which may be interactive or non-interactive). The requirement is that the worker runs in time that is polynomial in the size of f (when representing f as a circuit), and the delegator runs in time that is significantly shorter than the size of f (as otherwise, it would simply compute $f(x)$ on its own). In this work, we use the 2-round delegation protocol of [GKR08], which in turn uses a round reduction technique from [KR09]. The protocol has the following guarantees.

Theorem 8 *[GKR08, KR09] Assume the existence of a poly-logarithmic PIR scheme, as defined in Definition 5. Let* k *be the security parameter, and let* $\mathcal{F}$ *be the family of functions computable by* $\mathcal{L}$*-space uniform boolean circuits of size* S *and depth* $d \geq \log S$*. Then, there exists a delegation protocol for* $\mathcal{F}$ *with the following properties.*

1. *The worker runs in time* $\mathrm{poly}(S,k)$ *and the delegator runs in time* $n \cdot \mathrm{poly}(k,d)$*, where* n *is the length of the input.*

2. *The protocol has perfect completeness and soundness* $s \leq \frac{1}{2}$ *(can be made arbitrarily small), where soundness is against any cheating worker of size* $\leq 2^{k^3}$*.*

3. *The protocol consists of two messages, with communication complexity* $d \cdot \mathrm{poly}(k, \log S)$*. Moreover, the first message sent by the delegator depends only on her random coin tosses, and is independent of the statement being proved.*

4. *If the delegator is given oracle access to the low-degree extension of* x*, rather than being given the input* x *itself, then it runs in time* $\mathrm{poly}(k,d)$*, and the protocol still has all the properties described above, assuming the parameters* $\mathbb{H}, \mathbb{F}, m$ *of the low-degree extension satisfy the following:*

$$|\mathbb{H}| = \theta(d \cdot \log n),\ m = \theta\left(\frac{\log n}{\log d}\right),\ |\mathbb{F}| = \mathrm{poly}(|\mathbb{H}|)$$

where poly *is a large enough polynomial.*[4] *Moreover, the delegator queries the low-degree extension of* x *at a single point, which is uniformly random (over his coin tosses).*

Throughout this paper, we denote this protocol by GKR.

3.5 Merkle Tree Commitments

Definition 9 *Let* $h : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k$ *be a hash function. A Merkle tree commitment of a sting* $x \in \{0,1\}^n$ *w.r.t.* h*, denoted by* $T_h(x)$*, is a* k*-bit string, computed as follows: The input* x *is partitioned into* $m = \lceil n/k \rceil$ *blocks* $x = (B_1, \ldots, B_m)$*, each block of size* k*.*[5] *These blocks are partitioned into pairs* (B_{2i-1}, B_{2i})*, and the hash function* h *is applied to each pair, resulting in* $m/2$ *blocks. Then, again these* $m/2$ *blocks are partitioned into pairs, and the hash function* h *is applied to each of these pairs, resulting with* $m/4$ *blocks. This is repeated* $\log m$ *times, resulting in a binary tree of hash values, until one block remains. This block is* $T_h(x)$*.*

[4] The larger poly is, the smaller the soundness becomes.
[5] For simplicity, we assume that m is a power of 2.

Chapter 4

Our Entropy and Parallel Composition Lemmas

In this section, we are going to present two important lemmas for our constructions. The first one is a leakage lemma that argues a distribution that has high "computational entropy" still has high computational entropy even given a "short" leakage. We will elaborate the term *computational entropy* and state the precise quantity of *short* in the following section. The second lemma argues that t two-round protocols Π_1, Π_2, Π_t who share some common random secret, when composed in parallel, have soundness error bounded roughly by $\varepsilon_1 + \varepsilon_2, \ldots, \varepsilon_t$, where each ε_i is the soundness error of Π_i[1]. Thus, composing a polynomial number of two-round protocols that have negligible soundness error yields a two-round protocol that has negligible soundness error. This lemma plays an important role in that our final protocols of the schemes compose several 2-round protocols (with some common secret) in parallel. The lemma is crucial In order to argue the soundness of the final protocol.

4.1 Our Leakage Lemma

In this section, we define some machinery that is needed in order to prove the soundness of our streaming delegation scheme in Section 6.2. The main contribution of this section is a leakage lemma (Lemma 22), which essentially says that given an encryption $(\mathsf{pk}, \mathrm{Enc}_{\mathsf{pk}}(S))$ of a random

[1] Protocols Π_i's need some additional structure. We elaborate the details in the later section.

2-dimensional subspace $S \leftarrow \mathbb{F}^{m\times 2}$, and given any additional arbitrary (not necessarily efficient) bit of leakage $b = L(\mathsf{pk}, \mathrm{Enc}_{\mathsf{pk}}(S))$, then a random vector $z \leftarrow S$ is computationally indistinguishable from a truly random vector $u \leftarrow \mathbb{F}^m$. We formally state and prove this lemma in Section 4.1.2.

This lemma plays a central role in analyzing the (reusable) soundness of our streaming delegation scheme. In this scheme, the delegator has a secret state, and we need to prove that a cheating worker cannot learn any information about her secret state, even after running several delegation protocols with the delegator, and learning the bit of whether she accepted or rejected. In the soundness proof, this verdict bit is thought of as a leakage bit.

In the proof of Lemma 22, which is our main leakage lemma, we use another leakage lemma (Lemma 16), which is formally stated below and proved in Section 4.1.1. This leakage lemma roughly says that conditioning on a short leakage cannot decrease the conditional pseudo-entropy of a random variable too much.

In order to even state these lemmas formally, we first need to define the notion of conditional pseudo-entropy. There are several possible notions of conditional pseudo-entropy with subtle differences. In the following, we present our definition along with discussions on other possible notions. We start with the information-theoretic notion of min-entropy and conditional min-entropy.

Definition 10 (Min-Entropy) *Let X be a distribution over finite support. The* min-entropy *of X is defined as*

$$\mathbf{H}_\infty(X) = \min_{x\in\mathrm{supp}(X)} \log \frac{1}{\Pr[X=x]} = -\log\left(\max_{x\in\mathrm{supp}(X)} \Pr[X=x]\right).$$

Definition 11 (Conditional Min-Entropy) *Let (X, C) be a joint distribution over finite support. The (worst-case)* conditional min-entropy *of X conditioned on C is defined as*

$$\mathbf{H}_\infty(X|C) = \min_{(x,c)\in\mathrm{supp}(X,C)} \log \frac{1}{\Pr[X=x|C=c]} = \min_{c\in\mathrm{supp}(C)} \mathbf{H}_\infty(X|_{C=c}).$$

The above worst-case definition may seem too stringent as it requires X to have good min-entropy conditioned on every possible $c \in \mathrm{supp}(C)$. Several relaxed definitions have been used. For example, Renner and Wolf [RW05] defined a smooth version of the above definition, where X has ε-smooth conditional min-entropy n conditioned on C if (X, C) is ε-close in statistical distance to a distribution (X', C') with $\mathbf{H}_\infty(X'|C') = n$. Such a slackness will be implicitly allowed in

our definition of conditional pseudo-entropy. On the other hand, Dodis, Ostrovsky, Reyzin, and Smith [DORS08] defined an average-case version of conditional min-entropy, where $\mathbf{H}_{\infty}^{avg}(X|C) = -\log(\mathrm{E}_{c \leftarrow C}[\max_x\{\Pr[X = x|C = c]\}])$.[2]

In the computational setting, Hsiao, Lu, and Reyzin [HLR07] defined conditional HILL entropy. Informally, X has high conditional HILL entropy conditioned on C if there exists a random variable $Y = Y(C)$ such that (1) (X, C) is computationally indistinguishable from (Y, C), and (2) Y has high average conditional min-entropy conditioned on C (a la [DORS08]). In this work, we use a slightly different definition. The only difference between our definition and the [HLR07] definition is that we use the worse-case version of conditional min-entropy, as opposed to the average-case version. We work with the worst-case definition since it is more convenient for our application and makes the analysis simpler. For convenience, we refer to our notion also as conditional HILL entropy.

Definition 12 (Conditional HILL Entropy) *Let (X, C) be a joint distribution over a finite support, and let $n, s \in \mathbb{N}$ and $\varepsilon \in (0, 1)$ be parameters. We say that X conditioned on C has* conditional HILL entropy *at least n against circuits of size s with advantage ε, denoted by $\mathbf{H}^{\mathsf{HILL}}_{\varepsilon,s}(X|C) \geq n$, if there exists a distribution $Y = Y(C)$ jointly distributed with C such that (1) $\mathbf{H}_{\infty}(Y|C) \geq n$, and (2) (X, C) and (Y, C) are computationally indistinguishable against circuits of size s with advantage ε.*[3]

In the asymptotic setting where there is a security parameter k, we say $\mathbf{H}^{\mathsf{HILL}}(X|C) \geq n$, if for every constant $c \in \mathbb{N}$, $\mathbf{H}^{\mathsf{HILL}}_{k^{-c},k^c}(X|C) \geq n$.

Remark 13 Note that in the above definition, we only consider distributions (Y, C) that are indistinguishable from (X, C), i.e., we do not allow modifying the distribution of C. An alternative weaker definition is to consider all distributions (X', C') that are indistinguishable from (X, C). The two definitions may not be equivalent in general.[4] We emphasize that the more stringent definition seems more relevant for cryptographic applications, since C is often some leakage information on X learned by an adversary. We further emphasize that we do not claim that our definition is the "right" one, and we only use it as a tool to prove our main result.

Our goal is to show that if X has high conditional HILL entropy conditioned on C (say, $\mathbf{H}^{\mathsf{HILL}}(X|C) \geq n$) and $B = B(X, C)$ is an arbitrary, but short (say, one bit) leakage information on X, then after further conditioning on B, X still has high conditional HILL entropy (i.e.,

[2] We note that the two notions of [RW05] and [DORS08] are equivalent up to an additive $\log(1/\varepsilon)$ term. A detailed discussion can be found in Appendix B of [DORS08].

[3] Note that the ε slackness is inherent in the above HILL-type definition.

[4] We note that the two definitions are equivalent when the length of C is short ($\leq O(\log k)$).

$\mathbf{H}^{\mathsf{HILL}}(X|C,B) \geq n - t$ for some small t). When B can be efficiently generated from (X,C), this is very easy to prove. However, proving this for general $B = B(X,C)$ is not trivial. Indeed, in order to prove this, we need to further strengthen our definition of conditional HILL entropy.

Definition 14 (Conditional HILL Entropy w.r.t. Samplable Distributions) *Let k be a security parameter. For a finite distribution (X,C), we say $\mathbf{H}^{\mathsf{HILL}}(X|C) \geq n$ w.r.t.* samplable *distributions if there exists a distribution $Y = Y(C)$ such that the following holds.*

1. $\mathbf{H}_\infty(Y|C) \geq n$.
2. *(X,C) and (Y,C) are computationally indistinguishable.*
3. *There exists a* $\mathrm{poly}(k)$ *time algorithm* Smp *that on input $c \in \mathrm{supp}(C)$, outputs a sample $y \leftarrow (Y|_{C=c})$.*

Remark 15 This definition differs from Definition 12 in two ways. First, we require the distributions $Y|_{C=c}$ to be efficiently samplable for every c. Second, we require a single distribution $Y = Y(C)$ such that (X,C) and (Y,C) are indistinguishable for any $\mathrm{poly}(k)$-size distinguisher; whereas in Definition 12, we fix the size k^c of distinguisher first and require a distribution $Y = Y(C)$ such that (X,C) and (Y,C) are indistinguishable for k^c-size distinguishers.

Lemma 16 *Let k be a security parameter and n, ℓ, t be any parameters such that $n \leq \mathrm{poly}(k)$, $\ell = O(\log k)$, and $t = \omega(\log k)$. Let (X,C) be a joint distribution over $\{0,1\}^* \times \{0,1\}^*$ of* $\mathrm{poly}(k)$ *length. If $\mathbf{H}^{\mathsf{HILL}}(X|C) \geq n$ w.r.t. samplable distributions, then for any distribution $B = B(X,C)$ over $\{0,1\}^\ell$, we have*

$$\mathbf{H}^{\mathsf{HILL}}(X|C,B) \geq n - t.$$

The lemma says that further conditioning on $O(\log k)$ bits can only decrease the conditional HILL entropy by $\omega(\log k)$. Note that an upper bound of $O(\log k)$ on the length of B is necessary, since the pseudo-entropy of X could be generated from merely $\omega(\log k)$ bits of real entropy. For example, X can be the output of a pseudo-random generator (PRG) with sub-exponential stretch, and B can be the whole seed, if the length limit on B is relaxed. On the other hand, we do not know whether the samplability assumption on $Y(C)$ is necessary or not. Moreover, we do not know whether we inherently need $\omega(\log k)$ entropy loss, or whether one can prove $\ell = O(\log k)$ entropy loss.

Before presenting the proof of the lemma, we first compare it with previous results of [DP08, RTTV08]. Dziembowski and Pietrzak [DP08] (implicitly in Lemma 3), and Reingold, Trevisan, Tulsiani, and Vadhan [RTTV08] (Theorem 1.3, phrased in a different language of "dense model theorem") proved that if $\mathbf{H}^{\mathsf{HILL}}(X) \geq n$ and E is an event that occurs with probability $p \geq 1/\mathrm{poly}(k)$, then after conditioning on the event E, $\mathbf{H}^{\mathsf{HILL}}(X|_E) \geq n - \log(1/p)$.[5] This implies a special case of our Lemma 16 where C is not present: Suppose $\mathbf{H}^{\mathsf{HILL}}(X) \geq n$ and $B = B(X)$ is of length $O(\log k)$, then $\mathbf{H}^{\mathsf{HILL}}(X|B) \geq n - \omega(\log k)$.

In contrast, we consider a more general setting where a (possibly long) prior leakage information C is presented, which may information-theoretically determine X. Indeed, in our setting, C is an encryption of X and hence determines X.

4.1.1 Proof of Lemma 16

Preliminaries

We proceed to present the proof of Lemma 16. The first part of our proof follows the same line as previous results [DP08, RTTV08], where we convert (conditional) HILL-type entropy to (conditional) "metric-type" entropy, defined by Barak, Shaltiel, and Wigderson [BSW03]. On the other hand, the second part of our proof is more involved than previous results. We start by defining a conditional version of metric entropy.

Conditional Metric Entropy. Loosely speaking, metric entropy is weaker than HILL entropy and is defined by switching the order of quantifiers in the definition of HILL entropy. Recall that the HILL definition says that X has HILL entropy n if there exists a random variable Y with min-entropy n such that every small distinguisher D fails to distinguish between X and Y. In contrast, the definition of metric entropy requires that for every small distinguisher D, there exists a random variable Y (which may depend on D) with min-entropy n such that D fails to distinguish between X and Y.

Definition 17 (Conditional Metric Entropy) *Let (X, C) be a joint distribution over a finite support. Let $n, s \in \mathbb{N}$ and $\varepsilon \in (0, 1)$ be parameters. We say X conditioned on C has* conditional metric entropy *at least n against* randomized circuits *of size s with advantage ε, denoted by*

[5] This is also pointed out by Fuller and Reyzin [FR11].

$\mathbf{H}^{\mathsf{metric}}_{\varepsilon,s}(X|C) \geq n$, *if for every* randomized circuit D *of size at most* s, *there exists a distribution* $Y = Y(C)$ *jointly distributed with* C *such that* $\mathbf{H}_\infty(Y|C) \geq n$, *and*

$$|\Pr[D(X,C)=1] - \Pr[D(Y,C)=1]| \leq \varepsilon.$$

In the asymptotic setting where there is a security parameter k, *we say* $\mathbf{H}^{\mathsf{metric}}(X|C) \geq n$, *if it holds that* $\mathbf{H}^{\mathsf{metric}}_{k^{-c},k^c}(X|C) \geq n$ *for every constant* $c \in \mathbb{N}$.

We emphasize that, except for the natural generalization to the conditional version, our definition differs from that of [BSW03] in that we allow the distinguishers to be randomized, as opposed to deterministic.[6] The reason is that, as pointed out by Vadhan [Vad10] and Dziembowski and Pietrzak [DP08], the result of [BSW03] does not hold when deterministic distinguishers are considered, and randomized distinguishers should be used instead.[7]

Lemma 18 (Theorem 5.2 of [BSW03], generalized) *Let* (X,C) *be a joint distribution over* $\{0,1\}^{m_1} \times \{0,1\}^{m_2}$, *and let* $\varepsilon, \delta > 0$, $s, k \in \mathbb{N}$ *be parameters. If* $\mathbf{H}^{\mathsf{metric}}_{\varepsilon,s}(X|C) \geq n$, *then* $\mathbf{H}^{\mathsf{HILL}}_{\varepsilon+\delta,s'}(X|C) \geq n$ *for* $s' = s \cdot O(\delta^2/(m_1+m_2))$.

Lemma 18 is proved in exactly the same way as the proof in [BSW03], where one uses von-Neuman's min-max theorem [Neu28] to switch the order of quantifiers. For the sake of completeness, we give a proof sketch below.

Proof. (sketch) For the sake of contradiction, assume that $\mathbf{H}^{\mathsf{HILL}}_{\varepsilon+\delta,s'}(X|C) < n$. This means that for every distribution $Y = Y(C)$ with $\mathbf{H}_\infty(Y|C) \geq n$, there exists a size s' distinguisher D that distinguishes (X,C) from (Y,C) with advantage $\geq \varepsilon + \delta$. Applying min-max theorem, we obtain the following statement. There exists a *distribution* $\mathcal{D}$ over size-s' distinguishers such that for every $Y = Y(C)$ with $\mathbf{H}_\infty(Y|C) \geq n$, we have

$$\left| \mathop{\mathrm{E}}_{D \leftarrow \mathcal{D}}[D(X,C)] - \mathop{\mathrm{E}}_{D \leftarrow \mathcal{D}}[D(Y,C)] \right| \geq \varepsilon + \delta.$$

[6] Note that, for HILL-type entropy, randomized distinguishers and deterministic distinguishers are essentially equivalent, since one can turn a randomized distinguisher to a deterministic one by fixing the "best" coins that preserves the advantage for distinguishing two distributions. In contrast, for the case of metric entropy, it is unclear whether randomized distinguishers can be converted into deterministic ones since the distinguisher needs to work for all distributions.

[7] [DP08], instead of using randomized distinguishers, use deterministic $[0,1]$-valued distinguishers. We choose to use randomized circuit distinguishers since we find them to be more natural than circuits with $[0,1]$-valued output.

Now, it can be shown by standard Chernoff and union bounds that there exists a set $S = \{D_1, \ldots, D_{O((m_1+m_2)/\delta^2)}\}$ of circuits in $\mathrm{supp}(\mathcal{D})$ such that for every $(z, c) \in \{0,1\}^{m_1} \times \{0,1\}^{m_2}$,

$$\left| \mathop{\mathrm{E}}_{D \leftarrow \mathcal{D}}[D(z,c)] - \mathop{\mathrm{E}}_{D_i \leftarrow S}[D_i(z,c)] \right| \leq \delta/2.$$

Since this holds point-wise, it follows that for every $Y = Y(C)$ with $\mathbf{H}_\infty(Y|C) \geq n$,

$$\left| \mathop{\mathrm{E}}_{D_i \leftarrow S}[D_i(X,C)] - \mathop{\mathrm{E}}_{D_i \leftarrow S}[D_i(Y,C)] \right| \geq \varepsilon.$$

We obtain a contradiction by observing that choosing a random circuit $D_i \leftarrow S$ and outputting $D_i(z,c)$ can be implemented by a size $s = s' \cdot O((m_1+m_2)/\delta^2)$ *randomized* circuit. ■

As a corollary, the lemma implies that conditional HILL entropy and conditional metric entropy are equivalent in the asymptotic setting.

Corollary 19 *Let k be a security parameter. For every joint distribution (X, C) of polynomially bounded length $|(X,C)| \leq \mathrm{poly}(k)$, we have $\mathbf{H}^{\mathsf{HILL}}(X|C) = \mathbf{H}^{\mathsf{metric}}(X|C)$.*

Formal Proof of Lemma 16

Proof. Suppose for contradiction that there exists a distribution $B = B(X,C)$ such that $\mathbf{H}^{\mathsf{HILL}}(X|C,B) < n - t$. By Corollary 19, this implies that $\mathbf{H}^{\mathsf{metric}}(X|C,B) < n - t$. Namely, there exist some constant $c_0 \in \mathbb{N}$ and a *randomized* circuit D of size k^{c_0} such that for every distribution $Z = Z(C,B)$ with $\mathbf{H}_\infty(Z|C,B) \geq n - t$,

$$|\Pr[D(X,C,B) = 1] - \Pr[D(Z,C,B) = 1]| > k^{-c_0} \stackrel{\mathrm{def}}{=} \varepsilon. \tag{4.1}$$

On the other hand, the fact that $\mathbf{H}^{\mathsf{HILL}}(X|C) \geq n$ w.r.t. sampleable distributions implies that there exists a distribution $Y = Y(C)$ such that (1) $\mathbf{H}_\infty(Y|C) \geq n$, (2) (X,C) and (Y,C) are computationally indistinguishable, and (3) there exists a PPT algorithm Smp that on input $c \in \mathrm{supp}(C)$, outputs a sample $y \leftarrow (Y|_{C=c})$.

For notational convenience, let

$$p_{c,b}(z) \triangleq \Pr[D(z,c,b) = 1].$$

We construct a polynomial-size (randomized) circuit D' that distinguishes between (X, C) and (Y, C), as follows. On input (w, c) which comes from either (X, C) or (Y, C), D' does the following:

1. Use the sampling algorithm Smp to sample $s = (4 \cdot 2^\ell/\varepsilon)$ independent samples of $y_i \leftarrow Y|_{C=c}$.
2. For every $b \in \{0,1\}^\ell$, compute estimators for $p_{c,b}(w)$ and $p_{c,b}(y_i)$, denoted by $\tilde{p}_{c,b}(w)$ and $\tilde{p}_{c,b}(y_i)$, respectively. More specifically, run $D(w, c, b)$ (resp., $D(y_i, c, b)$) with fresh randomness $t \triangleq \Theta(\ell(\log^2 k)(\log s)/\varepsilon^2)$ times, and let $\tilde{p}_{c,b}(w)$ (resp., $\tilde{p}_{c,b}(y_i)$) be the average of the outputs.
3. If there exists some $b^* \in \{0,1\}^\ell$ such that

$$\tilde{p}_{c,b}(w) \geq \max_{y_i}\{\tilde{p}_{c,b}(y_i)\} + \varepsilon/4,$$

then output 1. Otherwise, output 0.

Note that D' can be implemented by a randomized circuit of size $\mathrm{poly}(k, 2^\ell, 1/\varepsilon) = \mathrm{poly}(k)$. We also note that the parameter $t = \Theta(\ell(\log^2 k)(\log s)/\varepsilon^2)$ defined in Step 2 is chosen so that, with overwhelming probability, *all* estimators have error less than $\varepsilon/8$, i.e.,

$$|\tilde{p}_{c,b}(w) - p_{c,b}(w)| < \varepsilon/8, \text{ and } |\tilde{p}_{c,b}(y_i) - p_{c,b}(y_i)| < \varepsilon/8. \tag{4.2}$$

This follows from a standard Chernoff bound,[8] which says that a single estimator has error less than $\varepsilon/8$ with probability $1 - e^{-\Omega(t\varepsilon^2)} = 1 - e^{-\Omega(\ell(\log^2 k)(\log s))}$. Since there are $2^\ell \cdot (s+1)$ estimators, by a union bound, the probability that all estimators have error less than $\varepsilon/8$ is at least

$$1 - e^{-\Omega(\ell(\log^2 k)(\log s))} \cdot 2^\ell \cdot (s+1) \geq 1 - \mathsf{ngl}(k).$$

We proceed to prove the following two claims, which jointly imply that D' distinguishes between (X, C) and (Y, C) with advantage $\varepsilon/4 - \mathsf{ngl}$, and thus complete the proof.

Claim 20 $\Pr[D'(Y, C) = 1] \leq \varepsilon/4$.

Claim 21 $\Pr[D'(X, C) = 1] \geq \varepsilon/2 - \mathsf{ngl}$.

[8] We use the following basic version of Chernoff bound: Let $A_1, \ldots, A_n$ be i.i.d. boolean random variables with $\Pr[A_i = 1] = p$, and let $\varepsilon \in (0, 1)$ be a parameter. Then

$$\Pr\left[\left|\left(\frac{1}{n}\sum A_i\right) - p\right| \geq \varepsilon\right] \leq e^{-\Omega(n\varepsilon^2)}.$$

Proof of Claim 20. Note that when $(w, c) \leftarrow (Y, C)$, then w and y_i's are i.i.d. copies of $(Y|_{C=c})$. Hence, for every $b \in \{0,1\}^\ell$,

$$\Pr[p_{c,b}(w) > \max_{y_i}\{p_{c,b}(y_i)\}] < 1/s.$$

By a union bound,

$$\Pr[\exists b^* \in \{0,1\}^\ell \text{ s.t. } p_{c,b^*}(w) > \max_{y_i}\{p_{c,b^*}(y_i)\}] < 2^\ell/s = \varepsilon/4.$$

Denote by E_{Good} the event that Equation (4.2) holds; i.e., the event that all estimators have error less than $\varepsilon/8$. Recall that we chose the parameter so that event E_{Good} holds with overwhelming probability (i.e., probability $1 - \mathsf{ngl}(k)$). Note that if event E_{Good} holds,

$$\tilde{p}_{c,b}(w) \geq \max_{y_i}\{\tilde{p}_{c,b}(y_i)\} + \varepsilon/4 \quad \Rightarrow \quad p_{c,b}(w) > \max_{y_i}\{p_{c,b}(y_i)\}$$

Therefore,

$$\begin{aligned}
&\Pr[D'(Y,C) = 1] \\
&= \Pr\left[\exists b \in \{0,1\}^\ell \text{ s.t. } \tilde{p}_{c,b}(w) \geq \max_{y_i}\{\tilde{p}_{c,b}(y_i)\} + \varepsilon/4\right] \\
&\leq \Pr\left[\left(\exists b \in \{0,1\}^\ell \text{ s.t. } \tilde{p}_{c,b}(w) \geq \max_{y_i}\{\tilde{p}_{c,b}(y_i)\} + \varepsilon/4\right) \wedge E_{Good}\right] + \Pr[\neg E_{Good}] \\
&\leq \Pr\left[\exists b \in \{0,1\}^\ell \text{ s.t. } p_{c,b}(w) > \max_{y_i}\{p_{c,b}(y_i)\}\right] + \mathsf{ngl}(k) \\
&\leq \varepsilon/4 + \mathsf{ngl}(k).
\end{aligned}$$

■

Proof of Claim 21. We first argue that we can assume, without loss of generality, that for every Z with $\mathbf{H}_\infty(Z|C,B) \geq n - t$,

$$\Pr[D(X,C,B) = 1] - \Pr[D(Z,C,B) = 1] > \varepsilon. \tag{4.3}$$

The reason is the following: Suppose for the sake of contradiction that there exists some distribution Z with $\mathbf{H}_\infty(Z|C,B) \geq n-t$ such that

$$\Pr[D(X,C,B)=1] - \Pr[D(Z,C,B)=1] > \varepsilon,$$

and yet there exists another distribution Z' with $\mathbf{H}_\infty(Z'|C,B) \geq n-t$ such that

$$\Pr[D(Z',C,B)=1] - \Pr[D(X,C,B)=1] > \varepsilon.$$

Then one can construct a distribution Z'', by taking an appropriate convex combination of Z and Z', such that $\mathbf{H}_\infty(Z''|C,B) \geq n-t$ and $\Pr[D(X,C,B)=1] = \Pr[D(Z'',C,B)=1]$, contradicting Equation (4.1).

For every pair $(c,b) \in \mathrm{supp}(C,B)$, let H_{cb} be a set of the "heaviest" 2^{n-t} points w that maximize $p_{cb}(w)$. Consider the distribution $Z^+ = Z^+(C,B)$ such that $Z^+|_{(C,B)=(c,b)}$ is the uniform distribution over $H_{c,b}$. Note that $\mathbf{H}_\infty(Z^+|C,B) = n-t$. For every $(c,b) \in \mathrm{supp}(C,B)$, define

$$p^+_{c,b} \triangleq \Pr[D(Z^+|_{C=c,B=b}, c, b) = 1].$$

Using these notations, Equation (4.3) implies that

$$\mathop{\mathrm{E}}_{(x,c,b)\leftarrow(X,C,B)}[p_{c,b}(x)] - \mathop{\mathrm{E}}_{(c,b)\leftarrow(C,B)}[p^+_{c,b}] \geq \varepsilon.$$

By a Markov argument, with probability at least $\varepsilon/2$ over $(x,c,b) \leftarrow (X,C,B)$,

$$p_{c,b}(x) - p^+_{c,b} \geq \varepsilon/2.$$

We next prove that in this case, $\Pr[D'(x,c)=1] \geq 1 - \mathsf{ngl}(k)$, which implies that

$$\Pr[D'(X,C)=1] \geq (1-\mathsf{ngl}(k)) \cdot (\varepsilon/2) \geq \varepsilon/2 - \mathsf{ngl}(k).$$

Fix any x, c, b such that $p_{c,b}(x) - p^+_{c,b} \geq \varepsilon/2$. It remains to prove that

$$\Pr[D'(x,c) = 1] \geq 1 - \mathsf{ngl}(k).$$

Note that by definition, $p_{cb}(w) \leq p^+_{c,b}$ for every $w \notin H_{cb}$. Recall that we choose the parameter so that with overwhelming probability, all estimators have error at most $\varepsilon/8$. As before, denote by E_{Good} the event that indeed all estimators have error at most $\varepsilon/8$.

$$\begin{aligned}
&\Pr[D'(x,c) = 1] \\
&\quad \geq \quad \Pr[\tilde{p}_{cb}(x) \geq \max_{y_i}\{\tilde{p}_{cb}(y_i)\} + \varepsilon/4] \\
&\quad \geq \quad \Pr[(\tilde{p}_{cb}(x) \geq \max_{y_i}\{\tilde{p}_{cb}(y_i)\} + \varepsilon/4) \wedge E_{Good}] - \Pr[\neg E_{Good}] \\
&\quad \geq \quad \Pr[p_{cb}(x) \geq \max_{y_i}\{p_{cb}(y_i)\} + \varepsilon/2] - \mathsf{ngl}(k) \\
&\quad \geq \quad \Pr[\forall i, y_i \notin H_{cb}] - \mathsf{ngl}(k) \\
&\quad \geq \quad (1 - \mathsf{ngl}(k)) - \mathsf{ngl}(k) \\
&\quad \geq \quad 1 - \mathsf{ngl}(k),
\end{aligned}$$

where the second-to-last inequality follows from the fact that $|H_{cb}| = 2^{n-t}$ and $\mathbf{H}_\infty(Y|_{C=c}) \geq n$.

■

■

4.1.2 Main Leakage Lemma

Throughout this section, we consider the following setting. Let $k \in \mathbb{N}$ be a security parameter. Let $\mathbb{F}$ be a finite field of size $q \geq 2^{\log^2 k}$, and let $E = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ be any semantic secure public-key encryption scheme. Let $m \leq \mathrm{poly}(k)$ be a parameter. We define the following random variables.

1. Let $S \in_R \mathbb{F}^{m \times 2}$ be a random m-by-2 matrix representing a random 2-dimensional linear subspace

   $$\{a_1 v_1 + a_2 v_2 : a_1, a_2 \in \mathbb{F}\},$$

   where v_1, v_2 are columns of S.

2. Let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{Gen}(1^k)$, and let $\hat{S} = \mathrm{Enc}_{\mathsf{pk}}(S)$.

3. Let $L : \{0,1\}^* \to \{0,1\}$ be an arbitrary (randomized, not necessarily efficient) leakage function that maps $(\hat{S}, \mathsf{pk})$ to one bit, and let $b = L(\hat{S}, \mathsf{pk})$.

4. Let $u \leftarrow \mathbb{F}^m$ be a random point in $\mathbb{F}^m$, and z be a random point in S. Specifically, $z = S \cdot a = a_1 v_1 + a_2 v_2$ where $a = (a_1, a_2)$ is a uniformly random vector in $\mathbb{F}^2$.

Our goal in this section is to prove the following lemma.

Lemma 22 *In the above setting, the distributions* $(z, \hat{S}, \mathsf{pk}, b)$ *and* $(u, \hat{S}, \mathsf{pk}, b)$ *are computationally indistinguishable.*

The lemma says that computationally, the encryption $(\hat{S}, \mathsf{pk})$ together with an arbitrary leakage bit b does not leak any information about z. Note that information-theoretically, $(\hat{S}, \mathsf{pk})$ does contain information about z, since we know that z is in S. Also note that when b is not present, semantic security readily implies that $(z, \hat{S}, \mathsf{pk})$ and $(u, \hat{S}, \mathsf{pk})$ are computationally indistinguishable. However, when the bit b is present, the proof becomes highly non-trivial, and in particular, our proof makes use of Lemma 16.

Proof. We first consider the distribution $(S, \hat{S}, \mathsf{pk})$. By the semantic security, $(S, \hat{S}, \mathsf{pk})$ is computationally indistinguishable from $(S', \hat{S}, \mathsf{pk})$, where S' is an i.i.d. copy of S. Note that this implies

$$\mathbf{H}^{\mathsf{HILL}}(S|\hat{S}, \mathsf{pk}) \geq 2m \cdot \log q$$

w.r.t. sampleable distributions. By Lemma 16,

$$\mathbf{H}^{\mathsf{HILL}}(S|\hat{S}, \mathsf{pk}, b) \geq (2m \cdot \log q) - t,$$

where we set $t = (\log q)/4 = \omega(\log k)$. Namely, for every constant $c \in \mathbb{N}$, there exists a distribution $T = T(\hat{S}, \mathsf{pk}, b)$ such that:

1. $\mathbf{H}_\infty(T|\hat{S}, \mathsf{pk}, b) \geq (2m \cdot \log q) - t$, and

2. $(S, \hat{S}, \mathsf{pk}, b)$ and $(T, \hat{S}, \mathsf{pk}, b)$ are computationally indistinguishable against circuits of size k^c with advantage k^{-c}.

Note that $z \leftarrow S$ is efficiently sampleable, say, using a circuit of size $m^3 = \mathrm{poly}(k)$. Therefore, the distributions $(z_S, S, \hat{S}, \mathsf{pk}, b)$ and $(z_T, T, \hat{S}, \mathsf{pk}, b)$, where $z_S \leftarrow S$ and $z_T \leftarrow T$ are random points

in S and T respectively, are computationally indistinguishable against circuits of size $(k^c - m^3)$ with advantage k^{-c}.

Clearly, we can remove S and T from the distributions while preserving the indistinguishability. Namely, the distributions $(z_S, \hat{S}, \mathsf{pk}, b)$ and $(z_T, \hat{S}, \mathsf{pk}, b)$ are computationally indistinguishable against circuits of size $(k^c - m^3)$ with advantage k^{-c}.

We next claim that for any distribution $T = T(\hat{S}, \mathsf{pk}, b)$ over $\mathbb{F}^{m \times 2}$, with

$$\mathbf{H}_\infty(T|\hat{S}, \mathsf{pk}, b) \geq (2m \cdot \log q) - t,$$

the distributions $(z_T, \hat{S}, \mathsf{pk}, b)$ and $(u, \hat{S}, \mathsf{pk}, b)$ are statistically close (i.e., have distance $\mathsf{ngl}(k)$). This would imply that $(z, \hat{S}, \mathsf{pk}, b)$ and $(u, \hat{S}, \mathsf{pk}, b)$ are computationally indistinguishable against circuits of size $(k^c - m^3)$ with advantage $k^{-c} + \mathsf{ngl}(k)$. Observing that the above argument holds for all constants $c \in \mathbb{N}$, we conclude that $(z, \hat{S}, \mathsf{pk}, b)$ and $(u, \hat{S}, \mathsf{pk}, b)$ are computationally indistinguishable, as desired.

Thus, it remains to prove that indeed $(z_T, \hat{S}, \mathsf{pk}, b)$ and $(u, \hat{S}, \mathsf{pk}, b)$ are statistically close. To this end, we use Lemma 23 below, which states that if a distribution T over $\mathbb{F}^{m \times 2}$ has min-entropy at least $(2m \cdot \log q) - t$ and $a = (a_1, a_2) \leftarrow \mathbb{F}^2$, then $z = T \cdot a$ is ε-close to uniform, where $\varepsilon \leq 2m \cdot q^{-1/4} = \mathsf{ngl}(k)$.

Recall that according to our definition of conditional min-entropy (which is a worse-case definition), $\mathbf{H}_\infty(T|\hat{S}, \mathsf{pk}, b) \geq (2m \cdot \log q) - t$ implies that $\mathbf{H}_\infty(T|_{(\hat{S}, \mathsf{pk}, b) = \sigma}) \geq (2m \cdot \log q) - t$ for every $\sigma \in \mathrm{supp}(\hat{S}, \mathsf{pk}, b)$. Thus, Lemma 23 implies that conditioned on any $(\hat{S}, \mathsf{pk}, b) = \sigma$, the random variable z_T is $\mathsf{ngl}(k)$-close to uniform. This clearly implies $(z_T, \hat{S}, \mathsf{pk}, b)$ and $(u, \hat{S}, \mathsf{pk}, b)$ are statistically close, as desired.

■

Lemma 23 *Let X be a distribution over $\mathbb{F}^{m \times 2}$ with $\mathbf{H}_\infty(X) \geq (2m \cdot \log q) - (\log q)/4$, and $a = (a_1, a_2) \leftarrow \mathbb{F}^2$. Then $(X \cdot a) \in \mathbb{F}^m$ is ε-close to uniform with $\varepsilon \leq 2m \cdot q^{-1/4}$.*

We prove the lemma in the next section.

4.1.3 Proof of Lemma 23

Let $x^1, \ldots, x^m \in \mathbb{F}^2$ denote the m rows of X, and let $\mathcal{H}$ be the following hash function family

$$\mathcal{H} = \{h_a : \mathbb{F}^2 \to \mathbb{F} \text{ s.t. } h_a(x) = a_1 x_1 + a_2 x_2 \quad \forall a \in \mathbb{F}^2\}.$$

In the above notation, $X \cdot a = (h_a(x^1), \ldots, h_a(x^m))$. Loosely speaking, Lemma 23 holds for the following reasons:

- All rows of X have high min-entropy. Note that there are only $(\log q)/4$ bits of entropy missing from X, so intuitively, all rows x^i of X have at least $2\log q - (\log q)/4$ bits of entropy.
- $\mathcal{H}$ is a 2-universal hash function family. By Leftover Hash Lemma (Lemma 27 below), $h_a \leftarrow \mathcal{H}$ can extract randomness from all rows x^i of X.

We proceed to present necessary preliminaries for proving Lemma 23.

Definition 24 (Block Source) *Let $X = (X_1, \ldots, X_m)$ be a distribution, and let $\ell \in \mathbb{N}$ be a parameter. We say that X is a* block ℓ-source *if for every $i \in [m]$ and every $(x_1, \ldots, x_{i-1}) \in \text{supp}(X_1, \ldots, X_{i-1})$, $\mathbf{H}_\infty(X_i|_{(X_1,\ldots,X_{i-1})=(x_1,\ldots,x_{i-1})}) \geq \ell$.*

The following lemma says that if $X = (X_1, \ldots, X_m)$ has sufficiently high min-entropy, then X is (statistically close to) a block source.

Lemma 25 (see, e.g., [Vad10]) *Let $X = (X_1, \ldots, X_m)$ be a distribution over $\{0,1\}^{m \times n}$. Let $\Delta \in \mathbb{N}$ and $\varepsilon \in (0,1)$ be parameters. If $\mathbf{H}_\infty(X) \geq mn - \Delta$, then X is $(m\varepsilon)$-close to a block k-source with $k = n - \Delta - \log(1/\varepsilon)$.*

Definition 26 (2-universal) *A hash function family $\mathcal{H} = \{h : \{0,1\}^{n_1} \to \{0,1\}^{n_2}\}$ is* 2-universal *if for every $x \neq x' \in \{0,1\}^{n_1}$,*

$$\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(x')] \leq 1/2^{n_2}.$$

It is easy to verify that the hash function family $\mathcal{H} = \{h_a : \mathbb{F}^2 \to \mathbb{F}\}$ defined above is 2-universal.

Lemma 27 (Leftorver Hash Lemma (see, e.g., [Vad10])) *Let $\mathcal{H} = \{h : \{0,1\}^{n_1} \to \{0,1\}^{n_2}\}$ be a 2-universal family of hash functions. Let $k \in \mathbb{N}$ and $\varepsilon > 0$ be parameters such that $n_2 \leq k - 2\log(1/\varepsilon)$. For any distribution X with $\mathbf{H}_\infty(X) \geq k$, the distribution $(h, h(X))$ is ε-close to uniform in statistical distance.*

Moreover, if $X = (X_1, \ldots, X_m)$ is a block k-source, then the distribution $(h, h(X_1), \ldots, h(X_m))$ is $(m\varepsilon)$-close to uniform.

Lemma 23 follows readily by Lemma 25 and 27.

Proof. (of Lemma 23) Let $\varepsilon = q^{-1/4}$, and $k = 1.5 \log q$. By Lemma 25, $X = (x^1, \ldots, x^m)$ is $m\varepsilon$-close to a block k-source. By Lemma 27, $X \cdot a = (h_a(x^1), \ldots, h_a(x^m))$ is $(m\varepsilon + m\varepsilon)$-close to uniform. ■

4.2 Parallel Composition Lemma

In this section we give soundness guarantees for a protocol Π that executes several protocols $\Pi_1, \ldots, \Pi_t$ in parallel, where in each $\Pi_i = \langle P_i, V_i \rangle$ the verifier V_i uses the same private randomness p. Such a parallel composition lemma will be used to prove soundness both of our memory delegation scheme (in Section 5.2) and the streaming delegation scheme (in Section 6.2). For the sake of simplicity, we focus on 2-message protocols, though our results hold for protocols with arbitrary number of messages.

Let $\Pi_1, \Pi_2, \ldots, \Pi_t$ be protocols, where each $\Pi_i = \langle P_i, V_i \rangle$ is a 2-message protocol (where the first message is sent by the verifier V_i and the second message is sent by the prover P_i) for proving $x_i \in L_i$. Let $\Pi = \langle P, V(p) \rangle(x_1, \ldots, x_t)$ be the two-message protocol that runs the protocols $\Pi_1, \ldots, \Pi_t$ in parallel, where each Π_i is run with the input x_i, and each verifier V_i uses the *same* private random coin tosses p (in addition to some independent private randomness which each V_i may use). V accepts $(x_1, \ldots, x_t)$ if and only if at least one of the V_i's accepts $x_i \in L_i$. Thus, $\Pi(x_1, \ldots, x_t)$ should be thought of as a proof that there exists $i \in [t]$ such that $x_i \in L_i$.

We say that a protocol Π_i has soundness error s_i if for every false statement $x \notin L_i$, and for every efficient cheating prover P^*,

$$\Pr_p[V_i \text{ accepts the interaction } \langle P^*, V_i(p) \rangle(x)] \leq s_i,$$

where the randomness is over p and over any additional random coins that V_i may use.

In what follows we prove that if in each protocol Π_i, the verifier's messages are computationally indistinguishable for all different p's (of length at most $\text{poly}(k)$), then the soundness of all the Π_i's implies the soundness of Π.

Lemma 28 *Let k be the security parameter and $t \leq \text{poly}(k)$. Suppose that a protocol Π consists of a parallel composition of $\Pi_1, \Pi_2, \ldots, \Pi_t$ of the above form, and suppose that for each i the following two properties hold:*

1. *Π_i has soundness error s_i.*
2. *Let $\{M_{V_i(x_i,p)}\}$ be the distribution of V_i's first message, where x_i is the common input of V_i and P_i, and p is the the common private random coins of $V_1, \ldots, V_t$. Then, for all x_i, p, p' (of length bounded by $\text{poly}(k)$), the distributions $\{M_{V_i(x_i,p)}\}$ and $\{M_{V_i(x_i,p')}\}$ are computationally indistinguishable.*

Then Π has soundness error at most $\sum_{i\in[t]} s_i + \mathsf{ngl}(k)$.

Proof. Suppose for the sake of contradiction that there exists an efficient (parallel) cheating prover P^* and a false input $x = (x_1, \ldots, x_t)$ (i.e., an input x such that for every $i \in [t]$, $x_i \notin L_i$) such that P^* succeeds in convincing the verifier V running Π to accept x with probability

$$\varepsilon > \sum_{i\in[t]} s_i + \alpha(k),$$

for some non-negligible function α. We argue that there exists a coordinate $i \in [t]$ and an efficient cheating prover P_i^* for the protocol Π_i that succeed in convincing V_i to accept the false x_i with probability greater than $s_i + \alpha/t - \mathsf{ngl}(k)$, which contradicts the assumption.

For every $i \in [t]$, let W_i be the event that P^* successfully cheats on the i'th coordinate in the protocol $\Pi(x)$, and define $\varepsilon_i \triangleq \Pr[W_i]$. By definition, if P^* cheats on Π then at least one W_i holds. Using the union bound, this implies that $\sum_{i\in[t]} \varepsilon_i \geq \varepsilon$.

Now for each $i \in [t]$ we define a cheating prover P_i^* for the protocol $\Pi_i(x_i)$ with the following strategy. P_i^*, upon receiving a message $M_{V_i(x_i,p)}$ from V_i, simulates the interaction between P^* and V by embedding the real message of V_i into the i-th coordinate, and setting the other V_j's messages to be $M_{V_j(x_j,0)}$ for $j \neq i$. Then P_i^* replies what P^* does in the i'th coordinate.

Denote the success probability of P_i^* by $\tilde{\varepsilon}_i$. By the message indistinguishability of $\{M_{V_j(x_j,0)}\}$ and $\{M_{V_j(x_j,p)}\}$ for $j \neq i$, we know that $\tilde{\varepsilon}_i > \varepsilon_i - \mathsf{ngl}(k)$; otherwise there is a distinguisher that distinguishes between the distributions $\{M_{V_k(x_k,0)}\}$ and $\{M_{V_k(x_k,p)}\}$ for some $k \neq i$ (by a standard hybrid argument).

Thus,

$$\sum_{i\in[t]} \tilde{\varepsilon}_i \geq \sum_{i\in[t]} \varepsilon_i - t \cdot \mathsf{ngl}(k) \geq \varepsilon - \mathsf{ngl}(k) \geq \sum_{i\in[t]} s_i + \alpha(k) - \mathsf{ngl}(k).$$

This implies that there exists i such that $\tilde{\varepsilon}_i \geq s_i + \alpha/t - \mathsf{ngl}(k)$, which contradicts the assumption. ■

We note that in the streaming delegation and memory delegation schemes, the delegator (verifier) uses an FHE scheme or a PIR scheme to achieve the property that $\{M_{V_i(x_i,p)}\}$ and $\{M_{V_i(x_i,p')}\}$ are computationally indistinguishable. This allows us to make use of Lemma 28.

Chapter 5

Memory Delegation

5.1 Memory Delegation Model

In this section, we formally define our memory delegation model. We present our memory delegation scheme in Section 5.2.

Definition 29 (Memory Delegation Scheme) *Let $\mathcal{F}, \mathcal{G}$ be two sets of functions. A* memory delegation scheme *for functions in $\mathcal{F}$ and updates in $\mathcal{G}$, is an interactive protocol $\mathsf{mDel}_{\mathcal{F},\mathcal{G}} = \langle \mathsf{D}, \mathsf{W} \rangle$ between a delegator D and a worker W, of the following form:*

1. *The scheme mDel consists of two stages: an offline/preprocessing stage and an online stage. The offline stage is executed only once before the online stage, whereas the online stage can be executed many times.*

2. *In the offline stage, both the delegator D and the worker W receive a security parameter 1^k and an input $x \in \{0,1\}^n$. The worker stores x, and the delegator computes a short (possibly secret) string σ_{D} and stores it for future use.*

3. *In the online stage, the delegator can interact with the worker via the following two operations.*

   - *A computation operation $\mathsf{Compute}(f)$ where both parties take as input a function $f \in \mathcal{F}$. This input is in addition to the inputs that the parties store throughout the delegation protocol: the security parameter 1^k, the current memory content $x \in \{0,1\}^n$ stored only*

by the worker W*, and the short (possibly secret) string* σ_{D} *stored only by the delegator* D*. Then,* W *proves to* D *that* $y = f(x)$*, for some* $y \in \{0,1\}^*$*.*

- *An update operation* $\mathsf{Update}(g)$ *where both parties take as input a function* $g : \{0,1\}^n \to \{0,1\}^{n'} \in \mathcal{G}$*. This is in addition to the inputs that the parties store throughout the delegation protocol: the security parameters* 1^k*, the current memory content* $x \in \{0,1\}^n$ *stored only by the worker* W*, and the short (possibly secret) string* σ_{D} *stored only by the delegator* D*. Then* W *and* D *interact, where* W *"helps"* D *update her secret* σ_{D}*. At the end of the interaction, if* D *accepts, she updates her secret to some* σ'_{D} *and believes that the stored* x *has been updated to* $g(x)$*. Otherwise, she keeps her secret* σ_{D} *(and thinks of the previous* x *as unchanged).*

At the end of each operation, D *sends* W *a decision bit* $b \in \{0,1\}$ *for her acceptance or rejection.*

For a delegation scheme to be meaningful, it needs to have efficiency, completeness and soundness properties, defined below.

Definition 30 (Efficiency) *A delegation scheme* $\mathsf{mDel}_{\mathcal{F},\mathcal{G}}$ *has an efficient delegator in the offline stage if* D *runs in time* $\mathrm{poly}(k,n)$ *in the offline stage. It has an efficient delegator in the online stage if* D *runs in time* $\mathrm{poly}(k)$ *(independent of* n*) during each operation in the online stage.*

A delegation scheme $\mathsf{mDel}_{\mathcal{F},\mathcal{G}}$ *has an efficient worker if the runtime of* W *is* $\mathrm{poly}(k,n)$ *during both the offline stage and during each operation in the online stage, where* n *is the length of the delegated memory.*

Definition 31 (Completeness) *For any sets of functions* $\mathcal{F}, \mathcal{G}$*, a delegation scheme* $\mathsf{mDel}_{\mathcal{F},\mathcal{G}} = \langle \mathsf{D}, \mathsf{W} \rangle$ *has* perfect completeness *if for every* $k, n \in \mathbb{N}$*, and for every* $x \in \{0,1\}^n$*, the following holds with probability* 1*:*[1] *When* D *and* W *run the offline stage with input* $(1^k, x)$*, and then run the online stage polynomially many times with the operations* $\mathsf{Update}(g)$*, for any* $g \in \mathcal{G}$*, and* $\mathsf{Compute}(f)$*, for any* $f \in \mathcal{F}$*, the delegator* D *always accepts (i.e., sends* W *the decision bit 1).*

The definition of soundness is more elaborate, and requires defining the following security game. We emphasize that our soundness definition is *reusable* in the sense that we require that a (computationally bounded) cheating worker cannot convince the delegator to accept a wrong statement,

[1] It has completeness $1 - \epsilon$ if the following holds with probability $1 - \epsilon$.

even after interacting with the delegator polynomially many times, and each time learning whether the delegator accepted or rejected the proof.

One could define security w.r.t. cheating workers of size $T(k)$ for any (possibly super-polynomial) function T. However, for the sake of simplicity of notation, we define soundness w.r.t. poly-size cheating workers.

Definition 32 (Reusable Security Game) *Let* $\mathsf{mDel}_{\mathcal{F},\mathcal{G}} = \langle \mathsf{D}, \mathsf{W} \rangle$ *be a delegation scheme. For a security parameter* $k \in \mathbb{N}$ *and for a* PPT *(cheating) worker* W^**, the* security game $\mathsf{G}^{\mathsf{W}^*}(k)$ *is defined as follows.*

The game starts with the offline stage of $\mathsf{mDel}_{\mathcal{F},\mathcal{G}}$*, and is followed by polynomially many rounds of the online stage.*

1. **(Initial Phase)** $\mathsf{W}^*(1^k)$ *first chooses a parameter* $n = \mathrm{poly}(k)$ *and an input* $x \in \{0,1\}^n$*. Then,* D *and* $\mathsf{W}^*(1^k)$ *run the offline stage on inputs* $(1^k, x)$*.*

2. **(Learning Phase)** *At the beginning of each round of the online stage,* W^* *can do one of the following:*

   (a) *Terminate this phase.*

   (b) *Choose a function* $f \in \mathcal{F}$ *and interact with* D *in the online stage with the operation* $\mathsf{Compute}(f)$*.*

   (c) *Choose a function* $g \in \mathcal{G}$ *and interact with* D *in the online stage with the operation* $\mathsf{Update}(g)$*.*

   After each round, if W^* *did not terminate the phase,* D *sends her decision bit to* W^**.*

3. **(Challenge Phase)** *If the learning phase is terminated,* W^* *chooses a function* $f' \in \mathcal{F}$*, and* D *and* W^* *execute* $\mathsf{Compute}(f')$*.*

W^* succeeds *in the game* $\mathsf{G}^{\mathsf{W}^*}(k)$ *if* D *accepts a wrong value* $y' \neq f'(x)$*, where* x *is the latest updated memory.*

Definition 33 (Reusable Soundness) *A delegation scheme* $\mathsf{mDel}_{\mathcal{F},\mathcal{G}} = \langle \mathsf{D}, \mathsf{W} \rangle$ *has* **reusable soundness error** ε *if for every* $k \in \mathbb{N}$ *and every* PPT *worker strategy* W^**,*

$$\Pr[\mathsf{W}^* \text{ succeeds in } \mathsf{G}^{\mathsf{W}^*}(k)] \leq \varepsilon(k),$$

where $\mathsf{G}^{\mathsf{W}^*}(k)$ *is the security game corresponding to* $\mathsf{mDel}_{\mathcal{F},\mathcal{G}}$*, as defined above. We say that* $\mathsf{mDel}_{\mathcal{F},\mathcal{G}}$ *is* **sound** *if it has a negligible soundness error.*

Remark. We stress that in the soundness definition, we allow the adversary W^* to learn the decision bit of the delegator D after each execution of the delegation protocol. This is in contrast to the two delegation schemes of [GGP10, CKV10], which are only sound if the adversary W^* does not learn the decision bit of the delegator D. We elaborate on this point when we discuss the streaming setting, in Section 6.1.

In what follows we define the notion of *one-time soundness.* The reason we need this definition is that the soundness proof of our memory delegation scheme (in Section 5.2), consists of two parts. We first prove that our scheme has one-time soundness, i.e., it is sound assuming the delegation protocol is executed only once. Then, we argue that the fact that it is one-time sound, implies that it is also sound from multiple interactions (i.e., has reusable soundness.)

Definition 34 (One-time Security Game and One-time Soundness) *Let* $\mathsf{mDel}_{\mathcal{F},\mathcal{G}} = \langle \mathsf{D}, \mathsf{W} \rangle$ *be a delegation scheme. For a security parameter* $k \in \mathbb{N}$ *and for a (cheating) worker* W^**, the* one-time security game $\mathsf{G}_1^{\mathsf{W}^*}(k)$ *is defined similarly to the security game in Definition 41, except that they do not execute the learning phase, and just proceed to the challenge phase directly from the initial phase.*

We say $\mathsf{mDel}_{\mathcal{F},\mathcal{G}} = \langle \mathsf{D}, \mathsf{W} \rangle$ *has* **one-time soundness error** ε *if for every* $k \in \mathbb{N}$ *and every* PPT *worker strategy* W^**,*

$$\Pr[\mathsf{W}^* \text{ succeeds in } \mathsf{G}_1^{\mathsf{W}^*}(k)] \leq \varepsilon(k).$$

We say that $\mathsf{mDel}_{\mathcal{F},\mathcal{G}}$ *is* **one-time sound** *if it has a negligible one-time soundness error.*

Theorem 35 (Memory Delegation) *Assume the existence of a poly-log PIR scheme (as defined in Definition 5) and a collision resistant hash function family. Then there exists a* 2*-round memory delegation scheme* mDel*, for delegating any function computable by an* $\mathcal{L}$*-uniform poly-size circuit. The delegation scheme,* mDel *has the following properties, for security parameter* k*.*

- *The scheme has perfect completeness and negligible (reusable) soundness error.*
- *The delegator and worker are efficient in the offline stage; i.e., both the delegator and the worker run in time* $\mathrm{poly}(k, n)$*.*

- *The worker is efficient in the online phase. More specifically, it runs in time* $\mathrm{poly}(k, S)$ *during each* $\mathsf{Compute}(f)$ *and* $\mathsf{Update}(f)$ *operation, where* S *is the size of the* $\mathcal{L}$*-uniform circuit computing* f*. The delegator runs in time* $\mathrm{poly}(k, d)$ *during each* $\mathsf{Compute}(f)$ *and* $\mathsf{Update}(f)$ *operation, where* d *is the depth of the* $\mathcal{L}$*-uniform circuit computing* f*.*[2]

5.2 Memory Delegation Scheme

In this section, we prove Theorem 35, by constructing a non-interactive memory delegation scheme with the desired properties.

5.2.1 Overview of our Memory Delegation Scheme

The initial idea behind our memory delegation scheme, is the observation of Goldwasswer *et. al.* [GKR08], that their delegation protocol can be verified *very* efficiently (in time sub-linear in the input size), if the delegator has oracle access to the low-degree extension of the input x (we refer the reader to Section 3.3 for the definition of a low-degree extension). Moreover, as observed by [GKR08], the delegator needs to access this low-degree extension LDE_x at a single point z, which depends only on the random coin tosses of the delegator.

This observation immediately gives rise to a memory delegation scheme with *one-time* soundness: The delegator's secret state will be $(z, \mathrm{LDE}_x(z))$. Then, she will use this secret state in order to verify computation using the GKR protocol. As was argued by Goldwasswer *et. al.*, this indeed works if the delegator runs the delegation protocol *once*. However, the soundness crucially relies on the fact that the delegator's secret state is indeed secret, and if the delegator uses this state more than once, then soundness breaks completely.

One idea, following the idea of Gennaro *et. al.* [GGP10], is to use a fully homomorphic encryption (FHE) scheme to encrypt all the communication, in order to hide the secret state. This indeed works if the worker does not learn whether the delegator accepts or rejects his proofs. However, if the worker does learn the verdict of the delegator, then there are known attacks that break soundness.

In the streaming setting, we follow this approach, and we succeed in overcoming this problem, and construct a scheme that is sound even if the worker does learn the verdict of the delegator. We could follow this approach in the memory delegation setting as well. However, for several reasons,

[2]Thus, for every constant $c \in \mathbb{N}$, if we restrict the depth of f to be at most k^c, then the delegator is considered efficient.

we choose to take a different approach. First, the approach above relies on the existence of an FHE scheme, whereas our memory delegation scheme relies on the existence of a poly-logarithmic PIR scheme, arguably a more reasonable assumption. Second, the approach above results with the delegator having a secret state, whereas in our memory delegation scheme, the state of the delegator is public. Finally, the construction and proof of the memory delegation scheme is simpler.

In our approach, instead of having $(z, \mathrm{LDE}_x(z))$ as the delegator's secret state, the delegator keeps a tree-commitment of the entire LDE_x as her secret state (recall the definition of a tree-commitment in Section 3.5). Namely, she chooses a random hash function h from a collision-resistant hash family, and keeps $(h, T_h(\mathrm{LDE}_x))$ as her state. In addition to giving the worker her memory x, she also gives him the hash function h. Notice that her state is not secret, which makes the proof of security significantly simpler than that in the streaming setting (where the delegator's state is secret).

When the delegator wishes to delegate the computation of a function f, they will execute $\mathsf{Compute}(f)$, by simply running the (non-interactive) delegation protocol $\mathrm{GKR}(f)$. Recall that at the end of the GKR protocol the delegator needs to verify the value of $\mathrm{LDE}_x(r)$ for a random r. However, she doesn't have x, since it was delegated to the prover, and all she has is the state $(h, T_h(\mathrm{LDE}_x))$. So, rather than computing the value of $\mathrm{LDE}_x(r)$ on her own, she will ask the worker to reveal to this value, by sending the augmented path in the Merkle tree corresponding to the leaf r.[3]

When the delegator wishes to update her memory from x to $g(x)$, she will need to update her secret state from $(h, T_h(\mathrm{LDE}_x))$ to $(h, T_h(\mathrm{LDE}_{g(x)}))$. As before, she cannot perform this operation on her own, and instead she will delegate this computation to the worker, by requesting a $\mathsf{Compute}(g')$ operation, where $g'(x) = T_h(\mathrm{LDE}_{g(x)})$.

Unfortunately the high-level description given above is a gross oversimplification of our scheme, and there are several technical issues that complicate matters.

The first technicality (the easiest one to deal with), is that the GKR delegation scheme does not have a negligible soundness error. In our setting, it is very important to have negligible soundness, since if the soundness is non-negligible, then a cheating worker may cheat in the update procedure (which is also being delegated). The problem is that if a worker cheats even once in an update procedure, all soundness guarantees are mute from that point on. So, we really need the soundness

[3] As we shall see in a few paragraphs, this is an oversimplification, and due to technical reasons, the actual protocol is more complicated.

error to be negligible. In order to reduce the soundness error, we will run the GKR protocol in parallel u times (for any parameter u such that $1/2^u = \mathsf{ngl}(k)$). We denote the u-fold parallel repetition of GKR by $\mathrm{GKR}^{(u)}$. As a result the worker will need to reveal to u augmented paths of the Merkle tree.

The other technical point is more subtle. In the offline stage, when the delegator computes the tree commitment $T_h(\mathrm{LDE}_x)$, she needs to choose the parameters $\mathbb{H}, \mathbb{F}, m$ for the low-degree extension. The typical choice for these parameters is: $|\mathbb{H}| = \mathrm{polylog}(n)$, $|\mathbb{F}| = \mathrm{poly}(|\mathbb{H}|)$, and $m = O\left(\frac{\log n}{\log\log n}\right)$, where $n = |x|$. However, when delegating the computation of a function f, the worker and delegator run $\mathrm{GKR}^{(u)}(f)$ and need to verify $\mathrm{LDE}_x(r_i) = v_i$ for random points $r_1, \ldots, r_u$. However, here the parameters of the low-degree extension LDE_x depend on the depth d of the circuit computing f. Namely, looking at the parameters in Theorem 8, the parameters of the low-degree extension are

$$|\mathbb{H}'| = \theta(d \cdot \log n),\ m' = \theta\left(\frac{\log n}{\log d}\right),\ |\mathbb{F}'| = \mathrm{poly}(|\mathbb{H}'|).$$

Therefore, the worker cannot simply send the augmented path, since the tree commitment is w.r.t. parameters $\mathbb{H}, \mathbb{F}, m$ whereas the delegator needs to verify $\mathrm{LDE}_x(r_i) = v_i$ w.r.t. the parameters $\mathbb{H}', \mathbb{F}', m'$.

We get around this technical problem by delegating the functions $g_{r_i}(x) \triangleq \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i)$. Luckily, Corollary 7 implies that these functions can be computed by a poly-size circuit of depth $\log^c(n)$ for some constant c (assuming the delegated function f is of poly-size). Again, we delegate the computation of each of these g_{r_i} using $\mathrm{GKR}^{(u)}$ to ensure negligible soundness. Thus, finally the worker will need to reveal the augmented paths of u^2 points in LDE_x (u points for each g_{r_i}).[4]

The final technical difficulty is that all these algorithms need to run in parallel, since we want our final memory delegation scheme to be non-interactive (i.e., to consist of only two messages). Typically, there is no problem in running several two-message protocols in parallel. However, in our case, the delegator uses a common *secret* input in these protocols. Namely, the delegator uses secret randomness $r_1, \ldots, r_u \in (\mathbb{F}')^{m'}$ in the parallel repetition of the delegation protocol $\mathrm{GKR}(f)$ which ends with her needing to verify that $\mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i) = v_i$ for every $i \in [u]$. In addition she uses these same r_i's in the delegation protocols $\mathrm{GKR}(g_{r_i})$. Moreover, at the end of each of the $\mathrm{GKR}(g_{r_i})$ protocols, the delegator needs to verify that $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j}) = w_{i,j}$ for

[4]We note that there are several ways to improve efficiency, such as thinking of $(g_{r_1}, \ldots, g_{r_u})$ as one function. However, for the sake of simplicity of exposition, we focus on the simplest (rather than most efficient) solution.

random points $z_{i,1}, \ldots, z_{i,u} \in \mathbb{F}^m$. Finally, they also run a reveal protocol for each $z_{i,j}$, denoted by Reveal$(z_{i,j})$, where the worker simply reveals to the augmented path of the leaf $z_{i,j}$ in the Merkle tree of $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}$.

We note that the protocol GKR(f) (resp. GKR(g)) is not sound if the r_i's (resp. $z_{i,j}$'s) are a priori known to the worker. To ensure that soundness still holds even if we run all these algorithms in parallel, we mask parts of the delegator's message using a PIR scheme, and then we use Lemma 28 to claim that the soundness error remains negligible.

5.2.2 Formal Description of our Memory Delegation Scheme

Our construction uses the following building blocks

1. A collision resistant hash family $\mathcal{H} = \{\mathcal{H}_k\}_{k\in\mathbb{N}}$, where every $h \in \mathcal{H}_k$ satisfies

$$h : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k.$$

2. The delegation scheme GKR $= \langle \mathsf{D}', \mathsf{W}' \rangle$ from [GKR08, KR09] (see Theorem 8 for the properties of this delegation scheme). The main property we use here, is that the delegator can verify proofs by accessing its input x at a single random point in LDE_x.

- **Parameters.** Let $k \in \mathbb{N}$ be the security parameter, and let $n \in \mathbb{N}$ be the length of the (initial) memory being delegated. Let $\mathbb{H}$ be an extension field of $\mathbb{GF}[2]$, and let $m \in \mathbb{Z}$ such that $|\mathbb{H}| = \mathrm{polylog}(n)$, $m = \theta\left(\frac{\log n}{\log\log n}\right)$, and let $\mathbb{F}$ be an extension field of $\mathbb{H}$ of size $|\mathbb{F}| = \mathrm{poly}(|\mathbb{H}|)$.
- **Offline Phase.** In the offline phase, both the delegator D and the worker W take as input the security parameter 1^k and a string $x \in \{0,1\}^n$. The worker W simply saves x. The delegator D does the following.
    1. Compute the low-degree extension of x w.r.t. $\mathbb{H}, \mathbb{F}, m$, denoted by $\mathrm{LDE}_x : \mathbb{F}^m \to \mathbb{F}$ (see Section 3.3 for the definition of a low-degree extension). She interprets LDE_x as a vector in $\mathbb{F}^{|\mathbb{F}|^m}$.
    2. Choose a random collision resistant hash-function $h \leftarrow \mathcal{H}_k$, and sends h to W.
    3. Compute the root of the Merkle tree of LDE_x with respect to the hash function h. Namely, compute $\sigma = T_h(\mathrm{LDE}_x)$, which is the root of the Merkle tree corresponding to the hash

function h (we refer the reader to Section 3.5 for the definition of a Merkle tree).

The delegator D saves (h, σ) as a short certificate for x.

- **Online-Phase.**
  - $\mathsf{Compute}(f)$**.** When the delegator D sends the worker W a computation request $\mathsf{Compute}(f)$, they run the following three protocols *in parallel*.
    1. Run the underlying delegation protocol $\mathrm{GKR} = \langle \mathsf{D}', \mathsf{W}' \rangle$ for delegating the computation of $f(x)$. However, since the soundness of the GKR protocol is only $1/2$, we amplify this soundness by repeating the GKR protocol u times in parallel. Namely, W and D run $\mathrm{GKR}^{(u)}$ which is a u-fold parallel repetition of of the GKR protocol, and thus has soundness $1/2^u + \mathsf{ngl}(k) = \mathsf{ngl}(k)$ (assuming we take u such that $1/2^u = \mathsf{ngl}(k)$).

       If D' rejects, then D rejects. Otherwise, at the end of this protocol the delegator D needs to verify that $\mathrm{LDE}_x(r_i) = v_i$ for some random points $r_1, \ldots, r_u$. Recall that these points depend only on the delegators random coin tosses, and can be efficiently computed by the delegator D before the protocol execution begins (see Theorem 8).

       Recall that the low-degree extension LDE_x, is not w.r.t. the parameters $\mathbb{H}, \mathbb{F}, m$, but rather w.r.t. parameters $\mathbb{H}', \mathbb{F}', m'$ that depend on the depth d of the $\mathcal{L}$-uniform circuit computing f (see Theorem 8 and the discussion in Section 5.2.1). Thus, the delegator cannot verify that

       $$\mathrm{LDE}_x^{\mathbb{F}', \mathbb{H}', m'}(r_i) = v_i$$

       by simply asking the worker to "decommit" to the leaves $r_1, \ldots, r_u$ of the Merkle tree, by sending their augmented paths (since the tree commitment was on the low-degree extension of x w.r.t. $\mathbb{H}, \mathbb{F}, m$). Instead they will run the following additional protocol (in parallel).
    2. The idea is to run for every $i \in [u]$, the delegation protocol $\mathrm{GKR}^{(u)} = \langle \mathsf{D}', \mathsf{W}' \rangle$ for

delegating the functions g_{r_i}, where

$$g_{r_i}(x) = \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i).$$

(As in Step 1, we use parallel repetition in order to reduce the soundness error to $\mathsf{ngl}(k)$). However, note that since these delegation protocols are running in parallel with the delegation protocol $\mathrm{GKR}^{(u)}$ of Step 1, the r_i's (which are part of the description of g_{r_i}) must be kept secret, to ensure the soundness of the $\mathrm{GKR}^{(u)}$ protocol of Step 1.

Thus, to ensure the secrecy of the r_i's, instead of running the $\mathrm{GKR}^{(u)}$ protocols of Step 2 "in the clear", we mask them using a PIR scheme. In what follows, we explain what a single masked $\mathrm{GKR}^{(u)}$ protocol for computing g_{r_i} looks like, and this protocol will be repeated in parallel u times (once for each $i \in [u]$).

Recall that in the GKR protocol (and thus in the $\mathrm{GKR}^{(u)}$ protocol), the message sent by the delegator D' depends only on the parameters (and her random coin tosses), and is independent of the actual function being delegated (see Theorem 8). Thus, this message can be sent in the clear, as it reveals no information about the function g_{r_i} being delegated (except for its size and depth), and thus reveals no information about the secret value r_i. On the other hand, the message sent by the worker W' obviously does depend on g_{r_i}, and thus on r_i. Since the r_i's should be kept secret, this message will be sent using a PIR scheme.

Namely, the worker W prepares a database DB' with $N' \triangleq |\mathbb{F}'|^{m'}$ entries, where the entry $r \in (\mathbb{F}')^{m'}$ contains the message he would have sent (in the parallel version $\mathrm{GKR}^{(u)}$) if the delegated function was $g_r(x) = \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r)$. The delegator D, in addition to sending a $\mathrm{GKR}^{(u)}$ message, also sends a query $q_i' \leftarrow Q^{\mathrm{PIR}}(k, r_i, N')$. Then, the worker W answers the PIR query q_i' using the database DB'; i.e., he sends $a_i' \leftarrow D^{\mathrm{PIR}}(k, \mathrm{DB}', q_i')$. Finally, the delegator D retrieves the "worker's message" using the Retrieve algorithm R^{PIR}, and accepts this message if and only if $(\mathsf{D}')^{(u)}$ would have accepted it, and if it is consistent with Step 1; i.e., if the worker proved that indeed $g_{r_i}(x) = v_i$, for the same value v_i obtained in Step 1.

As before, for every $i \in [u]$, to verify the i'th delegation protocol $\mathrm{GKR}^{(u)}$ for computing $g_{r_i}(x) = \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i)$, the delegator D' needs to verify that $\mathrm{LDE}_x(z_{i,j}) = w_{i,j}$, for random values $z_{i,1}, \ldots, z_{i,u}$ that depend only on the parameters and on the delegator's random coin tosses. However, here LDE_x is w.r.t. the parameters $\mathbb{H}, \mathbb{F}, m$, since the function g_{r_i} is computable by $\mathcal{L}$-uniform circuit of depth polylog(n) (follows from Corollary 7). In order to verify this, they run the following Reveal protocol u times per each $i \in [u]$, and thus altogether they run the Reveal protocol u^2 times.

3. The delegator and worker run a Reveal protocol u^2 times, where the delegator sends $z_{i,j} \in \mathbb{F}^m$ and the worker reveals the augmented path of the Merkle tree corresponding to the leaf $z_{i,j}$, denoted by $\mathrm{aug}(z_{i,j})$. However, since $z_{i,j}$ needs to remain secret for the $\mathrm{GKR}^{(u)}$ protocols in Step 2 to remain sound, this will be done using a PIR scheme. Namely, the worker W does the following. He prepares a database DB of size $N \triangleq |\mathbb{F}|^m$, where for any $z \in \mathbb{F}^m$ the z'th entry contains $\mathrm{aug}(z)$ (i..e, the augmented path corresponding to the leaf z in the Merkle tree of LDE_x). The delegator D sends a query $q_{i,j} \leftarrow Q^{\mathrm{PIR}}(k, z_{i,j}, N)$, and the worker W answers according to his database $a_{i,j} \leftarrow D^{\mathrm{PIR}}(k, \mathrm{DB}, q_{i,j})$. Finally, the delegator D retrieves the answer using the retrieving algorithm R^{PIR}, and accepts this answer if and only if the retrieved string is a valid augmented path of the Merkle tree corresponding to the leaf $z_{i,j}$, and if the leaf value is $w_{i,j}$.

We denote the delegation protocol of Step 1 by

$$\mathrm{GKR}^{(u)} = \langle \mathsf{W}'^{(u)}, \mathsf{D}'^{(u)}(r_1, \ldots, r_u)\rangle(f).$$

We denote the masked delegation protocols of Step 2 by

$$\mathrm{PIR}\left(\langle \mathsf{W}'^{(u)}, \mathsf{D}'^{(u)}(z_{i,1}, \ldots, z_{i,u})\rangle(g_{r_i})\right).$$

We denote the reveal protocols of Step 3 by $\mathrm{Reveal}(z_{i,j})$. Note that the memory x is implicit in all these notations. We summarize the $\mathsf{Compute}(f)$ protocol in Figure 5.1.

– $\mathsf{Update}(g)$**.** When the delegator D sends the worker W an update request $\mathsf{Update}(g)$ for some $g \in \mathcal{G}$, indicating that she wishes to update her memory x to $g(x)$, the worker and

$\mathsf{Compute}(f)$:

The delegator D stores a state (h, σ) where $\sigma = T_h(\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m})$ and wants to learn the value of $f(x)$ from the worker W, who stores $x \in \{0,1\}^n$.

1. D and W run $\mathrm{GKR}^{(u)} = \langle \mathsf{W}'^{(u)}, \mathsf{D}'^{(u)}(r_1, \ldots, r_u)\rangle(f)$.
   (a) If $\mathsf{D}'^{(u)}$ rejects, then the delegator D outputs "reject".
   (b) At the end of this protocol, the delegator D' needs to verify that $\mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i) = v_i$ for some values v_i.
2. For every $i \in [u]$, run $\mathrm{PIR}(\langle \mathsf{W}'^{(u)}, \mathsf{D}'^{(u)}(z_{i,1}, \ldots, z_{i,u})\rangle(g_{r_i}))$.
   (a) The delegator D makes sure that in these protocols the worker still claims that indeed $g_{r_i}(x) = v_i$, where $g_{r_i}(x) = \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i)$. If this is not the case, then the delegator D outputs "reject".
   (b) If at any point D' rejects, then the delegator D outputs "reject".
   (c) In order to verify these protocols, the delegator D needs to verify that $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j}) = w_{i,j}$ for some values $w_{i,j}$.
3. For every $i, j \in [u]$, run $\mathrm{Reveal}(z_{i,j})$.
   If for some $i, j \in [u]$, the worker fails in revealing to $w_{i,j}$, then the delegator D outputs "reject".
4. The delegator D outputs "accept", assuming he didn't output "reject" at any point.

Figure 5.1: Compute(f)

delegator do the following.

1. The delegator D uses the help of the worker W in order to update her short certificate. Specifically, D chooses a fresh hash function $h' \leftarrow \mathcal{H}_k$, and sends h' to W. Then, she delegates to the worker W the computation $\mathsf{Compute}(g')$, where

$$g'(x) = T_{h'}(\mathrm{LDE}_{g(x)}),$$

   where $\mathrm{LDE}_{g(x)}$ is w.r.t. $\mathbb{H}', \mathbb{F}', m'$, where letting $n' \triangleq |g(x)|$, the field $\mathbb{H}'$ is an extension field of $\mathbb{GF}[2]$ of size $|\mathbb{H}'| = \mathrm{polylog}(n')$, $m' = \theta\left(\frac{\log n'}{\log\log n'}\right)$, and $\mathbb{F}'$ is an extension field of $\mathbb{H}'$ of size $\mathrm{poly}(|\mathbb{H}'|)$. If she rejects this proof, then no update is performed. Otherwise, she updates her short certificate from σ to $\sigma' \triangleq T_{h'}(\mathrm{LDE}_{g(x)})$.
2. The worker W updates his state from x to $g(x)$.

Note that if the function g is computed by an $\mathcal{L}$-uniform circuit of depth d, then the function g' is computable by an $\mathcal{L}$-uniform circuit of depth $d + \mathrm{poly}(k) + \mathrm{polylog}(n') \leq \mathrm{poly}(d, k)$.

5.2.3 Proof of Theorem 35.

In this section, we prove that the construction above satisfies the properties of Theorem 35. The perfect completeness follows immediately from the completeness of the underlying delegation scheme GKR, the completeness of the PIR scheme, and the completeness of the Reveal protocol. The efficiency guarantees follow immediately from the efficiency guarantees of GKR and the efficiency guarantees of the underlying PIR scheme.

The main difficulty is in proving soundness. We shall prove the one-time soundness of our memory delegation scheme in Lemma 36, and establish the reusable soundness in Lemma 37. At a very high level,

- the one-time soundness of our scheme follows from the soundness of the GKR protocol, the security of tree commitments, and the parallel composition lemma (Lemma 28).
- the reusable soundness of our scheme follows from the one-time soundness, since the short certificate of the delegator is not secret, and the worker can compute this short certificate on his own.

Lemma 36 *The memory delegation scheme constructed in Section 5.2.2 is one-time sound, i.e., it has negligible one-time soundness error.*

Proof. Suppose for the sake of contradiction that there exists a PPT worker W^* and a polynomial q such that for infinitely many k's

$$\Pr[\mathsf{W}^* \text{ succeeds in } \mathsf{G}_1^{\mathsf{W}^*}(k)] \geq \frac{1}{q(k)}, \tag{5.1}$$

where G_1 is the one-time soundness game. Recall that in the game G_1, the worker $\mathsf{W}^*(1^k)$ first chooses a parameter $n = \text{poly}(k)$ and a string $x \in \{0,1\}^n$. Then, D and W^* run the offline phase, where D chooses a random hash function $h \leftarrow \mathcal{H}_k$, computes $\sigma \triangleq T_h(x)$, and sends h to W^*. Then, W^* chooses a function $f \in \mathcal{F}$, and D and W^* execute $\mathsf{Compute}(f)$.

Suppose that W^* succeeds in proving a false statement $f(x) = y'$. Namely, W^* and D run the protocol $\mathsf{Compute}(f)$ and at the end D accepts a wrong statement $f(x) = y'$.

Recall that the $\mathsf{Compute}(f)$ protocol consists of $1 + u + u^2$ sub-protocols: an execution of

$$\Pi_0 = \text{GKR}^{(u)} = \langle \mathsf{W}'^{(u)}, \mathsf{D}'^{(u)}(r_1, \ldots, r_u) \rangle (f),$$

for every $i \in [u]$, an execution of

$$\Pi_i = \mathrm{PIR}\left(\langle \mathsf{W}'^{(u)}, \mathsf{D}'^{(u)}(z_{i,1}, \ldots, z_{i,u})\rangle(g_{r_i})\right),$$

and for every $i, j \in [u]$, an execution of $\Pi_{i,j} = \mathrm{Reveal}(z_{i,j})$.

Suppose that in the $\mathsf{Compute}(f)$ protocol, Π_0 reduces verifying that $f(x) = y'$ to verifying the u statements

$$\mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i) = v_i,$$

and each Π_i reduces verifying that $\mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i) = v_i$ to verifying the u statements

$$\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j}) = w_{i,j}.$$

We note that for W^* to succeed in G_1, W^* must successfully "cheat" in at least one of Π_0, Π_i, or $\Pi_{i,j}$. Namely, one of the following cases holds.

1. **Case 1.** $y' \neq f(x)$, and for every $i \in [u]$, $v_i = \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i)$.
2. **Case 2.** There exists $i \in [u]$ such that $v_i \neq \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i)$, and for every $j \in [u]$, $w_{i,j} = \mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j})$.
3. **Case 3.** There exists $i, j \in [u]$ such that $w_{i,j} \neq \mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j})$.

Intuitively, each of the above cases should hold with only negligible probability. This is due to the soundness property of the GKR protocol and the security of the tree commitments.

We next use Lemma 28 to claim that each of the above cases holds with negligible probability, even when Π_0, Π_i, or $\Pi_{i,j}$ are executed in parallel, contradicting Equation (5.1).

In order to make use of Lemma 28, we consider a slightly modified version of the protocols Π_0, Π_i, and $\Pi_{i,j}$, denoted by Π'_0, Π'_i, and $\Pi'_{i,j}$. The messages sent in these protocols are identical to the ones sent in the original $\Pi_0, \Pi_i, \Pi_{i,j}$ protocols, and the only difference is in the verification procedure. Note that the protocols $\Pi_0, \Pi_i, \Pi_{i,j}$ are interleaved in the sense that the verifier of Π_0 doesn't actually verify the correctness of Π_0, but rather uses the protocols Π_i to verify the correctness of Π_0. Similarly, the verifier uses the protocols $\Pi_{i,j}$ to verify the correctness of Π_i. Instead, we define $\Pi'_0, \Pi'_i, \Pi'_{i,j}$ to be stand-alone protocols (with their own verification procedures) for recognizing the *empty* language.

Recall that $\Pi_0 = \mathrm{GKR}^{(u)} = \langle \mathsf{W}'^{(u)}, \mathsf{D}'^{(u)}(r_1, \ldots, r_u)\rangle(f)$ is a u-fold parallel repetition of GKR, where the delegator delegates the computation of $f(x)$ to the worker. In Π'_0, the verifier actually computes on her own (using x) the correct values of $y = f(x)$ and $v_i = \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i)$ for every $i \in [u]$[5], and the verifier accepts if and only if at the end of $\mathrm{GKR}^{(u)}$, the worker convinces the delegator to accept some incorrect $y' \neq f(x)$. Note that the the soundness of $\mathrm{GKR}^{(u)}$ implies that Π'_0 has negligible soundness error.

Recall that each $\Pi_i = \mathrm{PIR}(\langle \mathsf{W}'^{(u)}, \mathsf{D}'^{(u)}(z_{i,1}, \ldots, z_{i,u})\rangle(g_{r_i}))$ is a masked u-fold parallel repetition of GKR, where the delegator delegates the computation of $g_{r_i}(x) = \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i)$ to the worker, and r_i is a random point masked by the PIR scheme. In each Π'_i, the verifier actually computes on her own (using x) the correct value $v_i = \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i)$ and the values $w_{i,j} = \mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j})$ for every $j \in [u]$, and the verifier accepts if and only if the worker convinces the delegator to accept some incorrect $v'_i \neq \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i)$. Note that the soundness of $\mathrm{GKR}^{(u)}$ implies that Π'_i has negligible soundness error.[6]

Finally, recall that each $\Pi_{i,j} = \mathrm{Reveal}(z_{i,j})$ is a masked tree commitment reveal protocol, where the delegator asks the worker to reveal the value of $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j})$ by giving a valid augmented path $\mathrm{aug}(z_{i,j})$ of $T_h(\mathrm{LDE}_x)$, and $z_{i,j}$ is a random point masked by the PIR scheme. In each $\Pi'_{i,j}$, the verifier actually computes on her own (using x) the correct values of $w_{i,j} = \mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j})$, and the verifier accepts iff the worker convinces the delegator to accept some incorrect $w_{i,j} \neq \mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j})$. Note that the security of tree commitments implies that $\Pi'_{i,j}$ has negligible soundness error.

In all the protocols Π'_0, Π'_i, and $\Pi'_{i,j}$, we think of x as the common input, and we think of

$$p = \big((r_i)_{i\in[u]}, (z_{i,j})_{i,j\in[u]}\big)$$

as the common private randomness.

Let Π' be the corresponding parallel execution of Π'_0, Π'_i, and $\Pi'_{i,j}$, where the verifier of Π' accepts if and only if any one of Π'_0, Π'_i, and $\Pi'_{i,j}$ accepts. Lemma 28 implies that the fact that each of the protocols Π'_0, Π'_i, and $\Pi'_{i,j}$ has negligible soundness error, implies that Π' has negligible soundness as well. Recall that whenever W^* succeed in G_1, at least one of the above three cases holds. Namely, the verifier in Π'_0 accepts when Case 1 holds, the verifier in Π'_i accepts when Case 2 holds, and and

[5]Note that the verifier in these protocols runs in time poly($|x|$), which is too long in our setting, but this is only in the analysis.

[6]Note that we didn't use the soundness of the PIR scheme. Indeed, the PIR scheme is added only in order to later apply Lemma 28.

the verifier in $\Pi'_{i,j}$ accepts when Case 3 holds. Thus,

$$\Pr[\mathsf{W}^* \text{ succeeds in } \mathsf{G}_1^{\mathsf{W}^*}(k)] \leq \Pr[\mathsf{D} \text{ accepts in } \Pi'] \leq \mathsf{ngl}(k),$$

contradicting Equation (5.1).

■

Lemma 37 *The memory delegation scheme constructed in Section 5.2.2 is sound, i.e., it has negligible reusable soundness error.*

Proof. Suppose for the sake of contradiction that there exists a PPT worker W^* such that

$$\Pr[\mathsf{W}^* \text{ succeeds in } \mathsf{G}^{\mathsf{W}^*}(k)] \geq \alpha(k), \tag{5.2}$$

for a non-negligible function α, where G is the reusable soundness game. We construct a PPT worker W_1^* which succeeds in the one-time security game $\mathsf{G}_1^{\mathsf{W}_1^*}(k)$ with non-negligible probability.

Recall that the game G proceeds in three phases, as follows.

- In the initial phase, the worker $\mathsf{W}^*(1^k)$ first chooses a parameter $n = \text{poly}(k)$ and a string $x \in \{0,1\}^n$. Then, D and W^* run the offline phase, where D chooses a random hash function $h \leftarrow \mathcal{H}_k$, computes $\sigma \triangleq T_h(x)$, and sends h to W^*.

- In the learning phase, W^* and D execute polynomially many $\mathsf{Compute}(f)$ and $\mathsf{Update}(g)$ operations, where each $f \in \mathcal{F}$ and $g \in \mathcal{G}$ are chosen by W^*. In the $\mathsf{Update}(g)$, D chooses a fresh hash function $h' \leftarrow \mathcal{H}_k$, sends h' to W^*, and then D and W^* execute $\mathsf{Compute}(g')$ where $g'(x) = T_{h'}(\text{LDE}_{g(x)})$. If D accepts, then the memory is updated to $x' = g(x)$, and if D rejects, then the memory x remains unchanged.

- In the challenge phase, W^* chooses $f' \in \mathcal{F}$, and D and W^* execute $\mathsf{Compute}(f')$. W^* succeeds if D accepts a wrong value $y' \neq f'(x)$, where x is the latest updated memory.

Intuitively, noting that the short certificate of the delegator is public, one-time soundness seems to immediately imply reusable soundness, since in this case a (one-time) cheating worker W_1^* can simulate the learning phase of $\langle \mathsf{D}, \mathsf{W}^* \rangle$ on his own. However, this intuition is an oversimplification, since W^* may cheat in one of the Update computations, and thus convince the delegator D to update

her certificate to some incorrect value, which allows W^* to cheat easily in the challenge phase. In this case, simulating the learning phase does not help W_1^* cheat in the one-time security game G_1.

Therefore, instead of simply simulating the entire learning phase, roughly speaking, W_1^* does the following.

1. Guess the *first time* that W^* cheats successfully in either an update or a compute operation (we refer to this as the first cheating operation).

2. Guess the *last* valid update operation (that D accepts) before the cheating operation (we refer to this as the last update operation).

3. Embed the one-time game G_1 into the reusable game G, as follows: Simulate $\langle \mathsf{D}, \mathsf{W}^* \rangle$ in G up to the last update operation, and use the memory at that time to interact with D_1 in the initial phase of G_1. Then, continue to simulate $\langle \mathsf{D}, \mathsf{W}^* \rangle$ in G up to the first cheating operation, and interact with D_1 in the challenge phase of G_1 by using W^* in this first cheating operation in G.

In order to describe W_1^* more formally, we use the following notation. Let $L(k) \leq \text{poly}(k)$ be an upper bound on the total number of $\mathsf{Compute}$ and Update operations that W^* makes in the (reusable) game G. We call each such operation in G a round. We refer to the initial phase as round 0, the learning phase starts at round 1, and the challenge phase is the last round. Denote by x_i the memory content at the end of round i, and let f_i be the delegation function of the $\mathsf{Compute}(\cdot)$ operation in round i. Recall that $\mathsf{Update}(g)$ is implemented via a compute operation, so f_i is defined for every round.

Using this notation, we define W_1^* more formally, as follows:

1. Choose at random $j \leftarrow [L]$ and choose at random $i \leftarrow \{0, 1, \ldots, j-1\}$, where j is a guess for the first cheating operation of W^*, and i is a guess for the last successful Update operation of W^* before round j. ($i = 0$ corresponds to the guess that W^* doesn't update successfully the initial input x_0 before round j.)

2. Simulate the interaction of $\langle \mathsf{D}, \mathsf{W}^* \rangle$ in the reusable game G up until the end of round i. Denote by x_i the memory at the end of round i in the simulated game G.

3. Start the initial phase of G_1, with the memory x_i.

4. Upon receiving a hash function $h \leftarrow \mathcal{H}_k$ from D_1 in the offline phase, view this h as chosen by D in round i of the simulated (reusable game) G, and continue the simulation of G until the beginning of round j.

5. If at round j, W^* chooses to perform $\mathsf{Compute}(f_j)$, then start the challenge phase of G_1 with the function f_j, and interact with D_1 by simulating W^* (who supposedly executes $\mathsf{Compute}(f_j)$ with D).

We next analyze the success probability of W_1^*. To this end, for any $j \in [L]$ and any $i \in \{0, 1, \ldots, j-1\}$, let $W_{i,j}$ be the event that (1) the *first time* that W^* cheats successfully is in round j, and (2) the *last* valid update operation (that D accepts) before round j is in round i.

Note that by definition, whenever W^* succeeds in G, there must exist some such i, j such that event $W_{i,j}$ holds. Thus, a simple counting argument, together with Equation (5.2), implies that there exists some such i, j such that

$$\Pr[W_{i,j}] \geq \frac{\alpha}{L^2},$$

which implies that

$$\Pr[\mathsf{W}_1^* \text{ succeeds in } \mathsf{G}_1^{\mathsf{W}_1^*}(k)] \geq \frac{\alpha}{L^2},$$

contradicting Lemma 36, which asserts that the scheme is one-time secure.

■

Chapter 6

Streaming Delegation

6.1 Streaming Delegation Model

In this section, we formally define our streaming delegation model. We present our streaming delegation scheme in Section 6.2.

Definition 38 (Streaming Delegation Scheme) *Let $\mathcal{F}$ be a class of boolean functions. A* streaming delegation scheme $\mathsf{sDel}_{\mathcal{F}}$*, for a function class $\mathcal{F}$, consists of a streaming generator* S *and an interactive protocol* $\langle \mathsf{D}, \mathsf{W} \rangle$ *between a delegator* D *and a worker* W *with the following structure.*

1. *The scheme* sDel *starts at time* 0*, at which all parties receive a security parameter* 1^k *and a parameter N (in binary) which specifies the maximum length of the data stream. On input* $(1^k, N)$*,* D *generates some (possibly secret) state* σ_0*.*

2. *At each time* $t \in [N]$*,* S *generates a data item* $x_t \in \{0,1\}$*, which is received by both* D *and* W*. Upon receiving* x_t*,* D *updates her secret state from* σ_{t-1} *to* σ_t*, and* W *simply stores* x_t*. Let* $x^t = x^{t-1} \circ x_t$ *denote the current received data items, where* $\circ$ *represents a concatenation.*

3. *At any time* $t \in [N]$*,* D *may choose a delegation function* $f : \{0,1\}^t \to \{0,1\} \in \mathcal{F}$ *and run the delegation protocol* $\langle \mathsf{D}, \mathsf{W} \rangle$ *on input* (f, t)*. In addition to the common input* (f, t)*, the delegator D takes as input her secret state* σ_t*, and the worker* W *takes as input the data stream* x^t*.*[1]

[1] For simplicity of presentation, we omit the security parameter when it is clear from the context.

Remark 1. For the sake of readability, we try to keep the definition of the model as simple as possible. For example, we assume that data items are bits and delegation functions are boolean functions, while it is natural to consider non-boolean data items and non-boolean delegation functions. Also, we implicitly assume that D delegates at most one function of the current data stream at any given time t, while it is natural to allow D to delegate multiple functions at the same time. Nevertheless, it will be easy to see that our solution presented in Section 6.2 generalizes to these extensions readily.

Remark 2. Note that in Definition 38, we require that D updates her fingerprint of the data stream *on her own* efficiently (ideally, in time $\text{polylog}(N)$; see Definition 39 below). The reason is that in the streaming setting, the data stream arrives constantly at a high rate. Thus, if the update function would be delegated at time t, this delegation protocol may not end before time $t+1$. Hence, it may be infeasible and unreliable to ask the worker for his help in updating the delegator's fingerprint. We note that this is in contrast to the setting of memory delegation, where we do allow the update procedure to be delegated (see Section 5.1 for details).

Definition 39 (Efficiency) *A streaming delegation scheme* $\mathsf{sDel}_{\mathcal{F}}$ *has an* efficient *delegator if the runtime of* D *each time she updates her (secret) fingerprint is* $\text{polylog}(N)$*, and her runtime during each execution of the protocol* $\langle \mathsf{D}, \mathsf{W} \rangle(f,t)$ *is* $\text{poly}(k, \log N, \log S)$*, where* S *is the size of the circuit computing* f*. A streaming delegation scheme* $\mathsf{sDel}_{\mathcal{F}}$ *has an efficient worker if the runtime of* W *during an execution of* $\langle \mathsf{D}, \mathsf{W} \rangle(f,t)$ *is* $\text{poly}(k, \log N, S)$*, where* S *is the size of the circuit computing* f*.*

We proceed to define the completeness and soundness of a streaming delegation scheme.

Definition 40 (Completeness) *For any function class* $\mathcal{F}$*, a streaming delegation scheme* $\mathsf{sDel}_{\mathcal{F}}$ *has perfect completeness if for every parameters* $k, N \in \mathbb{N}$*,* $t \in [N]$*, every function* $f : \{0,1\}^t \to \{0,1\} \in \mathcal{F}$*, and every* $x^t \in \{0,1\}^t$ *generated by* S*, the following holds with probability* 1*:*[2] *When* D *and* W *run the delegation protocol* $\langle \mathsf{D}, \mathsf{W} \rangle(f,t)$*,* D *always accepts and outputs* $y = f(x^t)$*.*

The definition of the soundness property is more elaborate, since D and W may run the delegation protocol multiple times with different inputs. We provide the following game-based definition, where

[2]It has completeness $1-\epsilon$ if the following holds with probability $1-\epsilon$.

we allow the adversary W^* to choose the data stream and delegation functions, and W^* wins if he convinces D to accept an incorrect function value.

Definition 41 (Streaming Security Game) *Let k be a security parameter and N be a parameter. Let $\mathcal{F}$ be a function class and let $\mathsf{sDel}_{\mathcal{F}}$ be a delegation scheme for $\mathcal{F}$. The corresponding streaming security game $\mathsf{G}^{\mathsf{W}^*}(k, N)$ played by an (adversarial) worker W^* is defined as follows.*

1. *At time 0, the delegator D, on input $(1^k, N)$, generates her secret state σ_0.*
2. *At each time $t \in [N]$, W^* chooses a data item $x^t \in \{0,1\}$ and sends it to D, who then updates her secret state to σ_t. Furthermore, W^* may choose a function $f : \{0,1\}^t \to \{0,1\} \in \mathcal{F}$ and run with D the delegation protocol $\langle \mathsf{D}, \mathsf{W}^* \rangle$ on input (f, t).*

   At the end of each delegation protocol, W^* learns whether D accepts or rejects.
3. *W^* may terminate the game at any time $t \in [N]$.*

W^* succeeds *in the game $\mathsf{G}^{\mathsf{W}^*}(k, N)$ if there exists a time t such that W^* chooses to run the delegation protocol on input (f, t) for some function $f \in \mathcal{F}$, and convinces D to accept a wrong value $y \neq f(x^t)$.*

Definition 42 (Soundness) *Let k be a security parameter, and $\mathcal{F}$ a (boolean) function class that is poly-time computable. A delegation scheme $\mathsf{sDel}_{\mathcal{F}}$ has* **soundness error** *ε if for every worker strategy W^* with runtime $\mathrm{poly}(N)$, where $N = N(k)$,*

$$\Pr[\mathsf{W}^* \text{ succeeds in } \mathsf{G}^{\mathsf{W}^*}(k)] \leq \varepsilon(N),$$

where $\mathsf{G}^{\mathsf{W}^*}(k)$ is the security game corresponding to $\mathsf{sDel}_{\mathcal{F}}$, as defined above. We say that $\mathsf{sDel}_{\mathcal{F}}$ is **sound** *if it has a negligible soundness error in the parameter N.*

Remark. Note that in the above definition, we refer to k as the security parameter, but require the soundness to hold against any $\mathrm{poly}(N)$-time adversaries as opposed to standard $\mathrm{poly}(k)$-time adversaries. This is because the honest worker needs to run in time $\mathrm{poly}(N)$ even to evaluate the delegation function f. One should think of k as the security parameter of the cryptographic primitives used by D in the delegation scheme, where the primitives are required to be secure against $\mathrm{poly}(N)$-time adversaries.

Typically, one may assume that k and N are polynomially related. However, in the context of streaming algorithms, it is common to think of the data stream as having length super-polynomial in the computational resource of the streaming algorithms. For example, the space complexity of a streaming algorithm is typically limited to polylog(N), which usually implies the process time per data item is also polylog(N). We note that a stronger security assumption on the cryptographic primitives is necessary when the data stream is of length super-polynomial in the computational power of the delegator.

Remark. Note that, as in the memory delegation model, in the soundness definition, we allow the adversary W^* to learn the decision bit of the delegator D after each execution of the delegation protocol. This is in contrast to the two delegation schemes of [GGP10, CKV10], which are sound only if the adversary W^* does not learn the decision bit of the delegator D.

We stress that our streaming delegation scheme (in Section 6.2) has the property that the state of the delegator must be *secret*, in order to ensure soundness.[3] The only other delegation schemes that we are aware of which have this property, are [GGP10, CKV10]. However, these schemes are sound only if the adversary W^* does not learn the decision bit of the delegator D. The reason why in these schemes the decision bit needs to be kept secret is that the delegator uses her secret state to verify the worker's answer, and thus, her decision bit can potentially reveal one bit of information about her secret state. Indeed, in both schemes of [GGP10, CKV10], if D's decision bits are revealed, then there are known attacks to learn the secret state of D bit by bit and break the soundness of the schemes.

We didn't have to deal with this issue in our memory delegation scheme mDel, since the state of the delegator in mDel is not secret. In contrast, the delegator of our streaming delegation scheme constructed in Section 6.2 does hold a secret state, and handling this reusability issue is one of the main technical challenges of this work. It is for this reason that we need all the machinery that was developed in Section 4.1.

In what follows we define the notion of *one-time soundness*. The reason we need this definition, is that our soundness proof (for our streaming delegation scheme in Section 6.2), consists of two

[3] This is in contrast to our memory delegation scheme (in Section 5.2), where the state of the delegator was not secret.

parts: We first prove that our scheme has one-time soundness, i.e., it is sound assuming the delegation protocol is executed only once. Then, we argue that the one-time soundness implies reusable soundness.

Definition 43 (One-time Soundness) *Let k be a security parameter and let N be a parameter. Let $\mathcal{F}$ be a function class and let $\mathsf{sDel}_{\mathcal{F}}$ be a delegation scheme for $\mathcal{F}$. The corresponding* **one-time streaming security game** $\mathsf{G}_1^{\mathsf{W}^*}(k, N)$ *played by an (adversarial) worker W^* is defined the same as $\mathsf{G}^{\mathsf{W}^*}(k, N)$, except that the game is terminated after the first execution of the delegation protocol $\langle \mathsf{D}, \mathsf{W}^* \rangle$.*

We say that $\mathsf{sDel}_{\mathcal{F}}$ has **one-time soundness error** *ε if for every worker strategy W^* with runtime $\mathrm{poly}(N)$, where $N = N(k)$,*

$$\Pr[\mathsf{W}^* \text{ succeeds in } \mathsf{G}_1^{\mathsf{W}^*}(k, N)] \leq \varepsilon(N).$$

$\mathsf{sDel}_{\mathcal{F}}$ is **one-time sound** *if it has a negligible one-time soundness error (in parameter N).*

We proceed to state our main theorem for streaming delegation.

Theorem 44 (Streaming Delegation) *Let k be a security parameter, and let N be a parameter. Let $\mathcal{F}$ be the class of all $\mathcal{L}$-uniform poly-size boolean circuits. Assume the existence of a fully-homomorphic encryption scheme secure against $\mathrm{poly}(N)$-size adversaries. Then there exists a 2-round streaming delegation scheme $\mathsf{sDel}_{\mathcal{F}}$ for $\mathcal{F}$ with the following properties.*

- *$\mathsf{sDel}_{\mathcal{F}}$ has perfect completeness and negligible soundness error.*
- *D updates her secret state in time $\mathrm{polylog}(N)$, per data item.*
- *In the delegation protocol, when delegating a function $f \in \mathcal{F}$ computable by an $\mathcal{L}$-uniform circuit of size S and depth d, the delegator D runs in time $\mathrm{poly}(k, d, \log N)$, and the worker W runs in time $\mathrm{poly}(k, S, \log N)$.*

In particular, assuming the existence of a fully-homomorphic encryption scheme secure against adversaries of size $\mathrm{poly}(N)$, we obtain a streaming delegation scheme for $\mathcal{L}$-uniform **NC** computations, where the delegator D runs in time *poly-logarithmic* in the length of data stream.

We proceed to present our streaming delegation scheme in the next section.

6.2 Streaming Delegation Scheme

In this section, we prove Theorem 44, by constructing a non-interactive streaming delegation scheme with the desired properties.

6.2.1 Overview of our Streaming Delegation Scheme

Our streaming delegation scheme is similar to our memory delegation scheme mDel, presented in Section 5.2, and the main difference is in the way the certificate is generated and updated, and in the Reveal protocol of the Compute operation.

Generating and updating the certificate. Recall that in the memory delegation scheme, the certificate of the delegator D consists of a tree-commitment of the low-degree extension of her memory x. Namely, her certificate is $(h, T_h(\mathrm{LDE}_x))$, where h is a collision resistant hash function. Note that this certificate cannot be updated in a streaming manner, since any change to x changes the low-degree extension LDE_x almost everywhere.

Instead, in the streaming setting, we replace the tree commitment with an "*algebraic commitment*", which has the property that it can be updated efficiently when new data items arrive. The resulting certificate is a random point in the low-degree extension of the stream x; i.e., $(z, \mathrm{LDE}_x(z))$ for a random point z. Proposition 6 implies that this certificate is efficiently updatable,if we assume some upper-bound N on the size of the stream, and we take parameters $\mathbb{H}, \mathbb{F}, m$ such that $\mathbb{H}^m = \theta(N)$. The parameters we take are

$$|\mathbb{H}| = \mathrm{polylog}(N),\ m = \theta\left(\frac{\log N}{\log\log N}\right),\ |\mathbb{F}| = \mathrm{poly}(|\mathbb{H}|). \tag{6.1}$$

The Compute operation. The Compute operation of our streaming delegation scheme is very similar to the Compute operation of the memory delegation scheme mDel, and the main distinction is in the Reveal protocol. Namely, in Compute(f) the delegator and worker run $\mathrm{GKR}^{(u)}(f)$, which is the u-fold parallel repetition of $\mathrm{GKR}(f)$ (the parallel repetition is in order to get negligible soundness). In order to verify correctness, the delegator needs to verify the value of $\mathrm{LDE}_x(r_i)$ for random $r_1, \ldots, r_u$. Recall that this low-degree extension is w.r.t. the parameters $\mathbb{H}', \mathbb{F}', m'$ given in

Theorem 8. Namely,

$$|\mathbb{H}'| = \theta(d \cdot \log n),\ m' = \theta\left(\frac{\log n}{\log d}\right),\ |\mathbb{F}'| = \text{poly}(|\mathbb{H}'|).$$

where d is the depth of the circuit computing f, and n is the input length (i.e., the current size of the stream). Also, recall that the certificate of the delegator is of the form $(z, \text{LDE}_x(z))$, where here the low-degree extension is w.r.t. the parameters $\mathbb{H}, \mathbb{F}, m$ as in Equation (6.1).

In the memory delegation scheme we overcome this gap by delegating the computation of the functions g_{r_i}, which are defined by

$$g_{r_i}(x) \triangleq \text{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i),$$

using $\text{GKR}^{(u)}$ with respect to $(\mathbb{H}, \mathbb{F}, m)$. In order to verify the correctness of these u protocols, the delegator needs to verify u^2 values $\text{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j})$, where each $z_{i,j}$ is a random value in $\mathbb{F}^m$.

Remark. In our setting this approach is too costly since the running time of the worker W during the delegation protocols of g_{r_i} is polynomial in N (where N is an upper bound on the stream size), as opposed to polynomial in n, which is the actual stream size (see Theorem 8). However, it turns out that with a slight modification, we can ensure that W runs in time $\text{poly}(k, n, \log N)$ during these delegation protocols, where k is the security parameter. The idea is the following: Let $m'' = \lceil \frac{\log n}{\log \log N} \rceil$ so that $n \leq |\mathbb{H}|^{m''} \leq n \cdot \text{polylog}(N)$. The delegator will delegate the functions $g_{r_i}(x_i)$ by running $\text{GKR}^{(u)}$ with respect to $(\mathbb{H}, \mathbb{F}, m'')$. Note that the runtime of W during these protocols is $\text{poly}(n, k, \log N)$, as desired. At the end of these protocols the delegator needs to verify u^2 values $\text{LDE}_x^{\mathbb{F},\mathbb{H},m''}(z''_{i,j})$, where each $z''_{i,j}$ is a random value in $\mathbb{F}^{m''}$.

We next argue that for every $z'' \in \mathbb{F}^{m''}$,

$$\text{LDE}_x^{\mathbb{F},\mathbb{H},m''}(z'') = \text{LDE}_x^{\mathbb{F},\mathbb{H},m}(0^{m-m''}, z'').$$

To this end, recall (from Section 3.3) that

$$\text{LDE}_x^{\mathbb{F},\mathbb{H},m}(z) = \sum_{p \in \mathbb{H}^m} \tilde{B}(z, p) \cdot x_{\alpha(p)} \tag{6.2}$$

and similarly

$$\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m''}(z) = \sum_{p\in\mathbb{H}^{m''}} \tilde{B}(z,p)\cdot x_{\alpha''(p)}$$

where for every $z \in \mathbb{H}^m$ (or $z \in \mathbb{H}^{m''}$ respectively) it holds that $\tilde{B}(z,p) = 1$ if $z = p$, and $\tilde{B}(z,p) = 0$ otherwise. The functions $\alpha : \mathbb{H}^m \to \{0,1,\ldots,N-1\}$ and $\alpha'' : \mathbb{H}^{m''} \to \{0,1,\ldots,n-1\}$ are the lexicographic order, and thus for every $p \in \mathbb{H}^{m''}$ it holds that $\alpha''(p) = \alpha(0^{m-m''},p)$. Therefore, for every $x \in \{0,1\}^n$ and for every $z \in \mathbb{H}^{m''}$,

$$\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(0^{m-m''},z) = \mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m''}(z). \tag{6.3}$$

This, together with the Schwartz-Zippel lemma, and with the fact that both $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}$ and $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m''}$ are polynomials of degree at most $|\mathbb{H}|-1$ in each variable, implies that for every $z \in \mathbb{F}^{m''}$ it holds that

$$\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(0^{m-m''},z) = \mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m''}(z).$$

Therefore, it remains to verify the values of $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j})$, where $z_{i,j} \triangleq (0^{m-m''},z''_{i,j})$. In the memory delegation scheme this was done using the Reveal$(z_{i,j})$ protocol where the worker reveals the augmented path of the leaf $z_{i,j}$ in the Merkle tree-commitment of LDE_x. Here the Reveal protocol needs to be totally different, since the delegator cannot compute the tree-commitment of LDE_x.

Unfortunately, unlike in the memory delegation scheme, in the streaming setting constructing a *reusable* and *sound* reveal protocol is highly non-trivial.

The Reveal protocol. Our starting point is a basic reveal protocol Reveal_1 described in Figure 6.1. Note that the soundness of Reveal_1 relies on the secrecy of the certificate σ. Namely, assuming that W does not know the point z, it is not hard to see, by Schwartz-Zippel Lemma, that an adversarial worker can cheat with probability at most $d/|\mathbb{F}|$, where d is the (total) degree of LDE_x.

However, note that the Reveal_1 protocol is not reusable. Suppose that D uses the above reveal protocol to learn the value of LDE_x on two random points $s, s' \in \mathbb{F}^m$. From the two executions, an adversarial worker W^* receives two lines $\ell_{s,z}$ and $\ell_{s',z}$, and can learn the secret point z by taking the intersection of the two lines. Once W^* learns z, W^* can easily cheat by returning any polynomial p^* that agrees with LDE_x only on point z but disagrees on the remaining points.

Reveal$_1$ protocol: D stores a *secret* state $\sigma = (z, \mathrm{LDE}_x(z))$, where $x \in \{0,1\}^N$ and z is a random point in $\mathbb{F}^m$, and wants to learn the value of $\mathrm{LDE}_x(s)$ from W.

- D sends to W the line ℓ_{sz} that passes through the points s and z. More specifically, D chooses two random points $\alpha_1, \alpha_2 \leftarrow \mathbb{F}$, and defines $\ell_{s,z}$ to be the line that satisfies $\ell_{s,z}(\alpha_1) = z$ and $\ell_{s,z}(\alpha_2) = s$.
- W returns a univariate polynomial $p : \mathbb{F} \to \mathbb{F}$, which is the polynomial LDE_x restricted to the line $\ell_{s,z}$ (i.e., $p = \mathrm{LDE}_x|_{\ell_{s,z}}$).
- D checks whether $p(\alpha_1) = \mathrm{LDE}_x(z)$, and if so accepts the value $p(\alpha_2) = \mathrm{LDE}_x(s)$. Otherwise, she rejects.

Figure 6.1: Reveal$_1$ protocol

As observed by Gennaro *et. al.* [GGP10], a natural way to protect the secret point z, is to run the above Reveal protocol under a fully-homomorphic encryption (FHE) scheme. Namely, D generates a pair of keys $(\mathsf{pk}, \mathsf{sk})$ for a FHE $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{Eval})$, and sends pk and an encrypted line $\hat{\ell}_{s,z} = \mathrm{Enc}_{\mathsf{pk}}(\ell_{s,z})$ to W, who can compute the polynomial $p = \mathrm{LDE}_x|_\ell$ homomorphically under the encryption. Indeed, by the semantic security of FHE, an adversarial worker W^* cannot learn any information from D's message $\hat{\ell}_{s,z}$. This indeed makes the protocol reusable provided that W^* does not learn the decision bits of D, as proved in [GGP10, CKV10].

However, since the decision bit of D can potentially contain one bit information about the secret point z, it is not clear that security holds if W^* learns these decision bits. In fact, for both of the delegation schemes of [GGP10, CKV10], which use FHE to hide the delegator D's secret state, there are known attacks that learn the whole secret state of D bit-by-bit from D's decision bits.

Fortunately, we are able to show that a variant of the Reveal$_1$ protocol described in Figure 6.2 is reusable even if W^* learns the decision bits of D. The main difference between Reveal$_1$ and Reveal$_2$ is that in Reveal$_2$, the delegator D uses a random *two-dimensional* affine subspace instead of a line, and uses an FHE to mask the entire protocol.

Using our techniques developed in Section 4.1 (and in particular using Lemma 22), we show in Section 6.2.3 that no adversarial W^* can learn useful information about the secret point z from the Reveal$_2$ protocol. We note that the proof of the above statement is highly non-trivial, and is one of the main technical difficulties in this work. Informally, the proof first uses Lemma 16, which claims that the ciphertext $\hat{S}_{sz}$ and the decision bit b of D (which depend on the strategy of W^*) do not give too much information about S_{sz} to W^*. In other words, the random subspace $S_{s,z}$ still has high (pseudo-)entropy from the point of view of W^*. Then it uses an *information-theoretic* argument to

Reveal$_2$ protocol: D stores a secret state $\sigma = (z, \mathrm{LDE}_x(z))$, where $x \in \{0,1\}^N$ and z is a random point in $\mathbb{F}^m$, and wants to learn the value of $\mathrm{LDE}_x(s)$ from W.

- D does the following.
  1. Generate a pair of keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{Gen}(1^k)$ for a fully homomorphic encryption scheme FHE.
  2. Choose a random two-dimensional affine subspace $S_{s,z} \subset \mathbb{F}^m$ that contains the points s and z. More specifically, choose two random points $\alpha_1, \alpha_2 \leftarrow \mathbb{F}^2$ and let $S_{s,z} \subset \mathbb{F}^m$ be a random two-dimensional affine subspace that satisfies $S_{s,z}(\alpha_1) = z$ and $S_{s,z}(\alpha_2) = s$.
  3. Send $\hat{S}_{s,z} \leftarrow \mathrm{Enc}_{\mathsf{pk}}(S_{s,z})$ and pk to W.
- W homomorphically computes the two-variate polynomial $p = \mathrm{LDE}_x|_{S_{s,z}}$ under the FHE (denote the resulting ciphertext $\hat{p}$), and sends $\hat{p}$ to D.
- D decrypts and checks whether $p(\alpha_1) = \mathrm{LDE}_x(z)$, and if so accepts the value $p(\alpha_2) = \mathrm{LDE}_x(s)$.

Figure 6.2: Protocol Reveal$_2$

argue that a random point z in a sufficiently random (with high entropy) subspace $S_{s,z}$ is *statistically close* to a random point in $\mathbb{F}^m$, which implies that W^* does not learn useful information about z.

The Field Size. Recall that by Schwartz-Zippel Lemma, an adversarial worker can cheat with probability at most $d/|\mathbb{F}|$, where d is the (total) degree of LDE_x. Recall that in our setting of parameters:

$$|\mathbb{H}| = \mathrm{polylog}(N),\ m = O\left(\frac{\log N}{\log\log N}\right),\ |\mathbb{F}| = \mathrm{poly}(|\mathbb{H}|).$$

Thus, a cheating worker can cheat with probability $d/|\mathbb{F}| = O(1/\mathrm{polylog}(N))$, which is not low enough.

The idea is to reduce the cheating probability to negligible by simply increasing the field size to be super-polynomial. However, we cannot increase the field size in the GKR protocol, since it will increase the complexity of the worker. Instead, we use an extension field $\tilde{\mathbb{F}}$ of $\mathbb{F}$, of super-polynomial size, only in the certificate and the Reveal protocol, but run the GKR protocols as before. Namely, the secret state is $\sigma = (z, \mathrm{LDE}^{\tilde{\mathbb{F}},\mathbb{H},m}(z))$ where $z \leftarrow \tilde{\mathbb{F}}^m$, The GKR protocols are run exactly as before (one $\mathrm{GKR}^{(u)}$ protocol with the parameters $(\mathbb{H}', \mathbb{F}', m')$, and u $\mathrm{GKR}^{(u)}$ protocols with the parameters $(\mathbb{H}, \mathbb{F}, m'')$). In the Reveal protocol, the worker reveals to a point $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(s)$ as follows: The delegator sends an encryption of a random two-dimensional subspace $S_{s,z} \subset \tilde{\mathbb{F}}^m$, containing the points s and z, and the worker sends an encryption of $\mathrm{LDE}_x|_{S_{s,z}}$. The delegator

verifies correctness exactly as in Reveal_2.

Reveal Multiple Points in Parallel. Finally, recall that in memory delegation, the reveal protocol is executed in parallel u^2 times to reveal u^2 random points $z_{i,j}$. However, if all reveal protocols use the same algebraic commitment $(z, \text{LDE}_x(z))$, then proving that z remains secret after W^* learns the decision bits of D becomes somewhat tricky.[4] Instead, we could use u^2 independent commitments, one for each copy of the Reveal protocol. However, it blows up the size of D's secret state as well as her runtime during the update procedure. It also makes the analysis somewhat more complicated.

Therefore, instead of running multiple copies of the Reveal protocol in parallel, we use a classic "many-to-one" reduction to reduce the number of points to a *single* point, so that D only needs to run a single copy of the Reveal protocol. Briefly, the idea is to let D choose a random degree-u^2 curve ℓ, passing through $z_{i,j} \in \mathbb{F}^m$ for $i, j \in [u]$ and passing through an additional random point $s \leftarrow \tilde{\mathbb{F}}^m$, and send ℓ to W. More specifically, D will choose $u^2 + 1$ random points $\alpha_0, \alpha_{i,j} \leftarrow \tilde{\mathbb{F}}$ for $i, j \in [u]$, and let ℓ be the u^2-degree polynomial such that $\ell(\alpha_0) = s$ and $\ell(\alpha_{i,j}) = z_{i,j}$ for every $i, j \in [u]$. The worker W returns a univariate polynomial $p : \tilde{\mathbb{F}} \to \tilde{\mathbb{F}}$, which is supposed to be LDE_x restricted on the line ℓ. If W is honest, then D learns the values $\text{LDE}_x(z_{i,j}) = p(\alpha_{i,j})$ for $i, j \in [u]$, as desired. To verify that W was indeed honest, D and W run the Reveal protocol on the point s, to reveal to $\text{LDE}_x(s)$, and D checks whether indeed $p(\alpha_0) = \text{LDE}_x(s)$.

To summarize, our $\mathsf{Compute}(f)$ protocol runs the above many-to-one protocol and the Reveal protocol, in parallel, in order to reveal to $\{\text{LDE}_x(z_{i,j})\}_{i,j \in [u]}$.

6.2.2 Formal Description of Our Streaming Delegation Scheme

In this section, we give a formal description of our streaming delegation scheme for delegating $\mathcal{L}$-uniform depth-d circuits. Our construction uses the following building blocks

1. A fully homomorphic encryption scheme $\text{E} = (\text{Gen}, \text{Enc}, \text{Dec}, \text{Eval})$ where the semantic security holds against $\text{poly}(N)$-time adversaries (with negligible advantage in N).

2. The delegation scheme $\text{GKR} = \langle \mathsf{D}', \mathsf{W}' \rangle$ from [GKR08, KR09] (see Theorem 8 for the properties of this delegation scheme). The main property we use here, is that the delegator can verify

[4] We can make it work, but the analysis becomes more complicated. We decide to avoid this unnecessary complication.

proofs by accessing its input x at a single random point in LDE_x.

- **Parameters.** Let k be the security parameter, and let N be an upper bound on the length of the data stream. Let $\mathbb{H}$ be an extension field of $\mathbb{GF}[2]$ of size $|\mathbb{H}| = \mathrm{polylog}(N)$, let $m = \theta\left(\frac{\log N}{\log\log N}\right)$, and let $\mathbb{F}$ be an extension field of $\mathbb{H}$ of size $|\mathbb{F}| = \mathrm{poly}(|\mathbb{H}|)$. Let $\tilde{\mathbb{F}}$ be an extension field of $\mathbb{F}$ of size $|\tilde{\mathbb{F}}| = N^{\log N}$.

- **Generating and updating the secret state.**

  1. At the initial time $t = 0$, the delegator D chooses a random point $z \leftarrow \tilde{\mathbb{F}}^m$, and stores $\sigma_0 = (z, 0)$ as her secret state.
  2. At each time $t \in [N]$, when a data item $x_t \in \{0,1\}$ arrives, D updates her secret state from $\sigma_{t-1} = (z, \mathrm{LDE}_{x^{t-1}}(z))$ to $\sigma_t = (z, \mathrm{LDE}_{x^t}(z))$, by using Proposition 6.[5] (Recall that $x^t = (x_1, \ldots, x_t)$ denotes the entire data stream up until time t.)

- $\mathsf{Compute}(f, t)$**.** At any time $t \in [N]$ when the delegator wants the worker to compute some function f, where f is an $\mathcal{L}$-uniform depth-d circuit, they run the following protocols in parallel.

  1. Run $\mathrm{GKR}^{(u)}$, which is the u-fold parallel repetition of the underlying delegation protocol GKR, for delegating the computation of $f(x)$, where u is any parameter such that $1/2^u = \mathsf{ngl}(N)$. If at any time the GKR delegator rejects, the delegator D rejects.

     Let $\mathbb{F}', \mathbb{H}', m'$ be the parameters used by $\mathrm{GKR}^{(u)}$, and recall that at the end of the protocol, D needs to verify the value of $\mathrm{LDE}_{x^t}^{\mathbb{F}',\mathbb{H}',m'}$ at u random points $r_1, \ldots, r_u \in (\mathbb{F}')^{m'}$, i.e. it needs to check whether $\mathrm{LDE}_{x^t}^{\mathbb{F}',\mathbb{H}',m'}(r_i) = v_i$ for some values v_i.
  2. For each $i \in [u]$, run $\mathrm{GKR}^{(u)}$ for delegating the computation in $g_{r_i}(x^t) \triangleq \mathrm{LDE}_{x^t}^{\mathbb{F}',\mathbb{H}',m'}(r_i)$. These $\mathrm{GKR}^{(u)}(g_{r_i})$ protocols are done with respect to $(\mathbb{H}, \mathbb{F}, m'')$, where $m'' = \theta\left(\frac{\log t}{\log\log N}\right)$.[6] Moreover, these protocols are masked using a PIR scheme to keep the secrecy of the r_i's, which is necessary to ensure the soundness of the $\mathrm{GKR}^{(u)}$ protocol of Step 1.

[5] More explicitly, the update is done using the following formula.

$$\mathrm{LDE}_{x^t}(z) = \mathrm{LDE}_{x^{t-1}}(z) + x_t \cdot \mathrm{LDE}e_t(z),$$

where $e_t \in \{0,1\}^N$ denotes the N-bit string with 1 at the t-th bit, and 0 otherwise. Note that $\mathrm{LDE}_{e_t}(z)$ can be computed in $\mathrm{polylog}(N)$ time by interpolation.

[6] Recall that since computing $g_{r_i}(\cdot)$ can be done by a $\mathrm{polylog}(N)$-depth poly-size circuit, this setting of parameters works for $\mathrm{GKR}^{(u)}$.

We note that Steps 1 and 2 above are almost identical to Steps 1 and 2 of the $\mathsf{Compute}(f)$ operation of our memory delegation scheme, constructed in Section 5.2.2, and we refer the reader to Section 5.2.2 for a detailed description of the masked $\mathrm{GKR}^{(u)}(g_{r_i})$ protocols.[7]

If the GKR delegator rejects at any point, then the delegator D rejects. Otherwise, in order to verify these protocols, D needs to check that $\mathrm{LDE}_{x^t}^{\mathbb{F},\mathbb{H},m''}(z''_{i,j}) = w_{i,j}$ for u^2 random points $z''_{i,j} \in \mathbb{F}^{m''}$, and some values $w_{i,j} \in \mathbb{F}$. However, as was argued above (see Equation (6.3)), $\mathrm{LDE}_{x^t}^{\mathbb{F},\mathbb{H},m''}(z''_{i,j}) = \mathrm{LDE}_{x^t}^{\mathbb{F},\mathbb{H},m}(z_{i,j})$ where $z_{i,j} = (0^{m-m''}, z'_{i,j})$. Thus, D needs to check that $\mathrm{LDE}_{x^t}^{\mathbb{F},\mathbb{H},m}(z_{i,j}) = w_{i,j}$ for u^2 points $z_{i,j} \in \mathbb{F}^m$.

3. Run a many-to-one protocol to reduce computing $\mathrm{LDE}_{x^t}^{\mathbb{F},\mathbb{H},m}(z_{i,j})$ for u^2 points $z_{i,j} \in \mathbb{F}^m$, to computing $\mathrm{LDE}_{x^t}^{\tilde{\mathbb{F}},\mathbb{H},m}(s)$ for a single random point $s \leftarrow \tilde{\mathbb{F}}^m$. Note that s is a random point in the extension field $\tilde{\mathbb{F}}^m$, as opposed to $\mathbb{F}^m$. As before, the points $(z_{i,j})_{i,j\in[u]}$ must be kept secret to ensure the soundness of the $\mathrm{GKR}^{(u)}$ protocols of Step 2. We ensure this secrecy by simply running the protocol under an FHE scheme.

   We note that we cannot use a PIR scheme here, since $\tilde{\mathbb{F}}$ is of super-poly size, and therefore the use of a PIR scheme will result with the worker running in super-polynomial time. Instead we use an FHE scheme, which keeps both the worker and the delegator efficient.

   More specifically, the delegator D chooses a random $s \leftarrow \tilde{\mathbb{F}}^m$ and computes a canonical representation of the unique degree-u^2 curve $\ell_{\vec{z},s}$ passing through $\vec{z} = (z_{i,j})_{i,j\in[u]}$ and s. Then D generates $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathrm{Gen}(1^k)$, computes $\hat{\ell}_{\vec{z},s} = \mathrm{Enc}_{\mathsf{pk}_1}(\ell_{\vec{z},s})$, and sends $(\mathsf{pk}_1, \hat{\ell}_{\vec{z},s})$ to W. The worker W computes, homomorphically under encryption, a univariate polynomial $p_1 \triangleq \mathrm{LDE}_{x^t}^{\tilde{\mathbb{F}},\mathbb{H},m} \circ \ell_{\vec{z},s}$, and returns the resulting ciphertext $\hat{p}_1$ to D, who decrypts to obtain the polynomial p_1.

   Let $\alpha_0, \alpha_{i,j} \in \tilde{\mathbb{F}}$ for $i,j \in [u]$ be such that $\ell_{\vec{z},s}(\alpha_0) = s$ and $\ell_{\vec{z},s}(\alpha_{i,j}) = z_{i,j}$. D checks whether $w_{i,j} = p_1(\alpha_{i,j})$ (which are supposed to be $\mathrm{LDE}_{x^t}^{\mathbb{F},\mathbb{H},m}(z_{i,j})$). If not, she rejects.

4. In order to verify that indeed $p_1 = \mathrm{LDE}_{x^t}^{\tilde{\mathbb{F}},\mathbb{H},m} \circ \ell_{\vec{z},s}$, the delegator D checks whether $\mathrm{LDE}_{x^t}^{\tilde{\mathbb{F}},\mathbb{H},m}(s) = p_1(\alpha_0)$, using the Reveal protocol described in Figure 6.3, and using its certificate $\sigma_t = (z, \mathrm{LDE}_{x^t}^{\tilde{\mathbb{F}},\mathbb{H},m}(z))$.

   Note that in the Reveal protocol, we are specific about the representation of the random affine space $S_{s,z}$. This representation will be useful when we apply our main leakage

[7] We mention that, we can also mask the $\mathrm{GKR}^{(u)}$ protocol using an FHE scheme, as opposed to a polylog PIR scheme, which is what we do in Step 3.

lemma (Lemma 22), established in Section 4.1, to prove the reusable soundness of sDel in Section 6.2.3.

If D rejects in the Reveal(s) protocol, or if the accepted $p_2(\alpha_s)$ is not equal to the claimed value $p_1(\alpha_0)$, then D rejects.

5. The delegator accepts the interaction if and only if she did not reject in any place above.

Note that the first two steps are (almost) the same as that in our memory delegation scheme (Section 5.2.2). As before, we denote the delegation protocol of Step 1 by

$$\mathrm{GKR}^{(u)} = \langle \mathsf{W}'^{(u)}, \mathsf{D}'^{(u)}(r_1, \ldots, r_u)\rangle(f).$$

We denote the masked delegation protocols of Step 2 by

$$\mathrm{PIR}\left(\langle \mathsf{W}'^{(u)}, \mathsf{D}'^{(u)}(z''_{i,1}, \ldots, z''_{i,u})\rangle(g_{r_i})\right).$$

We denote the masked many-to-one protocol of Step 3 by u^2-to-1$(\vec{z}, s)$ and the reveal protocol of Step 4 by Reveal(s). Note that the streaming data x^t is implicit in all these notations. We summarize the $\mathsf{Compute}(f)$ protocol in Figure 6.4.

Remark. We note that our streaming delegation scheme can also be used as a memory delegation scheme. Recall that the only component that a memory delegation scheme has, and a streaming delegation scheme does not have, is the Update operation. Also recall that in our memory delegation scheme, the delegator D *delegates* the update of her state (from $(h, T_h(x))$ to $(h', T_{h'}(g(x)))$) to the worker W. The same idea can be applied to the streaming delegation scheme as well.[8] As mentioned in Section 5.2.1, our memory delegation scheme has several advantages over the one that could be obtained from our streaming delegation scheme.

6.2.3 Proof of Theorem 44.

In this section, we prove that the construction above satisfies the properties of Theorem 44.

[8] We note that to ensure that D's secret state is kept secret from the worker W after the Update operation, we would need to run the $\mathsf{Compute}$ operation (for updating D' secret state) under FHE.

Reveal(s):
The delegator D stores a secret algebraic commitment $\sigma_t = (z, \mathrm{LDE}_{x^t}^{\tilde{\mathbb{F}},\mathbb{H},m}(z))$ and wants to learn the value of $\mathrm{LDE}_{x^t}^{\tilde{\mathbb{F}},\mathbb{H},m}(s)$ from the worker W, who stores $x \in \{0,1\}^N$.

- D does the following.
  1. Choose a random two-dimensional affine subspace $S_{s,z} \subset \tilde{\mathbb{F}}^m$ that contains the points s and z.
  2. Let $X = S_{s,z} - s \triangleq \{v - s : v \in S_{s,z}\}$. Namely, X is a random two-dimensional subspace that contains the point $z - s$.
  3. Choose a random representation for X, by choosing two random points $x_1, x_2 \leftarrow X$, and represent X as a matrix in $\tilde{\mathbb{F}}^{m \times 2}$ where the first column is x_1 and the second column is x_2.
     Thus, every point $a \in S_{s,z}$ is represented by $a = X \cdot \alpha + s$ for some $\alpha \in \tilde{\mathbb{F}}^2$.
  4. Generate a pair of keys $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathrm{Gen}(1^k)$ and compute an encryption
     $$\hat{S}_{s,z} \triangleq (\hat{X}, \hat{s}) = (\mathrm{Enc}_{\mathsf{pk}_2}(X), \mathrm{Enc}_{\mathsf{pk}_2}(s)).$$
  5. Send $(\mathsf{pk}_2, \hat{S}_{s,z})$ to W.
- W homomorphically computes the two-variate polynomial $p_2 = \mathrm{LDE}_x \circ S_{s,z}$ under the FHE (denote the resulting ciphertext $\hat{p}_2$), and sends $\hat{p}_2$ to D.
- D decrypts $\hat{p}_2$ to obtain p_2. Let $\alpha_s, \alpha_z \in \tilde{\mathbb{F}}^2$ be such that $S_{s,z}(\alpha_s) = s$ and $S_{s,z}(\alpha_z) = z$. D checks whether $p_2(\alpha_z) = \mathrm{LDE}_{x^t}^{\tilde{\mathbb{F}},\mathbb{H},m}(z)$. If so, D accepts the value $\mathrm{LDE}_{x^t}^{\tilde{\mathbb{F}},\mathbb{H},m}(s) = p_2(\alpha_s)$. Otherwise, D rejects.

Figure 6.3: Formal description of the Reveal protocol.

The perfect completeness follows immediately from the completeness of the underlying GKR delegation scheme, the completeness of the FHE scheme, and the completeness of the Reveal protocol. The fact that the delegator D can update her secret state in time polylog(N) follows from Proposition 6. The efficiency guarantees of the $\mathsf{Compute}$ protocol follow immediately from the efficiency guarantees of GKR, the efficiency guarantees of the underlying PIR scheme, and the efficiency of the u^2-to-1 protocol and the Reveal protocol, under FHE.

The main difficulty is in proving soundness. First we observe that the streaming delegation scheme is one-time sound. This proof is very similar to the proof that our memory delegation scheme is one-time sound (Lemma 36). However, it is at all not clear how to reduce one-time soundness to many-time soundness, since the delegator uses the same secret state in all the executions, and each decision bit of the delegator may leak one bit information about this secret state.

$\mathsf{Compute}(f, t)$:

The delegator D stores a secret state $\sigma_t = (z, \mathrm{LDE}_x^{\tilde{\mathbb{F}},H,m}(z))$ and wants to learn the value of $f(x)$ from the worker W, who stores $x \in \{0,1\}^N$.

1. D and W run $\mathrm{GKR}^{(u)} = \langle \mathsf{W}'^{(u)}, \mathsf{D}'^{(u)}(r_1, \ldots, r_u)\rangle(f)$.
   (a) If $\mathsf{D}'^{(u)}$ rejects, then the delegator D outputs "reject".
   (b) At the end of this protocol, the delegator D' needs to verify that $\mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i) = v_i$ for some values v_i.
2. For every $i \in [u]$, run $\mathrm{PIR}(\langle \mathsf{W}'^{(u)}, \mathsf{D}'^{(u)}(z''_{i,1}, \ldots, z''_{i,u})\rangle(g_{r_i}))$ with parameters $(\mathbb{H}, \mathbb{F}, m'')$ where $m'' = \theta\left(\frac{\log t}{\log\log N}\right)$.
   (a) The delegator D makes sure that in these protocols the worker still claims that indeed $g_{r_i}(x) = v_i$, where $g_{r_i}(x) = \mathrm{LDE}_x^{\mathbb{F}',\mathbb{H}',m'}(r_i)$. If this is not the case, then the delegator D outputs "reject".
   (b) If at any point D' rejects, then the delegator D outputs "reject".
   (c) In order to verify these protocols, the delegator D needs to verify that $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m''}(z''_{i,j}) = w_{i,j}$ for some values $w_{i,j}$, which is equivalent to verifying that $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}(z_{i,j}) = w_{i,j}$ where $z_{i,j} = (0^{m-m''}, z''_{i,j}) \in \mathbb{F}^m$ (see Equation (6.3)).
3. Run u^2-to-1$(\vec{z}, s)$ where $\vec{z} = (z_{i,j})$.
   Recall that in this protocol, D sends W a message $(\mathsf{pk}_1, \hat{\ell}_{\vec{z},s})$, where $\ell_{\vec{z},s}$ is a canonical representation of the u^2-degree curve that passes through the points $z_{i,j}$ and the point s; namely, $\ell_{\vec{z},s}(\alpha_{i,j}) = z_{i,j}$ and $\ell_{\vec{z},s}(\alpha_0) = s$, for some $\alpha_{i,j}, \alpha_0 \in \tilde{\mathbb{F}}$. Then, W sends $\hat{p}_1$, where supposedly, $p_1 = \mathrm{LDE}_{x^t}^{\tilde{\mathbb{F}},\mathbb{H},m} \circ \ell_{\vec{z},s}$
   The delegator D verifies that $p_1(\alpha_{i,j}) = w_{i,j}$ for every $i, j \in [u]$. If this is not the case, then the delegator D outputs "reject".
4. The delegator D verifies that $\mathrm{LDE}_x^{\tilde{\mathbb{F}},\mathbb{H},m}(s) = p_1(\alpha_0)$ by running Reveal(s).
   If D rejects in the Reveal(s) protocol, or if $p_1(\alpha_0) \neq p_2(\alpha_s)$ (where p_2 and α_s are defined in the Reveal(s) protocol in Figure 6.3), then D rejects.
5. The delegator D outputs "accept", assuming she didn't output "reject" at any point.

Figure 6.4: Compute(f)

Soundness: high-level intuition. Intuitively, we would like to prove that in our construction, the decision bit of the delegator does not reveal any information about her secret state z, and thus this secret is "reusable", and hence the protocol is many-time sound.

Recall that in our Compute protocol, the delegator D uses her secret state z only in the Reveal protocol, where she sends the worker an encryption of a random two-dimensional affine subspace $S_{s,z}$ that contains z. Intuitively, the idea is to use our main leakage lemma (Lemma 22 in section 4.1) to argue that in the Reveal protocol, any one-bit leakage about the subspace $S_{s,z}$ does not reveal information about z (i.e., z looks totally random from the worker's view). While it is indeed true that any bit of leakage about $S_{s,z}$ does not reveal any information about z, note that the decision of the delegator in the Reveal protocol does depend on the actual point z, and not just on the subspace $S_{s,z}$ (as she checks the value of $p_2(\alpha_z)$).

One way of going around this, is by considering another delegator $\tilde{D}$, which is a "somewhat cautious" delegator, in the sense that she does not check the validity of the polynomial p_2 at the specific point α_z, as this depends on her secret point z. Instead, $\tilde{D}$ is an inefficient delegator that holds the entire stream $x \in \{0,1\}^t$, and checks that $p_2 = \mathrm{LDE}_x \circ S_{s,z}$. The point is that $\tilde{D}$ does not use the actual point z, but only the random subspace $S_{s,z}$.

Now we can try to use our main leakage lemma (Lemma 22) to claim that each execution of Compute does not reveal any information about the delegator's secret state z, and thus she can safely use this secret state again. It still remains to argue that if a cheating worker could cheat when talking to a "somewhat cautious" delegator, then he could also cheat when talking to the real delegator. The idea here is to use the Schwartz-Zippel lemma (with a computationally hidden z).

Towards our formal proof. In our formal proof, we take another route. We define a "cautious" delegator (as opposed to a "somewhat cautious") to be one that each time uses a fresh new secret state z'. As above, our cautious delegator is inefficient, and has the entire stream $x \in \{0,1\}^t$. The many-time soundness of this cautious scheme follows immediately from its one-time soundness.

Next, we consider a sequence of N hybrid games, $\mathsf{G}_0, \ldots, \mathsf{G}_N$, where in G_t the delegator is cautious during the first t Compute protocols, and runs the protocol of the original delegator D from the $t+1$'st execution onwards. Note that G_0 is the real soundness game (with the real delegator D), as defined in Definition 41, whereas G_N is the soundness game with the cautious delegator $\tilde{D}$. Thus, if there exists a cheating worker W^* that cheats in the original game G_0, by a standard hybrid

argument, there must exist two games G_{t-1} and G_t where there is a noticeable gap between the success probability of W^* in G_{t-1} and its success probability in G_t.

The idea is to use this fact to contradict our main leakage lemma (Lemma 22), as follows. Simulate the first $t-1$ $\mathsf{Compute}$ operations. Note that in both G_{t-1} and in G_t the delegator uses a fresh secret state in these executions. Then, in the t'th $\mathsf{Compute}$ operation, use the subspace given by the leakage lemma.

Recall that the leakage lemma claims that it is hard to distinguish between $(x, \mathrm{Enc}_{\mathsf{pk}}(X), \mathsf{pk}, b)$ and $(u, \mathrm{Enc}_{\mathsf{pk}}(X), \mathsf{pk}, b)$, where $x \leftarrow X$ and $u \leftarrow \tilde{\mathbb{F}}^m$. So, the idea is to use $\mathsf{pk}, \mathrm{Enc}_{\mathsf{pk}}(X)$ in the Reveal(s) protocol. Then continue the simulation of the rest of the $\mathsf{Compute}$ operations using the secret state $w + s$, where w is either a random element in X or a random element in $\tilde{\mathbb{F}}^m$. However, as was noted above, we cannot use the leakage lemma, since the original delegator actually used the secret point to check the validity of the polynomial p_2 sent by the worker. This, slightly complicates matters.

We define our "cautious" delegator to be *really cautious*, so that not only does he use a fresh secret state z' for each round, but he also checks the validity of the polynomial p_2 sent by the worker, by actually checking that $p_2 \equiv \mathrm{LDE}_x \circ S_{s,z'}$ (as opposed to checking its validity at a single point). Then, assuming we have a gap between G_{t-1} and G_t, we consider another hybrid game between these two games, which we denote by G^*. This game is similar to the game G_{t-1}, in the sense that the delegator uses the real secret state z in the t'th round, but it differs from G_{t-1} in that it checks the validity of p_2 by checking that $p_2 = \mathrm{LDE}_x \circ S_{s,z}$.

We use the Schwartz-Zippel lemma to prove that G_{t-1} and G^* are statistically indistinguishable, and we use our main leakage lemma (Lemma 22) to prove that G^* and G_t are computationally indistinguishable. The latter is done by contradiction. If there exists a distinguisher between the games G^* and G_t, then we construct a distinguisher that distinguishes between the distributions given the leakage lemma. This is done by simply embedding the input given by the leakage lemma in the t'th round of the Reveal protocol.

Remark 45 We remark that the security reduction is inherently non-uniform, due to the use of the machinery developed in Section 4.1. Specifically, the proof of Lemma 16 uses the equivalence of HILL and Metric entropy, which only holds in the non-uniform setting. Therefore, the cryptographic primitives that we rely on need to be secure against non-uniform adversaries. In contrast, the security

proof of our memory delegation scheme in Section 5.2 is uniform.

In what follows we formally define the notion of a cautious delegator (or a cautious streaming delegation scheme), followed by a formal proof of the (reusable) soundness of our streaming delegation scheme.

Cautious Streaming Delegation Scheme $\tilde{\mathsf{sDel}} = \langle \mathsf{W}, \tilde{\mathsf{D}} \rangle$. For the purpose of our analysis, we consider a variant of sDel, called *cautious streaming delegation scheme*, which we denote by $\tilde{\mathsf{sDel}} = \langle \mathsf{W}, \tilde{\mathsf{D}} \rangle$. In $\tilde{\mathsf{sDel}}$ the worker is exactly the same as the worker in sDel, but the delegator is "cautious."

More specifically, in $\tilde{\mathsf{sDel}}$, the delegator $\tilde{\mathsf{D}}$, instead of maintaining a secret state $\sigma_t = (z, \mathrm{LDE}_x(z))$, she stores the entire data stream $x^t \in \{0,1\}^t$. Recall that in the original sDel scheme, during the $\mathsf{Compute}$ protocol, the delegator uses her secret state σ_t only in the Reveal protocol. In $\tilde{\mathsf{sDel}}$, the $\mathsf{Compute}$ protocol is exactly as in sDel, except for the following modification to the Reveal(s) protocol.

- $\tilde{\mathsf{D}}$ generates her message in the same way as D, except that $\tilde{\mathsf{D}}$ chooses a random two-dimensional affine subspace $S_s \subset \tilde{\mathbb{F}}^m$ containing s, instead of choosing a random affine space $S_{s,z} \subset \tilde{\mathbb{F}}^m$ containing both s and z.
- To verify the polynomial p_2 returned by W (after decryption), $\tilde{\mathsf{D}}$ computes $\mathrm{LDE}_x \circ S_s$ on her own and checks if $p_2 = \mathrm{LDE}_x \circ S_s$, instead of checking its consistency with $\mathrm{LDE}_x \circ S_s$ on a single point.

In other words, $\tilde{\mathsf{D}}$ is doubly-cautious in the Reveal protocol: First, $\tilde{\mathsf{D}}$ doesn't reuse the same secret point z, but rather chooses the affine subspace at random, with the only restriction that it contains s. Second, she checks whether the polynomial p_2 returned by W is correct, instead of only checking if p_2 is correct on a random point z.[9] Our analysis proceeds in the following two steps.

- We first show, that $\tilde{\mathsf{sDel}}$ has negligible one-time soundness error, which immediately implies the (reusable) soundness of $\tilde{\mathsf{sDel}}$, since these executions are totally independent, as $\tilde{\mathsf{D}}$ does not use a secret state.

  The proof is very similar to the proof of the one-time soundness of our memory delegation scheme (Lemma 36).

[9] Note that $\tilde{\mathsf{D}}$ runs in time poly(N), which is not efficient in our setting, but this is only for the sake of the analysis.

- Let G and $\tilde{\mathsf{G}}$ be the (reusable) security games of sDel and $\tilde{\mathsf{sDel}}$, respectively. We shall show, that the main-leakage lemma (Lemma 22), together with a hybrid argument, implies that no poly(N)-size worker W^* succeeds with noticeably higher probability in G than in $\tilde{\mathsf{G}}$. In other words, there is no need to be cautious!

Lemma 46 *The streaming delegation scheme* sDel *constructed in Section 6.2.2 is one-time sound, i.e., it has negligible one-time soundness error.*

The proof of this lemma is essentially the same as the proof of Lemma 36, and is omitted.

Lemma 47 *The streaming delegation scheme* sDel *constructed in Section 6.2.2 is sound, i.e., it has negligible soundness error.*

Proof. Suppose for the sake of contradiction that there exists a poly(N)-size worker W^* such that

$$\Pr[\mathsf{W}^* \text{ succeeds in } \mathsf{G}^{\mathsf{W}^*}(k,N)] \geq \varepsilon(N), \tag{6.4}$$

for some noticeable $\varepsilon(N)$. Lemma 46 implies that $\tilde{\mathsf{sDel}}$ has negligible soundness error. Namely

$$\Pr[\mathsf{W}^* \text{ succeeds in } \tilde{\mathsf{G}}^{\mathsf{W}^*}(k,N)] \leq \mathsf{ngl}(N). \tag{6.5}$$

From Equation (6.4) and (6.5), we derive a contradiction to Lemma 22, using the following hybrid argument. Consider the following hybrid games G_t for every $t \in \{0,\ldots,N\}$.

- **Hybrid Game G_t:** The reusable security game for the streaming delegation scheme with a hybrid delegator D_t, who is cautious until (and including) time t, and behaves as the original delegator D after time t. More precisely, for every time $t' \leq t$, the hybrid delegator D_t stores the whole data stream $x_{t'}$ and behaves as the cautious delegator $\tilde{\mathsf{D}}$. At the beginning of time $t+1$, the hybrid delegator D_t generates a secret state $\sigma_{t+1} = (z, \mathrm{LDE}_{x^{t+1}}(z))$ by choosing a random $z \leftarrow \tilde{\mathbb{F}}^m$ and computing $\mathrm{LDE}_{x^{t+1}}(z)$. Then, for every time $t' \geq t+1$, the hybrid delegator D_t behaves as the original delegator D.

By definition, the hybrid games G_0 and G_N are simply G and $\tilde{\mathsf{G}}$, respectively. By a standard hybrid argument, Equations (6.4) and (6.5) imply that there exists some $t \in [N]$ such that

$$\Pr[\mathsf{W}^* \text{ succeeds in } \mathsf{G}_{t-1}^{\mathsf{W}^*}] - \Pr[\mathsf{W}^* \text{ succeeds in } \mathsf{G}_t^{\mathsf{W}^*}] \geq \varepsilon/N - \mathsf{ngl}(N). \tag{6.6}$$

Note that the only difference between G_{t-1} and G_t is at time t, where D_{t-1} behaves as the original delegator D, but D_t is still cautious. More precisely, at time t, the two delegators behave differently in the Reveal(s) protocol (if executed), as follows.

- D_{t-1} sends $(\hat{S}_{s,z}, \mathsf{pk}) = (\hat{X}, \hat{s}, \mathsf{pk})$ to W^*, where $S_{s,z}$ is a random 2-dimensional affine space containing s and z, and $X \in \tilde{\mathbb{F}}^{m\times 2}$ is a random representation of the two-dimensional subspace $S_{s,z} - s \triangleq \{v - x : v \in S_{s,z}\}$. Let $\alpha_z \in \tilde{\mathbb{F}}^2$ be such that $z = X \cdot \alpha_z + s$. Then D_{t-1} checks that $p_2(\alpha_z) = \mathrm{LDE}_{x^t}(z)$, where p_2 is the polynomial sent by W^* (after decryption).

- D_t sends $(\hat{S}_s, \mathsf{pk}) = (\hat{X}, \hat{s}, \mathsf{pk})$ to W^*, where $X \in \tilde{\mathbb{F}}^{m\times 2}$ is a random two-dimensional subspace, and $S = X + s$. Then D_t checks that $p_2 = \mathrm{LDE}_{x^t} \circ S_s$, where p_2 is the polynomial sent by W^* (after decryption).

Note that D_{t-1} continues to use z as her secret state after time t. Also note that in both cases, $X \in \tilde{\mathbb{F}}^{m\times 2}$ is a random representation of a random 2-dimensional linear subspace, so X is statistically close to a uniformly random m-by-2 matrix (with statistical distance $O(1/|\tilde{\mathbb{F}}|) = \mathsf{ngl}(N)$).

Looking ahead, we want to use the noticeable gap between the success probability of W^* in G_{t-1} and G_t (in Equation (6.6)) to contradict Lemma 22. To this end, we construct a distinguisher $\mathcal{A}$ that distinguishes distributions $(x, \hat{X}, \mathsf{pk}, b) \leftarrow \mathcal{D}_1$ and $(u, \hat{X}, \mathsf{pk}, b) \leftarrow \mathcal{D}_2$, where

- $X \leftarrow \tilde{\mathbb{F}}^{m\times 2}$ is a random m-by-2 matrix.

- pk is a public key generated by $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{Gen}(1^k)$, and $\hat{X} \leftarrow \mathrm{Enc}_{\mathsf{pk}}(X)$.

- $b = L(\hat{X}, \mathsf{pk})$ is a leakage bit, where $L : \{0,1\}^* \to \{0,1\}$ is a leakage function, to be specified later.

- $x \leftarrow X$ and $u \leftarrow \tilde{\mathbb{F}}^m$.

To this end, the distinguisher $\mathcal{A}$, upon receiving $(w, \hat{X}, \mathsf{pk}, b)$, distributed according to $\mathcal{D}_1$ or $\mathcal{D}_2$, tries to simulate the game G_{t-1} or G_t, respectively, by embedding the received distribution $(w, \hat{X}, \mathsf{pk}, b)$ in the game. At a high level, we let $\mathcal{A}$ embed the distribution in the Reveal protocol at time t (if executed), where $(\hat{X}, \mathsf{pk})$ is (part of) the delegator's message, and b is the *decision bit* indicating whether the delegator accepts or rejects in the Reveal protocol. However, note that (the

original) D and (the cautious) $\tilde{\mathsf{D}}$ decide whether to accept in the Reveal protocol in a different way, and in particular, D's decision depends on the secret point z.

Therefore, in addition to G_{t-1} and G_t, we consider an intermediate hybrid game G^*, which is a hybrid of G_{t-1} and G_t: It is the same as G_{t-1} and G_t except in the t'th protocol, where the delegator D^* sends the same message as D_{t-1} (which is less cautious since she uses z), but decide whether to accept in the same way as D_t (which is cautious). Namely,

- D^* sends $(\hat{S}_{s,z}, \mathsf{pk}) = (\hat{X}, \hat{s}, \mathsf{pk})$ to W^*, where $S_{s,z}$ is a random 2-dimensional affine space containing s and z, and $X \in \tilde{\mathbb{F}}^{m \times 2}$ is a random representation of $S_{s,z} - s$. Then D^* checks the polynomial p_2 returned by W^* (after decryption) by checking whether $p_2 = \mathrm{LDE}_{x^t} \circ S_{s,z}$.

Note that both D_{t-1} and D^* continue to use z in their secret state after time t.

We first argue that the hybrid games G_{t-1} and G^* are *statistically close*. Recall that the only difference between them is in the Reveal(s) protocol at time t, where D_{t-1} (resp., D^*) checks the polynomial p_2 sent by W^* (after decryption) by checking whether $p_2(\alpha_z) = \mathrm{LDE}_{x^t}(z)$ (resp., $p_2 = \mathrm{LDE}_{x^t} \circ S_{s,z}$). Note that D_{t-1} and D^*, being cautious before time t, use the (random) point z for the very first time at time t, so we can think of z as being generated from $z \leftarrow S_{s,z}$ *after* they send the message $(\hat{S}_{s,z}, \mathsf{pk})$.[10] It follows by the Schwartz-Zipple lemma that

$$\begin{aligned} &\Pr[\ \mathsf{D}_{t-1} \text{ and } \mathsf{D}^* \text{ make a different decision }] \\ &\quad \leq \quad \Pr[(p_2 \neq \mathrm{LDE}_{x^t} \circ S_{s,z}) \wedge (p_2(\alpha_z) = \mathrm{LDE}_{x^t}(z))] \\ &\quad \leq \quad O(d/|\tilde{\mathbb{F}}|) \leq \mathsf{ngl}(N), \end{aligned}$$

where $d = \mathrm{polylog}(N)$ is the total degree of LDE_{x^t}. Since this is the only difference between G_{t-1} and G^*, the statistical distance of the two hybrid is $\mathsf{ngl}(N)$.

Now, since G_{t-1} and G^* are statistically close, Equation (6.6) implies

$$\Pr[\mathsf{W}^* \text{ succeeds in } (\mathsf{G}^*)^{\mathsf{W}^*}] - \Pr[\mathsf{W}^* \text{ succeeds in } \mathsf{G}_t^{\mathsf{W}^*}] \geq (\varepsilon/N) - \mathsf{ngl}. \tag{6.7}$$

This time, we are able to use the noticeable gap between the success probability of W^* in G^* and G_t in Equation (6.7), to contradict Lemma 22.

[10] More precisely, first choosing a random $z \leftarrow \tilde{\mathbb{F}}^m$ and then choosing a random $S_{s,z}$ containing z (and s) is equivalent to first choosing a random S containing s, and then choosing $z \leftarrow S$.

Let us take a close look at the difference between G^* and G_t. Again, the only difference is in the Reveal(s) protocol at time t, where D^* (resp., D_t) sends $(\hat{S}_{s,z}, \mathsf{pk})$ (resp., $(\hat{S}_s, \mathsf{pk})$) to W^*. Note that, however, the distribution of both $S_{s,z}$ and S_s are actually the same, both being a random 2-dimensional affine space containing s. The real difference is that, after time t, $\tilde{\mathsf{D}}_{t-1}$ (resp., D_t) uses $z \leftarrow S_{s,z}$ (resp., $z \leftarrow \tilde{\mathbb{F}}^m$), in her secret state. Noting that this is exactly the difference between $(z, \hat{X}, \mathsf{pk}, b) \leftarrow \mathcal{D}_1$ and $(u, \hat{X}, \mathsf{pk}, b) \leftarrow \mathcal{D}_2$, we are ready to define the distinguisher $\mathcal{A}$, as follows, from which the choice of leakage bit $b = L(\hat{X}, \mathsf{pk})$ would become clear.

On input $(w, \hat{X}, \mathsf{pk}, b)$, distributed either according to $\mathcal{D}_1$ or according to $\mathcal{D}_2$, the distinguisher $\mathcal{A}$ does the following.

1. Simulate the interaction between W^* and $\tilde{\mathsf{D}}$ until the end of time $t-1$.
2. At time t, if there is a Compute execution, then simulate the Compute operation as follows.
   - Simulate for both parties all (sub-)protocols except the Reveal(s) protocol. This includes choosing $s \in \tilde{\mathbb{F}}^m$.
   - For the Reveal(s) protocol, simulate the delegator's message by $(\hat{X}, \hat{s}, \mathsf{pk})$, where $(\hat{X}, \mathsf{pk})$ is part of its input, and $\hat{s} \leftarrow \mathrm{Enc}_{\mathsf{pk}}(s)$. Then simulate W^*'s message. Finally, use the bit b (which is part of $\mathcal{A}$'s input) as the decision bit of the delegator. (Note that $\mathcal{A}$ does not have a secret key sk and cannot compute the delegator's decision bit efficiently.)
3. Compute $\sigma_t = (w + s, \mathrm{LDE}_{x^t}(w + s))$ and continue to simulate the interaction between W^* and D after time t using σ_t as D's secret state.
4. Output 1 if and only if W^* ever cheats successfully during the interaction.

We define the leakage function $L(\hat{X}, \mathsf{pk})$ to be the decision of D^* and D_t in the Reveal(s) protocol at time t, if executed, and 0 otherwise. More precisely, let $S = X + s$; $L(\hat{X}, \mathsf{pk}) = 1$ if and only if $p_2 = \mathrm{LDE}_{x^t} \circ S$. Note that L is a randomized function, which uses its randomness to simulate the game until the end of time t, and determine the decision of the delegator in the Reveal(s) protocol. Also note that the secret key sk is missing from $(\hat{X}, \mathsf{pk}, r)$, so L cannot be computed in poly(N) time.

Note that when $(z, \hat{X}, \mathsf{pk}, b) \leftarrow \mathcal{D}_1$, the distinguisher $\mathcal{A}$ perfectly simulates G^*, and when

$(u, \hat{X}, \mathsf{pk}, b) \leftarrow \mathcal{D}_2$, the distinguisher A perfectly simulates G_t. Therefore, Equation (6.7) implies

$$\Pr[\mathcal{A}(z, \hat{X}, \mathsf{pk}, b) = 1] - \Pr[\mathcal{A}(u, \hat{X}, \mathsf{pk}, b) = 1] \geq \varepsilon/N - \mathsf{ngl}(N),$$

contradicting Lemma 22. This completes the proof. ■

Chapter 7

Interactive Delegation of Any Efficient Computation

In this section, we construct memory and streaming delegation schemes based on universal arguments of Barak and Goldreich [BG02]. This allows the delegator to delegate computation for all of **P** rather than **NC**, at the price that the $\mathsf{Compute}(f)$ protocol becomes interactive with 4 rounds of message exchanges.

Recall that when we constructed our memory and streaming delegation schemes in Sections 5.2 and 6.2, the key property we need from the GKR protocol is that the verifier does not need to read the entire input, but rather only needs to access a few random points in the low-degree extension of the input. The main observation is that, the same property also holds for universal arguments, when the underlying PCP is substituted by an efficient PCP of Proximity (PCPP), a notion introduced by Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan [BSGH$^+$05] and Dinur and Reingold [DR06]. (We remark that this observation has been made independently by Cormode, Thaler, and Yi [CTY10] in the context of "streaming interactive proofs.") Therefore, we can use universal arguments with PCPP to construct memory and streaming delegation schemes for any (efficient) computation. In fact, the construction becomes simpler. Moreover, for memory delegation, we can avoid the use of poly-log PIR schemes, and only require the existence of collision-resistant hash functions.

Formally, we obtain the following theorems. For the sake of simplicity, we state the theorems for delegating computation of polynomial time Turing machines. The theorems extend readily to

non-uniform Turing machines, where the running time of the delegator and the worker in both the offline and the online stage depends polynomially on the length of the non-uniform advice.

Theorem 48 (Interactive Memory Delegation) *Let k be a security parameter, and let $\mathcal{F}$ be the class of all functions computable by polynomial time Turning machines. Assume the existence of collision-resistance hash functions. Then there exists a memory delegation scheme $\mathsf{sDel}_{\mathcal{F}}$ for $\mathcal{F}$ with the following properties.*

- *The scheme has perfect completeness and negligible (reusable) soundness error.*
- *The delegator and worker are efficient in the offline stage; i.e., both the delegator and the worker run in time $\mathrm{poly}(k, n)$, where n is the size of the memory.*
- *The worker is efficient in the online stage. More specifically, it runs in time $\mathrm{poly}(k, T(n))$ during each $\mathsf{Compute}(f)$ and $\mathsf{Update}(f)$ operation, where $T(n)$ is a time bound of the delegation function f on inputs of length n. The delegator runs in time $\mathrm{poly}(k, \log T)$ during each $\mathsf{Compute}(f)$ and $\mathsf{Update}(f)$ operation.*
- *Both $\mathsf{Compute}(f)$ and $\mathsf{Update}(f)$ consist of 4 rounds of message exchanges.*

Theorem 49 (Interactive Streaming Delegation) *Let k be a security parameter, and let N be a parameter bounding the maximum length of the stream. Let $\mathcal{F}$ be the class of all functions computable by polynomial time Turning machines. Assume the existence of a fully-homomorphic encryption scheme secure against $\mathrm{poly}(N)$-size adversaries. Then there exists a streaming delegation scheme $\mathsf{sDel}_{\mathcal{F}}$ for $\mathcal{F}$ with the following properties.*

- *$\mathsf{sDel}_{\mathcal{F}}$ has perfect completeness and negligible (reusable) soundness error.*
- *D updates her secret state in time $\mathrm{polylog}(N)$, per data item.*
- *In the delegation protocol, when delegating a function $f \in \mathcal{F}$ computable in time $T(n)$, the delegator D runs in time $\mathrm{poly}(k, \log N, \log T)$, and the worker W runs in time $\mathrm{poly}(k, \log N, T(n))$, where n is the length of the stream.*
- *The delegation protocol $\mathsf{Compute}(f)$ consists of 4 rounds of message exchanges.*

In Section 7.1, we present some necessary preliminaries, where we briefly review how to construct a (standard) delegation scheme using universal arguments and define PCP of Proximity. We then present a (standard) delegation scheme with the key property we need in Section 7.2. Finally, we construct the memory and streaming delegation schemes described in Theorem 48 and 49 in Sections 7.3 and 7.4, respectively.

7.1 Preliminaries

Universal Arguments

In this section, we briefly review how to delegate computation using universal arguments [BG02], as presented implicitly in [CKV10, Section 9].

Let k be the security parameter. To delegate the computation of a uniform function f (specified by a Turing machine M with a time bound T) on input $x \in \{0,1\}^n$,[1] the delegator D sends f, x to the worker W, who returns the answer $y = f(x)$ to D. Then they engage in a universal argument, where W proves to D that indeed $y = f(x)$. More specifically, W proves to D that (M, x, y, T) is in the following language L^{uni}.

$$L^{\text{uni}} \triangleq \{(M, x, y, T) : M \text{ is a Turing machine s.t. } M(x) \text{ outputs } y \text{ in} \leq T \text{ steps}\}.$$

In more detail, in the universal argument, the prover commits to a PCP proof π using a tree commitment $T_h(\pi)$ with the hash function $h \leftarrow \mathcal{H}$ chosen by the verifier. Then the verifier plays the role of a PCP verifier, with the prover answering her PCP queries by revealing the corresponding bits π_i's in the commitment $T_h(\pi)$.

The universal argument consists of 4 rounds of message exchanges, so the delegation protocol as described above requires 6 rounds of message exchanges. However, as argued in [CKV10], the first two rounds can be parallelized with the first two rounds of the universal argument, which yields a 4-round delegation protocol.

Clearly, the complexity of the delegation scheme depends on the complexity of the universal argument, which depends on the underlying PCPs. We note that the delegator needs to run in time $\Omega(n)$ since the verification of the underlying PCP proof π using standard PCPs (as opposed to PCPs

[1] Assume that $T \geq n$.

of proximity) depends on the whole input (M, x, y, T).

In the following theorem, we state the delegation scheme obtained by using the universal argument of [BG02].

Theorem 50 ([BG02]) [2] *Let k be the security parameter. Let n denote the input length, and let T be a time bound such that $T \geq n \geq k$. Let $\mathcal{F}$ be the family of boolean functions computable by time T Turing machines. Assume the existence of collision resistant hash functions secure against* $\mathrm{poly}(T)$*-time adversaries. Then, there exists a delegation protocol for $\mathcal{F}$ with the following properties.*

1. *The protocol has perfect completeness and negligible soundness error.*
2. *The worker runs in time* $\mathrm{poly}(T)$*, and the delegator runs in time* $\mathrm{poly}(n, \log T)$*.*
3. *The protocol consists of four messages, with communication complexity* $\mathrm{poly}(k, \log T)$*. Moreover, the protocol is public-coin.*

PCPs of Proximity

In this section, we present necessary preliminaries on PCPs of proximity and state the PCPP theorem of [BSGH$^+$05].

Definition 51 (Pair-language) *A* pair-language *L is simply a subset of the set of string pairs $L \subseteq \{0,1\}^* \times \{0,1\}^*$. For every $a \in \{0,1\}^*$, we denote $L_a = \{b \in \{0,1\}^* : (a,b) \in L\}$. We usually denote $\ell = |a|$ and $K = |b|$.*

The reader can think of a as a Turing machine M, and b as an input encoding, and $(a,b) \in L$ iff M accepts the input encoded by b within some time bound.

Definition 52 (PCPP **verifier [BSGH$^+$05])** *Let $r, q : \mathbb{N} \to \mathbb{N}$ and $t : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. An (r,q,t)-*PCPP verifier *V is a probabilistic oracle machine with the following structure.*

- *V receives as input a string $a \in \{0,1\}^*$ and a number $K \in \mathbb{N}$ (in binary), and has oracle access to two strings $b \in \{0,1\}^K$, and $\pi \in \{0,1\}^*$.*
- *V uses at most $r(|a|+K)$ coins, makes at most $q(|a|+K)$ non-adaptive queries to the two oracles (in total), runs in at most $t(|a|, K)$ time, and outputs a verdict bit, indicating her acceptance/rejection.*

[2] The theorem statement slightly differs from the one given in [BG02] and is tailored to our application.

The parameters r, q, t are the randomness*,* query*, and* time *complexity of V, respectively.*

The reader can think of r, q, t as being sub-linear.

Definition 53 (PCPP **for Pair-language [BSGH$^+$05])** *Let $r, q : \mathbb{N} \to \mathbb{N}$, $t : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, and $\varepsilon, \delta : \mathbb{N} \to [0,1]$. A pair-language $L \subseteq \{0,1\}^* \times \{0,1\}^*$ is in $\mathrm{PCPP}_{\varepsilon,\delta}[r, q, t]$ if there exists an (r, q, t)-*PCPP *verifier V with the following properties.*

- **(Completeness)** *If $(a, b) \in L$, then there exists a* PCPP *proof $\pi \in \{0,1\}^*$ such that $V^{b,\pi}(a, |b|)$ accepts with probability* 1*.*
- **(Soundness)** *If (a, b) is such that b is $\delta(|a| + |b|)$-far from $L_a \cap \{0,1\}^{|b|}$,*[3] *then for every $\pi \in \{0,1\}^*$, it holds that*

$$\Pr[V^{b,\pi}(a, |b|) = 1] \leq \varepsilon.$$

The parameter δ is called the proximity *parameter.*

Theorem 54 (Efficient PCPP **for Pair-language (Theorem 2.5 of [BSGH$^+$05]))** *Let $T : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function, and let $L = \{(a, b)\}$ be a pair-language. If L can be decided in time T,*[4] *then $L \in \mathrm{PCPP}_{1/2,\delta}[r, q, t]$ with*

- *proximity parameter $\delta = 1/\mathrm{polylog}(T)$,*
- *randomness complexity $r = \log_2 T + O(\log \log T)$,*
- *query complexity $q = \mathrm{polylog}(T)$, and*
- *verification time complexity $t(\ell, K) = \mathrm{poly}(\ell, \log K, \log T)$, where $\ell \triangleq |a|, K \triangleq |b|$, and $T = T(\ell + K)$.*

Furthermore, the PCPP *proof π for inputs in L (that makes V accepts) can be computed in time $\mathrm{poly}(T)$, and has length at most $q \cdot 2^r = T \cdot \mathrm{polylog}(T)$.*

We mention that the result of [BSGH$^+$05] holds for languages decidable in non-deterministic time T, but we are only interested in deterministic languages for the purpose of delegating computations.

[3] A string b is *δ-far* from a set $S \subset \{0,1\}^{|b|}$ if for every $b' \in S$, the relative Hamming distance $\Delta(b, b') \geq \delta$, where the relative Hamming distance is defined by $\Delta(b, b') \triangleq |\{i : b_i \neq b'_i\}|/|b|$.

[4] L can be decided in time T if there exists a Turing machine M such that for every input $(a, b) \in \{0,1\}^* \times \{0,1\}^*$, $M(a, b) = 1$ iff $(a, b) \in L$, and $M(a, b)$ runs in time $T(|a| + |b|)$.

We also mention that [BSGH$^+$05] does not discuss the complexity of constructing the PCPP proof, but the efficiency follows by a close inspection of their construction [Vad10].

We note that the soundness error $1/2$ can be reduced to $1/2^u$ by running V with independent coins u times. This blows up the randomness, query, and time complexity of V by a (multiplicative) factor of u (but does not increase the proof length).

7.2 Delegation Scheme using Universal Arguments with PCPP

In this section, we present a 4-round (standard) delegation scheme for any (efficient) computation, that possesses the same key property we need from the GKR protocol for memory and streaming delegation. Namely, the delegation protocol can be verified *very* efficiently (in sub-linear time in the input size), if the delegator has oracle access to the low-degree extension of the input x.

The starting point is the 4-round delegation scheme using universal argument [BG02] presented in Section 7.1. Recall that the delegator runs in time $\Omega(n)$ since the verification of the underlying PCP proof π depends on the whole input (M, x, y, T). To make the delegator run in sub-linear time in n, we substitute the underlying PCP for the language L^{uni} by the efficient PCPP of [BSGH$^+$05] from Theorem 54 for the following pair-language.

$$L^{\text{uni}-\text{pair}} \triangleq \{((M, y, T), \text{LDE}_x^{\mathbb{F},\mathbb{H},m}) : M \text{ is a Turing machine s.t. } M(x) \text{ outputs } y \text{ in} \leq T \text{ steps}\},$$

where the low-degree extension $\text{LDE}_x^{\mathbb{F},\mathbb{H},m}$ is a Reed-Muller code of $x \in \{0,1\}^n$ with

$$|\mathbb{H}| = \log n,\ m = \theta\left(\frac{\log n}{\log\log n}\right),\ |\mathbb{F}| = \log^2 n.$$

Since in Section 7.1, a language L is always defined to contain bit strings, we think of the codeword $\text{LDE}_x^{\mathbb{F},\mathbb{H},m}$ as a bit string, parsing every field element in $\mathbb{F}$ as $\log|\mathbb{F}|$ bits. Thus, the length of the codeword is $|\mathbb{F}|^m \cdot \log|\mathbb{F}| = \tilde{O}(n^2)$. The parameters $\mathbb{H}, \mathbb{F}, m$ are chosen so that the codewords have sufficient relative Hamming distance to each other, for establishing the soundness property (we discuss this later).

Note that there exists a Turing machine $\mathcal{M}$ that on input $((M, y, T), \tilde{x})$, decides whether $((M, y, T), \tilde{x}) \in L^{\text{uni}-\text{pair}}$ in time $\text{poly}(n, T)$ as follows: check if $\tilde{x}$ is a valid codeword (i.e., $\tilde{x} = \text{LDE}_x$ for some x), decode $\tilde{x}$ to obtain x, simulate $M(x)$ for T steps, and check if M outputs y. Hence, Theorem 54

implies the existence of a PCPP system Π_{pcpp} for $L^{\text{uni}-\text{pair}}$ such that on instance $((M, y, T), \text{LDE}_x)$, the PCPP verifier $V^{LDE_x,\pi}(M, y, T; |\text{LDE}_x|)$ runs in time polylog(T).

Therefore, when the underlying PCP of the delegation scheme in Section 7.1 is replaced by this PCPP system Π_{pcpp},[5] and when the delegator D is given oracle access to the low-degree extension LDE_x of x, D runs in time poly$(k, \log T)$, and makes at most polylog(T) queries to the oracle LDE_x. Furthermore, the queries to LDE_x depend only on D's coin tosses. On the other hand, the worker W runs in time poly(k, T), since the PCPP proof π can be computed in time poly(T). For the sake of completeness, a formal description of the modified protocol can be found in Figure 7.1.

$\mathsf{Del} = \langle \mathsf{D}, \mathsf{W} \rangle$:
D delegates the computation of a uniform function f, specified by a Turing machine M and a time bound T, on input $x \in \{0,1\}^n$ to W.

- D has (M, T) and x as the input.

1. D sends (M, T) and x to W.
2. W computes and sends $y = M(x)$ to D.
3. W and D engage in a universal argument with PCPP, where W proves to D that $((M, y, T), \text{LDE}_x) \in L^{\text{uni}-\text{pair}}$.
   - D sends a collision-resistant hash function $h \leftarrow \mathcal{H}$ to W.
   - W computes a PCPP proof π of the statement $((M, y, T), \text{Enc}(x)) \in L^{\text{uni}-\text{pair}}$, and sends the tree commitment $T_h(\pi)$ to D.
   - D runs the PCPP verifier V to generate a PCPP query q and sends it to W.
   - W reveals the bits π_i's queried by q from the commitment $T_h(\pi)$ to D.
   - D checks if π_i's are revealed validly, and runs the PCPP verifier $V^{\text{LDE}_x,\pi}(M, y, T; |\text{LDE}_x|)$.
4. D accepts $y = f(x)$ if D accepts in the universal argument.

Figure 7.1: A (standard) delegation protocol Del for any (efficient) computation.

We briefly check the completeness and soundness of Del. The completeness follows by the completeness of the PCPP. For the soundness, note that to convince the delegator D of an incorrect answer $y \neq f(x)$, an adversarial worker W^* needs to make D accept $((M, y, T), \text{LDE}_x) \notin L^{\text{uni}-\text{pair}}$ in the universal argument. Note that by the choice of parameters $\mathbb{F}, \mathbb{H}, m$ and Schwartz-Zippel Lemma,

[5] Namely, to prove $f(x) = y$, instead of proving $(M, x, y, T) \in L^{\text{uni}}$ using PCP, the worker W proves to D that $((M, y, T), \text{LDE}_x) \in L^{\text{uni}-\text{pair}}$ using universal argument with PCPPs.

the relative Hamming distance between any two codewords $\mathrm{LDE}_x, \mathrm{LDE}_{x'}$ is at least

$$\frac{\mathbb{F} - m \cdot |\mathbb{H}|}{|\mathbb{F}|} \cdot \frac{1}{\log |\mathbb{F}|} = \Omega\left(\frac{1}{\log \log n}\right) \geq \frac{1}{\mathrm{polylog}(T)}.$$

Hence, the soundness property of the PCPP implies that for every $((M, y, T), \mathrm{LDE}_x) \notin L^{\mathrm{uni-pair}}$ and for every $\pi \in \{0,1\}^*$,

$$\Pr[V^{\mathrm{LDE}_x, \pi}(M, y, T; |\mathrm{LDE}_x|) = 1] \leq 1/2.$$

Since the soundness of the universal argument follows by the security of collision-resistance hash functions and the soundness of the underlying PCP / PCPP, the delegator D would accept $y \neq f(x)$ with probability at most $1/2+\mathsf{ngl}$. Namely, the delegation protocol has soundness error $1/2+\mathsf{ngl}$. The soundness error can be reduced to negligible if the soundness of the underlying PCPP is negligible, which as mentioned, can be achieved by repeating the PCPP verifier with fresh randomness u times for some u satisfying $1/2^u \leq \mathsf{ngl}$.

As in Section 7.1, the protocol Del, as defined in Figure 7.1, consists of 6 message exchanges. However, note that the first message of D in the universal argument does not depend on the statement $((M, y, T), \mathrm{LDE}_x)$, and hence the worker's first message can be delayed to be sent together with the first prover's message of the universal argument, which yields a 4-message delegation protocol.

We summarize the properties of Del in the following theorem.

Theorem 55 (Interactive Delegation Scheme for Any (Efficient) Computation) *Let k be the security parameter. Let n denote the input length, and let T be a time bound such that $T \geq n \geq k$. Let $\mathcal{F}$ be the family of boolean functions computable by time T Turing machines. Assume the existence of collision resistant hash functions secure against* $\mathrm{poly}(T)$*-time adversaries. Then, there exists a delegation protocol for $\mathcal{F}$ with the following properties.*

1. *The protocol has perfect completeness and negligible soundness error.*
2. *The worker runs in time* $\mathrm{poly}(T)$*, and the delegator runs in time* $\mathrm{poly}(n, \log T)$*.*
3. *The protocol consists of four messages, with communication complexity* $\mathrm{poly}(k, \log T)$*. Moreover, the protocol is public-coin.*
4. *If the delegator is given oracle access to the low-degree extension of x, rather than being given*

the input x itself, and if the worker is given x as an input, then she runs in time $\mathrm{poly}(k, \log T)$*, and the protocol still has all the properties described above, for the following choice of parameters $\mathbb{H}, \mathbb{F}, m$ of the low-degree extension.*

$$|\mathbb{H}| = \log n,\ m = \theta\left(\frac{\log n}{\log\log n}\right),\ |\mathbb{F}| = \log^2 n.$$

Moreover, the delegator queries the low-degree extension of x at $\mathrm{polylog}(T)$ *points, depending only on her coin tosses.*

Remark 56 We remark that the parameters $\mathbb{F}, \mathbb{H}, m$ are chosen so that LDE_x is a Reed-Muller code with good rate and minimum distance. Let $N > n$ be a parameter. We can also set $\mathbb{F}, \mathbb{H}, m$ by

$$|\mathbb{H}| = \log N,\ m = \left\lceil\frac{\log n}{\log\log N}\right\rceil,\ |\mathbb{F}| = \log^2 N,$$

Then LDE_x has length at most $\mathrm{poly}(n, \log N)$ and (relative) minimum distance at least $\Omega(1/\log\log N)$. One can verify that the delegation scheme in Figure 7.1 is also sound with this setting of parameters (provided that $T \geq \log N$, which can be assumed without loss of generality by padding dummy steps), and the runtime of the delegator and the work are $\mathrm{poly}(k, \log N, \log T)$ and $\mathrm{poly}(k, \log N, T)$, respectively. This will be useful for our streaming delegation scheme presented in Section 7.4.

7.3 Memory Delegation Scheme Based on Theorem 55

In this section, we outline how to construct the memory delegation scheme (Theorem 48) using the above delegation scheme (Theorem 55). Let $x \in \{0,1\}^n$ be the memory being delegated. We observe that the delegator in Theorem 55 is efficient ($\mathrm{poly}(k, \log T)$) if she is given the oracle access to the low-degree extension of x with respect to some parameters $|\mathbb{F}| = \log^2 n$, $|\mathbb{H}| = \log n$, and $m = \theta(\log n/\log\log n)$. Thus, we can use similar techniques to the once in Section 5.2.2 where the delegator can verify points on the low-degree extension oracle using a tree commitment.

The Construction.

- **Offline Phase.** The delegator chooses parameters $\mathbb{F}, \mathbb{H}, m$ with $|\mathbb{F}| = \log^2 n$, $|\mathbb{H}| = \log n$, and $m = \theta(\log n/\log\log n)$, and a random function h from a collision resistant hash family $\mathcal{H}$. Then

the delegator computes the root of the Merkle tree of $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}$, namely $\sigma = T_h(\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m})$, and saves (h, σ) as a short certificate of x.

- **Online Plase.**
  - $\mathsf{Compute}(f)$: The worker and the delegator run the delegation scheme given by Theorem 55. In order to verify, the delegator needs to access $\mathrm{poly}\log T$ points in $\mathrm{LDE}_x^{\mathbb{F},\mathbb{H},m}$, and she can achieve this task by asking the worker to reveal the augmented paths corresponding to the Merkle tree. She accepts if and only if all the openings are accepted, and their corresponding values together with the answers to the PCPP queries are accepted in the verification of the delegation scheme.
  - $\mathsf{Update}(g)$: The delegator chooses a fresh collision resistant hash function $h' \leftarrow \mathcal{H}$, then the delegator and the worker run $\mathsf{Compute}(T_{h'}(\mathrm{LDE}_{g(x)}))$. Note that $T_{h'}(\mathrm{LDE}_{g(x)})$ is polynomial time computable (in x) given g. Finally, the worker replaces the memory x with $g(x)$.

Putting it together, we obtain Theorem 48. The proof is very similar to the proof of Theorem 35 in Section 5.2.3, and therefore is omitted. In what follows, we give a very high level overview of the proof.

Proof Overview of Theorem 48.

- The completeness follows from those of the delegation scheme from Theorem 55 and the tree commitment. The soundness can be proved in a similar way to the proof in Section 5.2.3.
- Both parties are efficient in the offline stage, i.e. run in time $\mathrm{poly}(k, n)$ since the computation of the root of the Merkle tree takes $\mathrm{poly}(k, |\mathbb{F}|^m) = \mathrm{poly}(k, n)$ time.
- In the online phase during each $\mathsf{Compute}(f)$ operation, the worker runs in time $\mathrm{poly}(k, T)$, and the delegator runs in time $\mathrm{poly}(k, \log T)$, where T is the running time of f. This follows from the efficiency of the delegation scheme from Theorem 55 and from the fact that only $\mathrm{poly}\log T$ leaves of the tree commitment need to be revealed.
- The protocol has 4 rounds of message exchanges, by running the delegation scheme from Theorem 55 and the Reveal protocols in parallel. Specifically, in the first two messages, the

delegator sends a random hash function and the worker uses it to commit to a PCPP proof. Then in the third message, the delegator queries both the PCPP proof and points on the low degree extension of the memory. The worker answers the corresponding queries in the fourth message. An illustration of the protocol can be found in Figure 7.2.

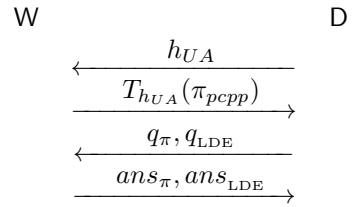

Figure 7.2: $\mathsf{Compute}(f)$ protocol of our interactive memory delegation scheme: h_{UA} is a collision resistant hash function chosen by the delegator in the universal argument. $T_{h_{UA}}(\pi_{pcpp})$ is the tree commitment of the PCPP proof π_{pcpp}. q_π and q_{LDE} denote the queries that the delegator makes to the PCPP proof and the Reed-Muller encoding LDE_x of the input x, respectively. ans_π and ans_{LDE} denote the corresponding answers together with the corresponding augmented paths.

7.4 Streaming Delegation Scheme Based on Theorem 55

In this section, we outline the construction of our streaming delegation scheme. Let N be the bound of the stream, and we choose the following parameters: $\tilde{\mathbb{F}}, \mathbb{F}, \mathbb{H}, m$ such that $|\mathbb{F}| = \log^2 N$, $|\mathbb{H}| = \log N$, $m = \theta(\log N / \log\log N)$, and $|\tilde{\mathbb{F}}| = N^{\log N}$ where $\tilde{\mathbb{F}}$ is an extension field of $\mathbb{F}$.

The Construction.

- **Generating and updating the secret state.** At time 0, the delegator keeps a secret $\sigma_0 = (z, 0)$, where $z \leftarrow \tilde{\mathbb{F}}^m$. At each time $t \in [N]$, when a data item $x_t \in \{0,1\}$ arrives, the delegator updates her secret state from $\sigma_{t-1} = (z, \mathrm{LDE}_{x^{t-1}}^{\tilde{\mathbb{F}},\mathbb{H},m}(z))$ to $\sigma_t = (z, \mathrm{LDE}_{x^t}^{\tilde{\mathbb{F}},\mathbb{H},m}(z))$, by using Proposition 6. (Recall that $x^t = (x_1, \ldots, x_t)$ denotes the entire data stream up until time t.)

- $\mathsf{Compute}(f, t)$**.** At any time t, when the delegator wants to execute $\mathsf{Compute}(f, t)$, both parties run the delegation scheme from Theorem 55 with a modified parameter setting

$$|\mathbb{H}| = \log N,\ m'' = \left\lceil \frac{\log n}{\log\log N} \right\rceil,\ |\mathbb{F}| = \log^2 N$$

as stated in Remark 56. Here in order to verify, the delegator needs to verify the value of $\mathrm{LDE}_{x^t}^{\mathbb{F},\mathbb{H},m''}$ on a few points $z_i'' \in \mathbb{F}^{m''}$. As argued in Section 6.2.1 (see Equation (6.3)), $\mathrm{LDE}_{x^t}^{\mathbb{F},\mathbb{H},m''}(z_i'') = \mathrm{LDE}_{x^t}^{\mathbb{F},\mathbb{H},m}(z_i)$ where $z_i = (0^{m-m''}, z_i)$. Hence, the delegator can instead verify the values $\mathrm{LDE}_{x^t}^{\mathbb{F},\mathbb{H},m}(z_i)$, which can be done by using a many-to-one protocol together with a Reveal protocol in exactly the same way as in Section 6.2.2 (see Steps 3 and 4 in Figure 6.4).

This gives us a streaming delegation scheme satisfying Theorem 49. As in Section 7.3, the proof is very similar to the proof of Theorem 44 in Section 6.2, and is therefore omitted.

Proof Overview of Theorem 49.

- The completeness follows from that of the delegation scheme from Theorem 55, the many-to-one protocol, and the Reveal protocol of the algebraic commitments (see Section 6.2 for details). The soundness can be proved in a similar way to the proof in Section 6.2.3.
- The delegator runs in time polylog$N$ to update her secret per data item.
- The worker runs in time $\mathrm{poly}(k, \log N, T)$ for the delegation of a time T computable function, and the delegator runs in time $\mathrm{poly}(k, \log N, \log T)$. This follows from the efficiency of the underlying delegation scheme from Theorem 55, the many-to-one protocol, and the Reveal protocol of the algebraic commitments.
- The protocol has 4 rounds of message exchanges, by running the the delegation scheme from Theorem 55, the many-to-one protocol, and the Reveal protocol in parallel, in a similar way to that of the memory delegation scheme described in Section 7.3. An illustration of the protocol can be found in Figure 7.3.

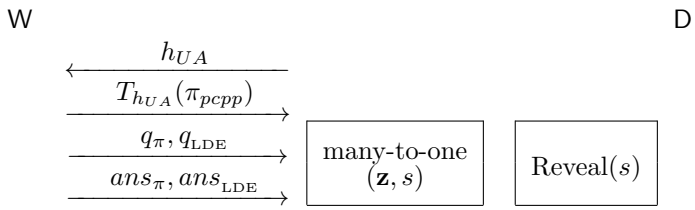

Figure 7.3: Compute(f) protocol of our interactive streaming delegation scheme, where $q_{\text{LDE}} = \mathbf{z} = \{z_i\}$ is the queries that the delegator makes to the Reed-Muller encoding LDE_x of the input x. In parallel to the universal argument, the delegator and the worker run a many-to-one protocol and a Reveal protocol in the same way as in Section 6.2.2 (see Steps 3 and 4 in Figure 6.4) to verify the values of LDE_x on points $\mathbf{z}$.

Part II

Tamper and Leakage Resilience in the Split-State Model

Chapter 8

Overview

We give an overview of our approach of achieving tamper and leakage resilient constructions. First we review a powerful cryptographic primitive – non-malleable code, which we use as our main building block. We present the high level concepts of the primitive and leave the overview of our new construction in Section 10.2. Then we present an overview of our security model and how we can achieve security against tampering and leakage attacks using non-malleable codes as building blocks.

8.1 Our building block: non-malleable codes

Non-malleable codes were defined by Dziembowski et al. [DPW10]. Let $\mathcal{E}nc$ be an encoding procedure and $\mathcal{D}ec$ be the corresponding decoding procedure. Consider the following tampering experiment [DPW10]: (1) A string s is encoded yielding a codeword $c = \mathcal{E}nc(s)$. (2) The codeword c is mauled by some function f to some $c^* = f(c)$. (3) The resulting codeword is decoded, resulting in $s^* = \mathcal{D}ec(c^*)$. $(\mathcal{E}nc, \mathcal{D}ec)$ constitutes a non-malleable code if tampering with c can produce only two possible outcomes: (1) f leaves c unchanged; (2) the decoded string s^* is unrelated to the original string s. Intuitively, this means that one cannot learn anything about the original string s by tampering with the codeword c.

It is clear [DPW10] that, without any restrictions on f, this notion of security is unattainable. For example, f could, on input c, decode it to s, and then compute $s^* = s + 1$ and then output $\mathcal{E}nc(s^*)$. Such an f demonstrates that no $(\mathcal{E}nc, \mathcal{D}ec)$ can satisfy this definition. However, for restricted classes

of functions, this definition can be instantiated.

Dziembowski et al. constructed non-malleable codes with respect to bit-wise tampering functions in the plain model, and with respect to split-state tampering functions in the random oracle model. They also show a compiler that uses non-malleable codes to secure any functionality against tampering attacks. In this paper, we improve their result in four ways: first, we construct a non-malleable code with respect to split-state tampering, in the CRS model (which is a significant improvement over the RO model). Second, our code has an additional property: it is leakage resilient. That is to say, for any constant $\varepsilon \in (0,1)$, any efficient shrinking split-state function $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{(1-\varepsilon)n} \times \{0,1\}^{(1-\varepsilon)n}$, $g(c)$ reveals no information about the s (where c is a codeword encoding s). Third, we prove that plugging in a leakage-resilient non-malleable code in the Dziembowski et al. compiler secures any functionality against tampering and leakage attacks at the same time. This gives a *randomized* secure implementation of any functionality. Fourth, we give another compiler that gives a *deterministic* secure implementation of any functionality where after initialization, the device (implementation) does not need access to a source of randomness.

8.2 Our continual tampering and leakage model

We consider the same tampering and leakage attacks as those of Liu and Lysyanskaya[LL10] and Kalai et al. [KKS11], which generalized the model of tampering-only [GLM+04, DPW10] and leakage-only [BKKV10, DHLAW10, LRW11, LLW11] attacks. (However, in this attack model we achieve stronger security, as discussed in the introduction.)

Let M be the memory of the device under attack. We view time as divided into discrete time periods, or rounds. In each round, the adversary A makes a leakage query g or a tampering query f; as a result, A obtains $g(M)$ or modifies the memory: $M := f(M)$. In this work, we consider both g, f to be split-state functions.

In this paper, we consider the simulation based security that generalized the Dziembowski et al. definition [DPW10]. On a high level, let the memory M be an encoded version of some secret s. Security means there exists a simulator who does not know s and only gets oracle access to the functionality $G(s,x)$, but can still respond to the adversary's attack queries in a way that is indistinguishable from the real game. This means that tampering and leakage attacks do not give the adversary more information than black box access to the functionality $G(s,x)$. This is captured

formally in Definition 67.

8.3 Our approach

We take a simple and natural approach given any leakage resilient non-malleable code. Let $G(s, x)$ be the functionality we want to secure, where s is some secret state and x is the user input. Given such coding scheme, our compiler takes G as input, outputs $G'(\mathcal{E}nc(s), x)$, where G' gets an encoded version of the state s, emulates $G(s, x)$ and re-encodes the new state at the end of each round. Then we will argue that even if the adversary can get partial information or tamper with the encoded state in every round, the compiled construction is still secure. Intuitively, by leakage resilience, the encoding of s gives no information from partial leakage of the codeword, and by non-malleability, tampering with the codeword will result in some unrelated message. Thus the adversary cannot gain any extra power via leakage and tampering attacks.

Chapter 9

Definitions and Models

9.1 Definitions

In this section, we formally define the tools we need for the construction. We need robust NIZK, one-time leakage-resilient encryption scheme, and universal one-way hash functions. Moreover, our construction needs these tools to have some additional properties. We describe these properties here and will show that they are without loss of generality.

Definition 57 (Robust NIZK [DDO$^+$01]) *$\Pi = (\ell, \mathcal{P}, \mathcal{V}, \mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2))$ is a robust NIZK proof/argument for the language $\mathbf{L} \in \mathbf{NP}$ with witness relation W if ℓ is a polynomial, and $\mathcal{P}, \mathcal{V}, \mathcal{S} \in \mathrm{PPT}$, there exists a negligible function $\mathsf{ngl}(\cdot)$ such that:*

- **(Completeness):** *For all $x \in \mathbf{L}$ of length k and all w such that $\mathrm{W}(x, w) = 1$, for all strings $\Sigma \in \{0,1\}^{\ell(k)}$, we have $\mathcal{V}(x, \mathcal{P}(x, w, \Sigma), \Sigma) = 1$.*

- **(Extractability):** *For all non-uniform PPT adversary A, we have*

$$\Pr\left[\begin{array}{r} (\Sigma, \tau) \leftarrow \mathcal{S}_1(1^k); (x, \pi) \leftarrow A^{\mathcal{S}_2(\cdot,\cdot,\Sigma,\tau)}(\Sigma); \\ w \leftarrow \mathrm{Ext}(\Sigma, \tau, x, \pi) : \\ (x, w) \in \mathrm{W} \vee (x, \pi) \in Q \vee \mathcal{V}(x, \pi, \Sigma) = 0 \end{array}\right] = 1 - \mathsf{ngl}(k)$$

  where Q denotes the successful statement-query pairs (x_i, p_i)'s that $\mathcal{S}_2$ has answered A.

- **(Multi-theorem Zero-Knowledge):** *For all non-uniform PPT adversary A, we have $|\Pr[X(k) =$*

$1] - \Pr[Y(k) = 1]| < \mathsf{ngl}(k)$ *where* X, Y *are binary random variables defined in the experiment below:*

$$X(k) = \left\{ \Sigma \leftarrow \{0,1\}^{\ell(k)}; X \leftarrow A^{\mathcal{P}(\cdot,\cdot,\Sigma)}(\Sigma) : X \right\};$$

$$Y(k) = \left\{ (\Sigma, \tau) \leftarrow \mathcal{S}_1(1^k); Y \leftarrow A^{\mathcal{S}_2(\cdot,\cdot,\Sigma,\tau)}(\Sigma) : Y \right\}.$$

Remark 58 We remark that in this paper, we assume a robust NIZK system that has an additional property that different statements must have different proofs. That is, suppose $\mathcal{V}(\Sigma, x, \pi)$ accepts, then $\mathcal{V}(\Sigma, x', \pi)$ must reject for all $x' \neq x$.

This property is not required by standard NIZK definitions, but can be achieved easily by appending the statement to its proof. In the construction of robust NIZK [DDO+01], if the underlying NIZK system has this property, then the transformed one has this property as well. Thus, we can assume this property without loss of generality.

Definition 59 (Universal One-way Hash Functions - UOWHF [HHR+10]) *A family of functions* $H_k = \{h_z : \{0,1\}^{n(k)} \to \{0,1\}^k\}_{z \in \{0,1\}^k}$ *is a universal one-way hash family if:*

- **(Efficient):** *given* $z \in \{0,1\}^k$*, and* $x \in \{0,1\}^{n(k)}$*, the value* $h_z(x)$ *can be computed in time* $\mathrm{poly}(k, n(k))$.
- **(Compressing):** *For all* k*,* $k \leq n(k)$.
- **(Universal One-way):** *For any non-uniform* PPT *adversary* A*, there exists a negligible function* $\mathsf{ngl}(\cdot)$*:*

$$\Pr\left[\begin{array}{l} x \leftarrow A(1^k); z \leftarrow \{0,1\}^k; x' \leftarrow A(1^k, z, x) : \\ x, x' \in \{0,1\}^{n(k)} \wedge x' \neq x \wedge h_z(x) = h_z(x') \end{array} \right] < \mathsf{ngl}(k).$$

Definition 60 (One-time Leakage Resilient Encryption [AGV09]) *Let* $\mathcal{E} = (\mathrm{KeyGen}, \mathrm{Encrypt}, \mathrm{Decrypt})$ *be an encryption scheme, and* $\mathcal{G}$ *be a set of functions. Let the random variable* $\mathsf{LE}_b(\mathcal{E}, A, k, \mathcal{G})$ *where* $b \in \{0,1\}$*,* $A = (A_1, A_2, A_3)$ *and* $k \in \mathbb{N}$ *denote the result of the following probabilistic experiment:*

$\mathsf{LE}_b(\mathcal{E}, A, k, \mathcal{G})$:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \text{KeyGen}(1^k)$.
- $g \leftarrow A_1(1^k, \mathsf{pk})$ *such that* g *is a leakage function in the class* $\mathcal{G}$.
- $(m_0, m_1, state_A) \leftarrow A_2(\mathsf{pk}, g(\mathsf{sk}))$ *s.t.* $|m_0| = |m_1|$.
- $c = \text{Encrypt}_{\mathsf{pk}}(m_b)$.
- *Output* $b' = A_3(c, state_A)$.

We say $\mathcal{E}$ *is semantically secure against* one-time leakage $\mathcal{G}$ *if* $\forall$ PPT *adversary* A*, the following two ensembles are computationally indistinguishable:*

$$\left\{\mathsf{LE}_0(\mathcal{E}, A, k, \mathcal{G})\right\}_{k\in\mathbb{N}} \approx_c \left\{\mathsf{LE}_1(\mathcal{E}, A, k, \mathcal{G})\right\}_{k\in\mathbb{N}}$$

Additional Properties. Our construction of LR-NM codes in Section 10 needs additional properties of the encryption scheme:

- Given a secret key sk, one can derive its corresponding public key pk deterministically and efficiently. This property is easy to achieve since we can just append public keys to secret keys.
- It is infeasible for non-uniform PPT adversaries that receive a random key pair $(\mathsf{pk}, \mathsf{sk})$ to output another valid key pair $(\mathsf{pk}, \mathsf{sk}')$ for some $\mathsf{sk}' \neq \mathsf{sk}$. This property is not guaranteed by standard definitions, but for leakage resilient encryption schemes, this is easy to achieve. We formalize this claim in the following lemma.

Lemma 61 *Let* $\mathcal{E} = (\text{KeyGen}, \text{Encrypt}, \text{Decrypt})$ *be a leakage resilient encryption scheme that allows* $t(k)$*-bit leakage for* $t(k) > k$*, and* $\mathcal{H}_k : \{h_z : \{0,1\}^{\text{poly}(k)} \to \{0,1\}^k\}_{s\in\{0,1\}^k}$ *be a family of universal one-way hash functions.*

Then there exists an encryption scheme $\mathcal{E}' = (\text{KeyGen}', \text{Encrypt}', \text{Decrypt}')$ *that is leakage resilient that allows* $(t-k)$*-bit leakage and has the following property: for all non-uniform* PPT *adversary* A*,*

$$\Pr_{(\mathsf{pk},\mathsf{sk})\leftarrow \mathrm{KeyGen}'(1^k)}[(\mathsf{sk}', \mathsf{pk}) \leftarrow A(\mathsf{sk}, \mathsf{pk}) : (\mathsf{sk}', \mathsf{pk}) \textit{ is a key pair and } \mathsf{sk}' \neq \mathsf{sk}] < \mathsf{ngl}(k).$$

Proof. [Sketch] The construction is as follows: $\mathrm{KeyGen}'(1^k)$: sample $z \leftarrow \{0,1\}^k$, and $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathrm{KeyGen}(1^k)$. Set $\mathsf{pk} = \mathsf{pk}_0 \circ z \circ h_s(\mathsf{sk}_0)$, and $\mathsf{sk} = \mathsf{sk}_0$.

The $\mathrm{Encrypt}'$ and $\mathrm{Decrypt}'$ follow directly from $\mathrm{Encrypt}, \mathrm{Decrypt}$. It is easy to see that, since it is safe to leak t bits of sk as the original cryptosystem, after publishing $h(\mathsf{sk})$ in the public key, it is still safe to leak $(t-k)$ bits. On the other hand, this additional property holds simply by the security of the universal one-way hash function and can be proved using a standard reduction.

■

In the rest of the paper, we will assume the encryption scheme has this property. Now we give an instantiation of one-time leakage resilient encryption scheme due to Naor-Segev[1]:

Theorem 62 ([NS09]) *Under the Decisional Diffie-Hellman assumption, for any polynomial $\ell(k)$, there exists an encryption scheme $\mathcal{E}$ that uses $\ell(k) + \omega(\log k)$ bits to represent its secret key and is semantically secure against one-time leakage $\mathcal{G}_\ell = \{$all efficient functions that have ℓ-bit output$\}$.*

9.2 Our Model

In this section, we define the function classes for split-state leakage and tampering attacks, $\mathcal{G}^{\mathsf{half}}$ and $\mathcal{F}^{\mathsf{half}}$, respectively. Then we define an adversary's interaction with a device that is vulnerable to such attacks. Finally, we give the definition of a compiler that transforms any cryptographic functionality $G(s,x)$ to a functionality $G'(s',x)$ that withstands these attacks.

Definition 63 *Define the following three function classes $\mathcal{G}_t, \mathcal{F}^{\mathsf{half}}, \mathcal{G}^{\mathsf{half}}_{t_1,t_2}$:*

- *Let $t \in \mathbb{N}$, and by $\mathcal{G}_t$ we denote the set of all polynomial-sized circuits that have output length t, i.e. $g : \{0,1\}^* \to \{0,1\}^t$.*
- *Let $\mathcal{F}^{\mathsf{half}}$ denote the set of length-preserving and polynomial-sized functions/circuits f that operate independently on each half of their inputs. I.e. $f : \{0,1\}^{2m} \to \{0,1\}^{2m} \in \mathcal{F}^{\mathsf{half}}$ if there*

[1] Actually the Naor-Segev scheme can tolerate more leakage up to $(1 - o(1)) \cdot |\mathsf{sk}|$, and the leakage function can even be computationally unbounded. In this work, this weaker version suffices for our purposes.

exist two polynomial-sized functions/circuits $f_1 : \{0,1\}^m \to \{0,1\}^m$, $f_2 : \{0,1\}^m \to \{0,1\}^m$ *such that for all* $x, y \in \{0,1\}^m$, $f(x,y) = f_1(x) \circ f_2(y)$.

- *Let* $t_1, t_2 \in \mathbb{N}$, *and we denote* $\mathcal{G}^{\mathsf{half}}_{t_1,t_2}$ *as the set of all polynomial-sized leakage functions that leak independently on each half of their inputs, i.e.* $g : \{0,1\}^{2m} \to \{0,1\}^{t_1+t_2} \in \mathcal{G}^{\mathsf{half}}_{t_1,t_2}$ *if there exist two polynomial-sized functions/circuits* $g_1 : \{0,1\}^m \to \{0,1\}^{t_1}$, $g_2 : \{0,1\}^m \to \{0,1\}^{t_2}$ *such that for all* $x, y \in \{0,1\}^m$, $g(x,y) = g_1(x) \circ g_2(y)$.

  We further denote $\mathcal{G}^{\mathsf{half}}_{t_1,\mathsf{all}}$ *as the case where* $g_1(x)$ *leaks* t_1 *bits, and* $g_2(y)$ *can leak all its input* y.

We remark that the security parameter k *with respect to which efficiency is measured is implicit in the definitions.*

Next, let us define an adversary's access to a functionality under tampering and leakage attacks. In addition to queries to the functionality itself (called $\mathsf{Execute}$ queries) an attacker has two more operations: he can cause the memory of the device to get tampered according to some function f, or he can learn some function g of the memory. Formally:

Definition 64 (Interactive Functionality Subject to Tampering and Leakage Attacks) *Let* $\langle G, s\rangle$ *be an interactive stateful system consisting of a public (perhaps randomized) functionality* $G : \{0,1\}^u \times \{0,1\}^k \to \{0,1\}^v \times \{0,1\}^k$ *and a secret initial state* $s \in \{0,1\}^k$. *We consider the following ways of interacting with the system:*

- $\mathsf{Execute}(x)$*: A user can provide the system with some query* $\mathsf{Execute}(x)$ *for* $x \in \{0,1\}^u$. *The system will compute* $(y, s_{\mathsf{new}}) \leftarrow G(s,x)$, *send the user* y, *and privately update its state to* s_{new}.
- $\mathsf{Tamper}(f)$*: the adversary can operate tampering attacks against the system, where the state* s *is replaced by* $f(s)$ *for some function* $f : \{0,1\}^k \to \{0,1\}^k$.
- $\mathsf{Leak}(g)$*: the adversary can obtain the information* $g(s)$ *of the state by querying* $\mathsf{Leak}(g)$.

Next, we define a compiler that compiles a functionality $\langle G, s\rangle$ into a hardware implementation $\langle G', s'\rangle$ that can withstand leakage and tampering attacks. A compiler will consist of two algorithms, one for compiling the circuit for G into another circuit, G'; the other algorithm is for compiling the memory, s, into s'. This compiler will be *correct*, that is to say, the resulting circuit and memory

will provide input/output functionality identical to the original circuit; it will also be tamper- and leakage-resilient in the following strong sense: there exists a simulator that, with oracle access to the original $\langle G, s\rangle$, will simulate the behavior of $\langle G', s'\rangle$ under tampering and leakage attacks. The following definitions formalize this:

Definition 65 *Let* CRS *be an algorithm that generates a common reference string, on input the security parameter* 1^k*. The algorithms* $(\mathsf{CircuitCompile}, \mathsf{MemCompile})$ *constitute a* correct *and* efficiency-preserving *compiler in the* $\mathsf{CRS}(1^k)$ *model if for all* $\Sigma \in \mathsf{CRS}(1^k)$*, for any* $\mathsf{Execute}$ *query* x*,* $\langle G', s'\rangle$*'s answer is distributed identically to* $\langle G, s\rangle$*'s answer, where* $G' = \mathsf{CircuitCompile}(\Sigma, G)$ *and* $s' \in \mathsf{MemCompile}(\Sigma, s)$*; moreover,* $\mathsf{CircuitCompile}$ *and* $\mathsf{MemCompile}$ *run in polynomial time and output* G' *and* s' *of size polynomial in the original circuit* G *and secret* s*.*

Note that this definition of the compiler ensures that the compiled functionality G' inherits all the security properties of the original functionality G. Also note that the compiler, as defined here, works separately on the functionality G and on the secret s, which means that it can be combined with another compiler that strengthens G' is some other way (for example, it can be combined with the compiler of Goldwasser and Rothblum [GR10]). This definition allows for both randomized and deterministic G'; as we discussed in the introduction, in general a deterministic circuit is more desirable.

Remark 66 Recall that G, and therefore G', are modeled as stateful functionalities. By convention, running $\mathsf{Execute}(\varepsilon)$ will cause them to update their states.

As defined above, in the face of the adversary's $\mathsf{Execute}$ queries, the compiled G' behaves identically to the original G. Next, we want to formalize the important property that whatever the adversary can learn from the compiled functionality G' using $\mathsf{Execute}$, Tamper and Leak queries, can be learned just from the $\mathsf{Execute}$ queries of the original functionality G.

We want the real experiment where the adversary interacts with the compiled functionality $\langle G', s'\rangle$ and issues $\mathsf{Execute}$, Tamper and Leak queries, to be indistinguishable from an experiment in which a simulator $\mathcal{S}im$ only has black-box access to the original functionality G with the secret state s (i.e. $\langle G, s\rangle$). More precisely, in every round, $\mathcal{S}im$ will get some tampering function f or leakage function g from A and then respond to them. In the end, the adversary halts and outputs its view. The simulator then may (potentially) output this view. Whatever view $\mathcal{S}im$ outputs needs

to be indistinguishable from the view A obtained in the real experiment. This captures the fact that the adversary's tampering and leakage attacks in the real experiment can be simulated by only accessing the functionality in a black-box way. Thus, these additional physical attacks do not give the adversary any additional power.

Definition 67 (Security Against $\mathcal{F}$ Tampering and $\mathcal{G}$ Leakage) *A compiler* (CircuitCompile, MemCompile) *yields an $\mathcal{F}$-$\mathcal{G}$ resilient hardened functionality in the CRS model if there exists a simulator $\mathcal{S}im$ such that for every efficient functionality $G \in$ PPT with k-bit state, and non-uniform* PPT *adversary A, and any state $s \in \{0,1\}^k$, the output of the following real experiment is indistinguishable from that of the following ideal experiment:*

Real Experiment $\mathsf{Real}(A, s)$**:** *Let $\Sigma \leftarrow \mathsf{CRS}(1^k)$ be a common reference string given to all parties. Let $G' \leftarrow \mathsf{CircuitCompile}(\Sigma, G)$, $s' \leftarrow \mathsf{MemCompile}(\Sigma, s)$. The adversary $A(\Sigma)$ interacts with the compiled functionality $\langle G', s' \rangle$ for arbitrarily many rounds where in each round:*

- *A runs* $\mathsf{Execute}(x)$ *for some $x \in \{0,1\}^u$, and receives the output y.*
- *A runs* $\mathsf{Tamper}(f)$ *for some $f \in \mathcal{F}$, and then the encoded state is replaced with $f(s')$.*
- *A runs* $\mathsf{Leak}(g)$*, and receives some $\ell = g(s')$ for some $g \in \mathcal{G}$, where s' is the current state. Then the system updates its memory by running* $\mathsf{Execute}(\varepsilon)$*, which will update the memory with a re-encoded version of the current state.*

Let $\text{view}_A = (state_A, x_1, y_1, \ell_1, x_2, y_2, \ell_2, \dots,)$ *denote the adversary's view where x_i's are the execute input queries, y_i's are their corresponding outputs, ℓ_i's are the leakage at each round i. In the end, the experiment outputs* (Σ, view_A).

Ideal Experiment $\mathsf{Ideal}(\mathcal{S}im, A, s)$**:** *$\mathcal{S}im$ first sets up a common reference string Σ, and $\mathcal{S}im^{A(\Sigma), \langle G, s \rangle}$ outputs $(\Sigma, \text{view}_{\mathcal{S}im}) = (\Sigma, (state_{\mathcal{S}im}, x_1, y_1, \ell_1, x_2, y_2, \ell_2, \dots))$, where (x_i, y_i, ℓ_i) is the input/output/leakage tuple simulated by $\mathcal{S}im$ with oracle access to $A, \langle G, s \rangle$.*

Note that we require that, in the real experiment, after each leakage query the device updates its memory. This is necessary, because otherwise the adversary could just keep issuing Leak query on the same memory content and, over time, could learn the memory bit by bit.

Also, note that, following Dziembowski et al. [DPW10] we require that each experiment faithfully record all the Execute queries. This is a way to capture the idea that the simulator cannot make

more queries than the adversary; as a result, an adversary in the real experiment (where he can tamper with the secret and get side information about it) learns the same amount about the secret as the simulator who makes the same queries (but does NOT get the additional tampering and leakage ability) in the ideal experiment.

Chapter 10

Main Tool – Leakage Resilient Non-Malleable Code

In this section, we present the definition of leakage resilient non-malleable codes (LR-NM codes), and our construction. We also extend the definition of Dziembowski et al. [DPW10] in two directions: we define a coding scheme in the CRS model, and we consider leakage resilience of a scheme. Also, our construction achieves the stronger version of non-malleability, so we present this version. For the normal non-malleability and the comparison, we refer curious readers to the paper [DPW10]. First we define a coding scheme in the plain model and in the CRS model.

10.1 Definitions

Definition 68 (Coding Scheme [DPW10]) *A (k, n) coding scheme consists of two algorithms: an encoding algorithm $\mathcal{E}nc : \{0,1\}^k \to \{0,1\}^n$, and decoding algorithm $\mathcal{D}ec : \{0,1\}^n \to \{0,1\}^k \cup \{\perp\}$ such that, for each $s \in \{0,1\}^k$, $\Pr[\mathcal{D}ec(\mathcal{E}nc(s)) = s] = 1$, over the randomness of the encoding/decoding algorithms.*

Definition 69 (Coding Scheme in the Common Reference String Model) *Let k be the security parameter, and $\mathcal{I}nit(1^k)$ be an efficient randomized algorithm that publishes a common reference string (CRS) $\Sigma \in \{0,1\}^{\mathrm{poly}(k)}$. We say $\mathcal{C} = (\mathcal{I}nit, \mathcal{E}nc, \mathcal{D}ec)$ is a coding scheme in the CRS model if for every k, $(\mathcal{E}nc(1^k, \Sigma, \cdot), \mathcal{D}ec(1^k, \Sigma, \cdot))$ is a $(k, n(k))$ coding scheme for some polynomial*

$n(k)$.

For simplicity, we will omit the security parameter and write $\mathcal{E}nc(\Sigma,\cdot), \mathcal{D}ec(\Sigma,\cdot)$ for the case in the CRS model.

Now we define the two properties of coding schemes: non-malleability and leakage resilience.

Definition 70 (Strong Non-malleability [DPW10]) *Let $\mathcal{F}$ be some family of functions. For each function $f \in \mathcal{F}$, and $s \in \{0,1\}^k$, define the tampering experiment*

$$\mathsf{Tamper}_s^f \stackrel{\text{def}}{=} \left\{ \begin{array}{c} c \leftarrow \mathcal{E}nc(s), \tilde{c} = f(c), \tilde{s} = \mathcal{D}ec(\tilde{c}) \\ \textit{Output} : \mathsf{same}^* \textit{ if } \tilde{c} = c, \textit{ and } \tilde{s} \textit{ otherwise.} \end{array} \right\}$$

The randomness of this experiment comes from the randomness of the encoding and decoding algorithms. We say that a coding scheme $(\mathcal{E}nc, \mathcal{D}ec)$ is strong non-malleable with respect to the function family $\mathcal{F}$ if for any $s_0, s_1 \in \{0,1\}^k$ and for each $f \in \mathcal{F}$, we have:

$$\{\mathsf{Tamper}_{s_0}^f\}_{k\in\mathbb{N}} \approx \{\mathsf{Tamper}_{s_1}^f\}_{k\in\mathbb{N}}$$

where $\approx$ can refer to statistical or computational indistinguishability.

When we refer to non-malleable codes in the common reference string model, for any CRS Σ we define

$$\mathsf{Tamper}_s^{f,\Sigma} \stackrel{\text{def}}{=} \left\{ \begin{array}{c} c \leftarrow \mathcal{E}nc(\Sigma, s), \tilde{c} = f^{\Sigma}(c), \tilde{s} = \mathcal{D}ec(\Sigma, \tilde{c}) \\ \textit{Output} : \ \mathsf{same}^* \textit{ if } \tilde{c} = c, \textit{ and } \tilde{s} \textit{ otherwise.} \end{array} \right\}.$$

We say the coding scheme $(\mathcal{I}nit, \mathcal{E}nc, \mathcal{D}ec)$ is strong non-malleable if we have $\{(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma})\}_{k\in\mathbb{N}} \approx \{(\Sigma, \mathsf{Tamper}_{s_1}^{f,\Sigma})\}_{k\in\mathbb{N}}$ where $\Sigma \leftarrow \mathcal{I}nit(1^k)$, any $s_0, s_1 \in \{0,1\}^k$, and $f \in \mathcal{F}$.

Definition 71 (Leakage Resilience) *Let $\mathcal{G}$ be some family of functions. We say a coding scheme $(\mathcal{I}nit, \mathcal{E}nc, \mathcal{D}ec)$ is leakage resilient with respect to $\mathcal{G}$ if for every function $g \in \mathcal{G}$, every two states $s_0, s_1 \in \{0,1\}^k$, and every efficient adversary A, we have $\Pr[A(\Sigma, g(\Sigma, \mathcal{E}nc(\Sigma, s_b)) = b] \leq 1/2 + \mathsf{ngl}(k)$, where b is a random bit, and $\Sigma \leftarrow \mathcal{I}nit(1^k)$.*

10.2 Construction Overview

In this section, we describe our construction of an LR-NM code. Before presenting our construction, we first consider two bad candidates.

Consider the following idea, inspired by Gennaro et al. [GLM+04]: a seemingly natural way to prevent malleability is to add a signature to the code; an attacker (it would seem) would have to forge a signature in order to tamper with the codeword. Thus, to encode a string s, we sample a signing and verification key pair $(\mathsf{sk}, \mathsf{vk})$ and set $M_1 = \mathsf{sk}$ and $M_2 = (\mathsf{vk}, \mathrm{Sign}_{\mathsf{sk}}(s))$. Intuitively, M_1 has no information about s, and M_2 cannot be tampered with by the unforgeability of the signature scheme. However, the problem is that the latter is true only as long as M_1 is not tampered with. An adversary can easily defeat this construction: first he resamples another key pair $(\mathsf{sk}', \mathsf{vk}')$ and then sets $M_1 = \mathsf{sk}'$, and $M_2 = (\mathsf{vk}', \mathrm{Sign}_{\mathsf{sk}'}(s))$. This creates a valid codeword whose underlying message is highly correlated to the original one, and thus it cannot satisfy the definition.

Another possible approach (inspired by the work on non-malleable cryptography [DDN00]) is to use a non-malleable encryption scheme. To encode a string s, we sample a key pair $(\mathsf{pk}, \mathsf{sk})$ and set $M_1 = \mathsf{sk}$ and $M_2 = (\mathsf{pk}, \mathrm{Encrypt}_{\mathsf{pk}}(s))$. If the adversary tampers with the ciphertext $\mathrm{Encrypt}_{\mathsf{pk}}(s)$ only, then by the definition of non-malleable encryption, the tampered message cannot be related to s, which is what we need. However, if the adversary tampers with the keys as well, it is unclear how non-malleability can be guaranteed. In fact, we are not aware of any encryption scheme that has this type of non-malleability in the face of key tampering.

Although we just saw that non-malleable encryption does not work directly, the techniques of how to achieve non-malleability, due to Naor and Yung [NY90], Sahai [Sah99] and Dolev et al. [DDN00], give us a good starting point. In particular, these works used a non-interactive zero-knowledge (NIZK) proof to enforce consistency such that the adversary cannot generate valid ciphertexts by mauling the challenger's ciphtertext. Here we consider a similar technique that uses an encryption scheme and an NIZK proof, and sets $M_1 = \mathsf{sk}$, $M_2 = (\mathsf{pk}, \hat{s} = \mathrm{Encrypt}_{\mathsf{pk}}(s), \pi)$ where π is a proof of consistency (i.e. it proves that there exists a secret key corresponding to pk and that $\hat{s}$ can be decrypted using this secret key).

Does this work yet? If the attacker modifies $\hat{s}$, then the proof π has to be modified as well. If the underlying proof system is malleable, then it could be possible to modify both at the same time, so that the attacker could obtain an encoding of a string that is related to the original s. So we require

that the proof system be *non-malleable*; specifically we use the notion of *robust NIZK* given by de Santis et al. [DDO+01], in which, informally, the adversary can only output new proofs for which he knows the corresponding witnesses, even when given black-box access to a simulator that produces simulated proofs on demand; there exists an extractor that can extract these witnesses.

Now let us try to give a high-level proof of security. Recall that we need to show: for any poly-time adversary A that breaks the non-malleability with some split-state tampering function $f = (f_1, f_2)$, there exists an efficient reduction that breaks the semantic security of the encryption. Given a public key pk, and a ciphertext c, it is the reduction's job to determine whether c is an encryption of s_0 or s_1, with the help of the adversary that distinguishes $\mathsf{Tamper}^f_{s_0}$ and $\mathsf{Tamper}^f_{s_1}$. A natural way for the reduction is to pretend that $M_1 = \mathsf{sk}$, and put the public key pk and the ciphertext $\hat{s} = c$ with a simulated proof into M_2, setting $M_2 = (\mathsf{pk}, \hat{s}, \pi_{\mathcal{S}im})$. Then the reduction simulates the tampering experiment Tamper^f_s. Clearly, irrespective of f_1 the reduction can compute $f_2(M_2) = (\mathsf{pk}', \hat{s}', \pi_{\mathcal{S}im})$, and intuitively, the non-malleability of the proof assures that the adversary can only generate valid $(\mathsf{pk}', \hat{s}')$ if he knows sk' and s'. So at first glance, the outcome of the tampering experiment (i.e. the decoding of the tampered codeword) should be s', which can be simulated by the reduction. Thus, the reduction can use A to distinguish the two different experiments.

However, there are several subtle missing links in the above argument. The reduction above does not use any property of f_1, which might cause a problem. Suppose $f_1(\mathsf{sk}) = \mathsf{sk}'$, then the decoding of the tampered codeword is really s', so the reduction above simulates the tampering experiment faithfully. However, if not, then the decoding should be $\perp$ instead. Thus, the reduction crucially needs one bit of information: $\mathsf{sk}' \stackrel{?}{=} f_1(\mathsf{sk})$. If the reduction could get leakage $f_1(\mathsf{sk})$ directly, then it could compute this bit. However, the length of $f_1(\mathsf{sk})$ is the same as that of sk itself, and therefore no leakage-resilient cryptosystem can tolerate this much leakage. If the reduction, instead, tried to guess this bit, then A will be able to tell that it is dealing with the reduction rather than with the correct experiment, and may cancel out its advantage. (This is a common pitfall in indistinguishability reductions: they often don't go through if the adversary can tell that he is not operating "in the wild.")

Our novel observation here is that actually a small amount of leaked information about the secret key sk is sufficient for the reduction to tell the two cases apart. Let h be a hash function that maps input strings to strings of length ℓ. Then, in order to check whether $f_1(\mathsf{sk}) = \mathsf{sk}'$, it is very likely (assuming appropriate collision-resistance properties of h) sufficient to check if $h(f_1(\mathsf{sk})) = h(\mathsf{sk}')$.

So if we are given a cryptosystem that can tolerate ℓ bits of leakage, we can build a reduction that asks that $h(f_1(\mathsf{sk}))$ be leaked, and this (in addition to a few other technicalities that we do not highlight here) enables us to show that the above construction is non-malleable.

Besides non-malleability, the above code is also leakage-resilient in the sense that getting partial information about a codeword does not reveal any information about the encoded string. Intuitively, this is because the NIZK proof hides the witness, i.e. the message, and partial leakage of the secret key does not reveal anything about the message, either. Thus, this construction achieves non-malleability and leakage resilience at the same time.

10.3 The Construction

Recall $\mathcal{G}_t$ is the function class that includes all poly-sized circuits with t-bit output. Now we are ready to describe our tools and coding scheme.

Our tools: Let t be a polynomial, $\mathcal{E} = (\text{KeyGen}, \text{Encrypt}, \text{Decrypt})$ be an encryption scheme that is semantically secure against one-time leakage $\mathcal{G}_t$, and $\Pi = (\ell, \mathcal{P}, \mathcal{V}, \mathcal{S})$ be a robust NIZK proof system (see Definitions 60 and 57 in Section 9.1). The encryption scheme and robust NIZK needs to have some additional properties, and we briefly summarize them here: (1) given a secret key sk, one can efficiently derive it corresponding public key pk; (2) given a key pair $(\mathsf{pk}, \mathsf{sk})$, it is infeasible to find another valid $(\mathsf{pk}, \mathsf{sk}')$ where $\mathsf{sk} \neq \mathsf{sk}'$; (3) different statements of the proof system must have different proofs.

In Section 9.1 we give formal definitions of these additional properties and show that simple modifications of leakage-resilient crypto systems and robust NIZK proof systems satisfy them. Now, we define a coding scheme $(\mathcal{I}nit, \mathcal{E}nc, \mathcal{D}ec)$ as follows:

The coding scheme:

- $\mathcal{I}nit(1^k)$: sample a common reference string at random, i.e. $\Sigma \leftarrow \{0,1\}^{\ell(k)}$.
- $\mathcal{E}nc(\Sigma, s)$: on input message $s \in \{0,1\}^k$, sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \text{KeyGen}(1^k)$. Then consider the language $\mathbf{L}$ with the witness relation W defined as following:

$$\mathbf{L} = \left\{ (\mathsf{pk}, \hat{m}) : \exists w = (\mathsf{sk}, m) \text{ such that } \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \text{ forms a public-key secret-key pairs for } \mathcal{E} \text{ and} \\ m = \text{Decrypt}_{\mathsf{sk}}(\hat{m}). \end{array} \right\},$$

and W is the natural witness relation defined in the above language $\mathbf{L}$.

Compute the proof $\pi \leftarrow \mathcal{P}((\mathsf{pk}, \hat{s}), (\mathsf{sk}, s, r), \Sigma)$ of the statement that $(\mathsf{pk}, \hat{s}) \in \mathbf{L}$. Then output the encoding $c = (\mathsf{sk}; \mathsf{pk}, \hat{s} = \text{Encrypt}_{\mathsf{pk}}(s), \pi)$.

- $\mathcal{D}ec(\Sigma, c)$: If (1) $\mathcal{V}((\mathsf{pk}, \hat{s}), \pi, \Sigma)$ accepts and (2) $(\mathsf{pk}, \mathsf{sk})$ form a valid key pair, output $\text{Decrypt}_{\mathsf{sk}}(\hat{s})$. Otherwise, output $\perp$.

Let $n = n(k)$ be the polynomial that is equal to the length of $\mathsf{sk} \circ \mathsf{pk} \circ \hat{s} \circ \pi$. Without loss of generality, we assume that n is even, and $|\mathsf{sk}| = n/2$, and $|\mathsf{pk} \circ \hat{s} \circ \pi| = n/2$ (these properties can be easily guaranteed by padding the shorter side with 0's). Thus, a split-state device where $n(k)$-bit memory M is partitioned into M_1 and M_2 could store sk in M_1 and $(\mathsf{pk}, \hat{s}, \pi)$ in M_2.

Remark 72 Note that the decoding algorithm $\mathcal{D}ec$ is deterministic if the verifier $\mathcal{V}$ and the decryption algorithm Decrypt are both deterministic; as almost all known instantiations are. In the rest of the paper, we will assume that the decoding algorithm is deterministic.

Theorem 73 *Let $t : \mathbb{N} \to \mathbb{N}$ be some non-decreasing polynomial, and $\mathcal{G}_t, \mathcal{F}^{\mathsf{half}}, \mathcal{G}^{\mathsf{half}}_{t,\mathsf{all}}$ be as defined above. Suppose the encryption scheme $\mathcal{E}$ is semantically secure against one-time leakage $\mathcal{G}_t$; the system Π is a robust NIZK as stated above; and $\mathcal{H}_k : \{h_z : \{0,1\}^{\text{poly}(k)} \to \{0,1\}^k\}_{z \in \{0,1\}^k}$ is a family of universal one-way hash functions.*

Then the coding scheme is strong non-malleable (Def 70) with respect to $\mathcal{F}^{\mathsf{half}}$, and leakage resilient (Def 71) with respect to $\mathcal{G}^{\mathsf{half}}_{t,\mathsf{all}}$.

Proof. The proof contains two parts: showing that the code is non-malleable and that it is leakage resilient. The second part is easy so we only give the intuition. First let us look at $M_2 = (\mathsf{pk}, \hat{s}, \pi)$. Since π is a NIZK proof, it reveals no information about the witness (sk, s). For the memory $M_1 = \mathsf{sk}$, since the encryption scheme is leakage resilient, getting partial information about sk does not hurt the semantic security. Thus, for any $g \in \mathcal{G}^{\mathsf{half}}_{t,\mathsf{all}}$, $g(M_1, M_2)$ hides the original input string. We omit the formal details of the reduction, since they are straightforward.

Now we focus on the proof of non-malleability. In particular, we need to argue that for any $s_0, s_1 \in \{0,1\}^k$, and $f \in \mathcal{F}^{\mathsf{half}}$, we have $(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma}) \approx_c (\Sigma, \mathsf{Tamper}_{s_1}^{f,\Sigma})$ where $\Sigma \leftarrow \mathcal{I}nit(1^k)$. We show this by contradiction: suppose there exist $f = (f_1, f_2) \in \mathcal{F}^{\mathsf{half}}$, s_0, s_1, some $\varepsilon = 1/\mathrm{poly}(k)$, and a distinguisher D such that $\Pr[D(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma}) = 1] - \Pr[D(\Sigma, \mathsf{Tamper}_{s_1}^{f,\Sigma}) = 1] > \varepsilon$, then we can construct a reduction that breaks the encryption scheme $\mathcal{E}$.

The reduction will work as discussed in the overview. Before describing it, we first make an observation: D still distinguishes the two cases of the Tamper experiments even if we change all the real proofs to the simulated ones. More formally, let $(\Sigma, \tau) \leftarrow \mathcal{S}_1(1^k)$, and define $\mathsf{Tamper}_s^{f,\Sigma,\tau}$ be the same game as $\mathsf{Tamper}_s^{f,\Sigma}$ except proofs in the encoding algorithm $\mathcal{E}nc(\Sigma, \cdot)$ are computed by the simulator $\mathcal{S}_2(\cdot, \Sigma, \tau)$ instead of the real prover. We denote this distribution as Tamper_s^{f*}. We claim that D also distinguishes $\mathsf{Tamper}_{s_0}^{f*}$ from $\mathsf{Tamper}_{s_1}^{f*}$.

Suppose not, i.e. D, who distinguishes $\mathsf{Tamper}_{s_0}^{f,\Sigma}$ from $\mathsf{Tamper}_{s_1}^{f,\Sigma}$ does not distinguish $\mathsf{Tamper}_{s_0}^{f*}$ from $\mathsf{Tamper}_{s_1}^{f*}$. Then one can use D, f, s_0, s_1 to distinguish real proofs and simulated ones using standard proof techniques. This violates the multi-theorem zero-knowledge property of the NIZK system Π. Thus, we have $\Pr[D(\Sigma, \mathsf{Tamper}_{s_0}^{f*}) = 1] - \Pr[D(\Sigma, \mathsf{Tamper}_{s_1}^{f*}) = 1] > \varepsilon/2$.

In the following, we are going to define a reduction Red to break the leakage resilient encryption scheme $\mathcal{E}$. The reduction Red consists of an adversary $A = (A_1, A_2, A_3)$ and a distinguisher D' defined below.

The reduction (with the part A) plays the game $\mathsf{LE}_b(\mathcal{E}, A, k, \mathcal{F})$ with the challenger defined in Definition 60, and with the help of the distinguisher D and the tampering function $f = (f_1, f_2)$.

- First A_1 samples $z \in \{0,1\}^{t-1}$ (this means A_1 samples a universal one-way hash function $h_z \leftarrow \mathcal{H}_{t-1}$), and sets up a simulated CRS with a corresponding trapdoor $(\Sigma, \tau) \leftarrow \mathcal{S}_1(1^k)$.
- A_1 sets $g : \{0,1\}^{n/2} \to \{0,1\}^t$ to be the following function, and sends this leakage query to the challenger.

$$g(\mathsf{sk}) = \begin{cases} 0^t & \text{if } f_1(\mathsf{sk}) = \mathsf{sk}, \\ 1 \circ h_z(f_1(\mathsf{sk})) & \text{otherwise.} \end{cases}$$

  This leakage value tells A_1 if the tampering function f_1 alters sk.

- A_2 chooses m_0, m_1 to be s_0, and s_1 respectively. Then the challenger samples $(\mathsf{pk}, \mathsf{sk})$ and sets $\hat{m} = \mathrm{Encrypt}_{\mathsf{pk}}(m_b)$ to be the ciphertext, and sends $\mathsf{pk}, g(\mathsf{sk}), \hat{m}$ to the adversary.

- Then A_3 computes the simulated proof $\pi = \mathcal{S}_2(\mathsf{pk}, \hat{m}, \Sigma, \tau)$, and sets $(\mathsf{pk}', \hat{m}', \pi') = f_2(\mathsf{pk}, \hat{m}, \pi)$. Then A_3 does the following:

  1. If $g(\mathsf{sk}) = 0^t$, then consider the following cases:
     (a) $\mathsf{pk}' \neq \mathsf{pk}$, set $d = \perp$.
     (b) Else ($\mathsf{pk}' = \mathsf{pk}$),
        i. if $(\hat{m}', \pi') = (\hat{m}, \pi)$, set $d = \mathsf{same}^*$.
        ii. if $\hat{m}' \neq \hat{m}, \pi' = \pi$, set $d = \perp$.
        iii. else ($\pi' \neq \pi$), check whether $\mathcal{V}((\mathsf{pk}', \hat{m}'), \pi', \Sigma)$ accepts.
           A. If no, set $d = \perp$.
           B. If yes, use the extractor Ext to compute $(\mathsf{sk}'', m'') \leftarrow \mathrm{Ext}(\Sigma, \tau, x' = (\mathsf{pk}', \hat{m}'), \pi')$, where the list $Q = ((\mathsf{pk}, \hat{m}), \pi)$. If the extraction fails, then set $d = \perp$; otherwise $d = m''$.

  2. Else if $g(\mathsf{sk}) = 1 \circ h_z(f_1(\mathsf{sk})) \stackrel{\text{def}}{=} 1 \circ \mathit{hint}$, then consider the following case:
     (a) if $\pi' = \pi$, then set $d = \perp$.
     (b) else, check if $\mathcal{V}(\mathsf{pk}', \pi', crs)$ verifies, if not set $d = \perp$. Else, compute $(\mathsf{sk}'', m'') \leftarrow \mathrm{Ext}(\Sigma, \tau, x' = (\mathsf{pk}', \hat{m}'), \pi')$, where the list $Q = ((\mathsf{pk}, \hat{m}), \pi)$. If the extraction fails, then set $d = \perp$; otherwise consider the following two cases:
        i. If $h_z(\mathsf{sk}'') \neq \mathit{hint}$, then set $d = \perp$.
        ii. Else, set $d = m''$.

- Finally, A_3 outputs d, which is the output of the game $\mathsf{LE}_b(\mathcal{E}, A, k, \mathcal{F}^{\mathsf{half}})$.

Define the distinguisher D' on input d outputs $D(\Sigma, d)$. Then we need to show that A, D' break the scheme $\mathcal{E}$ by the following lemma. In particular, we will show that the above A's strategy simulates the distributions $\mathsf{Tamper}_{s_b}^{f^*}$, so that the distinguisher D's advantage can be used by D' to break $\mathcal{E}$.

Claim 74 *Given the above A and D', we have*

$$\Pr[D'(\mathsf{LE}_0(\mathcal{E}, A, k, \mathcal{F}^{\mathsf{half}})) = 1] - \Pr[D'(\mathsf{LE}_1(\mathcal{E}, A, k, \mathcal{F}^{\mathsf{half}})) = 1] > \varepsilon/2 - \mathsf{ngl}(k).$$

To prove this claim, we argue that the output d does simulate the decoding of the tampered codeword $\tilde{c} = (f_1(\mathsf{sk}), f_2(\mathsf{pk}, \hat{m}, \pi)) = (\mathsf{sk}', \mathsf{pk}', \hat{m}', \pi')$. Here A does not know $f_1(\mathsf{sk})$ so he cannot decode $\tilde{c}$ directly. Although A can get the help from leakage functions, however, $f_1(\mathsf{sk})$, as sk itself, has $n/2$ bits of output, which is too long so A cannot learn all of them. Our main observation is that getting a hash value of $f_1(\mathsf{sk})$ is sufficient for A to simulate the decoding. In particular, we will show that with the leakage $g(\mathsf{sk})$, A can simulate the decoding with at most a negligible error.

Proof of claim: First we make the following observations. Consider the case where $\mathsf{sk} = \mathsf{sk}' \stackrel{\text{def}}{=} f_1(\mathsf{sk})$ (the tampering function did not modify the secret key).

- If $\mathsf{pk}' \neq \mathsf{pk}$, since pk can be derived from sk deterministically as pointed out in Definition 60 and its remark, the correct decoding will be $\perp$ by the consistency check, which is that A_3 outputs. (case 1a).
- If f_2 does not modify its input either, the correct decoding equals $d = \mathsf{same}^*$, as A_3 says (case 1(b)i).
- if $\mathsf{pk}' = \mathsf{pk}, \hat{m}' \neq \hat{m}$ but $\pi' = \pi$, then the correct decoding will agree with A_3 and outputs $\perp$. This is because $\mathcal{V}(\mathsf{pk}', \hat{m}', \pi, \Sigma)$ will output a rejection since the statement has changed and the old proof to another statement cannot be accepted, by the robustness of NIZK (case 1(b)ii).
- if $\mathsf{pk}' = \mathsf{pk}$, $\pi' \neq \pi$, the correct decoding algorithm will first check $\mathcal{V}(\mathsf{pk}', \hat{m}', \pi')$. If it verifies, by the extractability of the proof system, the extractor Ext will output a witness $w = (\mathsf{sk}'', m'')$ of the relation W. Then A will use m'' as the outcome of the decoding. The only difference between the decoding simulated by A and the correct decoding algorithm (that knows sk and can therefore decrypt $\hat{m}'$) is the case when the extraction fails. By the property of the proof system, we know this event happens with at most $\nu(k)$, which is a negligible quantity. (case 1(b)iii).

Then we consider the case where $\mathsf{sk}' \neq \mathsf{sk}$ (the tampering adversary modified the secret key).

- If $\pi' = \pi$, then the correct decoding will be $\perp$ with probability $1 - \mathsf{ngl}(k)$. This is by the two additional properties: (1) the property of the encryption scheme stated in

Lemma 61 that no efficient adversary can get a valid key pair $(\mathsf{pk}, \mathsf{sk}')$ from $(\mathsf{pk}, \mathsf{sk})$ with non-negligible probability. (2) the proof of statement x cannot be used to prove other statements $x' \neq x$.

Thus, in this case A_3 agrees with the correct decoding algorithm with overwhelming probability $(1 - \mathsf{ngl}(k))$. (case 2a).

- If $\pi' \neq \pi$, and $\mathcal{V}(\mathsf{pk}', \hat{m}', \pi', \Sigma)$ accepts, then with probability $1 - \nu(k)$ the extractor will output a witness (sk'', m''). The correct decoding algorithm checks whether $(\mathsf{pk}', \mathsf{sk}')$ forms a key pair. Here A emulates this check by checking whether $h_z(\mathsf{sk}'') = h_z(\mathsf{sk}')$. Since h_z is a universal one-way hash function, the probability that $h_z(\mathsf{sk}'') = h_z(\mathsf{sk}') \wedge \mathsf{sk}'' \neq \mathsf{sk}'$ is at most $\mathsf{ngl}(k)$. Otherwise, we can construct another reduction B who simulates these games to break the universal one-wayness. B simulates both the adversary and the challenger of the interaction $\mathsf{LE}_b(\mathcal{E}, A, k, \mathcal{F}^{\mathsf{half}})$, and when A queries the leakage g that contains a description of f_1, B sets its x to be $\mathsf{sk}' = f_1(\mathsf{sk})$. Then B receives a index z, and then B continue to simulate the game. Then B can find out another $x' = \mathsf{sk}''$ where $h_z(x) = h_z(x') \wedge x' \neq x'$ from in the game with non-negligible probability. This is a contradiction.

  Thus by a union bound, with probability $1 - \nu(k) - \mathsf{ngl}(k)$, A emulates the decoding algorithm faithfully. (case 2b).

Let event E_1 be the one where Ext extracts a valid witness $w = (\mathsf{sk}'', m'')$ in cases 1(b)iii and 2b, . Let event E_2 be the one where in case 2b, $h(\mathsf{sk}'') = h(\mathsf{sk}') \wedge \mathsf{sk}'' = \mathsf{sk}'$.

By the above observations, we have

$$\Pr\left[(\Sigma, \mathsf{LE}_b(\mathcal{E}, A, k, \mathcal{F}_{\mathsf{half}})) = \mathsf{Tamper}_{s_b}^{f*} \middle| E_1 \wedge E_2\right] = 1, \text{ and } \Pr[\neg E_1] + \Pr[\neg E_2] < \mathsf{ngl}(k).$$

Thus we have $\Pr\left[(\Sigma, \mathsf{LE}_b(\mathcal{E}, A, k, \mathcal{F}_{\mathsf{half}})) = \mathsf{Tamper}_{s_b}^{f*}\right] > 1 - \mathsf{ngl}(k)$, which implies the claim directly.

□

This completes the proof of the Theorem.

■

Chapter 11

The Construction of Compiler

In this section, we present two compilers that use our LR-NM code to secure any functionality G from split-state tampering and leakage attacks. The first compiler, as an intermediate result, outputs a compiled functionality G' that has access to fresh random coins. The second one outputs a deterministic functionality by derandomizing G' using a pseudorandom generator.

11.1 Randomized Implementation

Let $G(s, x)$ be an interactive functionality with a k-bit state s that we want to protect, and let $\mathcal{C} = (\mathcal{I}nit, \mathcal{E}nc, \mathcal{D}ec)$ be the LR-NM coding scheme we constructed in the previous section. Our compiler works as follows: first it generates the common parameters $\Sigma \leftarrow \mathcal{I}nit(1^k)$. Then $\mathsf{MemCompile}(\Sigma, s)$ outputs an encoding of s, $(M_1, M_2) \leftarrow \mathcal{E}nc(\Sigma, s)$; and $\mathsf{CircuitCompile}(G, \mathcal{C}, \Sigma)$ outputs a randomized functionality G' such that $\langle G', \mathcal{E}nc(\Sigma, s)\rangle$ works in the following way: on user input x, first G' decodes the memory using the decoding algorithm $\mathcal{D}ec$. If the outcome is $\perp$, then G' will always output $\perp$ (equivalently, self-destruct); otherwise it obtains s. Then G' computes $(s_{\mathsf{new}}, y) \leftarrow G(s, x)$ and outputs y. Finally G' re-encodes its memory: $(M_1, M_2) \leftarrow \mathcal{E}nc(\Sigma, s_{\mathsf{new}})$. There are two places where G' uses fresh randomness: the functionality G itself and the re-encoding step.

We denote this randomized hardware implementation of the compiler as $\mathsf{Hardware}_{\mathrm{rand}}(\mathcal{C}, G) \stackrel{\mathrm{def}}{=} \langle G', \mathcal{E}nc(s)\rangle$. Obviously the compiler is correct, i.e. the implementation's input/output behavior is the same as that of the original functionality. Next, we will show it is also secure against leakage and tampering attacks.

Theorem 75 *Let $t : \mathbb{N} \to \mathbb{N}$ be some non-decreasing polynomial, and $\mathcal{G}_t, \mathcal{F}^{\mathsf{half}}, \mathcal{G}^{\mathsf{half}}_{t,\mathsf{all}}$ be as defined above.*

Suppose we are given a cryptosystem $\mathcal{E} = (\mathrm{KeyGen}, \mathrm{Encrypt}, \mathrm{Decrypt})$ that is semantically secure against one-time leakage $\mathcal{G}_t$; a robust NIZK $\Pi = (\ell, \mathcal{P}, \mathcal{V}, \mathcal{S})$; and $\mathcal{H}_k : \{h_z : \{0,1\}^{\mathrm{poly}(k)} \to \{0,1\}^k\}_{z \in \{0,1\}^k}$, a family of universal one-way hash functions. Then the randomized hardware implementation presented above is secure against $\mathcal{F}^{\mathsf{half}}$ tampering and $\mathcal{G}^{\mathsf{half}}_{t,\mathsf{all}}$ leakage.

Let us explain our proof approach. In the previous section, we have shown that the coding scheme is leakage-resilient and non-malleable. This intuitively means that one-time attacks on the hardware implementation $\mathsf{Hardware}_{\mathrm{rand}}(\mathcal{C}, G)$ are useless. Therefore, what we need to show is that these two types of attacks are still useless even when the adversary has launched a continuous attack.

Recall that, by definition, to prove tamper and leakage resilience, we need to exhibit a simulator that simulates the adversary's view of interaction with $\mathsf{Hardware}_{\mathrm{rand}}(\mathcal{C}, G)$ based solely on black-box access to $\langle G, s \rangle$. The simulator computes M_1 and M_2 almost correctly, except it uses $s_0 = 0^k$ instead of the correct s (which, of course, it cannot know). The technically involved part of the proof is to show that the resulting simulation is indistinguishable from the real view; this is done via a hybrid argument in which an adversary that detects that, in round i, the secret changed from s_0 to the real secret s, can be used to break the LR-NM code, since this adversary will be able to distinguish $\mathsf{Tamper}^{f,\Sigma}_{s_0}$ from $\mathsf{Tamper}^{f,\Sigma}_{s}$ or break the leakage resilience of the code. In doing this hybrid argument, care must be taken: by the time we even get to round i, the adversary may have overwritten the state of the device; also, there are several different ways in which the security may be broken and our reduction relies on a careful case analysis to rule out each way.

Proof. [Theorem 75] To prove the theorem, we need to construct a simulator $\mathcal{S}im$ that gets black-box access to any adversary A who issues $\mathsf{Execute}$, Tamper, and Leak queries, and functionality $\langle G, s \rangle$ that only answers $\mathsf{Execute}$ queries, and outputs an indistinguishable view from that of the real experiment, in which A talks directly to the harden functionality for $\langle G, s \rangle$. Define $\mathcal{S}im$ as the following procedure:

On input 1^k, $\mathcal{S}im$ first samples a common reference string $\Sigma \leftarrow \{0,1\}^{\ell(k)}$. (Recall ℓ is the parameter in the NIZK $\Pi = (\ell, \mathcal{P}, \mathcal{V}, \mathcal{S})$). In the first round, the simulator starts with the normal mode defined below:

- Normal mode, while the adversary keeps issuing queries, respond as follows:

  - When the adversary queries $\mathsf{Execute}(x)$, the simulator queries the input x to $\langle G, s\rangle$ and forwards its reply y back to A.
  - When the adversary queries $\mathsf{Tamper}(f)$ for some $f \in \mathcal{F}^{\mathsf{half}}$, the simulator samples t from the distribution $\mathsf{Tamper}_{0^k}^{f,\Sigma}$. If $t = \mathsf{same}^*$, then $\mathcal{S}im$ does nothing. Otherwise, go to the overwritten mode defined below with the state t.
  - When the adversary queries $\mathsf{Leak}(g)$ for some $g \in \mathcal{G}_{t,\mathsf{all}}^{\mathsf{half}}$, the simulator samples a (random) encoding of 0^k, $\mathcal{E}nc(0^k)$, and sends $g(\mathcal{E}nc(0^k))$ to the adversary.

- Overwritten mode with state t, while the adversary keeps issuing queries, respond as follows:
  - The simulator simulates the hardened functionality with state t, i.e. $\langle G', \mathcal{E}nc(t)\rangle$, and answers execute, tampering and leakage queries accordingly.

- Suppose A halts and outputs $\text{view}_A = (state_A, x_1, \ell_1, \dots)$ where x_i denotes the query, and ℓ_i is the leakage in the i-th round. Then the simulator sets $\text{view}_{\mathcal{S}im} = \text{view}_A$, and outputs $(\Sigma, \text{view}_{\mathcal{S}im})$ at the end. We remark that if in the i-th round, A did not make an $\mathsf{Execute}$ query, then $x_i = \phi$; similarly if he did not query Leak, then $\ell_i = \phi$.

Intuitively, the normal mode simulates the adversary's queries before he mauls the secret state, and the overwritten mode simulates those after he mauls it. Intuitively, the coding scheme is non-malleable, so the adversary can either keep the secret state unchanged or change it to something he knows. This is captured by the above two modes. On the other hand, the (one-time) leakage resilient encryption protects the secret against leakage attacks.

In the end of each round, the secret state is re-encoded with fresh randomness. Thus we can use a hybrid argument to show that the hardened functionality is secure for many rounds. We remark that since there are three possible queries and two different modes in each round, in our hybrid argument, a case study of many options should be expected.

In the rest of the proof, we are going to formalize this intuition and show that this simulated view is indistinguishable from that of the real experiment. In particular, we will establish the following lemma:

Lemma 76 *Let $\mathcal{S}im$ be the simulator defined above. Then for any adversary A and any state $s \in \{0,1\}^k$, $\mathsf{Real}(A, s) = (\Sigma, \text{view}_A)$ is computationally indistinguishable from $\mathsf{Ideal}(\mathcal{S}im, A, s) = (\Sigma, \text{view}_{\mathcal{S}im})$.*

Proof. Suppose there exists an adversary A running the experiment for at most $L = \mathrm{poly}(k)$ rounds, a state s, and a distinguisher D such that $\Pr[D(\Sigma, \mathrm{view}_{Real}) = 1] - \Pr[D(\Sigma, \mathrm{view}_{\mathcal{S}im}) = 1] > \varepsilon$ for some non-negligible ε, then we will construct a reduction that will find a function $f \in \mathcal{F}^{\mathsf{half}}$, two states s_0, s_1, and a distinguisher D' that distinguishes $(\Sigma, \mathsf{Tamper}^{f,\Sigma}_{s_0})$ from $(\Sigma, \mathsf{Tamper}^{f,\Sigma}_{s_1})$. This breaks non-malleability of the coding scheme, which contradicts to Theorem 73.

To show this, we define the following hybrid experiments for $i \in [L]$:

Experiment $\mathcal{S}im^{(i)}(A, s)$:

- $\mathcal{S}im^{(i)}$ setups the common reference string to be $\Sigma \leftarrow \{0,1\}^{\ell(k)}$.
- In the first i rounds, $\mathcal{S}im^{(i)}$ does exactly the same as $\mathcal{S}im$.
- From the $i+1$-th round, if $\mathcal{S}im^{(i)}$ has already entered the overwritten mode, then do the simulation as the overwritten mode. Otherwise, let s_{curr} be the current state of the functionality, and the simulation does the following modified normal mode:
  - When the adversary queries $\mathsf{Execute}(x)$, the simulator queries the functionality $(y, s_{\mathsf{new}}) \leftarrow G(x, s_{\mathsf{curr}})$. Then it forwards y, and set $s_{\mathsf{curr}} = s_{\mathsf{new}}$.
  - When the adversary queries $\mathsf{Tamper}(f)$ for some $f \in \mathcal{F}$, the simulator samples t from the distribution $\mathsf{Tamper}^{f,\Sigma}_{s_{\mathsf{curr}}}$. If $t = \mathsf{same}^*$, then the simulator does nothing. Otherwise, go to the overwritten mode with the state t.
  - When the adversary queries $\mathsf{Leak}(g)$ for some $g \in \mathcal{G}$, the simulator samples a (random) encoding of s_{curr}, $\mathcal{E}nc(s_{\mathsf{curr}})$, and replies $g(\mathcal{E}nc(s_{\mathsf{curr}}))$ to the adversary.

We remark that $\mathcal{S}im^{(i)}$ behaves like $\mathcal{S}im$ in the first i rounds, and in the later rounds, it behaves exactly the same as $\langle G^{\Sigma,\mathcal{E}nc,\mathcal{D}ec}, \mathcal{E}nc(\Sigma, s_{\mathsf{curr}})\rangle$ if the simulation does not enter the overwritten mode. Then we observe that $\mathcal{S}im^{(0)}(A, s)$ is the output of the real experiment $(\Sigma, \mathrm{view}_A)$, and $\mathcal{S}im^{(L)}(A, s)$ is that of the ideal experiment $(\Sigma, \mathrm{view}_{\mathcal{S}im})$. By an averaging argument, there exists some $j \in [L]$ such that

$$\Pr[D(\Sigma, \mathcal{S}im^{j}(A, s)) = 1] - \Pr[D(\Sigma, \mathcal{S}im^{j+1}(A, s)) = 1] > \varepsilon/L.$$

Since $\mathcal{S}im^{(j)}$ and $\mathcal{S}im^{(j+1)}$ only differ at round $j+1$ and D can distinguish one from the other, our reduction will take the advantage of D on this round. First we define the following four possible events that can happen in round $j+1$:

- E_1: the simulation has entered the overwritten mode by the $j+1$-st round.
- E_2: the simulation is in the normal mode and the adversary queries $\mathsf{Execute}$ in the $j+1$-st round.
- E_3: the simulation is in the normal mode and the adversary queries Leak in the $j+1$-st round.
- E_4: the simulation is in the normal mode and the adversary queries Tamper in the $j+1$-st round.

Claim 77 *The probability of $E_3 \vee E_4$ is non-negligible.*

Proof of claim: We can easily see that conditioning on the events E_1, E_2, $\mathcal{S}im^{(j)}$ and $\mathcal{S}im^{(j+1)}$ are identical. Thus if $E_3 \vee E_4$ happens with negligible probability, then $\mathcal{S}im^{(j)}$ and $\mathcal{S}im^{(j+1)}$ are statistically close up to negligible probability, which is a contradiction to the fact that D distinguishes them with non-negligible probability. □

Then we are going to show the following claim:

Claim 78 $\Pr[E_4] > \alpha$ *for some non-negligible* α.

Proof of claim: We will show this by contradiction. Suppose $\Pr[E_4] = \mathsf{ngl}(k)$. Then we are going to construct a reduction B that breaks the encryption scheme $\mathcal{E}$. First we observe an easy fact that $\Pr[E_3]$ is non-negligible. This follows from the previous claim, and our premise that $\Pr[E_4] = \mathsf{ngl}(k)$.

Let $\mathsf{LE}_b \stackrel{\text{def}}{=} \mathsf{LE}_b(\mathcal{E}, B, k, \mathcal{G}_t)$ be the game and B does the following:

- First B receives pk, and then B sets up a common reference string along with a trapdoor from the NIZK simulator, i.e. $(\Sigma, \tau) \leftarrow \mathcal{S}_1(1^k)$.
- Then B simulates the interaction of $\mathcal{S}im^{(j)}(A, s)$ for the first j rounds except whenever the simulation requires a proof, B uses $\mathcal{S}_2(\cdot, \cdot, \Sigma, \tau)$ to generate it. We remark that to simulate this experiment, B needs to run the adversary A and the functionality $G(\cdot, \cdot)$. In particular, B keeps tracks of the current state of $\langle G, s\rangle$ at each round, and let let s_{curr} be the current state at the end of the j-th round.

- In the $j+1$-st round, if the event E_3 does not happen, then B gives up: B simply sends any dummy messages m_0, m_1 to the challenger, but then guesses a bit at random on input challenge ciphertexts.
- Otherwise if the adversary queries $\mathsf{Leak}(g)$ for some $g = (g_1, g_2) \in \mathcal{G}^{\mathsf{half}}_{t,\mathsf{all}}$, B chooses $m_0 = 0^k$ and $m_1 = s_{\mathsf{curr}}$ and then asks for the leakage $g_1(\mathsf{sk})$.
- Then B receives $\mathsf{pk}, \hat{m}_b = \mathrm{Encrypt}_{\mathsf{pk}}(m_b), g_1(\mathsf{sk})$, and then B computes a simulated proof π. Then B sends to A $g_1(\mathsf{sk}), g_2(\mathsf{pk}, \hat{m}_b, \pi)$ as the response to $\mathsf{Leak}(g)$ and simulates the rest of $\mathcal{S}im^{j+1}$. Once A halts, A outputs a view, and B set $\mathrm{view}'_{\mathcal{S}im}$ to be that view.
- In the end, B outputs $D(\Sigma, \mathrm{view}'_{\mathcal{S}im})$: if D thinks that his view came from $\mathcal{S}^{(j)}$ then B outputs m_0, else m_1.

Then we are going to show that $|\Pr[\mathsf{LE}_0 = 1] - \Pr[\mathsf{LE}_1 = 1]| > \varepsilon'$ for some non-negligible ε'. First we observe that

$$\begin{aligned}
&\Pr[\mathsf{LE}_0 = 1] - \Pr[\mathsf{LE}_1 = 1] \\
&= \sum_{i \in [4]} \left(\Pr\left[\mathsf{LE}_0 = 1 \middle| E_i\right] \cdot \Pr[E_i] - \Pr\left[\mathsf{LE}_1 = 1 \middle| E_i\right] \cdot \Pr[E_i] \right) \\
&= \left(\Pr\left[\mathsf{LE}_0 = 1 \middle| E_3\right] - \Pr\left[\mathsf{LE}_1 = 1 \middle| E_3\right] \right) \cdot \Pr[E_3].
\end{aligned}$$

This follows from the fact that conditioning on $\neg E_3$, the output of LE_b is uniformly at random from the construction of the adversary B. In the following, we are going to show this is a noticeable quantity.

Let $\mathcal{S}im^{(j)'}$ denote the experiment identical with $\mathcal{S}im^{(j)}$ except that the common reference string and all the proofs are set up by the NIZK simulator $\mathcal{S}$. Similarly we have $\mathcal{S}im^{(j+1)'}$. By the zero knowledge property, we have,

$$\left| \Pr_{\Sigma \leftarrow \{0,1\}^{\ell(k)}}[D(\Sigma, \mathcal{S}im^{(j)}) = 1] - \Pr_{\Sigma \leftarrow \mathcal{S}(1^k)}[D(\Sigma, \mathcal{S}im^{(j)'}) = 1] \right| < \mathsf{ngl}(k),$$

$$\left| \Pr_{\Sigma \leftarrow \{0,1\}^{\ell(k)}}[D(\Sigma, \mathcal{S}im^{(j+1)}) = 1] - \Pr_{\Sigma \leftarrow \mathcal{S}(1^k)}[D(\Sigma, \mathcal{S}im^{(j+1)'}) = 1] \right| < \mathsf{ngl}(k).$$

From the assumption we know

$$\left| \Pr_{\Sigma \leftarrow \{0,1\}^{\ell(k)}}[D(\Sigma, \mathcal{S}im^{(j)}) = 1] - \Pr_{\Sigma \leftarrow \{0,1\}^{\ell(k)}}[D(\Sigma, \mathcal{S}im^{(j+1)}) = 1] \right| > \varepsilon/L.$$

Thus we have

$$\left| \Pr_{\Sigma \leftarrow \mathcal{S}(1^k)}[D(\Sigma, \mathcal{S}im^{(j)'}) = 1] - \Pr_{\Sigma \leftarrow \mathcal{S}(1^k)}[D(\Sigma, \mathcal{S}im^{(j+1)'}) = 1] \right| > \varepsilon/L - \mathsf{ngl}(k).$$

Then we express this equation with the four conditioning probabilities:

$$\begin{aligned}
&\Pr_{\Sigma \leftarrow \mathcal{S}(1^k)}[D(\Sigma, \mathcal{S}im^{(j)'}) = 1] - \Pr_{\Sigma \leftarrow \mathcal{S}(1^k)}[D(\Sigma, \mathcal{S}im^{(j+1)'}) = 1] \\
&= \sum_{i \in [4]} \left(\Pr_{\Sigma \leftarrow \mathcal{S}(1^k)} \left[D(\Sigma, \mathcal{S}im^{(j)'}) = 1 \middle| E_i \right] \cdot \Pr[E_i] - \Pr_{\Sigma \leftarrow \mathcal{S}(1^k)} \left[D(\Sigma, \mathcal{S}im^{(j+1)'}) = 1 \middle| E_i \right] \cdot \Pr[E_i] \right) \\
&= \Delta_3 \cdot \Pr[E_3] + \Delta_4 \cdot \Pr[E_4] \\
&\geq \varepsilon/L - \mathsf{ngl}(k),
\end{aligned}$$

where $\Delta_3 = \Pr_{\Sigma \leftarrow \mathcal{S}(1^k)} \left[D(\Sigma, \mathcal{S}im^{(j)'}) = 1 \middle| E_3 \right] - \Pr_{\Sigma \leftarrow \mathcal{S}(1^k)} \left[D(\Sigma, \mathcal{S}im^{(j+1)'}) = 1 \middle| E_3 \right]$, and $\Delta_4 = \Pr_{\Sigma \leftarrow \mathcal{S}(1^k)} \left[D(\Sigma, \mathcal{S}im^{(j)'}) = 1 \middle| E_4 \right] - \Pr_{\Sigma \leftarrow \mathcal{S}(1^k)} \left[D(\Sigma, \mathcal{S}im^{(j+1)'}) = 1 \middle| E_4 \right]$.

The first equality follows from the Bayes' equation. The second equality follows from the fact that conditioning on E_1 or E_2, $\mathcal{S}im^{(j)'}$ and $\mathcal{S}im^{(j+1)'}$ are identically distributed. Recall that the two distributions become identical once the simulation has entered the overwritten mode before round $j+1$. If the adversary queries $\mathsf{Execute}$ with the normal mode in the $j+1$-th round, the two experiments are the same also. The last inequality just follows from the above equation.

Then from the premise, we have $\Pr[E_4] = \mathsf{ngl}(k)$, we have $\Delta_4 \cdot \Pr[E_4] = \mathsf{ngl}(k)$, and thus: $\Delta_3 \cdot \Pr[E_3] \geq \varepsilon/L - \mathsf{ngl}(k)$.

Then we observe that for B's strategy, conditioning on the event E_3, if $b = 0$, B will simulate according to $\mathcal{S}im^{(j)'}$, and if $b = 1$, $\mathcal{S}im^{(j+1)'}$. This means $\Pr[\mathsf{LE}_0 =$

$1|E_3] - \Pr[\mathsf{LE}_1 = 1|E_3] = \Delta_3$, and thus from the previous calculations, we have

$$\begin{aligned}
&\Pr[\mathsf{LE}_0 = 1] - \Pr[\mathsf{LE}_1 = 1] \\
&= \left(\Pr\left[\mathsf{LE}_0 = 1 \middle| E_3\right] - \Pr\left[\mathsf{LE}_1 = 1 \middle| E_3\right]\right) \cdot \Pr[E_3] \\
&= \Delta_3 \cdot \Pr[E_3] \\
&\geq \varepsilon/L - \mathsf{ngl}(k).
\end{aligned}$$

This means B breaks the scheme $\mathcal{E}$ with non-negligible probability.

□

We wish to show that the simulator $\mathcal{S}im$ we give satisfies Definition 67. So far we have shown that if it does not provide a good simulation, then there exists some state s, index j such that $\Pr[E_4]$ happens with non-negligible probability. We must now construct a reduction that with s and j as advice, and with access to the adversary A, breaks non-malleability of the coding scheme. The idea is to use A's tampering query in round $j+1$, which we know A makes such query with non-negligible probability.

The reduction we will construct needs to find with advice s, j, two strings s_0, s_1, and a tampering function $f^\Sigma = (f_1^\Sigma, f_2^\Sigma) \in \mathcal{F}^{\mathsf{half}}$, and distinguishes $(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma})$ from $(\Sigma, \mathsf{Tamper}_{s_1}^{f,\Sigma})$.

Both the reduction and the function f^Σ will run $\mathcal{S}im^{(j)}$ as a subroutine, and will have oracle access to Σ. A subtlety in this approach is that $\mathcal{S}im^{(j)}$ is a randomized algorithm while f^Σ is deterministic (a polynomial-sized circuit). To overcome this, our reduction will simply fix the randomness of $\mathcal{S}im$. Let R be a random tape. By $\mathcal{S}im^{(j)}[R]$ we denote that $\mathcal{S}im^{(j)}$ uses randomness R; similarly $\mathcal{S}im^{(j+1)}[R]$.

Now we describe the reduction. First it picks R uniformly at random as the randomness for the simulator. It runs $\mathcal{S}im^{(j)}[R](A, s)$ for j rounds, to obtain the current state s_{curr}. Then it sets $s_0 = 0^k$, $s_1 = s_{\mathsf{curr}}$. Next the reduction computes a description of the polynomial-sized circuits for $f^\Sigma = (f_1^\Sigma, f_2^\Sigma) \in \mathcal{F}^{\mathsf{half}}$. This f^Σ is the tampering function that A outputs when running $\mathcal{S}im^{(j)}[R](A, s)$ at round $j+1$. If A does not query Tamper or the simulation has entered the overwritten mode at this round (the event E_4 does not happen), then let f^Σ be a constant function that always outputs $\perp$. We call this event Bad (i.e. $\mathsf{Bad} = \neg E_4$). We remark that there is an efficient algorithm that on input circuits A, $\langle G, s\rangle$, $\mathcal{S}im^{(j)}[R]$, outputs the function f.

Next let us argue that with s_0, s_1, $f^{\Sigma} = (f_1^{\Sigma}, f_2^{\Sigma})$ as above, one can distinguish $(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma})$ from $(\Sigma, \mathsf{Tamper}_{s_1}^{f,\Sigma})$. We construct a distinguisher D' as follows.

On input $(\Sigma, \mathsf{Tamper}_{s_b}^{f,\Sigma})$, D' first uses R to do the simulation of the first j rounds of $\mathcal{S}im^{(j)}[R](A, s)$. Then if the event Bad happens, D outputs 0 or 1 uniformly at random. Otherwise, D' uses the outcome of $\mathsf{Tamper}_{s_b}^{f,\Sigma}$ and continues to simulate the remaining rounds from round $j+2$ to L. Let view_b be the view of this simulation in the end. Then D' runs $D(\Sigma, \text{view}_b)$; if D thinks that he was interacting with $\mathcal{S}im^{(j)}$, D' outputs 1; else D' outputs 0.

From the above arguments, we know that (1) conditioning on the event $\neg\mathsf{Bad}$, view_0 is exactly the view of $\mathcal{S}im^{(j+1)}$, and view_1 is exactly that of $\mathcal{S}im^{(j)}$; (2) conditioning on Bad, the output of D' is randomly over 0/1; (3) $\Pr[\neg\mathsf{Bad}] > \alpha$ for some non-negligible α by the above claim. Thus we have

$$
\begin{aligned}
&\Pr\left[D'(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma}) = 1\right] - \Pr\left[D'(\Sigma, \mathsf{Tamper}_{s_1}^{f,\Sigma}) = 1\right] \\
&= \left(\Pr\left[D'(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma}) = 1 \middle| \neg\mathsf{Bad}\right] \cdot \Pr\left[\neg\mathsf{Bad}\right] + \Pr\left[D'(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma}) = 1 \middle| \mathsf{Bad}\right] \cdot \Pr\left[\mathsf{Bad}\right]\right) - \\
&\quad \left(\Pr\left[D'(\Sigma, \mathsf{Tamper}_{s_1}^{f,\Sigma}) = 1 \middle| \neg\mathsf{Bad}\right] \cdot \Pr\left[\neg\mathsf{Bad}\right] + \Pr\left[D'(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma}) = 1 \middle| \mathsf{Bad}\right] \cdot \Pr\left[\mathsf{Bad}\right]\right) \\
&= \left(\Pr\left[D(\Sigma, \text{view}_0) = 1\right] - \Pr\left[D(\Sigma, \text{view}_1) = 1\right]\right) \cdot \Pr\left[\neg\mathsf{Bad}\right] \\
&= \left(\Pr\left[D(\Sigma, \mathcal{S}im^{(j)}) = 1\right] - \Pr\left[D(\Sigma, \mathcal{S}im^{(j+1)}) = 1\right]\right) \cdot \Pr\left[\neg\mathsf{Bad}\right] \\
&\geq \varepsilon/L \cdot \alpha, \text{ a non-negligible quantity.}
\end{aligned}
$$

This completes the proof of the lemma.

■

This proof of the theorem follows directly from the construction of $\mathcal{S}im$ and the lemma.

■

11.2 Deterministic Implementation

In the previous section, we showed that the hardware implementation $\mathsf{Hardware}_{\text{rand}}$ with the LR-NM code is leakage- tampering-resilient. In this section, we show how to construct a deterministic implementation by derandomizing the construction. Our main observation is that, since the coding scheme also hides its input string (like an encryption scheme), we can store an encoding of a random

seed, and then use a pseudorandom generator to obtain more (pseudo) random bits. Since this seed is protected, the output of the PRG will be pseudorandom, and can be used to update the encoding and the seed. Thus, we have pseudorandom strings for an arbitrary (polynomially bounded) number of rounds. The intuition is straitforward yet the reduction is subtle: we need to be careful to avoid a circular argument in which we rely on the fact that the seed is hidden in order to show that it is hidden.

To get a deterministic implementation for any given functionality $G(\cdot,\cdot)$, we use the coding scheme $\mathcal{C} = (\mathcal{I}nit, \mathcal{E}nc, \mathcal{D}ec)$ defined in the previous section, and a pseudorandom generator $g : \{0,1\}^k \to \{0,1\}^{k+2\ell}$, where ℓ will be defined later. Let $s \in \{0,1\}^k$ be the secret state of $G(\cdot,\cdot)$, and $\mathsf{seed} \in \{0,1\}^k$ be a random k-bit string that will serve as a seed for the PRG. Now we define the compiler. The compiler first generates the common parameters $\Sigma \leftarrow \mathcal{I}nit(1^k)$. Then on input $s \in \{0,1\}^k$, $\mathsf{MemCompile}(s)$ first samples a random seed $\mathsf{seed} \in \{0,1\}^k$ and outputs $(M_1, M_2) \leftarrow \mathcal{E}nc(\Sigma, s \circ \mathsf{seed})$ where $\circ$ denotes concatenation. $\mathsf{CircuitCompile}(G)$ outputs a deterministic implementation $\mathsf{Hardware}_{\mathrm{det}}(\mathcal{C}, G) \stackrel{\mathrm{def}}{=} \langle G^{*,\Sigma,\mathcal{E}nc,\mathcal{D}ec}, \mathcal{E}nc(\Sigma, s \circ r)\rangle$ that works as follows:

On input x:

- G^* first decodes $\mathcal{E}nc(\Sigma, s \circ \mathsf{seed})$ to obtain $s \circ \mathsf{seed}$. Recall that the decoding scheme $\mathcal{D}ec$ is deterministic.
- Then G^* computes $\mathsf{seed}' \circ r_1 \circ r_2 \leftarrow g(\mathsf{seed})$, where $\mathsf{seed}' \in \{0,1\}^k$, and $r_1, r_2 \in \{0,1\}^\ell$.
- G^* calculates $(s_{\mathsf{new}}, y) \leftarrow G(s, x)$ (using the string r_1 as a random tape if G is randomized), then outputs y, and updates the state to be s_{new}.
- G^* calculates the encoding of $s' \circ \mathsf{seed}'$ using the string r_2 as a random tape. Then it stores the new encoding $\mathcal{E}nc(\Sigma, s_{\mathsf{new}} \circ \mathsf{seed}')$.

In this implementation $\mathsf{Hardware}_{\mathrm{det}}$, we only use truly random coins when initializing the device, and then we update it deterministically afterwards. Let us show that the implementation $\mathsf{Hardware}_{\mathrm{det}}(\mathcal{C}, G)$ is also secure against $\mathcal{F}^{\mathsf{half}}$ tampering and $\mathcal{G}^{\mathsf{half}}_{t,\mathsf{all}}$ leakage. We prove the following theorem.

Theorem 79 *Let $t : \mathbb{N} \to \mathbb{N}$ be some non-decreasing polynomial, and $\mathcal{G}_t, \mathcal{F}^{\mathsf{half}}, \mathcal{G}^{\mathsf{half}}_{t,\mathsf{all}}$ be as defined in the previous section.*

Suppose we are given a crypto system $\mathcal{E} = (\text{KeyGen}, \text{Encrypt}, \text{Decrypt})$ *that is semantically secure against one-time leakage* $\mathcal{G}_t$*; a robust NIZK* $\Pi = (\ell, \mathcal{P}, \mathcal{V}, \mathcal{S})$*; and* $\mathcal{H}_k : \{h_z : \{0,1\}^{\text{poly}(k)} \to \{0,1\}^k\}_{z \in \{0,1\}^k}$*, a family of universal one-way hash functions. Then the deterministic hardware implementation presented above is secure against* $\mathcal{F}^{\mathsf{half}}$ *tampering and* $\mathcal{G}^{\mathsf{half}}_{t,\mathsf{all}}$ *leakage.*

Combining the above theorem and Theorem 62, we are obtain the following corollary.

Corollary 80 *Under the decisional Diffie-Hellman assumption and the existence of robust NIZK, for any polynomial* $t(\cdot)$*, there exists a coding scheme with the deterministic hardware implementation presented above that is secure against* $\mathcal{F}^{\mathsf{half}}$ *tampering and* $\mathcal{G}^{\mathsf{half}}_{t,\mathsf{all}}$ *leakage.*

To show this theorem, we need to construct a simulator $\mathcal{S}im$ such that for any non-uniform PPT adversary A, any efficient interactive stateful functionality G, any state s we have the experiment $\mathsf{Real}(A, s) \approx_c \mathsf{Ideal}(\mathcal{S}im, A, s)$. Recall that $\mathsf{Real}(A, s)$ is the view of the adversary when interacting with $\mathsf{Hardware}_{\text{det}}(\mathcal{C}, G)$. We will show that the simulator constructed in the proof of Theorem 75 provides a good simulation for this case as well.

First, we define a related modification of the implementation. For any interactive stateful system $\langle G, s\rangle$, define $\langle \tilde{G}, s \circ s'\rangle$ as the system that takes the state $s \circ s'$ and outputs $G(s, x)$, for any state $s \in \{0,1\}^k$, $s' \in \{0,1\}^k$, and input x. I.e. $\tilde{G}$ simply ignores the second part of the state, and does what G does on the first half of its input.

Claim 81 *For any efficient interactive stateful functionality* G*, any state* s*, any non-uniform* PPT *adversary* A*, the following two distributions are computationally indistinguishable: (1)* A*'s view when interacting with* $\mathsf{Hardware}_{\text{rand}}(\mathcal{C}, \tilde{G}) \stackrel{\text{def}}{=} \langle \tilde{G}^{\Sigma, \mathcal{E}nc, \mathcal{D}ec}, \mathcal{E}nc(\Sigma, s \circ 0^k)\rangle$ *(running* Execute*,* Tamper*, and* Leak *queries), and (2)* A*'s view when interacting with* $\mathsf{Hardware}_{\text{det}}(\mathcal{C}, G) \stackrel{\text{def}}{=} \langle G^{*, \Sigma, \mathcal{E}nc, \mathcal{D}ec}, \mathcal{E}nc(\Sigma, s \circ r)\rangle$*.*

Let us see why this claim is sufficient to prove Theorem 79. From Theorem 75, we know that there exists a simulator $\mathcal{S}im$ such that for any adversary A, $\mathsf{Ideal}(\mathcal{S}im, A, s \circ 0^k)$ is indistinguishable from the real experiment when A is interacting with $\mathsf{Hardware}_{\text{rand}}(\mathcal{C}, \tilde{G})$. Also since $\tilde{G}$ ignores the second half of the input, we can easily see from the construction of $\mathcal{S}im$ that $\mathsf{Ideal}(\mathcal{S}im, A, s \circ 0^k)$ who gets oracle access to $\langle \tilde{G}, s \circ 0^k\rangle$ is identical to $\mathsf{Ideal}(\mathcal{S}im, A, s)$ who gets oracle access to $\langle G, s\rangle$. Therefore, A's view when interacting with $\mathsf{Hardware}_{\text{rand}}(\mathcal{C}, \tilde{G})$ is indistinguishable from $\mathsf{Ideal}(\mathcal{S}im, A, s)$. Once we have established the claim that A cannot distinguish

from $\mathsf{Hardware}_{\mathrm{rand}}(\mathcal{C}, \tilde{G})$ from $\mathsf{Hardware}_{\mathrm{det}}(\mathcal{C}, G)$, it will follow that A's view when interacting with $\mathsf{Hardware}_{\mathrm{det}}(\mathcal{C}, G)$ is indistinguishable from $\mathsf{Ideal}(\mathcal{S}im, A, s)$. This completes the proof of the theorem.

Let us prove Claim 81. Denote by $\vec{r}$ the set of strings $\{(\mathsf{seed}_i, r_1^{(i)}, r_2^{(i)})\}_{i\in[L]}$. Let $R(i)$ be the distribution over $\vec{r}$ where for $j \leq i$, $(\mathsf{seed}_j, r_1^{(j)}, r_2^{(j)})$ are truly random; for $j > i$, we have $(\mathsf{seed}_j, r_1^{(j)}, r_2^{(j)}) = g(\mathsf{seed}_{j-1})$ where g is the pseudorandom generator.

Given an adversary A, for every $i \in [L]$, define new experiments $\mathsf{Real}_i(A, s)[R(i)]$ where the adversary is interacting with the following hybrid variant of implementation of $\langle G, s\rangle$ with the random tape $R(i)$:

- For every round j, the implementation computes $(s_{j+1}, y) \leftarrow G(s_j, x)$ using $r_1^{(j)}$ as its random tape, where s_j denotes the state at round j and similarly s_{j+1}.
- For rounds $j \leq i$, the implementation computes and stores $\mathcal{E}nc(\Sigma, s_j \circ 0^k)$ using $r_2^{(j)}$ as its random tape.
- For round $j > i$, it computes and stores $\mathcal{E}nc(\Sigma, s_j \circ \mathsf{seed}_j)$ using $r_2^{(j)}$ as its random tape.
- In the end, A outputs his view.

We define experiments $\mathsf{Real}'_i(A, s)[R(i)]$ to be the same as $\mathsf{Real}_i(A, s)[R(i)]$ except in the i-th round, the implementation computes and stores $\mathcal{E}nc(\Sigma, s_i \circ \mathsf{seed}_i)$. In the following sometimes we will omit the $A, s, R(i)$ and only write Real'_i and Real_i for the experiments if it is clear from the context.

We observe that Real_0 is the view of A when interacting with $\mathsf{Hardware}_{\mathrm{det}}(\mathcal{C}, G)$ and Real_L is the view when interacting with $\mathsf{Hardware}_{\mathrm{rand}}(\mathcal{C}, \tilde{G})$. Thus we need to show that $\mathsf{Real}_0 \approx_c \mathsf{Real}_L$. We do this by showing the following neighboring hybrid experiments are indistinguishable by the following two claims:

Claim 82 *For any non-uniform* PPT *adversary A, state s, and every $i \in [L]$, $\mathsf{Real}_{i-1} \approx_c \mathsf{Real}'_i$.*

> **Proof of claim:** This follows directly from the fact that g is a PRG. Suppose there exist A, s such that D can distinguish $\mathsf{Real}_{i-1} \approx \mathsf{Real}'_i$. Then there is a reduction that can distinguish $X = (\mathsf{seed}_i, r_1^{(i)}, r_2^{(i)})$ from $Y = g(\mathsf{seed}_{i-1})$ where X is truly random.

The reduction first simulates the first $i-1$ rounds of the experiment Real_{i-1} using truly random strings as the random tape, and then embeds the input as the random tape for round i, and then simulate the remaining rounds. By this way X will produce exactly the distribution Real'_i and Y will produce Real_{i-1}. Thus the reduction can use D to distinguish the two distributions.

□

Claim 83 *For any non-uniform* PPT *adversary A, state s, and every $i \in [L]$, $\mathsf{Real}'_i \approx_c \mathsf{Real}_i$.*

Proof of claim: Before proving the claim, first we make the following observations. Given any adversary A, let $\text{view}^A_{0^k}$ denote A's view when interacting with $\mathsf{Hardware}_{\text{rand},0^k} \stackrel{\text{def}}{=} \langle \tilde{G}^{\Sigma,\mathcal{E}nc,\mathcal{D}ec}, \mathcal{E}nc(s \circ 0^k)\rangle$; let $\text{view}^A_{\mathsf{seed}}$ denote A's view when interacting with $\mathsf{Hardware}_{\text{rand},\mathsf{seed}} \stackrel{\text{def}}{=} \langle \tilde{G}^{\Sigma,\mathcal{E}nc,\mathcal{D}ec}, \mathcal{E}nc(s \circ \mathsf{seed})\rangle$. Let $\mathcal{S}im$ be the simulator defined in the proof of Theorem 75, and $\text{view}^{\mathcal{S}im}_{0^k}$ be the output of $\mathcal{S}im$ when interacting with A and $\langle \tilde{G}, s \circ 0^k\rangle$; let $\text{view}^{\mathcal{S}im}_{\mathsf{seed}}$ be the output of $\mathcal{S}im$ when interacting with A and $\langle \tilde{G}, s \circ \mathsf{seed}\rangle$. From the construction of the simulator and the fact that $\tilde{G}$ simply ignores the second half of the input and acts as G does, we know that the in both $\text{view}^{\mathcal{S}im}_{0^k}$ and $\text{view}^{\mathcal{S}im}_{\mathsf{seed}}$, the simulator gets exactly the same distribution of input/output behavior from $\tilde{G}$. Thus, $\text{view}^{\mathcal{S}im}_{0^k}$ and $\text{view}^{\mathcal{S}im}_{\mathsf{seed}}$ are identical. Putting it together, we know that $\text{view}^A_{\mathsf{seed}} \approx_c \text{view}^{\mathcal{S}im}_{\mathsf{seed}} = \text{view}^{\mathcal{S}im}_{0^k} \approx_c \text{view}^A_{0^k}$.

Now we are ready to prove the claim. Suppose there exist a adversary A, a state s, and a distinguisher D_A that distinguishes $\mathsf{Real}'_i(A,s)[R(i)]$ from $\mathsf{Real}_i(A,s)[R(i)]$. Then we can construct a reduction B such that $\text{view}^B_{0^k}$ and $\text{view}^B_{\mathsf{seed}}$ are distinguishable. The reduction gets as input a random seed, a state s, and interacts with either $\mathsf{Hardware}_{\text{rand},0^k}$ or $\mathsf{Hardware}_{\text{rand},\mathsf{seed}}$ for a random seed. The goal of the reduction is to output a view such that a distinguisher can tell $\mathsf{Hardware}_{\text{rand},0^k}$ from $\mathsf{Hardware}_{\text{rand},\mathsf{seed}}$.

B will do the following:

- B first simulates $i-1$ rounds of the interaction of A with $\mathsf{Hardware}_{\text{rand},0^k} \stackrel{\text{def}}{=} \langle \tilde{G}^{\Sigma,\mathcal{E}nc,\mathcal{D}ec}, \mathcal{E}nc(\Sigma, s \circ 0^k)\rangle$. This simulation is exactly the same distribution as the first i rounds of the interaction of $\mathsf{Real}_i(A,s)[R(i)]$, which is identical to $\mathsf{Real}'_i(A,s)[R(i)]$.
- In the i-th round, B routes A's query to the challenge device.
- Then B sets $\mathsf{seed}_i = \mathsf{seed}$.

- From the remaining rounds $j > i$, B simulates the interaction of A with the deterministic implementation $\mathsf{Hardware}_{\mathrm{det}}(\mathcal{C}, \tilde{G}) \stackrel{\mathrm{def}}{=} \langle G^{*,\Sigma,\mathcal{E}nc,\mathcal{D}ec}, \mathcal{E}nc(\Sigma, s \circ \mathsf{seed}_j)\rangle$.
- In the end, B simply outputs view_B as the output view of A.

Now we construct a distinguisher D_B as follows: on input B's output view, D_B runs $D_A(\mathrm{view}_B)$. If D_A thinks it is Real_i, then D_B outputs $\mathsf{Hardware}_{\mathrm{rand},0^k}$, otherwise $\mathsf{Hardware}_{\mathrm{rand},\mathsf{seed}}$.

To analyze the reduction, observe that if B's challenge device is $\mathsf{Hardware}_{\mathrm{rand},0^k}$, then view_B will be identical to $\mathsf{Real}_i(A, s)$; if it is $\mathsf{Hardware}_{\mathrm{rand},\mathsf{seed}}$, then view_B will be identical to $\mathsf{Real}'_i(A, s)$. Therefore, D_A can distinguish one from the other, so the reduction B produces a distinguishable view. This contradicts to the previous observation we have made.

□

Claim 81 follows from Claims 82 and 83 by a standard hybrid argument. Thus, we complete the proof to the theorem.

We remark that in the proof above, we only rely on the security of the PRG and the randomized hardware implementation. Thus, we can prove a more general statement:

Corollary 84 *Suppose a coding scheme $\mathcal{C}$ with the randomized implementation $\mathsf{Hardware}_{\mathrm{rand}}$ is secure against $\mathcal{F}$ tampering and $\mathcal{G}$ leakage where $\mathcal{F}$ and $\mathcal{G}$ are subclasses of efficient functions. Then $\mathcal{C}$ is also secure against $\mathcal{F}$ tampering and $\mathcal{G}$-leakage with the deterministic implementation $\mathsf{Hardware}_{\mathrm{det}}$ presented in this section.*

11.3 Discussion of Complexity Assumptions and Efficiency

We just showed a leakage and tampering resilient construction for any stateful functionality in the split-state model. Our construction relied on the existence of (1) a semantically secure one-time (bounded) leakage resilient encryption scheme (LRE), (2) a robust NIZK, (3) a universal one-way hash family (UOWHF), and (4) a pseudorandom generator (PRG). In terms of the complexity assumptions that we need to make for these four building blocks to exist, we note that UOWFHs and PRGs exist if and only if one-way functions (OWFs). (Rompel [Rom90] showed that (OWFs)

imply UOWHFs and Håstad et al. [HILL99] showed OWFs imply PRGs; and both UOWFGs and PRGs imply OWFs); thus both UOWHFs and PRGs are implied by the existence of a semantically secure cryptosystem. So we are left with assumptions (1) and (2).

It is not known how LRE relates to robust NIZK. No construction of LRE is known from general assumptions such as the existence of trapdoor permutations (TDPs). LRE has been proposed based on specific assumptions such as the decisional Diffie-Hellman assumption (DDH) and its variants, or the learning with error assumption (LWE) and its variants [AGV09, NS09, ADW09, KV09]. Robust NIZK [DDO$^+$01] has been shown based on the existence of dense cryptosystems (i.e. almost every string can be interpreted as a public key for this system), and a multi-theorem NIZK, which in turn has been shown from TDPs [KP98, FLS99] or verifiable unpredictable functions [GO92, Lys02].

Note that using general NIZK for all NP from TDPs may not be desirable in practice because those constructions rely on the Cook-Levin reduction. Therefore, finding a more efficient NIZK for the specific language we use is desirable. Note that, if we use the DDH-based Naor-Segev cryptosystem, then the statement that needs to be proved using the robust NIZK scheme is just a statement about relations between group elements and their discrete logarithms. Groth [Gro06] gives a robust NIZK for proving relations among group elements (based on the XDH assumption which is stronger than DDH), and in combination with a technique due to Meiklejohn [Mei09] it can be used as a robust NIZK for also proving knowledge of discrete logarithms of these group elements. Groth's proof system's efficiency is a low-degree polynomial in the security parameter, unlike the general NIZK constructions. Therefore, we get a construction that is more suitable for practical use.

Bibliography

[AARR02] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The em side-channel(s). In *CHES*, pages 29–45, 2002.

[ADL13] Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. Cryptology ePrint Archive, Report 2013/201, 2013. `http://eprint.iacr.org/`.

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS*, pages 106–115, 2001.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349, 2012.

[BCE+08] Mira Belenkiy, Melissa Chase, C. Christopher Erway, John Jannotti, Alptekin Küpçü, and Anna Lysyanskaya. Incentivizing outsourced computation. In *NetEcon*, pages 85–90, 2008.

[BG02] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity*, pages 194–203, 2002.

[BGV11] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, pages 111–131, 2011.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[BIN97] Mihir Bellare, Russell Impagliazzo, and Moni Naor. Does parallel repetition lower the error in computationally sound protocols? In *FOCS*, pages 374–383, 1997.

[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.

[BR97] Mihir Bellare and Phillip Rogaway. Minimizing the use of random oracles in authenticated encryption schemes. In *ICICS*, pages 1–16, 1997.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO*, pages 868–886, 2012.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, pages 513–525, 1997.

[BSGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Short pcps verifiable in polylogarithmic time. In *IEEE Conference on Computational Complexity*, pages 120–134, 2005.

[BSW03] Boaz Barak, Ronen Shaltiel, and Avi Wigderson. Computational analogues of entropy. In *RANDOM-APPROX*, pages 200–215, 2003.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *FOCS*, pages 97–106, 2011.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM*, 51(4):557–594, 2004.

[CHS05] Ran Canetti, Shai Halevi, and Michael Steiner. Hardness amplification of weakly verifiable puzzles. In *TCC*, pages 17–33, 2005.

[CKKC13] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Carlos Cid. Multi-client non-interactive verifiable computation. In *TCC*, pages 499–518, 2013.

[CKM11] Seung Geol Choi, Aggelos Kiayias, and Tal Malkin. Bitr: Built-in tamper resilience. In *ASIACRYPT*, pages 740–758, 2011.

[CKV10] Kai-Min Chung, Yael Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.

[CTY10] G. Cormode, J. Thaler, and K. Yi. Verifying computations with streaming interactive proofs. Technical Report TR10-159, ECCC Report, 2010.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.

[DDO+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.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, pages 54–74, 2012.

[DHLAW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[DKO13] Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Non-malleable codes from two-source extractors. Unpublished Manusript, 2013.

[DLWW11] Yevgeniy Dodis, Allison Lewko, Brent Waters, and Daniel Wichs. Storing secrets on continually leaky devices. Cryptology ePrint Archive, Report 2011/369, 2011. `http://eprint.iacr.org/`.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 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.

[DPW10] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *ICS*, pages 434–452, 2010.

[DR06] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the pcp theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006.

[FL93] Lance Fortnow and Carsten Lund. Interactive proof systems and alternating time-space complexity. *Theoretical Computer Science*, 113(1):55–73, 1993.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999.

[FPV11] Sebastian Faust, Krzysztof Pietrzak, and Daniele Venturi. Tamper-proof circuits: How to trade leakage for tamper-resilience. In *ICALP (1)*, pages 391–402, 2011.

[FR11] Benjamin Fuller and Leonid Reyzin. Computational entropy and information leakage. *Manuscript.*, 2011.

[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.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the fiat-shamir paradigm. pages 102–113, 2003.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[GLM+04] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (atp) security: Theoretical foundations for security against hardware tampering. In *TCC*, pages 258–277, 2004.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.

[GO92] Shafi Goldwasser and Rafail Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent (extended abstract). In *CRYPTO*, pages 228–245, 1992.

[GR10] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, pages 59–79, 2010.

[Gro06] Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *ASIACRYPT*, pages 444–459, 2006.

[HHR+10] Iftach Haitner, Thomas Holenstein, Omer Reingold, Salil P. Vadhan, and Hoeteck Wee. Universal one-way hash functions via inaccessible entropy. In *EUROCRYPT*, pages 616–637, 2010.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[HL05] Susan Hohenberger and Anna Lysyanskaya. How to securely outsource cryptographic computations. In *TCC*, pages 264–282, 2005.

[HL11] Shai Halevi and Huijia Lin. After-the-fact leakage in public-key encryption. In *TCC*, pages 107–124, 2011.

[HLAWW12] Carmit Hazay, Adriana Lopez-Alt, Hoeteck Wee, and Daniel Wichs. Leakage-resilient cryptography from minimal assumptions. Cryptology ePrint Archive, Report 2012/604, 2012. `http://eprint.iacr.org/`.

[HLR07] Chun-Yuan Hsiao, Chi-Jen Lu, and Leonid Reyzin. Conditional computational entropy, or toward separating pseudoentropy from compressibility. In *EUROCRYPT*, pages 169–186, 2007.

[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.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits ii: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[JV10] Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO*, pages 41–58, 2010.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

[KKS11] Yael Tauman Kalai, Bhavana Kanukurthi, and Amit Sahai. Cryptography with tamperable and leaky memory. In *CRYPTO*, pages 373–390, 2011.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.

[KP98] Joe Kilian and Erez Petrank. An efficient noninteractive zero-knowledge proof system for np with general assumptions. *J. Cryptology*, 11(1):1–27, 1998.

[KR09] Yael Tauman Kalai and Ran Raz. Probabilistically checkable arguments. In *CRYPTO*, 2009.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[LL10] Feng-Hao Liu and Anna Lysyanskaya. Algorithmic tamper-proof security under probing attacks. In *SCN*, pages 106–120, 2010.

[LLW11] Allison B. Lewko, Mark Lewko, and Brent Waters. How to leak on key updates. In *STOC*, pages 725–734, 2011.

[LRW11] Allison B. Lewko, Yannis Rouselakis, and Brent Waters. Achieving leakage resilience through dual system encryption. In *TCC*, pages 70–88, 2011.

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the dh-ddh separation. In *CRYPTO*, pages 597–612, 2002.

[Mei09] Sarah Meiklejohn. An extension of the Groth-Sahai proof system. *Master's Thesis*, 2009.

[Mic94] Silvio Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453, 1994.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *CRYPTO*, pages 96–109, 2003.

[Neu28] John Von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437, 1990.

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *TCC*, pages 222–242, 2013.

[PTT11] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. In *CRYPTO*, pages 91–110, 2011.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, pages 387–394, 1990.

[RTTV08] Omer Reingold, Luca Trevisan, Madhur Tulsiani, and Salil P. Vadhan. Dense subsets of pseudorandom sets. In *FOCS*, pages 76–85, 2008.

[RW05] Renato Renner and Stefan Wolf. Simple and tight bounds for information reconciliation and privacy amplification. In *ASIACRYPT*, pages 199–216, 2005.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553, 1999.

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

[Vad10] Salil Vadhan. Psuedorandomness. *Book draft, available at "http://people.seas.harvard.edu/~salil/pseudorandomness/pseudorandomness-Aug10.pdf"*, 2010.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.