Tamper and Leakage Resilience in the Split-State Model

Feng-Hao Liu
Brown University
fenghao@cs.brown.edu

Anna Lysyanskaya
Brown University
anna@cs.brown.edu

May 28, 2012

Abstract

It is notoriously difficult to create hardware that is immune from side channel and tampering attacks. A lot of recent literature, therefore, has instead considered *algorithmic* defenses from such attacks.

In this paper, we show how to algorithmically secure any cryptographic functionality from continual split-state leakage and tampering attacks. A split-state attack on cryptographic hardware is one that targets separate parts of the hardware separately. Our construction does not require the hardware to have access to randomness. In contrast, prior work on protecting from continual combined leakage and tampering [28] required true randomness for each update. Our construction is in the common reference string (CRS) model; the CRS must be hard-wired into the device. We note that prior negative results show that it is impossible to algorithmically secure a cryptographic functionality against a combination of arbitrary continual leakage and tampering attacks without true randomness; therefore restricting our attention to the split-state model is justified.

Our construction is simple and modular, and relies on a new construction, in the CRS model, of non-malleable codes with respect to split-state tampering functions, which may be of independent interest.

1 Introduction

Recently, the cryptographic community has been extensively studying 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 [31, 22]. 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 [4, 1]. What can be done about protecting the security of the functionality of the device?

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, Liu and Lysyanskaya [34] showed that the adversary will eventually learn the entire secret state. Further, even with tampering alone, Gennaro et al. [17] 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 [5, 8, 33, 32]. 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 [2], where the attacker can learn arbitrary information about the secret, as long as the total amount is bounded by some prior parameter [2, 38, 3, 29].

If only tampering is allowed, Gennaro et al. [17] 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. [13] 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. Very recently, Choi, Kiayias, and Malkin [6] 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 [28]. 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 [34]. 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 this paper, 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 [12, 9] and tampering-only [13] attacks.

Our main result. 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 [38] relies on DDH and is good enough for our purposes; in fact it gives more security than we require.

(2) The existence of robust non-interactive zero-knowledge proof systems for an appropriate NP language. For example, de Santis et al.'s [7] construction of robust NIZK for all languages in NP suffices; although a construction for a more specialized language suffices as well.

In Section 5 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 [20] and a technique due to Meiklejohn [36], 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. [13] 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.

Prior work. 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").

Type of leakage	Type of tampering	Local coins	Known results about continual attacks
None	Any	No	Signature and decryption in the CRS model [17]
Any	None	No	Trivially impossible
Bits	Any	No	Impossible [34]
Any	None	Yes	Signature and encryption in the plain model [5, 8, 33, 32]
None	Bits	Yes	All functionalities in the plain model [13]
None	SS	Yes	All functionalities in the RO model [13]
None	Blocks	Yes	All functionalities in the plain model [6]
Any	Any	Yes	Signature and encryption in the CRS model [28]
SS	SS	No	All functionalities in the CRS model [**This work**]

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 [32] 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 [26, 37, 25, 12, 40, 10, 15, 19, 27, 14]. 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 [19], or Juma and Vahlis [27] 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 [23] 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 [5, 33, 32, 28], even if one does not care about trusted local randomness and tamper-resilience.

Our building block: non-malleable codes We use non-malleable codes, defined by Dziembowski et al. [13], as our building block.

Let $\mathcal{E}nc$ be an encoding procedure and $\mathcal{D}ec$ be the corresponding decoding procedure. Consider the following tampering experiment [13]: (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 [13] 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.

Our continual tampering and leakage model. We consider the same tampering and leakage attacks as those of Liu and Lysyanskaya[34] and Kalai et al. [28], which generalized the model of tampering-only [17, 13] and leakage-only [5, 8, 33, 32] attacks. (However, in this attack model we achieve stronger security, as discussed above.)

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 [13]. 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 5.

Our approach. Let $G(s,x)$ be the functionality we want to secure, where s is some secret state and x is the user input. Our compiler takes the leakage-resilient non-malleable code and 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.

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 1 *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 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 2 (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 3 *Let* CRS *be an algorithm that generates a common reference string, on input the security parameter* 1^k*. The algorithms* (CircuitCompile, 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* 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,* CircuitCompile *and* 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 [19]). This definition allows for both randomized and deterministic G'; as we discussed in the introduction, in general a deterministic circuit is more desirable.

Remark 4 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 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 Execute, Tamper and Leak queries, can be learned just from the 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 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 5 (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 $\mathrm{view}_A = (state_A, x_1, y_1, \ell_1, x_2, y_2, \ell_2, \ldots,)$ *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* $(\Sigma, \mathrm{view}_A)$.

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 $(\Sigma, \mathrm{view}_{\mathcal{S}im}) = (\Sigma, (state_{\mathcal{S}im}, x_1, y_1, \ell_1, x_2, y_2, \ell_2, \ldots))$, 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. [13] 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.

3 Leakage Resilient Non-malleable Codes

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. [13] 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 [13]. First we define a coding scheme in the plain model and in the CRS model.

3.1 Definition

Definition 6 (Coding Scheme [13]) *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 7 (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) $\Sigma \in \{0,1\}^{\text{poly}(k)}$. 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 8 (Strong Non-malleability [13]) *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*

$$\mathsf{Tamper}_s^f \stackrel{\text{def}}{=} \left\{ \begin{array}{c} c \leftarrow \mathcal{E}nc(s), \tilde{c} = f(c), \tilde{s} = \mathcal{D}ec(\tilde{c}) \\ Output : \mathsf{same}^* \text{ if } \tilde{c} = c, \text{ and } \tilde{s} \text{ otherwise.} \end{array} \right\}$$

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

$$\mathsf{Tamper}_s^{f,\Sigma} \stackrel{\text{def}}{=} \left\{ \begin{array}{c} c \leftarrow \mathcal{E}nc(\Sigma, s), \tilde{c} = f^{\Sigma}(c), \tilde{s} = \mathcal{D}ec(\Sigma, \tilde{c}) \\ Output : \ \mathsf{same}^* \text{ if } \tilde{c} = c, \text{ and } \tilde{s} \text{ otherwise.} \end{array} \right\}.$$

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 9 (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 $\Sigma \leftarrow \mathcal{I}nit(1^k)$.*

3.2 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. [17]: 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 [11]) 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 [39] and Dolev et al. [11] give us a good starting point. In particular, both 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. [7], 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.

3.3 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 26 and 23 in Appendix A). 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 Appendix A 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 $\mathbf{L}$.

Let $\pi \leftarrow \mathcal{P}((\mathsf{pk}, \hat{s}), (\mathsf{sk}, s, r), \Sigma)$ be an NIZK proof computed by the prover's strategy of the proof system Π with CRS Σ 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 10 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 11 *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 8) with respect to $\mathcal{F}^{\mathsf{half}}$, and leakage resilient (Def 9) 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 $(\Sigma, \mathsf{Tamper}^{f,\Sigma}_{s_0}) \approx_c (\Sigma, \mathsf{Tamper}^{f,\Sigma}_{s_1})$ 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 $\varepsilon = 1/\text{poly}(k)$, and a distinguisher D such that $\Pr[D(\Sigma, \mathsf{Tamper}^{f,\Sigma}_{s_0}) = 1] - \Pr[D(\Sigma, \mathsf{Tamper}^{f,\Sigma}_{s_1}) = 1] > \varepsilon$, 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 $\mathsf{Tamper}^{f,\Sigma,\tau}_s$ be the same game as $\mathsf{Tamper}^{f,\Sigma}_s$ 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^{f*}_s. We claim that D also distinguishes $\mathsf{Tamper}^{f*}_{s_0}$ from $\mathsf{Tamper}^{f*}_{s_1}$.

Suppose not, i.e. D, who distinguishes $\mathsf{Tamper}^{f,\Sigma}_{s_0}$ from $\mathsf{Tamper}^{f,\Sigma}_{s_1}$ does not distinguish $\mathsf{Tamper}^{f*}_{s_0}$ from $\mathsf{Tamper}^{f*}_{s_1}$. 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 $\Pr[D(\Sigma, \mathsf{Tamper}^{f*}_{s_0}) = 1] - \Pr[D(\Sigma, \mathsf{Tamper}^{f*}_{s_1}) = 1] > \varepsilon/2$.

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 26, 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 $\hat{m} = \text{Encrypt}_{\mathsf{pk}}(m_b)$ 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 $(\mathsf{sk}'', m'') \leftarrow \text{Ext}(\Sigma, \tau, x' = (\mathsf{pk}', \hat{m}'), \pi')$, 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 $(\mathsf{sk}'', m'') \leftarrow \text{Ext}(\Sigma, \tau, x' = (\mathsf{pk}', \hat{m}'), \pi')$, 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 12 *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 26 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 27 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.

■

4 Our Compilers

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.

4.1 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 13 *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 13] 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 $\mathsf{Tamper}^{f,\Sigma}_{0^k}$. 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 $g \in \mathcal{G}^{\mathsf{half}}_{t,\mathsf{all}}$, 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 14 *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 $L = \text{poly}(k)$ rounds, a state s, and a distinguisher D such that $\Pr[D(\Sigma, \text{view}_{Real}) = 1] - \Pr[D(\Sigma, \text{view}_{\mathcal{S}im}) = 1] > \varepsilon$ 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 11.

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 (Σ, view_A), and $\mathcal{S}im^{(L)}(A,s)$ is that of the ideal experiment $(\Sigma, \text{view}_{\mathcal{S}im})$. 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 15 *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 16 $\Pr[E_4] > \alpha$ *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 $\mathsf{pk}, \hat{m}_b = \text{Encrypt}_{\mathsf{pk}}(m_b), g_1(\mathsf{sk})$, 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 $\text{view}'_{\mathcal{S}im}$ to be that view.
- In the end, B outputs $D(\Sigma, \text{view}'_{\mathcal{S}im})$: 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 5. 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

$$
\begin{aligned}
&\Pr\left[D'(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma}) = 1\right] - \Pr\left[D'(\Sigma, \mathsf{Tamper}_{s_1}^{f,\Sigma}) = 1\right] \\
&= \left(\Pr\left[D'(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma}) = 1 \middle| \neg\mathsf{Bad}\right] \cdot \Pr[\neg\mathsf{Bad}] + \Pr\left[D'(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma}) = 1 \middle| \mathsf{Bad}\right] \cdot \Pr[\mathsf{Bad}]\right) - \\
&\quad \left(\Pr\left[D'(\Sigma, \mathsf{Tamper}_{s_1}^{f,\Sigma}) = 1 \middle| \neg\mathsf{Bad}\right] \cdot \Pr[\neg\mathsf{Bad}] + \Pr\left[D'(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma}) = 1 \middle| \mathsf{Bad}\right] \cdot \Pr[\mathsf{Bad}]\right) \\
&= (\Pr[D(\Sigma, \text{view}_0) = 1] - \Pr[D(\Sigma, \text{view}_1) = 1]) \cdot \Pr[\neg\mathsf{Bad}] \\
&= \left(\Pr\left[D(\Sigma, \mathcal{S}im^{(j)}) = 1\right] - \Pr\left[D(\Sigma, \mathcal{S}im^{(j+1)}) = 1\right]\right) \cdot \Pr[\neg\mathsf{Bad}] \\
&\geq \varepsilon/L \cdot \alpha, \text{ a non-negligible quantity.}
\end{aligned}
$$

This completes the proof of the lemma.

■

This proof of the theorem follows directly from the construction of $\mathcal{S}im$ and the lemma.

■

4.2 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 17 *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 $\mathcal{E} = (\mathrm{KeyGen}, \mathrm{Encrypt}, \mathrm{Decrypt})$ 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 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 28, we are obtain the following corollary.

Corollary 18 *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 $\mathsf{Hardware}_{\mathrm{det}}(\mathcal{C}, G)$. We will show that the simulator constructed in the proof of Theorem 13 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 19 *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 17. From Theorem 13, 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 $\mathsf{Hardware}_{\text{rand}}(\mathcal{C}, \tilde{G})$ from $\mathsf{Hardware}_{\text{det}}(\mathcal{C}, G)$, it will follow that A's view when interacting with $\mathsf{Hardware}_{\text{det}}(\mathcal{C}, G)$ is indistinguishable from $\mathsf{Ideal}(\mathcal{S}im, A, s)$. This completes the proof of the theorem.

Let us prove Claim 19. 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 $\mathsf{Hardware}_{\text{det}}(\mathcal{C}, G)$ and Real_L is the view when interacting with $\mathsf{Hardware}_{\text{rand}}(\mathcal{C}, \tilde{G})$. 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 20 *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 21 *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 13, 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 $\mathsf{Hardware}_{\text{det}}(\mathcal{C}, \tilde{G}) \stackrel{\text{def}}{=} \langle G^{*,\Sigma,\mathcal{E}nc,\mathcal{D}ec}, \mathcal{E}nc(\Sigma, s \circ \mathsf{seed}_j) \rangle$.
- 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 $D_A(\text{view}_B)$. If D_A thinks it is Real_i, then D_B outputs $\mathsf{Hardware}_{\text{rand},0^k}$, otherwise $\mathsf{Hardware}_{\text{rand},\mathsf{seed}}$.

To analyze the reduction, observe that if B's challenge device is $\mathsf{Hardware}_{\text{rand},0^k}$, then view_B will be identical to $\mathsf{Real}_i(A, s)$; if it is $\mathsf{Hardware}_{\text{rand},\mathsf{seed}}$, 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 19 follows from Claims 20 and 21 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 22 *Suppose a coding scheme $\mathcal{C}$ with the randomized implementation* $\mathsf{Hardware}_{\text{rand}}$ *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* $\mathsf{Hardware}_{\text{det}}$ *presented in this section.*

5 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 [41] showed that (OWFs) imply UOWHFs and Håstad et al. [24] 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 [2, 38, 3, 29]. Robust NIZK [7] 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 [30, 16] or verifiable unpredictable functions [18, 35].

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 [20] 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 [36] 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.

Acknowledgement. Anna Lysyanskaya was supported by NSF grants 1012060, 0964379, 0831293, and by a Sloan Foundation fellowship. Feng-Hao Liu is supported by National Science Foundation for partial support under grant CNS-0347661 and CNS-0831293.

References

[1] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The em side-channel(s). In *CHES*, pages 29–45, 2002.

[2] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[3] J. Alwen, Y. Dodis, and D. Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[4] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, pages 513–525, 1997.

[5] Z. Brakerski, Y. T. Kalai, J. Katz, and V. Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *FOCS*, pages 501–510, 2010.

[6] S. G. Choi, A. Kiayias, and T. Malkin. Bitr: Built-in tamper resilience. In *ASIACRYPT*, pages 740–758, 2011.

[7] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, pages 566–598, 2001.

[8] Y. Dodis, K. Haralambiev, A. López-Alt, and D. Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[9] Y. Dodis, A. Lewko, B. Waters, and D. Wichs. Storing secrets on continually leaky devices. Cryptology ePrint Archive, Report 2011/369, 2011. `http://eprint.iacr.org/`.

[10] Y. Dodis and K. Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on Feistel networks. In *CRYPTO*, pages 21–40, 2010.

[11] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.

[12] S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[13] S. Dziembowski, K. Pietrzak, and D. Wichs. Non-malleable codes. In *ICS*, pages 434–452, 2010.

[14] S. Faust, K. Pietrzak, and D. Venturi. Tamper-proof circuits: How to trade leakage for tamper-resilience. In *ICALP (1)*, pages 391–402, 2011.

[15] S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT*, pages 135–156, 2010.

[16] U. Feige, D. Lapidot, and A. Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999.

[17] R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, and T. Rabin. Algorithmic tamper-proof (atp) security: Theoretical foundations for security against hardware tampering. In *TCC*, pages 258–277, 2004.

[18] S. Goldwasser and R. Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent (extended abstract). In *CRYPTO*, pages 228–245, 1992.

[19] S. Goldwasser and G. N. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, pages 59–79, 2010.

[20] J. Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *ASIACRYPT*, pages 444–459, 2006.

[21] I. Haitner, T. Holenstein, O. Reingold, S. P. Vadhan, and H. Wee. Universal one-way hash functions via inaccessible entropy. In *EUROCRYPT*, pages 616–637, 2010.

[22] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security Symposium*, pages 45–60, 2008.

[23] S. Halevi and H. Lin. After-the-fact leakage in public-key encryption. In *TCC*, pages 107–124, 2011.

[24] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[25] Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner. Private circuits ii: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.

[26] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[27] A. Juma and Y. Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO*, pages 41–58, 2010.

[28] Y. T. Kalai, B. Kanukurthi, and A. Sahai. Cryptography with tamperable and leaky memory. In *CRYPTO*, pages 373–390, 2011.

[29] J. Katz and V. Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[30] J. Kilian and E. Petrank. An efficient noninteractive zero-knowledge proof system for np with general assumptions. *J. Cryptology*, 11(1):1–27, 1998.

[31] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.

[32] A. B. Lewko, M. Lewko, and B. Waters. How to leak on key updates. In *STOC*, pages 725–734, 2011.

[33] A. B. Lewko, Y. Rouselakis, and B. Waters. Achieving leakage resilience through dual system encryption. In *TCC*, pages 70–88, 2011.

[34] F.-H. Liu and A. Lysyanskaya. Algorithmic tamper-proof security under probing attacks. In *SCN*, pages 106–120, 2010.

[35] A. Lysyanskaya. Unique signatures and verifiable random functions from the dh-ddh separation. In *CRYPTO*, pages 597–612, 2002.

[36] S. Meiklejohn. An extension of the Groth-Sahai proof system. *Master's Thesis*, 2009.

[37] S. Micali and L. Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.

[38] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[39] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437, 1990.

[40] K. Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[41] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, pages 387–394, 1990.

A Definitions

In this section, we formally define the tools we need for the construction. Recall that 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 23 (Robust NIZK [7]) $\Pi = (\ell, \mathcal{P}, \mathcal{V}, \mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2))$ *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 24 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 [7], 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 25 (Universal One-way Hash Functions - UOWHF [21]) *A family of functions* $H_k = \{h_z : \{0,1\}^{n(k)} \rightarrow \{0,1\}^k\}_{z \in \{0,1\}^k}$ *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* $\text{poly}(k, n(k))$*.*
- **(Compressing):** *For all* k*,* $k \leq n(k)$*.*
- **(Universal One-way):** *For any non-uniform* PPT *adversary* A*, there exists a negligible function* $\mathsf{ngl}(\cdot)$*:*

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

Definition 26 (One-time Leakage Resilient Encryption [2]) *Let* $\mathcal{E} = (\text{KeyGen}, \text{Encrypt}, \text{Decrypt})$ *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 3 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 27 *Let $\mathcal{E} = (\mathrm{KeyGen}, \mathrm{Encrypt}, \mathrm{Decrypt})$ be a leakage resilient encryption scheme that allows $t(k)$-bit leakage for $t(k) > k$, and $\mathcal{H}_k : \{h_z : \{0,1\}^{\mathrm{poly}(k)} \to \{0,1\}^k\}_{s \in \{0,1\}^k}$ be a family of universal one-way hash functions.*

Then there exists an encryption scheme $\mathcal{E}' = (\mathrm{KeyGen}', \mathrm{Encrypt}', \mathrm{Decrypt}')$ 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 28 ([38]) *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$\}$.*

[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.