Tamper and Leakage Resilience in the Split-State Model

Feng-Hao Liu and Anna Lysyanskaya

Brown University
{fenghao,anna}@cs.brown.edu

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 [23] 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 [25,18]. 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

R. Safavi-Naini and R. Canetti (Eds.): CRYPTO 2012, LNCS 7417, pp. 517–532, 2012.
© International Association for Cryptologic Research 2012

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 [28] showed that the adversary will eventually learn the entire secret state. Further, even with tampering alone, Gennaro et al. [16] 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.

For the leakage-only case, positive results are known for continual attacks assuming an on-device source or randomness [5,8,27,26]. The one-time leakage case has also been studied [2,30,3,24]. For the tampering-only case, positive results are known as well for different setup and tampering models [16,13,6].

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 [23]. 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 [28]. 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, but with access to a trusted common reference string (CRS). 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 assumes the existence of a one-time leakage resilient public-key cryptosystem that allows leakage of any poly-time computable $g(\mathsf{sk})$ of length $c|\mathsf{sk}|$ for some constant c, for example one due to Naor and Segev [30]. Further, we need robust non-interactive zero-knowledge proof systems [7] for an appropriate NP language. See the full version of this paper for further detailed discussions.

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 [16]
Any	None	No	Trivially impossible
Bits	Any	No	Impossible [28]
Any	None	Yes	Signature and encryption in the plain model [5,8,27,26]
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 [23]
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 [26] 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 [21,29,20,12,31,10,15,17,22,14]. 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 [17], or Juma and Vahlis [22] 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.

A decryption device produced by our compiler will have stronger leakage resilience properties than most previous work [5,27,26,23] on leakage resilient encryption: it will tolerate *after-the-fact* leakage defined by Halevi and Lin [19]; since this will be guaranteed on a continual basis, our results solve a problem left explicitly open by Halevi and Lin.

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. Third, we prove that plugging in a leakage-resilient non-malleable code in the Dziembowski et al. compiler secures any functionality against *both* tampering and leakage attacks. Fourth, we *derandomize* the compiled construction such that it no longer requires a trusted source of randomness for updates.

Our continual tampering and leakage model. We consider the same tampering and leakage attacks as those of Liu and Lysyanskaya[28] and Kalai et al. [23], which generalized the model of tampering-only [16,13] and leakage-only [5,8,27,26] 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. We consider a simulation-based definition of security against such attacks.

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

Definition 1. *Define the following three function classes* $\mathcal{G}_t, \mathcal{F}^{\mathsf{half}}, \mathcal{G}^{\mathsf{half}}_{t_1,t_2}$*:*

- *Let* $t \in \mathbb{N}$*. By* $\mathcal{G}_t$ *we denote the set of poly-sized circuits with output length* t*.*
- *Let* $\mathcal{F}^{\mathsf{half}}$ *denote the set of functions of the following form:* $f : \{0,1\}^{2m} \to \{0,1\}^{2m} \in \mathcal{F}^{\mathsf{half}}$ *if there exist two poly-sized circuits* $f_1, f_2 : \{0,1\}^m \to \{0,1\}^m$*, 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* $\mathcal{G}^{\mathsf{half}}_{t_1,t_2}$ *be the set of all poly-sized leakage functions that leak independently on each half of their inputs,* t_1 *bits on the first half and* t_2 *bits on the second half.*
  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*.*

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 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:*

- $\mathsf{Execute}(x)$*: For* $x \in \{0,1\}^u$*, the system will compute* $(y, s_{\mathsf{new}}) \leftarrow G(s, x)$*, privately update state to* s_{new}*, and output* y*.*
- $\mathsf{Tamper}(f)$*: the state* s *is replaced by* $f(s)$*.*
- $\mathsf{Leak}(g)$*: the adversary can obtain the information* $g(s)$*.*

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* ($\mathsf{CircuitCompile}$, $\mathsf{MemCompile}$) *constitute a* correct *and* efficiency-preserving *compiler in the* $\mathsf{CRS}(1^k)$ *model if for all* $\Sigma \in \mathsf{CRS}(1^k)$*, for any* $\mathsf{Execute}$ *query* x*,* $\langle G', s'\rangle$*'s answer is distributed identically to* $\langle G, s\rangle$*'s answer, where* $G' = \mathsf{CircuitCompile}(\Sigma, G)$ *and* $s' \in \mathsf{MemCompile}(\Sigma, s)$*; moreover,* $\mathsf{CircuitCompile}$ *and* $\mathsf{MemCompile}$ *run in polynomial time and output* G' *and* s' *of size polynomial in the original circuit* G *and secret* s*.*

Note that this definition of the compiler ensures that the compiled functionality G' inherits all the security properties of the original functionality G. Also the compiler 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 [17]). This definition allows for both randomized and deterministic G'; as we discussed in the introduction, in general a deterministic circuit is more desirable.

Remark 1. Recall that G, and therefore G', are modeled as stateful functionalities. By convention, running $\mathsf{Execute}(\varepsilon)$ will cause them to update their states.

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

We want the real experiment where the adversary interacts with the compiled functionality $\langle G', s'\rangle$ and issues $\mathsf{Execute}$, Tamper and Leak queries, to be indistinguishable from an experiment in which a simulator $\mathcal{S}im$ only has black-box access to the original functionality G with the secret state s (i.e. $\langle G, s\rangle$). More precisely, in every round, $\mathcal{S}im$ will get some tampering function f or leakage function g from A and then respond to them. In the end, the adversary halts and outputs its view. The simulator then may (potentially) output this view. Whatever view $\mathcal{S}im$ outputs needs to be indistinguishable from the view A obtained in the real experiment. This captures the fact that the adversary's tampering and leakage attacks in the real experiment can be simulated by only accessing the functionality in a black-box way. Thus, these additional physical attacks do not give the adversary any additional power.

Definition 4 (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].

Definition 5 (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 outputs 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 $n(k)$: i.e. for each $s \in \{0,1\}^k, \Sigma$,*
$\Pr[\mathcal{D}ec(\Sigma, \mathcal{E}nc(\Sigma, s)) = s] = 1$. *For simplicity, we will omit the security parameter.*

Now we define the two properties of coding schemes: non-malleability and leakage resilience. We extend the definition of the strong non-malleability by Dziembowski et al. [13] to the CRS model.

Definition 6 (Strong Non-malleability in the CRS Model). *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 in the common reference string model. For any CRS Σ, we define*

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

where the randomness of this experiment comes from the randomness of the encoding and decoding algorithms.

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 $f \in \mathcal{F}$, and $\approx$ can refer to statistical or computational indistinguishability.

Definition 7 (Leakage Resilience). *Let $\mathcal{G}$ be some family of functions. 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 $\Sigma \leftarrow \mathcal{I}nit(1^k)$.*

How do we realize this definition? Consider a technique reminiscent of non-malleable encryption [11,32]: set $M_1 = \mathsf{sk}$, $M_2 = (\mathsf{pk}, \hat{s} = \text{Encrypt}_{\mathsf{pk}}(s), \pi)$ 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? If the underlying proof system is malleable, then it could be possible to modify both parts 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. 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 $\mathsf{Tamper}_{s_0}^f$ and $\mathsf{Tamper}_{s_1}^f$. A natural way for the reduction is to pretend that $M_1 = \mathsf{sk}$, and put

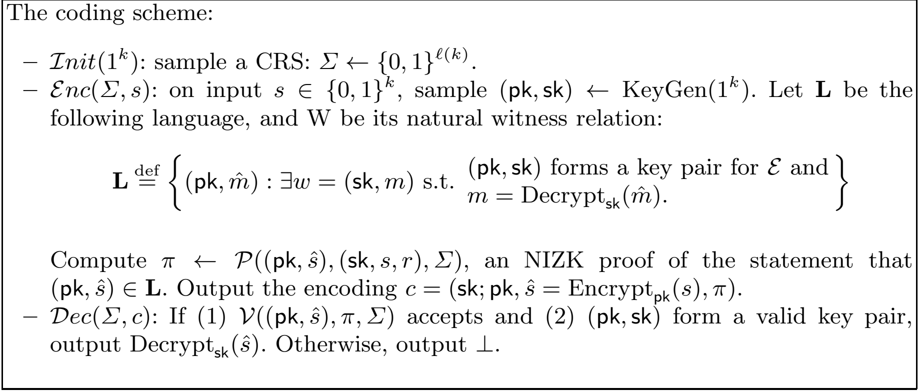
the public key pk and the ciphertext $\hat{s} = c$ with a simulated proof into M_2, setting $M_2 = (\mathsf{pk}, \hat{s}, \pi_{\mathcal{S}im})$. Then the reduction simulates Tamper_s^f. Clearly, irrespective of f_1 the reduction can compute $f_2(M_2) = (\mathsf{pk}', \hat{s}', \pi_{\mathcal{S}im})$, 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.

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, to check whether $f_1(\mathsf{sk}) = \mathsf{sk}'$, it is very likely (assuming proper collision-resistance properties of h) sufficient to check if $h(f_1(\mathsf{sk})) = h(\mathsf{sk}')$. So if 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.

The Construction. 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 (we defer the formal definitions to the full version of this paper). The encryption scheme and robust NIZK need 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 the full version of this paper, 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. We define a coding scheme $(\mathcal{I}nit, \mathcal{E}nc, \mathcal{D}ec)$ in Figure 1.



The coding scheme:

- $\mathcal{I}nit(1^k)$: sample a CRS: $\Sigma \leftarrow \{0,1\}^{\ell(k)}$.
- $\mathcal{E}nc(\Sigma, s)$: on input $s \in \{0,1\}^k$, sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{KeyGen}(1^k)$. Let $\mathbf{L}$ be the following language, and W be its natural witness relation:

$$\mathbf{L} \stackrel{\text{def}}{=} \left\{ (\mathsf{pk}, \hat{m}) : \exists w = (\mathsf{sk}, m) \text{ s.t. } \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \text{ forms a key pair for } \mathcal{E} \text{ and} \\ m = \mathrm{Decrypt}_{\mathsf{sk}}(\hat{m}). \end{array} \right\}$$

  Compute $\pi \leftarrow \mathcal{P}((\mathsf{pk}, \hat{s}), (\mathsf{sk}, s, r), \Sigma)$, an NIZK proof of the statement that $(\mathsf{pk}, \hat{s}) \in \mathbf{L}$. Output the encoding $c = (\mathsf{sk}; \mathsf{pk}, \hat{s} = \mathrm{Encrypt}_{\mathsf{pk}}(s), \pi)$.
- $\mathcal{D}ec(\Sigma, c)$: If (1) $\mathcal{V}((\mathsf{pk}, \hat{s}), \pi, \Sigma)$ accepts and (2) $(\mathsf{pk}, \mathsf{sk})$ form a valid key pair, output $\mathrm{Decrypt}_{\mathsf{sk}}(\hat{s})$. Otherwise, output $\perp$.

Fig. 1. The coding scheme

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 2. 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.

Then we are able to achieve the following theorem:

Theorem 1. *Let $t : \mathbb{N} \to \mathbb{N}$ be some non-decreasing polynomial, and $\mathcal{G}_t$, $\mathcal{F}^{\mathsf{half}}, \mathcal{G}_{t,\mathsf{all}}^{\mathsf{half}}$ 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 $\mathcal{H}_k : \{h_z : \{0,1\}^{\mathrm{poly}(k)} \to \{0,1\}^k\}_{z \in \{0,1\}^k}$ is a family of universal one-way hash functions.*

Then the coding scheme is strong non-malleable (Def 6) with respect to $\mathcal{F}^{\mathsf{half}}$, and leakage resilient (Def 7) with respect to $\mathcal{G}_{t,\mathsf{all}}^{\mathsf{half}}$.

Proof (Sketch). 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 $g \in \mathcal{G}_{t,\mathsf{all}}^{\mathsf{half}}$, $g(M_1, M_2)$ hides the original input string. We omit the formal details of the reduction, since they are straightforward.

Now we focus on the proof of non-malleability. In particular, we need to argue that for any $s_0, s_1 \in \{0,1\}^k$, and $f \in \mathcal{F}^{\mathsf{half}}$, we have $(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma}) \approx_c (\Sigma, \mathsf{Tamper}_{s_1}^{f,\Sigma})$ where $\Sigma \leftarrow \mathcal{I}nit(1^k)$. We show this by contradiction: suppose

there exist $f = (f_1, f_2) \in \mathcal{F}^{\mathsf{half}}$, s_0, s_1, some $\varepsilon = 1/\mathrm{poly}(k)$, and a distinguisher D such that $\Pr[D(\Sigma, \mathsf{Tamper}_{s_0}^{f,\Sigma}) = 1] - \Pr[D(\Sigma, \mathsf{Tamper}_{s_1}^{f,\Sigma}) = 1] > \varepsilon$, then we are going to construct a reduction that breaks the encryption scheme $\mathcal{E}$.

The reduction will work as discussed in the overview. Before describing it, we first make an observation: D still distinguishes the two cases of the Tamper experiments even if we change all the real proofs to the simulated ones. More formally, let $(\Sigma, \tau) \leftarrow \mathcal{S}_1(1^k)$, and define $\mathsf{Tamper}_s^{f,\Sigma,\tau}$ be the same game as $\mathsf{Tamper}_s^{f,\Sigma}$ except proofs in the encoding algorithm $\mathcal{E}nc(\Sigma, \cdot)$ are computed by the simulator $\mathcal{S}_2(\cdot, \Sigma, \tau)$ instead of the real prover. We denote this distribution as Tamper_s^{f*}. We claim that D also distinguishes $\mathsf{Tamper}_{s_0}^{f*}$ from $\mathsf{Tamper}_{s_1}^{f*}$.

Suppose not, i.e. D, who distinguishes $\mathsf{Tamper}_{s_0}^{f,\Sigma}$ from $\mathsf{Tamper}_{s_1}^{f,\Sigma}$ does not distinguish $\mathsf{Tamper}_{s_0}^{f*}$ from $\mathsf{Tamper}_{s_1}^{f*}$. Then one can use D, f, s_0, s_1 to distinguish real proofs and simulated ones using standard proof techniques. This violates the multi-theorem zero-knowledge property of the NIZK system Π. Thus, we have:

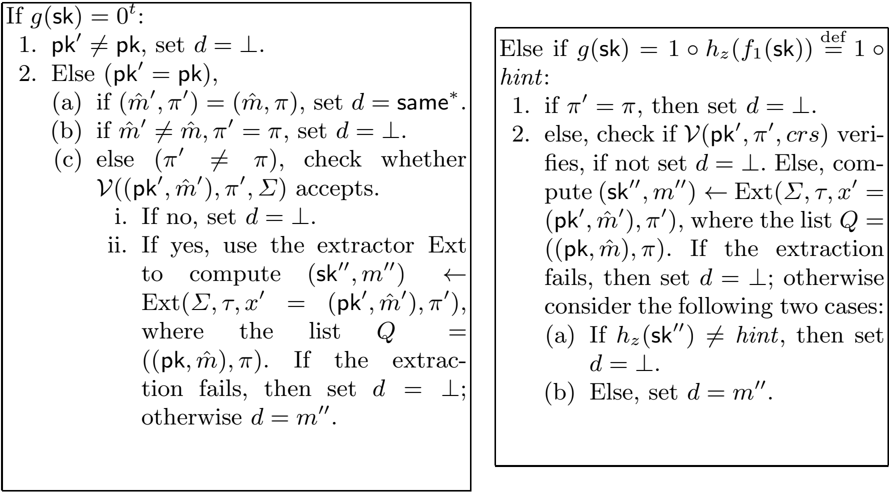
$\Pr[D(\Sigma, \mathsf{Tamper}_{s_0}^{f*}) = 1] - \Pr[D(\Sigma, \mathsf{Tamper}_{s_1}^{f*}) = 1] > \varepsilon/2$.

In the following, we are going to define a reduction Red to break the leakage resilient encryption scheme $\mathcal{E}$. The reduction Red consists of an adversary $A = (A_1, A_2, A_3)$ and a distinguisher D' defined below.

The reduction (with the part A) plays the leakage-resilience game $\mathsf{LE}_b(\mathcal{E}, A, k, \mathcal{F})$ with the challenger, and with the help of the distinguisher D and the tampering function $f = (f_1, f_2)$. Informally speaking of the game, the adversary first sends a leakage function in $g \in \mathcal{F}$ (using A_1), and the challenger replies $g(\mathsf{sk})$. Then A_2 chooses two messages m_0, m_1, and the challenger encrypts either of them, and sends a challenge ciphertext. Finally, A_3 determines which message the challenge was generated from. We defer the formal definition of the game to the full version of this paper. Now we describe the reduction:

- 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 $(\Sigma, \tau) \leftarrow \mathcal{S}(1^k)$.
- A_1 sets $g : \{0,1\}^{n/2} \to \{0,1\}^t$ to be the following function, and sends this leakage query to the challenger: $g(\mathsf{sk}) = \begin{cases} 0^t & \text{if } f_1(\mathsf{sk}) = \mathsf{sk}, \\ 1 \circ h_z(f_1(\mathsf{sk})) & \text{otherwise.} \end{cases}$
  This leakage value tells A_1 if the tampering function f_1 alters sk.
- A_2 chooses m_0, m_1 to be s_0, and s_1 respectively. Then the challenger samples $(\mathsf{pk}, \mathsf{sk})$ and sets $\hat{m} = \mathrm{Encrypt}_{\mathsf{pk}}(m_b)$ to be the ciphertext, and sends $\mathsf{pk}, g(\mathsf{sk}), \hat{m}$ to the adversary.
- Then A_3 computes the simulated proof $\pi = \mathcal{S}_2(\mathsf{pk}, \hat{m}, \Sigma, \tau)$, and sets $(\mathsf{pk}', \hat{m}', \pi') = f_2(\mathsf{pk}, \hat{m}, \pi)$. Then A_3 computes a bit b using one of the algorithms in figure 2, depending on the outcome of $g(\mathsf{sk})$.
- 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 $\mathcal{E}$.



If $g(\mathsf{sk}) = 0^t$:
1. $\mathsf{pk}' \neq \mathsf{pk}$, set $d = \perp$.
2. Else $(\mathsf{pk}' = \mathsf{pk})$,
   (a) if $(\hat{m}', \pi') = (\hat{m}, \pi)$, set $d = \mathsf{same}^*$.
   (b) if $\hat{m}' \neq \hat{m}, \pi' = \pi$, set $d = \perp$.
   (c) else $(\pi' \neq \pi)$, check whether $\mathcal{V}((\mathsf{pk}', \hat{m}'), \pi', \Sigma)$ accepts.
      i. If no, set $d = \perp$.
      ii. If yes, use the extractor Ext to compute $(\mathsf{sk}'', m'') \leftarrow \mathrm{Ext}(\Sigma, \tau, x' = (\mathsf{pk}', \hat{m}'), \pi')$, where the list $Q = ((\mathsf{pk}, \hat{m}), \pi)$. If the extraction fails, then set $d = \perp$; otherwise $d = m''$.

Else if $g(\mathsf{sk}) = 1 \circ h_z(f_1(\mathsf{sk})) \stackrel{\text{def}}{=} 1 \circ hint$:
1. if $\pi' = \pi$, then set $d = \perp$.
2. else, check if $\mathcal{V}(\mathsf{pk}', \pi', crs)$ verifies, if not set $d = \perp$. Else, compute $(\mathsf{sk}'', m'') \leftarrow \mathrm{Ext}(\Sigma, \tau, x' = (\mathsf{pk}', \hat{m}'), \pi')$, where the list $Q = ((\mathsf{pk}, \hat{m}), \pi)$. If the extraction fails, then set $d = \perp$; otherwise consider the following two cases:
   (a) If $h_z(\mathsf{sk}'') \neq hint$, then set $d = \perp$.
   (b) Else, set $d = m''$.

Fig. 2. The two cases for the reduction

To analyze the reduction, we are going to establish the following claim. We defer the formal proof to the full version of this paper.

Claim. Given the above A and D', we have

$$\Pr[D'(\mathsf{LE}_0(\mathcal{E}, A, k, \mathcal{F}^{\mathsf{half}})) = 1] - \Pr[D'(\mathsf{LE}_1(\mathcal{E}, A, k, \mathcal{F}^{\mathsf{half}})) = 1] > \varepsilon/2 - \mathsf{ngl}(k).$$

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.

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 $\mathsf{Hardware}_{\text{rand}}(\mathcal{C}, G) \stackrel{\text{def}}{=} \langle G', \mathcal{E}nc(s)\rangle$. Obviously the compiler is correct, i.e. the implementation's input/output behavior is the same as that of the original functionality. Then we are able to achieve the following theorem:

Theorem 2. *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 $\mathcal{E} = (\text{KeyGen}, \text{Encrypt}, \text{Decrypt})$ that is semantically secure against one-time leakage $\mathcal{G}_t$; a robust NIZK $\Pi = (\ell, \mathcal{P}, \mathcal{V}, \mathcal{S})$; and $\mathcal{H}_k : \{h_z : \{0,1\}^{\text{poly}(k)} \to \{0,1\}^k\}_{z\in\{0,1\}^k}$, a family of universal one-way hash functions. Then the 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 $\mathsf{Hardware}_{\text{rand}}(\mathcal{C}, G)$ 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 $\mathsf{Hardware}_{\text{rand}}(\mathcal{C}, G)$ 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. The formal proof appears in the full version of this paper.

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:

G^* on input x does the followings:

- Decode $\mathcal{E}nc(\Sigma, s \circ \mathsf{seed})$ to obtain $s \circ \mathsf{seed}$. Recall that $\mathcal{D}ec$ is deterministic.
- Compute $\mathsf{seed}' \circ r_1 \circ r_2 \leftarrow g(\mathsf{seed})$, where $\mathsf{seed}' \in \{0,1\}^k, r_1, r_2 \in \{0,1\}^\ell$.
- Calculate $(s_{\mathsf{new}}, y) \leftarrow G(s, x)$ (using the string r_1 as a random tape if G is randomized), then outputs y, and updates the state to be s_{new}.
- Calculate the encoding of $s' \circ \mathsf{seed}'$ using the string r_2 as a random tape. Then it stores the new encoding $\mathcal{E}nc(\Sigma, s_{\mathsf{new}} \circ \mathsf{seed}')$.

Fig. 3. The deterministic implementation

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 achieve the following theorem.

Theorem 3. *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 the Naor-Segev Leakage-resilient encryption scheme [30], we are obtain the following corollary.

Corollary 1. *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.*

The formal proof appears in the full version of this paper.

Acknowledgement. We thank Yevgeniy Dodis for useful discussions. This work was supported by NSF grants 1012060, 0964379, 0831293.

References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM Side-Channel(s). In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 29–45. Springer, Heidelberg (2003)
2. Akavia, A., Goldwasser, S., Vaikuntanathan, V.: Simultaneous Hardcore Bits and Cryptography against Memory Attacks. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 474–495. Springer, Heidelberg (2009)
3. Alwen, J., Dodis, Y., Wichs, D.: Leakage-Resilient Public-Key Cryptography in the Bounded-Retrieval Model. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 36–54. Springer, Heidelberg (2009)
4. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)
5. Brakerski, Z., Kalai, Y.T., Katz, J., Vaikuntanathan, V.: Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In: FOCS, pp. 501–510. IEEE (2010)
6. Choi, S.G., Kiayias, A., Malkin, T.: BiTR: Built-in Tamper Resilience. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 740–758. Springer, Heidelberg (2011)
7. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust Non-interactive Zero Knowledge. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 566–598. Springer, Heidelberg (2001)
8. Dodis, Y., Haralambiev, K., López-Alt, A., Wichs, D.: Cryptography against continuous memory attacks. In: FOCS, pp. 511–520 (2010)
9. Dodis, Y., Lewko, A.B., Waters, B., Wichs, D.: Storing secrets on continually leaky devices. In: FOCS, pp. 688–697 (2011)
10. Dodis, Y., Pietrzak, K.: Leakage-Resilient Pseudorandom Functions and Side-Channel Attacks on Feistel Networks. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 21–40. Springer, Heidelberg (2010)
11. Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM J. Comput. 30(2), 391–437 (2000)
12. Dziembowski, S., Pietrzak, K.: Leakage-resilient cryptography. In: FOCS, pp. 293–302. IEEE (2008)
13. Dziembowski, S., Pietrzak, K., Wichs, D.: Non-malleable codes. In: ICS, pp. 434–452 (2010)
14. Faust, S., Pietrzak, K., Venturi, D.: Tamper-Proof Circuits: How to Trade Leakage for Tamper-Resilience. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 391–402. Springer, Heidelberg (2011)
15. Faust, S., Rabin, T., Reyzin, L., Tromer, E., Vaikuntanathan, V.: Protecting Circuits from Leakage: the Computationally-Bounded and Noisy Cases. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 135–156. Springer, Heidelberg (2010)
16. Gennaro, R., Lysyanskaya, A., Malkin, T., Micali, S., Rabin, T.: Algorithmic Tamper-Proof (ATP) Security: Theoretical Foundations for Security against Hardware Tampering. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 258–277. Springer, Heidelberg (2004)
17. Goldwasser, S., Rothblum, G.N.: Securing Computation against Continuous Leakage. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 59–79. Springer, Heidelberg (2010)

18. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: Cold boot attacks on encryption keys. In: USENIX Security Symposium, pp. 45–60 (2008)
19. Halevi, S., Lin, H.: After-the-Fact Leakage in Public-Key Encryption. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 107–124. Springer, Heidelberg (2011)
20. Ishai, Y., Prabhakaran, M., Sahai, A., Wagner, D.: Private Circuits II: Keeping Secrets in Tamperable Circuits. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 308–327. Springer, Heidelberg (2006)
21. Ishai, Y., Sahai, A., Wagner, D.: Private Circuits: Securing Hardware against Probing Attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
22. Juma, A., Vahlis, Y.: Protecting Cryptographic Keys against Continual Leakage. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 41–58. Springer, Heidelberg (2010)
23. Kalai, Y.T., Kanukurthi, B., Sahai, A.: Cryptography with Tamperable and Leaky Memory. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 373–390. Springer, Heidelberg (2011)
24. Katz, J., Vaikuntanathan, V.: Signature Schemes with Bounded Leakage Resilience. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 703–720. Springer, Heidelberg (2009)
25. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
26. Lewko, A.B., Lewko, M., Waters, B.: How to leak on key updates. In: STOC, pp. 725–734. ACM (2011)
27. Lewko, A., Rouselakis, Y., Waters, B.: Achieving Leakage Resilience through Dual System Encryption. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 70–88. Springer, Heidelberg (2011)
28. Liu, F.-H., Lysyanskaya, A.: Algorithmic Tamper-Proof Security under Probing Attacks. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 106–120. Springer, Heidelberg (2010)
29. Micali, S., Reyzin, L.: Physically Observable Cryptography (Extended Abstract). In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 278–296. Springer, Heidelberg (2004)
30. Naor, M., Segev, G.: Public-Key Cryptosystems Resilient to Key Leakage. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 18–35. Springer, Heidelberg (2009)
31. Pietrzak, K.: A Leakage-Resilient Mode of Operation. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 462–482. Springer, Heidelberg (2009)
32. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: FOCS, pp. 543–553. IEEE (1999)