An extended abstract of this paper is accepted at the 38th International Colloquium on Automata, Languages and Programming – ICALP 2011. This is the full version.

Tamper-Proof Circuits: How to Trade Leakage for Tamper-Resilience

Sebastian Faust[1], Krzysztof Pietrzak[2] and Daniele Venturi[3]

June 14, 2011

Abstract

Tampering attacks are cryptanalytic attacks on the implementation of cryptographic algorithms (e.g. smart cards), where an adversary introduces faults with the hope that the tampered device will reveal secret information. Inspired by the work of Ishai et al. [Eurocrypt'06], we propose a compiler that transforms any circuit into a new circuit with the same functionality, but which is resilient against a well-defined and powerful tampering adversary. More concretely, our transformed circuits remain secure even if the adversary can *adaptively* tamper with *every* wire in the circuit as long as the tampering fails with some probability $\delta > 0$. This additional requirement is motivated by practical tampering attacks, where it is often difficult to guarantee the success of a specific attack.

Formally, we show that a q-query tampering attack against the transformed circuit can be "simulated" with only black-box access to the original circuit and $\log(q)$ bits of additional auxiliary information. Thus, if the implemented cryptographic scheme is secure against $\log(q)$ bits of leakage, then our implementation is tamper-proof in the above sense. Surprisingly, allowing for this small amount of information leakage – and not insisting on perfect simulability like in the work of Ishai et al. – allows for much more efficient compilers, which moreover do not require randomness during evaluation.

Similar to earlier work our compiler requires *small, stateless and computation-independent* tamper-proof gadgets. Thus, our result can be interpreted as reducing the problem of shielding arbitrary complex computation to protecting *simple* components.

[1]K.U. Leuven ESAT-COSIC/IBBT and Aarhus University.
[2]CWI Amsterdam.
[3]SAPIENZA University of Rome.

1 Introduction

Modern security definitions usually consider some kind of game between an adversary and the cryptosystem under attack, where the adversary interacts with the system and finally must break it. A distinctive feature of such notions is that the adversary has only *black-box* access to the cryptosystem. Unfortunately, in the last decade it became evident that such black-box notions do not capture adversaries that attack the physical implementation of a cryptosystem. Recently, significant progress has been made to close this gap [24, 10, 28, 2, 12, 19, 26, 31, 9]. With few exceptions most of this research is concerned with "side-channel attacks". These are attacks where the adversary measures information that is leaked from the cryptodevice during computation.

In this work we explore *active* physical attacks which so far got much less attention from the theory community. We study the security of cryptographic implementations when the adversary can not only measure, but actively tamper with the computation of the physical device, e.g. by introducing faults. Such attacks, often called *fault analysis* or *tampering attacks*, are a serious threat to the security of real-world implementations and often allow to completely break otherwise provably secure schemes. In this work we investigate the general question whether *any* cryptographic scheme can be implemented *efficiently* such that it resists a very powerful adversary tampering with the whole computation and the memory.

Many techniques to induce faults into the computation of a cryptodevice have been proposed. Important examples include heating up the device, expose it to infrared radiation or alter the internal power supply or clock [3, 8, 30]. One might think that an adversary that obtains the result of faulty computation will not be very useful. In a seminal paper Boneh et al. [8] show that already a single fault may allow to completely break an RSA based digital signature scheme. Different methods to counter such attacks have been proposed. Most such countermeasures have in common that they protect against specific adversarial strategies and come without a rigorous security analysis. This is different from the *provable security* approach followed by modern cryptography, where one first defines a precise adversarial model and then proves that no (resource-bounded) attacker exists who breaks the scheme.

An influential work on provable security against tampering attacks is the work on *private circuits* of Ishai et al. [19, 18]. Informally, such "private circuits" carry out a specific cryptographic task (e.g., signing), while protecting the internal secret state against a well-defined class of tampering attacks. The authors construct a general circuit compiler that transforms *any* Boolean circuit C into a functionally equivalent "private circuit" $\widehat{C}$. It is shown that an adversary who can tamper[1] with at most t wires in-between any two invocations of $\widehat{C}$, cannot learn anything more about the secret state than an adversary having just black-box access to C. Security is proven by a simulation argument: for any adversary $\mathcal{A}$ that can mount a tampering attack on the circuit, there exists an (efficient) simulator $\mathcal{S}$ having only black-box access to the circuit, such that the output distribution of $\mathcal{A}$ and $\mathcal{S}$ are statistically close.

To achieve this goal their transformation uses techniques from multi-party computation and combines randomized computation with redundant encodings to detect faulty computation. If tampering is detected a self-destruction mechanism is triggered that overwrites the complete state, so that, from there on, regular and tampering queries to the circuit can be trivially simulated. One difficulty that needs to be addressed is that this self-destruction mechanism itself is exposed

[1]Tampering with a wire means permanently *set* its value to the constant 1 or 0 or *toggle* the wire, which means that whatever value is put on the wire gets inverted.

to tampering attacks. In particular, an adversary could just try to cancel any trigger for self-destruction and from then on apply arbitrary attacks without being in danger of detection. Ishai et al. face this problem by spreading and propagating errors that appear during the computation. As discussed later we will use similar techniques in our work. We discuss some further related work in the Appendix A.

1.1 Our Contribution

The "holy grail" in this line of research is to find an efficient general circuit compiler that provably resists *arbitrary* tampering attacks to an *unlimited number* of wires. This goal might be too ambitious since an adversary might just "reprogram" the circuit such that it outputs its complete secret state. Hence, to have any hope to formally analyze the security against tampering attacks we will need to limit the power of the adversary. As just discussed, Ishai et al. limit the adversary to tamper with at most t wires in each round. Their construction blows up the circuit size quite significantly and makes extensive use of randomness: for a statistical security parameter k, the size of the transformed circuit is by a factor of $O(k^3t)$ larger and requires $O(k^2)$ bits of randomness in each invocation.

"Noisy" tampering. In this work we consider a somewhat different (and incomparable) adversarial model, where the adversary can tamper with *every* wire (not just some small number t) in the circuit, but tampering with every wire will fail (independently) with some probability $\delta \geq 0$.[2] This model is partially motivated by existing attacks [27, 7, 8]. Concrete examples of attacks covered by our model are e.g. optical attacks and Eddy currents attacks (cf. [27] for details).

Leakage. Another crucial difference between our work and [18] is that we use a relaxed security definition which allows the tampering adversary to learn a logarithmic amount of information about the secret state (in total, not per invocation). This relaxation buys us a lot in terms of efficiency and simplicity. In particular, for security parameter k we achieve statistical security by blowing up the circuit size by only $O(k)$ and requiring *no* randomness during run-time (and only $2k$ bits during production).

If q is the number of queries and n the size of the output of the original circuit, the amount of leakage is $\log(q \cdot n)$ bits. Intuitively, the only advantage an adversary gets by being able to tamper the transformed circuit is to "destroy" its internal state, but the point in the computation where the destruction happens can depend on the secret state. The information leaked by allowing tampering is this point of failure.

If we apply our transformation to a particular cryptosystem in order to get a tamper-resilient version $\widehat{C}$ of it, it is crucial that the scheme C remains secure even given Λ. Some primitives like public-key encryption [1] or digital signatures [14, 12] are always secure against a logarithmic amount of arbitrary leakage, but a logarithmic amount of leakage can decrease the security of the PKE or signature scheme by a polynomial factor. Recently signature schemes [22, 2] and public-key encryption schemes [1, 26] have been constructed where the security does not degrade at all, even if the amount of leakage is quite substantial. Using such schemes we can avoid the loss in security due to the leakage.

[2]The adversary can tamper the same wire several times, but only once in-between every two invocations. As tampering is persistent, after a sufficiently large number of attempts the tampering will succeed almost certainly, i.e. with probability $1 - \delta^l$ after l rounds.

Overview of the construction. Starting with any Boolean (probabilistic) circuit C our transformation Φ outputs a transformed circuit $\widehat{C}$ that consists of k subcircuits (which we will call the *core*). Each subcircuit is made out of special constant size *tamper-proof* masked "Manchester gadgets". Instead of doing simple Boolean operations, these gadgets compute with encodings, so called *masked Manchester encodings* (which encode a single bit into 4 bits). If the inputs are not valid encodings, this gadgets output the invalid state 0000. Since each of the k subcircuits is made solely out of such special gadgets, errors introduced by a tampering attack will propagate to the output of the core and are input to a self-destruction phase (essentially identical to the one introduced by [18]). In contrast to the core of the circuit which is built with gadgets of constant size, the self-destruction phase (in the following also called cascade phase) will require (simple, stateless and deterministic) tamper-proof gadgets of linear (in k) size. Ishai et al. require tamper-proof gadgets of linear (in kt) size in the *entire* circuit, albeit simpler ones than we do.[3] Unlike [18], we do not require so called "reversible" NOT gates. Such gadgets propagate a fault on the output side to their input side and are highly non-standard.

It is not difficult to see that the transformation as outlined above is not secure in the setting of Ishai et al. (i.e. when $\delta = 0$ and the adversary can tamper with up to t wires in each round). Nevertheless, we show in Appendix B how to adjust our compiler to achieve similar efficiency improvement when $\delta = 0$ and some small amount of leakage is allowed.

On tamper-proof gadgets. Assuming simple components that withstand active physical attacks has been frequently made in the literature [18, 14] (and several others in the context of leakage [13, 21, 16]). Of course, the simpler the components are, the stronger the result is that one gets.

1. The components we use are fixed, standard and universal elements that can be added to once standard cell library. This is far better than designing over and over task specific tamper-proof components.

2. Our gadgets are simple, stateless and deterministic. In particular, the gadgets used in the core of the circuit have a small constant size.

3. Our transformation for the core is independent of the cascade phase, which uses linear size gadgets. Thus one can implement a universal "tamper-proof cascade phase", and only has to compile and implement the core.

Outline of the security proof. We construct an efficient simulator $\mathcal{S}$ that – having only black-box access to the circuit C – can simulate access to the transformed circuit $\widehat{C}$ including adversarial tampering queries. The main challenge here is consistency, that is, answers computed by $\mathcal{S}$ must have the same distribution as an adversary would see when tampering with $\widehat{C}$.

If an adversary tampers with $\widehat{C}$, a subsequent invocation of $\widehat{C}$ can have one of three different outcomes:

1. Nothing happens: the invocation goes through as if no tampering did happen (this is e.g. the case if a wire is set to 0, but its value during the invocation is 0 anyway).

2. Self-destruct: the redundancy added to $\widehat{C}$ "detects" tampering, and the entire state is deleted.

[3] More precisely, they require tamper-proof AND gadgets that take $4kt$ bits as input.

3. Successful Tampering: the outcome of $\widehat{C}$ changes as a consequence of the tampering, and this tampering was not detected.

In a first step, we show that case 3 will not happen but with exponentially small probability. To show this, we use the fact that tampering with any particular wire fails with probability δ, and moreover that every bit carried by a wire in C is encoded in $\widehat{C}$ with a highly redundant and randomized encoding. This guarantees that the chance of an adversary to change a valid encoding of a bit to its complement is tiny: either she has to be lucky – in the sense that she tampers with many wires at once and all attacks succeed – or she has to guess the randomness used in the encoding.

As we ruled out case 3., we must only build a simulator $\mathcal{S}$ that simulates $\widehat{C}$ as if no tampering has happened (i.e. case 1.). This is easy as $\mathcal{S}$ has access to C which is functionally equivalent. Moreover, at some point $\mathcal{S}$ has to simulate a self-destruct (i.e. case 2.). Unfortunately there is no way for $\mathcal{S}$ to know when the self-destruct happens (as the probability of this event can be correlated with the secret state). As explained before, we provide the exact point of failure as auxiliary input to $\mathcal{S}$.

The simulator has to continue simulation even after the self-destruct. This seems easy, as now all the secret state has been deleted. There is one important technicality though. As tampering is permanent, even after self-destruct the simulator $\mathcal{S}$ must simulate a circuit in a way that is consistent with the simulation so far. A priori the simulator only knows which wires the adversary tried to tamper, but recall that each tampering is only successful with probability $1 - \delta$. For this reason, we let the simulator choose all the randomness used, including the randomness of the compiler (which generates $\widehat{C}$ from C) and the randomness that determines the success of the tampering attacks. Knowledge of this randomness, allows the simulator to continue simulation after self-destruct.

Note that the above-mentioned auxiliary information (i.e., the point at which self-destruct is triggered) can be computed as a function of this randomness, and the randomness used by the adversary.

2 Definitions

Notation. If D is a distribution over a set $\mathcal{D}$, then $x \leftarrow \mathsf{D}$ means a random variable x is drawn from D (if $\mathcal{D}$ is a set with no distribution specified, then $x \leftarrow \mathcal{D}$ denotes a random variable with uniform distribution over $\mathcal{D}$). If D is an algorithm, then $y \leftarrow \mathsf{D}(x)$ means that y is the output of D on input x; in particular when D is probabilistic, y is a random variable. Two distributions D and D' are *ϵ-close*, written $\mathsf{D} \approx_\epsilon \mathsf{D}'$, if their statistical distance $\frac{1}{2}\sum_{x\in\mathcal{D}} |\mathsf{D}(x) - \mathsf{D}'(x)|$ is less than or equal to ϵ.

We write $\mathcal{A}^{\mathcal{O}(\cdot)}$ to denote an algorithm $\mathcal{A}$ with oracle access to $\mathcal{O}(\cdot)$.

Given two codewords $x, y \in \{0,1\}^n$ their Hamming distance, $0 \leq d_H(x,y) \leq n$, is the number of positions in which x and y differ.

2.1 Our Model

Our physical model of computation is very similar to [18]. We consider (probabilistic) stateful Boolean circuits C and present circuit compilers Φ that transform any such circuit into a new circuit $\widehat{C}$ resistant against a well-defined class of tampering attacks. Details follow below.

Circuits. A Boolean circuit C is a directed acyclic graph whose vertices are standard Boolean gates and whose edges are the wires. The depth of C, denoted $\mathit{depth}(C)$, is the longest path from an input to an output. A circuit is *clocked* if it evolves in clock cycles (or rounds). The input and output values of the circuit C in clock cycle i are denoted by X_i and Y_i, respectively. A circuit is *probabilistic* if it uses internal randomness as part of its logic. We call such probabilistic logic *randomness gates* and denote them with \$. In each clock cycle \$ outputs a fresh random bit.

Additionally, a circuit may contain *memory gates.* Memory gates, which have a single incoming edge and any number of outgoing edges, maintain state: at any clock cycle, a memory gate sends its current state down its outgoing edges and updates it according to the value of its incoming edge. Any cycle in the circuit graph must contain at least one memory gate. The state of all memory gates at clock cycle i is denoted by M_i, with M_0 denoting the initial state. When a circuit is run in state M_{i-1} on input X_i, the circuit will output Y_i and the memory gates will be in a new state M_i. We will denote this by $(Y_i, M_i) \leftarrow C[M_{i-1}](X_i)$.

Adversarial model. We consider adversaries that can adaptively tamper in q clock cycles with up to t wires. In this paper we are particularly interested in the case where t is unbounded, i.e. the adversary can tamper with an *arbitrarily large* number of wires in the circuit in *every* round. For each wire we allow the adversary to choose between the following types of attacks: *set*, i.e. setting a wire to 1, *reset*, i.e. setting a wire to 0 and *toggle*, i.e. flipping the value on the wire. For each wire such an attack fails independently with some probability. This is captured by the global parameter δ, where $\delta = 0$ means that the attack succeeds always, and $\delta = 1$ that no tampering takes place. The model of [18] considers the case in which t is a small integer and tampering is always successful, i.e. $\delta = 0$.

When an attack fails for one wire the computation continues with the original value on that wire. Notice that once a fault is successfully placed it stays *permanently.* Let us stress that we do allow the adversary to "undo" (with zero error probability) persistent attacks induced in previous rounds (this captures so called transient faults). We call such an adversary, that can adaptively tamper with a circuit for up to q clock cycles attacking up to t wires per round, an (t, δ, q)-adversary and denote the attack strategy for each clock cycle as $\mathsf{W} = \{(w_1, a_1), \ldots, (w_t, a_t)\}$. The first element in each such tuple specifies which wire in the circuit is attacked and the second element specifies the type of attack (i.e. *set*, *reset* or *toggle*). When the number of faults per clock cycle is unbounded, we will explicitely write $t = \infty$.

2.2 Tamper-Proof Security

The definitions below are given for (∞, δ, q)-adversaries, but can be adapted to the case where the number t of faults in every clock cycle is bounded in a straight forward way.

Transformation. A circuit transformation Φ takes as input a security parameter k, a (probabilistic) circuit C and an initial state M_0 and produces a transformed initial state $\widehat{M_0}$ and a transformed (probabilistic) circuit $\widehat{C}$. This is denoted by $(\widehat{C}, \widehat{M_0}) \leftarrow \Phi(C, M_0)$. The compiled circuit can use a different set of gates, and this will be the case for the compiler we construct. The transformation itself can be randomized and we let ρ_Φ denote the random coins of the transformation. We say that the transformation Φ is *functionality preserving* if for all C, M_0 and any set of public inputs

$X_1, X_2, \ldots, X_q$ the original circuit C starting with state M_0 and the transformed circuit $\widehat{C}$ starting with state $\widehat{M_0}$ result in an identical output distribution.

Following [18], we define security of circuit transformations against tampering attacks by a simulation-based argument, but we augment the simulation by allowing auxilliary input. Loosely speaking, for every (∞, δ, q)-adversary $\mathcal{A}$ tampering with $\widehat{C}$, there exists a simulator $\mathcal{S}_\lambda$, that gets as input some λ-bounded auxiliary information Λ and only has black-box access to the original circuit C, such that the output distribution of $\mathcal{A}$ and $\mathcal{S}_\lambda$ are close. We will specify the nature of the auxiliary information below.

Real world experiment. The adversary $\mathcal{A}$ can in each round i adaptively specify an input X_i and an attack strategy W_i that is applied to the transformed circuit $\widehat{C}$ when run on input X_i with secret state $\widehat{M_{i-1}}$. The output Y_i resulting from the (possibly) faulty computation is given to the adversary and the state is updated to $\widehat{M_i}$ for the next evaluation. To formally describe such a process we introduce a special oracle, Tamper, that can be queried on (X_i, W_i) to return the result Y_i. More precisely, for any (∞, δ, q)-adversary $\mathcal{A}$, any circuit C and any initial state M_0, we define the following experiment:

Experiment $\mathsf{Exp}_\Phi^{\mathrm{Real}}(\mathcal{A}, C, M_0)$:
$(\widehat{C}, \widehat{M_0}) \leftarrow \Phi(C, M_0)$
Output $\mathcal{A}^{\mathsf{Tamper}(\widehat{C}, \widehat{M_0}, \cdot, \cdot)}(C)$

Simulation. The simulator $\mathcal{S}_\lambda$ simulates the adversary's view, however, she has to do so without having tampering access to the transformed circuit. More precisely, the simulator only has oracle access to $C[M_0](\cdot)$. Additionally, we will give the simulator some λ-bounded auxiliary information. This is described by letting $\mathcal{S}_\lambda$ choose an arbitrary function $f: \{0,1\}^* \to \{0,1\}^\lambda$ and returning the result of f evaluated on input the secret state M_0, i.e. $\Lambda \overset{\mathsf{def}}{=} f(M_0)$. For a simulator $\mathcal{S}_\lambda$ we define the following experiment for any circuit C, any initial state M_0 and any (∞, δ, q)-adversary $\mathcal{A}$:

Experiment $\mathsf{Exp}_\Phi^{\mathrm{Sim}}(\mathcal{S}_\lambda, C, M_0, \mathcal{A})$:
$f \leftarrow \mathcal{S}_\lambda(\mathcal{A}, C)$ where $f: \{0,1\}^* \to \{0,1\}^\lambda$
Output $\mathcal{S}_\lambda^{C[M_i](\cdot)}(\Lambda)$ where $\Lambda = f(M_0)$

Tamper-resilient circuit transformations. A circuit transformation is said to be tamper-resilient if the outputs of the two experiments are statistically close. More formally:

Definition 1. (**Tamper-Resilience of Circuit Transformation**). *A circuit transformation* Φ *is* $(\infty, \delta, q, \lambda, \epsilon)$-tamper-resilient *if for any* (∞, δ, q)*-adversary* $\mathcal{A}$*, for every circuit* C *and any initial state* M_0*, there exists a simulator* $\mathcal{S}_\lambda$ *such that*

$$\mathsf{Exp}_\Phi^{\mathrm{Sim}}(\mathcal{S}_\lambda, C, M_0, \mathcal{A}) \approx_\epsilon \mathsf{Exp}_\Phi^{\mathrm{Real}}(\mathcal{A}, C, M_0),$$

where the probabilities are taken over all the random coin tosses involved in the experiments.[4]

[4]The parameters δ, q, λ and ϵ are all parameterized by the security parameter k.

3 A Compiler Secure against (∞, δ, q)-Adversaries

We describe a compiler Φ which is secure against (∞, δ, q)-adversaries. A different construction for the case of a small t and $\delta = 0$ – i.e. when the number of faults per round is *bounded* but attacks succeed always – is given in Appendix B.

Instead of computing with bits the compiled circuit $\widehat{C}$ will operate on redundant and randomized encodings of bits.

3.1 Encodings

Our transformation is based on three encoding schemes, where each is used to encode the previous one. The first encoding, so called *Manchester encoding*, can be described by a deterministic function that takes as input a bit $b \in \{0,1\}$ and has the following output: $\mathsf{MC}(b) \overset{\mathsf{def}}{=} (b, \bar{b})$.. Decoding is done just by outputting the first bit. The output $(b, \bar{b})$ is given as input to the next level of our encoding procedure where we use a probabilistic function $mask : \{0,1\}^2 \times \{0,1\}^2 \to \{0,1\}^4$. Such a function uses as input additionally two random bits for *masking* its output. More precisely, we have $mask(\mathsf{MC}(b), (r, r')) \overset{\mathsf{def}}{=} (b \oplus r, r, \bar{b} \oplus r', r')$, with $(r, r') \leftarrow \{0,1\}^2$. We denote with $\mathsf{MMC} \subset \{0,1\}^4$ the set of valid masked Manchester encoded bits, and with $\overline{\mathsf{MMC}} \overset{\mathsf{def}}{=} \{0,1\}^4 \setminus \mathsf{MMC}$ the non-valid encodings. Our final encoding consists of k independent masked Manchester encodings:

$$\mathsf{Enc}(b, \vec{r}) \overset{\mathsf{def}}{=} mask(\mathsf{MC}(b), (r_1, r'_1)), \ldots, mask(\mathsf{MC}(b), (r_k, r'_k)), \qquad (1)$$

with $\vec{r} = (r_1, r'_1, r_2, r'_2, \ldots, r_k, r'_k) \in \{0,1\}^{2k}$. Thus it has length $4k$ bits and uses $2k$ bits of randomness. When the randomness in an encoding is omitted, it is uniformly sampled, e.g. $\mathsf{Enc}(b)$ denotes the random variable $\mathsf{Enc}(b, \vec{r})$ where $\vec{r} \in \{0,1\}^{2k}$ is sampled uniformly at random.

We denote with $\mathsf{Enc} \subset \{0,1\}^{4k}$ the set of all valid encodings and with $\overline{\mathsf{Enc}} \overset{\mathsf{def}}{=} \{0,1\}^{4k} \setminus \mathsf{Enc}$ the non-valid ones.

3.2 The Compiler

Consider any (probabilistic) Boolean circuit C that consists of Boolean NAND gates, randomness gates \$ and memory cells. We assume that the original circuit handles fanout through special copy gates taking one bit as input and outputting two copies. If k copies are needed, the original value is passed through a subcircuit of $k-1$ copy gadgets arranged in a tree structure. Let us first describe the transformation for the secret state. On factory setup $2k$ random bits $\rho_\Phi = (r_1, r'_1, \ldots, r_k, r'_k)$ are sampled uniformly. Then, each bit of the secret state m_i is encoded by $\mathsf{Enc}(m_i, \rho_\Phi)$. Putting all these encodings together we get the initial transformed secret state $\widehat{M}_0$. The encoded secret state will be stored in the memory cells of $\widehat{C}$, but we will discuss this below. Notice that we use the same randomness for each encoding.

The global picture of our transformer consists of four different stages: the encoder, the input/output cascade phase, the transformation for the core and the decoder. These stages are connected as shown in Figure 5 on page 21 and are described below.

The encoder and the decoder. Since the compiled circuit computes with values in encoded form, we need to specify how to encode and decode the public inputs and outputs of $\widehat{C}$. The encoder (which is deterministic and build from copy and negation gates) encodes every bit of the input using randomness ρ_Φ:

$$\mathsf{Encoder}(x_1, \dots, x_t) \overset{\mathsf{def}}{=} \mathsf{Enc}(x_1, \rho_\Phi), \dots, \mathsf{Enc}(x_t, \rho_\Phi) \quad \text{where} \quad x_1, \dots, x_t \in \{0,1\}.$$

The decoding phase simply outputs the XORs of the first two bits of every encoding:

$$\mathsf{Decoder}(X_1, \dots, X_{t'}) \overset{\mathsf{def}}{=} X_1[1] \oplus X_1[2], \dots, X_{t'}[1] \oplus X_{t'}[2] \quad \text{where} \quad X_i \in \{0,1\}^{4k}.$$

The input and output cascade phases. For self-destruction we use a tool already introduced by Ishai et al. – the *cascade phase* (cf. Figure 6 on page 21). In our construction we make use of two cascade phases: an *input* cascade phase and an *output* cascade phase. As shown in Figure 5 on page 21 the input cascade phase takes as input the output of the encoder and the encoded secret state. The output cascade phase takes as inputs the output of the core and the updated secret state.[5] As discussed later in Section 4, for technical reasons we require that the secret state is always in the top part and the public output is always on the bottom part of the cascade phase. For ease of description we call the part of the cascade phase that takes the inputs as the first half and the part that produces the outputs as the second half. This is shown in Figure 6 on page 21.

Inside the cascade phase we make use of special *cascade gadgets* $\Pi : \{0,1\}^{8k} \to \{0,1\}^{8k}$. The gadgets behave like the identity function if the inputs are valid encodings using randomness ρ_Φ, and output 0^{8k} otherwise, i.e.

$$\Pi(A, B) = \begin{cases} A, B & \text{if } A, B \in \{\mathsf{Enc}(0, \rho_\Phi), \mathsf{Enc}(1, \rho_\Phi)\} \\ 0^{8k} & \text{otherwise.} \end{cases}$$

The gadgets are assumed to be *tamper-proof*, i.e. the adversary is allowed to tamper with their inputs and outputs, but she cannot modify their internals.

The core. With $\widehat{\mathsf{NAND}}_{r,r'} : \{0,1\}^{2\times 4} \to \{0,1\}^4$ we define a NAND gate which works on masked Manchester encodings using randomness r, r' (on input and output). If the input contains anything else than a valid masked Manchester encoding , the output is $0^4 \in \overline{\mathsf{MMC}}$. The truth table of these gadgets is given in Figure 1. Similarly we denote with $\widehat{\mathsf{copy}}_{r,r'} : \{0,1\}^4 \to \{0,1\}^{2\times 4}$ a copy

1st Input	2nd Input	Output
$mask(\mathsf{MC}(0,r,r'))$	$mask(\mathsf{MC}(0,r,r'))$	$mask(\mathsf{MC}(1,r,r'))$
$mask(\mathsf{MC}(0,r,r'))$	$mask(\mathsf{MC}(1,r,r'))$	$mask(\mathsf{MC}(1,r,r'))$
$mask(\mathsf{MC}(1,r,r'))$	$mask(\mathsf{MC}(0,r,r'))$	$mask(\mathsf{MC}(1,r,r'))$
$mask(\mathsf{MC}(1,r,r'))$	$mask(\mathsf{MC}(1,r,r'))$	$mask(\mathsf{MC}(0,r,r'))$
$\star$	$\star$	0^4

Figure 1: Truth table of $\widehat{\mathsf{NAND}}_{r,r'} : \{0,1\}^{2\times 4} \to \{0,1\}^4$.

gate which takes as input a masked Manchester encoding using randomness r, r' and outputs two

[5]Notice that the input and the output cascade phases might have a different number of inputs/outputs.

copies of it. Whenever the input contains anything else than a masked Manchester encoding using randomness r, r', the output is $0^8 \in \overline{\mathsf{MMC}}$.

$$\widehat{\mathsf{copy}}_{r,r'}(A) = \begin{cases} A, A & \text{if } A \in \{\mathit{mask}(\mathsf{MC}(0,r,r')), \mathit{mask}(\mathsf{MC}(1,r,r'))\} \\ 0^8 & \text{otherwise.} \end{cases}$$

Finally we let $\widehat{\$}_{r,r'}$ denote a randomness gadget outputting a fresh masked Manchester encoded random bit.

With $\widehat{C}_{r,r'}$ we denote the circuit we get by replacing every wire in C with 4 wires (carrying an encoding in MMC using randomness r, r') and every NAND gate (resp. copy gate, \$ gate) in C with a $\widehat{\mathsf{NAND}}_{r,r'}$ (resp. $\widehat{\mathsf{copy}}_{r,r'}$, $\widehat{\$}_{r,r'}$). Similar to the Π gadgets, we require the $\widehat{\mathsf{NAND}}_{r,r'}$, $\widehat{\mathsf{copy}}_{r,r'}$, $\widehat{\$}_{r,r'}$ gadgets to be *tamper-proof*, i.e. the adversary is allowed to tamper with their inputs and outputs, but cannot modify the internals. Note that if we want to avoid the use of $\widehat{\$}_{r,r'}$ gadgets we can derandomize the original circuit C replacing the \$ gates with the output of a PRG. The core of the transformed circuit consists of the k circuits $\widehat{C}_{r_1,r'_1}, \ldots, \widehat{C}_{r_k,r'_k}$ (where the r_i, r'_i are from ρ_Φ).

4 The Simulator $\mathcal{S}_\lambda$

In this section we define the simulator $\mathcal{S}_\lambda$ and show that she has the property as required by Definition 1, thus proving the tamper-resilience of the compiler defined in the previous section.

4.1 The Bad Events

Consider the experiment $\mathsf{Exp}_\Phi^{\mathrm{Real}}(\mathcal{A}, C, M_0)$. We will say that the adversary "triggered" the self-destruct, if in an invocation of $\widehat{C}$ we get (as a consequence of tampering with $\widehat{C}$) an invalid encoding at the input to a cascade gadget Π. The high level idea behind our circuit compiler, which will make simulation possible, can be summarised as follows:

1. Any kind of tampering attempt is much more likely to trigger self-destruct than to succeed in changing a valid encoding of b into an encoding of $1 - b$.
2. Once self-destruct is triggered, the entire secret state of $\widehat{C}$ gets erased with overwhelming probability.

The reason the latter could actually fail is that even though we have an invalid encoding at the input to Π, the adversary can also tamper with its output (which will be all zeros), potentially changing it back to a valid encoding. We define two event bad_1 and bad_2 which hold if the first or second of the unlikely cases above occurs.

$\mathsf{bad}_1 = 1$ if in the experiment $\mathsf{Exp}_\Phi^{\mathrm{Real}}(\mathcal{A}, C, M_0)$ at some point the encoding $\mathsf{Enc}(b, \rho_\Phi)$ occurs either at the output of the core of $\widehat{C}$ or within a cascade phase, whereas in the untampered circuit this value should be $\mathsf{Enc}(1 - b, \rho_\Phi)$. Moreover this happens before self-destruct was triggered.

$\mathsf{bad}_2 = 1$ if self-destruct is triggered, but "undone" before the entire state has been erased. Notice that if the Π gadget where self-destruct is triggered lies in the first half of the (input or output) cascade phase, then the entire state is erased if the inputs to all the following Π's in this phase are not valid, as in this case the output of the cascade phase is all zeros. If the

Π gadget lies in the second half of the input (resp. output) cascade phase, then the state is erased if all inputs to the Π gadgets in the first half of the next output (resp. input) cascade phase are not valid encodings.

4.2 Description of $\mathcal{S}_\lambda$

The simulator $\mathcal{S}_\lambda$ must "fake" the experiment $\mathsf{Exp}_\Phi^{\mathrm{Real}}(\mathcal{A}, C, M_0)$, which basically means to simulate the answers coming from the tampering oracle $\mathsf{Tamper}(\widehat{C}, \widehat{M}_0, \cdot, \cdot)$, having only black-box access to $C[M_0](\cdot)$.

$\mathcal{S}_\lambda$ first samples all the coin tosses for the experiment $\mathsf{Exp}_\Phi^{\mathrm{Real}}(\mathcal{A}, C, M_0)$. This includes the following uniform coin tosses: the coins $\rho_\mathcal{A}$ for the adversary and the coins ρ_Φ for the transformation. Additionally, $\mathcal{S}_\lambda$ samples a sufficiently long string ρ_E which will model the failure of each tampering attempt. The bits of ρ_E are iid. and each bit is 1 with probability δ ($\rho_E[i] = 1$ meaning the ith tampering attempt fails).

Next, the simulator can define the auxiliary input function $f : \{0,1\}^* \to \{0,1\}^\lambda$. The function f gets M_0 as input, and thus can completely simulate the experiment $\mathsf{Exp}_\Phi^{\mathrm{Real}}(\mathcal{A}, C, M_0)$ using the randomness just sampled. This experiment defines the following three values, which are f's output:

- $\mathsf{abort} \in \{0,1\}$ is a predicate which is 1 if either of the predicates bad_1 or bad_2 (as defined above) is 1.
- $q^* \in [1, q]$ specifies the round (if there is any) in which a self-destruct is triggered, that is, the first round where an input from $\overline{\mathsf{Enc}}$ appears as input to a cascade gadget Π.
- $n^* \in [0, n]$ specifies which cascade gadget this is, as illustrated in Figure 8 on page 22. If this is not one of the gadgets computing the final (public and secret) output we set $n^* = 1$. If this is one of the gadgets computing the secret output we set $n^* = 0$. Otherwise n^* specifies the gadget exactly.

After getting this auxiliary input the simulator checks if $\mathsf{abort} = 1$, in this case the simulator stops with output "simulation failed". Otherwise $\mathcal{S}_\lambda$ runs the experiment $\mathsf{Exp}_\Phi^{\mathrm{Real}}(\mathcal{A}, C, M_0)$ using $\rho_\mathcal{A}$ as $\mathcal{A}$'s randomness. We will show in Lemma 1 below that $\mathcal{S}_\lambda$ can do so perfectly. Then we show in Lemma 2 that the probability that $\mathsf{abort} = 1$ is very low.

Lemma 1. *There exists a simulator $\mathcal{S}_\lambda$ (and we define it in the proof of this lemma) such that whenever $\mathsf{abort} = 0$ the simulation is perfect, i.e. the output is exactly the output of the real experiment $\mathsf{Exp}_\Phi^{\mathrm{Real}}(\mathcal{A}, C, M_0)$ (where the experiment uses the same randomness as sampled by $\mathcal{S}_\lambda$).*

Lemma 2. *The probability that f returns $\mathsf{abort} = 1$ is at most $3(1 - \delta/2)^k$.*

These lemmas, whose proofs we postpone for a moment, imply our main theorem.

Theorem 1 (Tamper-resilience against (∞, δ, q)-adversaries). *Let $0 < \delta < 1/2$, $k > 0$. The compiler Φ of Section 3 is $(\infty, \delta, q, \lambda, \epsilon)$-tamper resilient, where $\lambda = \log(q) + \log(n+1) + 1$ and $\epsilon = 3(1 - \delta/2)^k$.*

Proof. Let $\mathcal{R}$ be the randomness space of the experiment $\mathsf{Exp}_\Phi^{\mathrm{Real}}(\mathcal{A}, C, M_0)$. For $\rho \in \mathcal{R}$ let $\mathsf{Exp}_\Phi^{\mathrm{Real}}(\mathcal{A}, C, M_0)[\rho]$ denote the outcome of the experiment when using randomness ρ. Similarly let $\mathsf{Exp}^{\mathrm{Sim}}_{\Phi}(\mathcal{S}_\lambda, C, M_0, \mathcal{A})[\rho]$ denote the outcome of the experiment $\mathsf{Exp}^{\mathrm{Sim}}_{\Phi}(\mathcal{S}_\lambda, C, M_0, \mathcal{A})$, assuming $\mathcal{S}_\lambda$ initially samples randomness ρ. For any ρ where $\mathsf{abort} = 0$ we have by Lemma 1

$$\mathsf{Exp}^{\mathrm{Sim}}_{\Phi}(\mathcal{S}_\lambda, C, M_0, \mathcal{A})[\rho] = \mathsf{Exp}^{\mathrm{Real}}_{\Phi}(\mathcal{A}, C, M_0)[\rho].$$

By Lemma 2 for a random $\rho \in \mathcal{R}$ we have $\mathsf{abort} = 1$ with probability at most ϵ. This implies that for a random ρ, the output of the two experiments is ϵ-close, i.e.

$$\mathsf{Exp}^{\mathrm{Sim}}_{\Phi}(\mathcal{S}_\lambda, C, M_0, \mathcal{A}) \approx_\epsilon \mathsf{Exp}^{\mathrm{Real}}_{\Phi}(\mathcal{A}, C, M_0),$$

as claimed. □

4.3 Proof of Lemma 1

Proof. We have to specify how (in the case $\mathsf{abort} = 0$) the simulator answers $\mathcal{A}$'s queries to the $\mathsf{Tamper}(\widehat{C}, \widehat{M}_0, \cdot, \cdot)$ oracle. The answers must be exactly the same as the answers in the experiment $\mathsf{Exp}^{\mathrm{Real}}_{\Phi}(\mathcal{A}, C, M_0)[\rho]$ when using the same randomness $\rho = \{\rho_E, \rho_{\mathcal{A}}, \rho_\Phi\}$ as chosen by $\mathcal{S}_\lambda$.

The first $q^* - 1$ queries. For $i = 1, \ldots, q^* - 1$, if $\mathcal{A}$ makes the query (X_i, W_i) the simulator forwards X_i to $C[M_0](\cdot)$ and gives the output Y_i to $\mathcal{A}$.

Queries $q^* + 1$ to q. As $\mathsf{abort} = 0$ (and thus $\mathsf{bad}_2 = 0$), in the query q^* or $q^* + 1$ there is some point in the evaluation of $\widehat{C}$ where all wires are 0, and the simulator knows this point (from q^*, n^*). Moreover, as the simulator chooses the randomness ρ of the experiment, she knows the state of $\widehat{C}$ exactly, in particular which wires were successfully tampered. Thus from the point where all wires are 0, the simulator can continue to compute the answers of the $\mathsf{Tamper}(\widehat{C}, \widehat{M}_0, \cdot, \cdot)$ oracle itself.

The q^*th query. We haven't yet covered the query q^*. This query is a hybrid between the two cases considered above. If $n^* = 0$ (i.e. the self-destruct is triggered at the end of the output cascade phase after the public output is already computed) we can handle this query exactly like the first $q^* - 1$ queries. Also if $n^* = 1$ (i.e. self-destruct is triggered early in the query, before the cascade gadgets outputting the final public output were invoked), this query can be handled like in the previous case.

If $n^* > 1$ the simulator first queries (as in the first $q^* - 1$ queries) for Y_{q^*}, but before forwarding this value to $\mathcal{A}$ she must adapt it to take into account that parts of it were deleted: the first $n^* - 1$ bits of Y_{q^*} remain unchanged, the others are set to 0.

Finalising. Finally $\mathcal{S}_\lambda$ outputs whatever $\mathcal{A}$ outputs. By inspection, the queries above have exactly the same distribution as the outputs of the $\mathsf{Tamper}(\widehat{C}, \widehat{M}_0, \cdot, \cdot)$ oracle in the experiment $\mathsf{Exp}^{\mathrm{Real}}_{\Phi}(\mathcal{A}, C, M_0)[\rho]$. □

4.4 Proof of Lemma 2

We will first state some properties of the transformed circuit $\widehat{C}$ which will be used in the proof of the lemma. The proofs of these properties can be found in Appendix C. We first show (Lemma 3 below) that tampering with (a non-empty subset of) four wires holding a masked Manchester encoded value – where tampering with a wire fails independently with probability δ and the adversary may know the encoded value, but not the randomness used in the encoding – will result in an invalid value with probability $\delta/2$.

Tampering with Masked Manchester Encodings. Let $0 < \delta < 1/2$ and consider the following game.

- An adversary chooses $b \in \{0, 1, \perp\}$ and four functions $f_1, f_2, f_3, f_4 : \{0, 1\} \to \{0, 1\}$ of which at least one is not the identity function.
- If $b = \perp$ set $(x_1, x_2, x_3, x_4) = (0, 0, 0, 0)$, otherwise sample random $r, r' \in \{0, 1\}$ and let $(x_1, x_2, x_3, x_4) = \mathit{mask}(\mathsf{MC}(b), (r, r'))$ be the masked Manchester encoding of b.
- For each $i \in \{1, 2, 3, 4\}$, with probability $1 - \delta$ set $y_i = f_i(x_i)$ and $y_i = x_i$ otherwise.

Lemma 3 (**Tampering with** MMC)**.** *The probability that at the end of the experiment just described* $(y_1, y_2, y_3, y_4) \in \overline{\mathsf{MMC}}$ *(i.e. it is not a valid masked Manchester encoding) is at least* $\delta/2$.

Recall that $\mathsf{abort} = 1$ if either bad_1 or bad_2 (as defined at the beginning of this section) are 1. We upper bound the probability that $\mathsf{bad}_1 = 1$ and $\mathsf{bad}_2 = 1$ below.

Bounding $\Pr[\mathsf{bad}_1 = 1]$**.** By Lemma 3, tampering with the output wires of a $\widehat{\mathsf{NAND}}_{r,r'}$ (or $\widehat{\mathsf{copy}}_{r,r'}$) gate (which is 0^4 or $\mathit{mask}(\mathsf{MC}(b), r, r')$) will result in an invalid encoding with probability at least $\delta/2$. We next show a general *composition lemma*, which essentially states that the property proven in Lemma 3 *composes* to an arbitrary subcircuit made of masked Manchester gadgets as used in $\widehat{C}$.

Lemma 4 (**Tampering with the core of** $\widehat{C}$)**.** *The probability that in the experiment* $\mathsf{Exp}_{\Phi}^{\mathrm{Real}}(\mathcal{A}, C, M_0)$ *we have an encoding* $\mathsf{Enc}(b, \rho_\Phi)$ *at the output of the core, where in the untampered circuit this values would be* $\mathsf{Enc}(1 - b, \rho_\Phi)$ *and this happens before self-destruct is triggered, is at most* $(1 - \delta/2)^k$.

The above lemma does not cover all cases of the $\mathsf{bad}_1 = 1$ case. The other possibility to get $\mathsf{bad}_1 = 1$ is by tampering within the cascade phase, hoping to change $\mathsf{Enc}(b, \rho_\Phi)$ into $\mathsf{Enc}(1-b, \rho_\Phi)$. The probability that this succeeds can also be bounded by $(1 - \delta/2)^k$. As the proof is similar (but somewhat simpler) that the proof of Lemma 4 we omit it here. Taking both cases into account, we get

$$\Pr[\mathsf{bad}_1 = 1] \leq 2(1 - \delta/2)^k.$$

Bounding $\Pr[\mathsf{bad}_2 = 1]$**.** Recall that $\mathsf{bad}_2 = 1$ if the adversary manages to change the output 0^{8k} of a Π gadget (which was queried on an invalid input) back to something valid, i.e. $\mathsf{Enc}(b, \rho_\Phi), \mathsf{Enc}(b', \rho_\Phi)$ for some bits $b, b' \in \{0, 1\}$. Even in the case $\delta = 0$ (i.e. when tampering always succeeds), we can upper bound the probability that the adversary can "undo" tampering on one particular Π gadget by 2^{-2k}. To see this assume an adversary changes 0^{8k} to a valid encoding with advantage ϵ. In this case we can extract ρ_Φ from its tampering queries, but as $\rho_\Phi \in \{0, 1\}^{2k}$ is uniform (and the answers from the Tamper oracle are independent of ρ_Φ), it follows that the adversary has $2k$ bits min-entropy about ρ_Φ and thus $\epsilon \leq 2^{-2k}$. Taking into account that the adversary can use different guesses for ρ_Φ on the outputs of the different Π gadgets, the probability that $\mathsf{bad}_2 = 1$ is $t \cdot 2^{-2k}$ where t is the number of Π gadgets in $\widehat{C}$. Assuming $t \cdot 2^{-2k} \leq (1 - \delta/2)^k$ (which is the case for any interesting range of parameters) we get

$$\Pr[\mathsf{bad}_2 = 1] \leq (1 - \delta/2)^k.$$

Acknowledgments

We thank Yuval Ishai and Manoj Prabhakaran for helpful discussions on their work in [18].

References

[1] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 474–495. Springer, 2009.

[2] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In Halevi [17], pages 36–54.

[3] Ross Anderson and Markus Kuhn. Tamper resistance: a cautionary note. In *WOEC'96: Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1–1, Berkeley, CA, USA, 1996. USENIX Association.

[4] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. to appear in the 2nd Symposium on Innovations in Computer Science (ICS 2011), 2011. Full version available on `http://eprint.iacr.org/`.

[5] Mihir Bellare and David Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In Rabin [29], pages 666–684.

[6] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 491–506. Springer, 2003.

[7] Johannes Blömer and Jean-Pierre Seifert. Fault based cryptanalysis of the advanced encryption standard (AES). In Rebecca N. Wright, editor, *Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2003.

[8] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.

[9] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In Michael Mitzenmacher, editor, *STOC*, pages 621–630. ACM, 2009.

[10] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.

[11] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In Andrew Chi-Chih Yao, editor, *ICS*, pages 434–452. Tsinghua University Press, 2010.

[12] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In Daniele Micciancio, editor, *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 2010.

[13] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 135–156. Springer, 2010.

[14] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In Naor [25], pages 258–277.

[15] David Goldenberg and Moses Liskov. On related-secret pseudorandomness. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 255–272. Springer, 2010.

[16] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In Rabin [29], pages 59–79.

[17] Shai Halevi, editor. *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*. Springer, 2009.

[18] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits II: Keeping secrets in tamperable circuits. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 308–327. Springer, 2006.

[19] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[20] Antoine Joux, editor. *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*. Springer, 2009.

[21] Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In Rabin [29], pages 41–58.

[22] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 703–720. Springer, 2009.

[23] Stefan Lucks. Ciphers secure against related-key attacks. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 359–370. Springer, 2004.

[24] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Naor [25], pages 278–296.

[25] Moni Naor, editor. *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*. Springer, 2004.

[26] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In Halevi [17], pages 18–35.

[27] Martin Otto. *Fault Attacks and Countermeasures.* PhD thesis, University of Paderborn, Germany, 2006.

[28] Krzysztof Pietrzak. A leakage-resilient mode of operation. In Joux [20], pages 462–482.

[29] Tal Rabin, editor. *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*. Springer, 2010.

[30] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

[31] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Joux [20], pages 443–461.

A Related Work

A special case of tampering attacks are "related-key" attacks, where an adversary is allowed to query a primitive not only under the target key, but under other "related" (chosen by her) keys derived from it. A theoretical framework for this setting has been developed for pseudorandom functions and permutations [6, 23, 5, 15] and semantic security [4].

Gennaro et al. [14] consider a model where an adversary can attack a circuit G with secret state S; G could e.g. compute digital signatures where S contains the secret key. The adversary can make regular queries to the cryptosystem, where on input X she receives $G(S, X)$. Additionally she can make "tamper" queries, where she chooses a function f, and the secret state is replaced with $f(S)$. The solution they propose is to sign the state. More precisely, they compile the circuit/state pair G, S into G', S' where the new state $S' = \{S, \sigma\}$ contains a digital signature $\sigma = sign(sk, S)$. The circuit $G'(S')$ parses $S' = \{S, \sigma\}$, checks if σ is a valid signature for S, and if so, outputs $G(S)$; otherwise it "self-destructs". The main advantage of this solution is that it works for any *efficient tampering function* f, but some problems remain. For example G has to be stateless (i.e. it cannot overwrite its state) as the public G' cannot contain the secret signing key sk. Moreover, as in our work, the attacker can learn some information about S by using tamper-queries (though at most $log(q)$ bits where q is the number of tamper-queries).

In [11] the notion of "non-malleable codes" is proposed. It is shown that by encoding the state S (instead of appending a signature) one can avoid most of the above problems. We refer to [11] for a detailed discussion.

There are two fundamental difference between our work and the line of research presented in [6, 23, 5, 15, 4, 14, 11]. On the one hand the adversary considered in [6, 23, 5, 15, 4, 14, 11] is much stronger, since she can apply tampering attacks from a much larger class of tampering functions (in the work of [14] even arbitrary polynomial time functions). On the other hand, we (as well as [18]) consider adversaries that tamper with the entire computation. This is in contrast to [6, 23, 5, 15, 4, 14, 11], where the adversary is restricted solely to attack the memory, but the computation is assumed to be completely tamper proof.

Other Circuit Compilers. We will briefly discuss some other circuit compilers that protect against physical attacks. In [19] Ishai et al. consider general circuit compilers that produce circuits resilient to passive observing adversaries. In particular, they present a circuit compiler such that no adversary will succeed in breaking the cryptographic scheme when probing up to t wires. Notice that here probes are released in every clock cycle so huge parts of the computation stay oblivious to the adversary. Recently, this result has been generalized in [13]. In this work circuit compilers are proposed, where the compiled circuit remains secure even if the attacker can learn a global low-complexity (or noisy) function of the secret state and all intermediate results. Both compilers make heavy use of randomness and have a blow-up in size of $O(k^2)$ for some security parameter k.

More recently Juma and Vahlis [21] and Goldwasser and Rothblum [16] proposed circuit compilers for leakage that adheres the "only computation leaks information" assumption. As in this work, the compilers from [13, 21, 16] require special leak-free hardware.

1st Input	2nd Input	Output
0^{2kt}	0^{2kt}	1^{2kt}
0^{2kt}	1^{2kt}	1^{2kt}
1^{2kt}	0^{2kt}	1^{2kt}
1^{2kt}	1^{2kt}	0^{2kt}
$\star$	$\star$	$0^{kt}1^{kt}$

Figure 2: Truth table of a $\widehat{\mathsf{NAND}}$ gadget in the case of $(t, 0, q)$-adversaries.

1st Input	2nd Input	Output
0^{2kt}	0^{2kt}	$(0^{2kt}, 0^{2kt})$
0^{2kt}	1^{2kt}	$(0^{2kt}, 1^{2kt})$
1^{2kt}	0^{2kt}	$(1^{2kt}, 0^{2kt})$
1^{2kt}	1^{2kt}	$(1^{2kt}, 1^{2kt})$
$\star$	$\star$	$(0^{kt}1^{kt}, 0^{kt}1^{kt})$

Figure 3: Truth table of a cascade gadget in the case of $(t, 0, q)$-adversaries.

B A Compiler Secure against $(t, 0, q)$-Adversaries

In [18] Ishai et al. proposed a circuit transformation Φ_{IPSW} for $\delta = 0$ and t being a small integer. In this section we consider this setting but allow the adversary to learn a small amount of information (this is captured in our definition by the leakage Λ). Similar to the transformation from the previous section, aiming for a weaker security notion results in significant efficiency improvements.

Before we discuss the details of our new compiler Φ_t, let us outline why the transformation from Section 3 is not secure when the failure probability of attacks on individual wires is highly correlated (the model of [18] is such a case). In a nutshell, the reason is that the gadgets in the core operate on encodings of constant length c. A $(t, 0, q)$-adversary can successfully tamper with such encodings if $c \leq t$ (e.g. she can fix it to a fixed encoding). In this section we outline how to significantly improve the efficiency of the compiler Φ_{IPSW} when a small amount of leakage is tolerated.

For readers familiar with [18] we recall the compiler Φ_{IPSW} here. The transformation Φ_{IPSW} uses randomization and redundant encodings and works in two steps. First, using the transformation from [19] the original circuit $C[M_0]$ is transformed into a randomized circuit $C'[M_0']$. That is, each wire w in C is represented by a k-bit wire bundle in C' of the form $(r_1, \ldots, r_{k-1}, r_1 \oplus \ldots \oplus r_{k-1} \oplus w)$. The gates in C are replaced by gadgets that operate on such randomized encodings. The gadgets have to satisfy the invariant that learning up to $k - 1$ wires does not reveal any information about the encoded computation. This first transformation blows up the size of the circuit by $O(k^2)$.

The circuit C' uses standard Boolean gates. In the next step each wire in C' (that is part of a wire bundle) is encoded with a $2kt$ bit long redundant encoding. The encoding that is used maps 0 to 0^{2kt} and 1 to 1^{2kt}. The value $0^{kt}1^{kt}$ is a special invalid state. The Boolean gates in C' are then replaced using the $\widehat{\mathsf{NAND}}$ gadgets given in Figure 2. These gadgets compute with the encoding just outlined above. This concludes the description of the core of the transformed circuit $\widehat{C}$. The output of the core is given as input to a cascade phase. This is essentially identical to the one described in Section 3 but built with the cascade gadgets shown in Figure 3. Notice that the gadgets in Figure 2 and 3 are *not* atomic. Instead they can be built from standard Boolean gates and tamper-proof AND gates that take $4kt$ bits as input. The result of the above outlined transformation gives us $\widehat{C}[\widehat{M}_0]$.

If we allow the tampering adversary to learn a small amount of information, then we can eliminate most of the overhead resulting from the first step. Our transformation Φ_t essentially follows the transformation Φ_{IPSW} with two changes. First, instead of randomizing each wire in C with $k-1$ random bits (to achieve statistical security with security parameter k), in C' we use only a single bit of randomness. More precisely, each wire w_i in C is represented in C' by two wires that carry the values $(w_i \oplus r_i, r_i)$. The computation in C' is done in such a way that learning the value of a *single* wire in C' does not reveal information. This can be done with the techniques introduced in [19] (i.e. replacing each gate in C by gadgets that are constructed from standard Boolean gates and probabilistic gates). Second, the gadgets in Figure 2 and 3 are tamper-proof. That is, an adversary can tamper with its inputs and outputs but the internals are not subject to attacks. These tamper-proof gadgets can be implemented in size linear in kt.

We give some further details below. For security parameter k we define the probabilistic encoding scheme $\mathsf{Enc}_t : \{0,1\} \to \{0,1\}^{4kt}$ as $\mathsf{Enc}_t(b) = (b^{2kt} \oplus r^{2kt}, r^{2kt})$, where $r \leftarrow \{0,1\}$. Each bit b of the initial secret state M_0 is then replaced by $\mathsf{Enc}_t(b)$. Hence, the memory cells in C are replaced by $4kt$ memory cells.

As the transformation in Section 3 $\widehat{C}$ is structured in three phases: the encoder/decoder, the input/output cascade phases and the core. The encoder encodes every input bit b with Enc_t. The decoder takes the first bit of the encoding and the $(2kt+1)$th bit and outputs the XOR. The input/output cascade phase is similar to the one described in Section 3, with the difference that it makes use of the tamper-proof cascade gadgets of Figure 3.

Transformation of the core. Essentially, $\widehat{C} \leftarrow \Phi_t(C)$ operates with encodings Enc_t instead of plain bits. To achieve this it proceeds in two steps. First every gate in C is replaced by gadgets (built from Boolean and randomness gates) that operate with masked bits, i.e. $(b \oplus r, r)$. This can for instance be done with the transformation Φ_{ISW} from [19]. Next each gate in C' is replaced with $\widehat{\mathsf{NAND}}$ gadgets of Figure 2. This gives us the transformed circuit $\widehat{C}$. Notice that even if the original circuit C was deterministic, the transformed circuit $\widehat{C}$ will contain randomness gates (this is due to the transformation Φ_{ISW}). We can use the PRG construction proposed in [18] to de-randomize $\widehat{C}$.[6]

It is easy to see that changing the transformation Φ_{IPSW} as outlined above decreases the size of $\widehat{C}$ by a factor of $O(k^2)$. Also, the amount of randomness is decreased from $O(sk^2)$ to $O(s)$ (where s is the size of C).

We prove security of Φ_t using the techniques developed in Section 4. This requires to show that any attempt to change a valid encoding in the core (or the cascade phases) results with high probability in an invalid encoding at the output of $\widehat{C}$'s core.

Lemma 5 (Tampering with the core and/or with the cascade phase of $\widehat{C}$). *Consider any circuit C and its transformation $\widehat{C} = \Phi_t(C)$. Every $(t, 0, q)$-adversary tampering inside the core of $\widehat{C}$ either lets the computation in $\widehat{C}$ unchanged or results with probability at least $1 - 2^{-2k}$ to an invalid encoding on some wire bundle at the output of the core.*

Proof. Since $\widehat{C}$ only uses tamper-proof $\widehat{\mathsf{NAND}}$ and cascade gadgets, the attack strategy of the adversary can only include the inputs and outputs of such gadget. Notice that the minimal Hamming distance between two *valid* encodings is $d_H(0^{2kt}, 1^{2kt}) = 2kt$. Since the adversary is only allowed to tamper with at most t wires in every round, either fixing or changing an encoding will require $2k$ rounds. The main observation is that the randomness of the masking is refreshed in each round.

[6]Notice that here we require a special PRG (namely the one from [18]), while in Section 3 we could just use any PRG for de-randomization. The reason for this is that in Section 3, if C is deterministic, then so is $\widehat{C}$. On the other hand if C is probabilistic we can make it deterministic using any standard PRG, and only afterwards apply our transformation.

1st Input	2nd Input	Output
$\mathsf{MC}(0)$	$\mathsf{MC}(0)$	$\mathsf{MC}(1)$
$\mathsf{MC}(0)$	$\mathsf{MC}(1)$	$\mathsf{MC}(1)$
$\mathsf{MC}(1)$	$\mathsf{MC}(0)$	$\mathsf{MC}(1)$
$\mathsf{MC}(1)$	$\mathsf{MC}(1)$	$\mathsf{MC}(0)$
$\star$	$\star$	$\perp^*$

Figure 4: Truth table of a $\widehat{\mathsf{NAND}}$ gadget when only *reset* attacks are allowed.

Hence, the adversary is successful in each round with probability at most 1/2. This concludes the proof. □

Similar to the proof of Section 1 we can use Lemma 5 to show tamper-resilience against $(t, 0, q)$-adversaries.

Theorem 2 (**Tamper-resilience against** $(t, 0, q)$**-adversaries**). *Let $k > 0$ be the security parameter and $t \in \mathbb{N}$. Let C be a Boolean circuit with n bits of public output. For any $q = q(k)$, the compiler Φ_t described above is $(t, 0, q, \lambda, \epsilon)$-tamper resilient, where $\lambda = \log(q) + \log(n+1) + 1$ and $\epsilon = 2^{-2k}$.*

Proof. The proof is along the lines of Theorem 1 and is therefore omitted. □

C Proofs Omitted from the Main Body

C.1 Proof of Lemma 3

We first look at the case where at least one of the f_i's – for concreteness say f_4 – is a toggle function, i.e. $f_4(x_4) = 1 - x_4$. Any three bits of a masked Manchester encoding uniquely determine the fourth. In particular, if $(y_1, y_2, y_3, y_4) \in \mathsf{MMC}$ then $y_1 \oplus y_2 \oplus y_3 \oplus 1 = y_4$. Recall that $y_4 = x_4$ with probability δ (i.e. we don't apply f_4) and $y_4 = 1 - x_4$ with probability $1 - \delta$. In one of the two cases we get $(y_1, y_2, y_3, y_4) \in \overline{\mathsf{MMC}}$, thus the encoding is invalid with probability at least δ.

Consider now the case where at least one of the f_i's – for concreteness say f_4 – is a set function, i.e. $f_4(x_4) = 1$ (the proof for the reset case is identical). As $\Pr[x_4 = 0] = 1/2$ and $y_4 = x_4 = 0$ with probability δ, we get $\Pr[y_4 = 0] = \delta/2$. Again, only in one of the two cases ($y_4 = 0$ or $y_4 = 1$) we get $(y_1, y_2, y_3, y_4) \in \overline{\mathsf{MMC}}$, thus the encoding is invalid with probability at least $\delta/2$.

C.2 Proof of Lemma 4

Recall that the core of $\widehat{C}$ is made-up of k independent subcircuits $\widehat{C}_{r_1, r_1'}, \ldots, \widehat{C}_{r_k, r_k'}$, each using *independent* randomness r_i, r_i'. The output of each subcircuits when no tampering took place, consists of the original output in masked Manchester encoded form (masked using the same randomness).

The first important observation is that if the adversary wants to change one of the encodings at the output of the core, she must tamper with *every* subcircuit (either in a single round or in different rounds). Consider the very last level where the adversary applies a tampering attack in a given subcircuit, say $1 \leq \ell \leq \mathit{depth}(\widehat{C}_{r_i,r'_i})$.[7] The value $\ell = 1$ means that the adversary tampers only with the input of the subcircuit, whereas the value $\ell = \mathit{depth}(\widehat{C}_{r_i,r'_i})$ means that the attack involves also the encoded output. Since any tampering strategy ending at level ℓ involves the output of some masked Manchester gadget, Lemma 3 says that the output of that gadget is in $\overline{\mathsf{MMC}}$ with probability at least $\delta/2$. The key observation here is that, since ℓ is the last level where the adversary applies an attack, the invalid encoding will propagate to the output of the subcircuit. Thus the output of that subcircuit has at least one value in $\overline{\mathsf{MMC}}$ with probability at least $\delta/2$. Moreover this argument applies to all the subcircuits since they use *independent* randomness. As a consequence with probability at least $1 - (1 - \delta/2)^k$ one of the wire bundles at the output carries a value in $\overline{\mathsf{Enc}}$, and thus triggers self-destruct.

[7]Since the transformation in the core is gate-by-gate the depth of $\widehat{C}_{r_i,r'_i}$ is the same as the depth of the original circuit C.

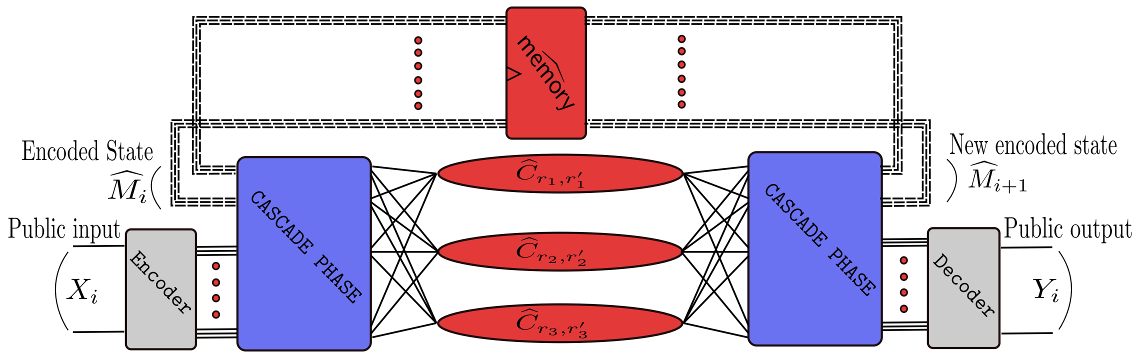

Figure 5: A global picture of our compiler in the case $k = 3$. In the red-coloured parts we rely on gadgets of constant size, whereas in the blue-coloured parts gadgets of linear size (in the security parameter k) are used.

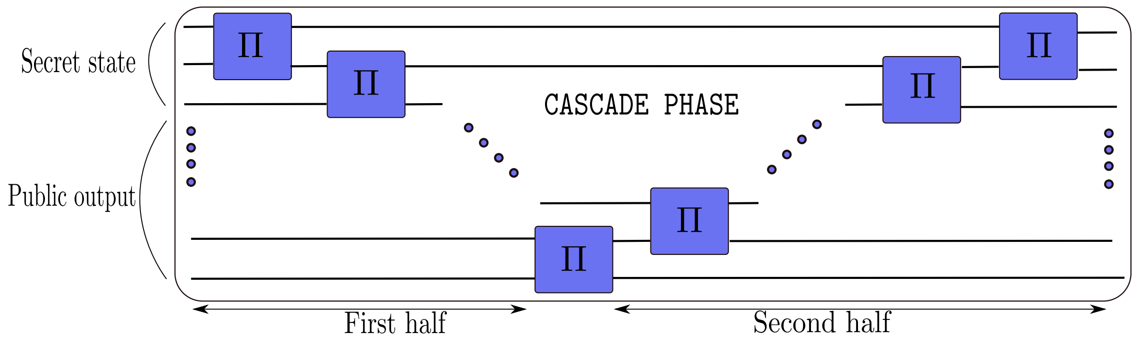

Figure 6: The cascade phase for error propagation and self-destruction. Every cascade gadget (in blue) has size linear in the security parameter k.

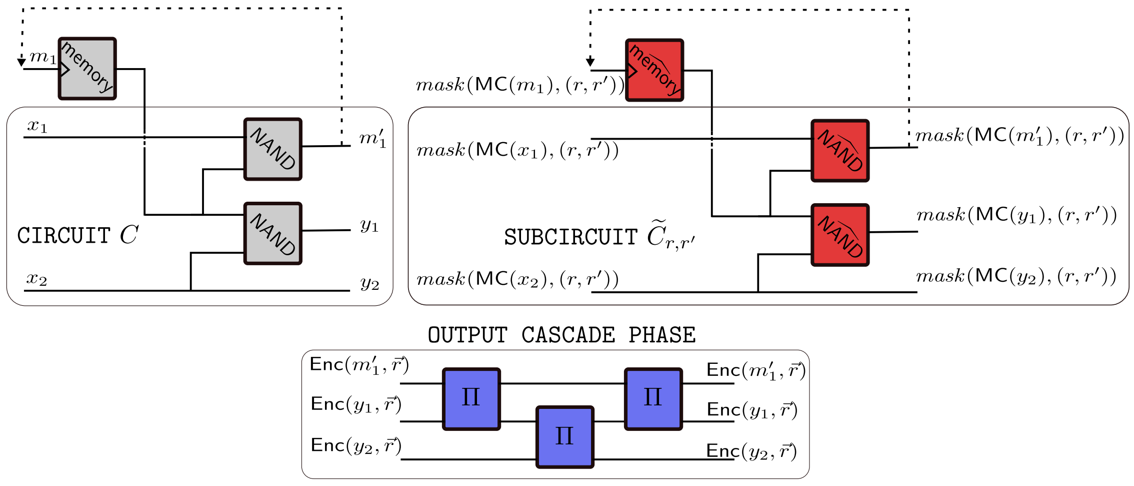
m_1
memory
x_1
NAND
m'_1
CIRCUIT C
y_1
x_2
y_2
$mask(\mathsf{MC}(m_1),(r,r'))$
$mask(\mathsf{MC}(x_1),(r,r'))$
$mask(\mathsf{MC}(m'_1),(r,r'))$
SUBCIRCUIT $\widetilde{C}_{r,r'}$
$mask(\mathsf{MC}(y_1),(r,r'))$
$mask(\mathsf{MC}(x_2),(r,r'))$
$mask(\mathsf{MC}(y_2),(r,r'))$
OUTPUT CASCADE PHASE
$\mathsf{Enc}(m'_1,\vec{r})$
$\mathsf{Enc}(y_1,\vec{r})$
$\mathsf{Enc}(y_2,\vec{r})$
Π

Figure 7: The compiler Φ applied to a concrete circuit C (in the top left picture). The top right picture shows one of the subcircuits $\widehat{C}_{r,r'}$ in the core of the compiled circuit $\widehat{C}$. The gadgets used in $\widehat{C}_{r,r'}$ are all of constant size. Finally the bottom picture shows the output cascade phase.

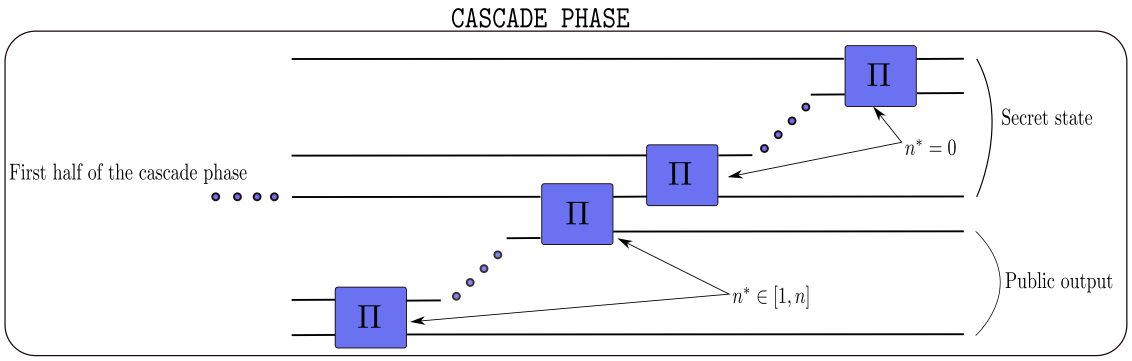

Figure 8: Cascade gadgets in the second half of the output cascade phase are assigned a label in $[1, n]$. The auxiliary information Λ includes the label n^* of the *first* cascade gadget where $\perp^*$ appears as an input.