# Securing Circuits Against Constant-Rate Tampering

Dana Dachman-Soled and Yael Tauman Kalai

Microsoft Research New England

**Abstract.** We present a compiler that converts any circuit into one that remains secure even if a *constant fraction* of its wires are tampered with. Following the seminal work of Ishai *et. al.* (Eurocrypt 2006), we consider adversaries who may choose an arbitrary set of wires to corrupt, and may set each such wire to 0 or to 1, or may toggle with the wire. We prove that such adversaries, who *continuously* tamper with the circuit, can learn at most *logarithmically many bits* of secret information (in addition to black-box access to the circuit). Our results are information theoretic.

**Key words:** side-channel attacks, tampering, circuit compiler, PCP of proximity

## 1 Introduction

In recent years, there has been a proliferation of physical attacks on cryptographic schemes. Such attacks exploit the implementation (rather than the functionality) of the scheme. For instance, Kocher *et al.* [46] demonstrated how one can possibly learn the secret key of an encryption scheme by measuring the power consumed during an encryption operation, or by measuring the time it takes for the operation to complete [45]. Other types of physical attacks include: inducing faults to the computation [5, 7, 46], using electromagnetic radiation [30, 55, 54], and several others [54, 43, 47, 35]. These physical attacks have proven to be a significant threat to the practical security of cryptographic devices.

Traditional cryptographic models do not take such attacks into account since they idealize the parties' interaction and implicitly assume that an adversary may only observe an honest party's input-output behavior. Recently, a large and growing body of research has sought to introduce more realistic models and to secure cryptographic systems against such physical attacks. The vast majority of these works focus on securing cryptographic schemes against various *leakage* attacks (e.g. [10, 38, 49, 31, 37, 20, 51, 1, 50, 42, 18, 17, 24, 39, 34]). In these attacks an adversary plays a passive role, learning information about the honest party through side-channels but not attempting to interfere with the honest party's computation. However, as mentioned above, physical attacks are not limited to leakage, and include active *tampering* attacks, where an adversary may actively modify the honest party's memory or circuit. The focus of this work is to construct schemes that are secure even in the presence of tampering (we elaborate on related work in Section 1.3).

### 1.1 Our Results

We show a general compiler that converts any circuit into one that is resilient to (a certain form of) tampering. We consider the tampering model of Ishai *et al.* [37]. Namely, we consider *adversarial* faults, where during each run the computationally unbounded adversary can tamper with any (bounded) set of wires of his choice.

We note that we cannot hope to get correctness, since the adversary may simply tamper with the final output wire. Thus, following [37], we do not attempt to guarantee correctness, but instead devote our efforts to ensuring *privacy*. More specifically, we consider circuits that are associated with a secret state (such as a cryptographic key). We model such circuits as standard circuits (with AND, OR, and NOT gates), with additional secret memory that contains the secret state. The circuit itself is thought of as being public and known to the adversary, whereas the memory content is secret. Following the terminology of [37], we refer to such circuits as *private circuits*. Our goal is to protect the secret state of the circuit even when an adversary may run the circuit on arbitrary inputs while continuously tampering with the circuit.

Unlike the leakage regime, where we build cryptographic schemes that are secure against *arbitrary* (bounded) leakage, in the tampering regime, we focus on limited types of adversarial tampering attacks. Indeed, it is not hard to see that it is impossible to construct private circuits resilient to arbitrary tampering attacks, since an adversary may modify the circuit so that it simply outputs the entire secret state in memory. As in [37], we only allow the adversary to tamper with individual wires [37, 25] and individual memory gates [13, 31, 21]. More specifically, in each run of the circuit we allow the adversary to specify a set of tampering instructions, where each instruction is of the form: Set a wire (or a memory gate) to $0$ or $1$, or toggle with the value on a wire (or a memory gate). However, in contrast to [37], we allow the adversary to tamper with any *constant fraction* of wires and memory gates in the circuit. We note that the tampering allowed in [37] is very local: To be resilient against tampering with $t$ wires per run, they blow up the size of the circuit by a factor of at least $t$; thus their overall tampering rate is very small compared to the size of the circuit.

As noted by [31], it is impossible to construct private circuits resilient against tampering on wires without allowing feedback into memory, i.e. without allowing the circuit to overwrite its own memory. Otherwise, an adversary may simply set to $0$ or $1$ one memory gate at a time and observe whether the final output is modified or not. Thus, the adversary may often learn the entire internal state of the circuit by setting one wire at a time.

Even if we allow feedback, and place limitations on the type of tampering we allow, it is not a priori clear how to build tamper-resilient circuits. As pointed out in [37], the fundamental problem is that the part of the circuit which is supposed to detect tampering and overwrite the memory, may itself be tampered with. Indeed, this "self-destruct" mechanism itself needs to be resilient to tampering.

As in [37], we prove security using a simulation based definition, where we require that for any adversary who continually tampers with the circuit (as described above), there exists a simulator who simulates the adversary's view. However, whereas [37] give the simulator only black-box access to the circuit, we also give the simulator logarithmic number of leakage bits on the secret state. We note that for many applications, leakage

of only logarithmically many bits of the secret key does not break the security of the primitive. This is because an adversary may simply *guess* these bits of leakage on his own.

We also note that our compiler is deterministic, and for deterministic constructions such leakage is necessary. Loosely speaking, this is the case since an adversary can always leak a memory gate of its choice, by checking whether setting that memory gate to $0$ affects the functionality of the circuit. We note that if the compiler is deterministic then many of the memory gates contain "sensitive" information. This is not necessarily the case if the compiler is randomized since then the secret state can be secret-shared in memory, in which case each memory gate on its own may contain a truly random bit.

**Our Results More Formally** We present a general compiler $\mathcal{T}$ that converts a circuit $C$ with a secret state $s$ (denoted by $C_s$) into a circuit $\tilde{C}_{\tilde{S}}$ with a secret state $\tilde{S}$. We consider computationally unbounded adversaries $\mathcal{A}$ who receive access to $\tilde{C}_{\tilde{S}}$ and behave in the following way: $\mathcal{A}$ runs the circuit many times with arbitrary and adaptively chosen inputs. In addition, during each run of the circuit the adversary $\mathcal{A}$ may specify tampering instructions of the form "set wire $w$ to 1","set wire $w$ to 0", "flip value of wire $w$", as well as "set memory gate $g$ to 1", "set memory gate $g$ to 0", "flip value of memory gate $g$", for any wire $w$ or memory gate $g$. We restrict the number of tampering instructions $\mathcal{A}$ may specify per run to be at most $\lambda \cdot \sigma$, where $\lambda > 0$ is some constant and $\sigma$ is the size of the circuit $\tilde{C}_{\tilde{S}}$. Thus, in each run, $\mathcal{A}$ may tamper with a constant fraction of wires and memory gates. Again, we emphasize that our result does not rely on computational assumptions and is completely information theoretic. Moreover, our result does not rely on any source of randomness and our compiler is deterministic.

**Theorem 1 (Main Theorem, Informal).** *There exists a universal constant $\lambda > 0$ and an efficient transformation $\mathcal{T}$ which takes as input any circuit $C_s$ with private state $s$ and outputs a circuit $\tilde{C}_{\tilde{S}}$ with private state $\tilde{S}$ such that the following two conditions hold:*

*Correctness: For every input $x$, $C_s(x) = \tilde{C}_{\tilde{S}}(x)$.*

*Tamper-Resilience: For every adversary $\mathcal{A}$, which may tamper with a $\lambda$-fraction of wires and memory gates in $\tilde{C}_{\tilde{S}}$ per run, there exists a simulator $\mathsf{Sim}$, which can simulate the view of $\mathcal{A}$ given only black-box access to $C_s$ and $\log(T)$ bits of leakage on $s$, where $T$ is an upper bound on the number of times that $\mathcal{A}$ runs the (tampered) circuit.*[1] *Moreover, the runtime of $\mathsf{Sim}$ is polynomial in the runtime of $\mathcal{A}$ and $C_s$.*

Intuitively, the theorem asserts that adversaries who may observe the input-output behavior of the circuit while tampering with at most a $\lambda$-fraction of wires and memory gates in each run, do not get too much extra knowledge over what they could learn from just input-output access to the circuit. More specifically, running the (tampered) circuit $\mathrm{poly}(|s|)$ times leaks at most $O(\log |s|)$ bits.

We remark that this relaxation of allowing $O(\log |s|)$ bits of leakage was first considered by Faust *et. al.* [25]. We additionally mention, as we argued previously, that leaking at most $O(\log |s|)$ bits does not compromise security of many applications.

[1] The reader can think of $T$ as polynomial in the security parameter.

This is because an adversary receiving only input-output access to the circuit may simply *guess* the $O(\log |s|)$ bits of leakage, and his guess will be correct with probability $1/2^{O(\log |s|)} = 1/\operatorname{poly}(|s|)$. Thus, we don't lose much by leaking $O(\log |s|)$ bits of the secret key.

### 1.2 Comparison with Ishai *et al.* [37]

Our work is inspired by the work of [37]. As in our work, they too consider circuits with memory gates, and consider the same type of faults as we do. Similarly to us, they construct a general compiler that converts any private circuit into a tamper resilient one. However our work differs from their work in several ways.

- The tamper resilient circuits in [37] require the use of "randomness gates", which output a fresh random bit in each run of the circuit. Alternatively, [37] can get rid of these randomness gates at the cost of relying on a computational assumption. Their computational result, which does not require randomness gates, relies on the existence of one-way functions and considers only polynomially bounded adversaries. In contrast, our compiler and the resulting tamper-resilient circuits are deterministic, and provide information theoretical security.
- As mentioned previously, [37] constructs tamper resilient circuits that are resilient only to local tampering. More specifically, in order to be resilient to tampering with $t$ wires per run, they blow up the circuit size by a factor of at least $t$. In contrast, our tamper-resilient circuits are resilient to a constant fraction of tampering anywhere in the circuit, and thus are robust to global tampering.
- In the tamper resilient circuits of [37], there is one gate $G$ that has a large fan-out. In practice, however, large fan-out gates do not exist, and instead they are implemented using a *splitter*, which takes as input the output wire of a gate and duplicates the wire many times. Unfortunately, such an implementation is not resilient to tampering with the output wire of $G$ (which is the input wire to the splitter), since if this wire is tampered with, then it causes an immediate tampering with *all* the output wires of the splitter. In contrast, in our work all the gates of the tamper resilient circuits have constant fan-out (and some gates use splitters). However, due to lack of space, in this extended abstract we focus on the initial constructions that do assume the existence of a gate with large fan-out. We refer the reader to the full version [14] for details on how we remove this large fan-out gate.
- One advantage of the tampering model of [37] over our model, is that their model allows for"persistent faults", e.g, if a value of some wire is fixed during one run, it remains set to that value in subsequent runs. We note that in our case, we cannot in general allow persistent faults across runs of the circuit. However, we allow "persistent faults" on memory gates so if a memory value is modified during one run, it remains modified for all subsequent runs.
- Another advantage of [37] is that their security definition is slightly stronger than ours. They guarantee that any adversary that continually tampers with the circuit (as described above) can be simulated given only black-box access to the circuit, whereas in our security definition, we give the simulator in addition logarithmically many bits of information.

**Extending our result to allow for both Leakage and Tampering.** Note that so far, we considered adversaries who only tamper with the wires and the memory of the circuit. Our result can be extended to consider adversaries who both tamper and leak. More specifically, we show a general compiler that converts any circuit into one that is resilient against both tampering (in the model described above) and leakage. The leakage model that we consider is OCL ("only computation leaks") leakage, as defined by Micali and Reyzin [49].

Unfortunately, being resilient to leakage (in addition to tampering) comes at a price: First, the resulting circuit no longer tolerates a constant-fraction of tampering, but rather security is ensured as long as $1/\operatorname{poly}(k)$-fraction of the wires (or memory gates) can be tampered with. In addition, the result is no longer information theoretical, and relies on cryptographic assumptions. Specifically it relies on the existence of a non-committing encryption scheme [11] (which can be instantiated from hardness of factoring Blum integers, CDH and LWE [11, 15, 12]), and on the existence of a fully homomorphic encryption scheme (which can be instantiated from LWE [32, 9, 8]). Due to lack of space, we defer our result in the leakage setting to the full version.

### 1.3 Related Work

The problem of constructing error resilient circuits dates back to the work of Von Neumann from 1956 [56]. Von Neumann studied a model of *random* errors, where each gate has an (arbitrary) error independently with small fixed probability, and his goal was to obtain *correctness* (as opposed to privacy). There have been numerous follow up papers to this seminal work, including [16, 53, 52, 27, 23, 36, 28, 22], who considered the same noise model, ultimately showing that any circuit of size $\sigma$ can be encoded into a circuit of size $O(\sigma \log \sigma)$ that tolerates a fixed constant noise rate, and that any such encoding must have size $\Omega(\sigma \log \sigma)$.

There has been little work on constructing circuits resilient to *adversarial* faults, while guaranteeing correctness. The main works in this arena are those of Kalai *et al.* [41], Kleitnam *et al.* [44], and Gál and Szegedy [29]. The works of [44] and [41] consider a different model where the only type of faults allowed are short-circuiting gates. [29] consider a model that allows arbitrary faults on gates, and show how to construct tamper-resilient circuits for symmetric Boolean functions. We note that [29] allow a constant fraction $\delta$ of adversarial faults *per level* of the circuit. Moreover, if there are less than $1/\delta$ gates on some level, they allow no tampering at all on that level. [29] also give a more general construction for any efficiently computable function which relies on PCP's. However, in order for their construction to work, they require an entire PCP proof $\pi$ of correctness of the output to be precomputed and handed along with the input to the tamper-resilient circuit. Thus, they assume that the input to the circuit is already encoded via an encoding which depends on the *output* value of that very circuit. We also use the PCP methodology in our result, but do not require any precomputations or that the input be encoded in some special format.

Recently, the problem of physical attacks has come to the forefront in the cryptography community. From the viewpoint of cryptography, the main focus is no longer to ensure correctness, but to ensure *privacy*. Namely, we would like to protect the honest party's secret information from being compromised through the physical attacks

of an adversary. There has been much work on protecting circuits against leakage attacks [38, 49, 20, 51, 19, 26, 39, 34]. However, there has not been much previous work on constructing circuits resilient to tampering attacks. In this arena, there have been two categories of works. The works of [31, 21, 13, 48, 40] allow the adversary to only tamper with and/or leak on the *memory* of the circuit in between runs of the circuit, but do not allow the adversary to tamper with the circuit itself. Due to lack of space we do not elaborate on these related works in this extended abstract, and refer the reader to the full version for details. We note that this model of allowing tampering only with memory is very similar to the problem of "related key attacks" (see [3, 2] and references therein). In contrast, in our work, as well as in the works of [37, 25], the focus is on constructing circuits resilient to tampering with both the memory as well as the wires of the circuit.

Faust *et al.* [25] consider a model that is reminiscent to the model of [37] and to the model we consider here. They consider adversarial faults where the adversary may actually tamper with all wires of the circuit but each tampering attack fails independently with some probability $\delta$. As in our case, they also allow the adversary to learn a logarithmic number of bits of information on the secret key. However, their result requires the use of small tamper-proof hardware components.

### 1.4 Our Techniques

Our conceptual approach is similar to [37]: The tamper-resilient circuit has an "error-detection" mechanism, and if an error is detected then the circuit self-destructs (i.e. the memory is zeroed out). However, our techniques for achieving this are very different.

Recall that our goal is to guarantee security against an adversary who may tamper with a *constant fraction* of the wires and memory gates. We begin with the simple observation that circuits in $NC^0$ are inherently tamper-resilient. In particular, since each output bit of an $NC^0$ circuit is computed by a constant-size circuit (say of size $\tilde{\sigma}$), if we allow $\alpha \cdot 1/\tilde{\sigma}$-fraction of tampering for some constant $\alpha < 1$, then we ensure that at least $(1 - \alpha)$-fraction of the output bits are exactly correct.

In order to construct a good error-detection mechanism for our tamper-resilient circuit, we need to construct a mechanism that detects errors and is itself resilient to a constant-fraction of tampering. More specifically, we would like to construct a "robust" verifier in $NC^0$ for the computation of the original circuit, such that if an error is detected in the computation, then most of the circuit's output are set to 0. Fortunately, the PCPP (PCP of Proximity) Theorem [4] provides us with exactly such a verifier. Informally, in a PCPP for an NP language $\mathcal{L}$, the verifier is given oracle access to a PCP $\pi$ and is also given oracle access to an instance $z$. The verifier tosses logarithmically many random coins and it queries a *constant* number of bits of $\pi$ and $z$. It accepts if $z \in \mathcal{L}$ and if the PCP $\pi$ was generated honestly, and it rejects (with probability $1/2$) if $z$ is far from being in the language (even if $\pi$ was generated maliciously). We refer the reader to Section 2 for details.

Note that for any fixed setting of the random coins, the corresponding verifier is of constant size since it has only a constant number of input bits. Thus, by constructing a separate verifier corresponding to each possible outcome of the random coins and outputting the output bits of all verifiers, we obtain a single (larger) verifier in $NC^0$ with the property that if $z$ is far from $\mathcal{L}$ then at least a $1/2$-fraction of the verifier's

outputs are 0. Note that the soundness property only holds when $z$ is "far" from every string in $\mathcal{L}$. Thus, our transformed circuit has the following basic paradigm: We encode the secret state $s$ and the input $x$ using an error-correcting code to obtain $S = \mathsf{ECC}(s)$ and $X = \mathsf{ECC}(x)$. We compute $b = C_s(x)$, and we compute a PCPP proof $\pi$ that $(b, S \circ X) \in \mathcal{L}$ where $\mathcal{L} = \{(b, S \circ X) \mid \exists s, x : S = \mathsf{ECC}(s), X = \mathsf{ECC}(x), b = C_s(x)\}$. Then we verify the proof and self-destruct (i.e. erase all memory) if any of the output bits of the verifier are 0. Additional work is required to go from tolerating a constant-fraction of tampering in the error-detection stage to tolerating a constant-fraction of tampering overall. We elaborate on these in Section 4.2.

We mention that the resulting tamper-resilient circuit has one gate with large fan-out. Additional ideas are needed in order to remove the need of such a large fan-out gate. Due to lack of space we elaborate on these in the full version.

We note that the techniques mentioned so far are used to get resilience only against tampering attacks (and not leakage attacks). In order to get security against both (continual) leakage and tampering, we rely on techniques in the leakage regime, and in particular rely on recent results in the OCL model [49, 39, 34, 33].[2] Due to lack of space we defer the details to the full version.

## 2 PCP of Proximity

In this section, we present necessary preliminaries on PCP of proximities (denoted by PCPP), and state the efficient PCPP theorem of [4]. A PCP of proximity is a relaxation of a standard PCP, that only verifies that the input is *close* to an element of the language. The advantage of this relaxation is that it allows the possibility that the verifier may read only a small number of bits from the input. For greater generality, the input is divided into two parts $(a, z)$, where $a$ is given explicitly to the verifier, and $z$ is given only as an oracle. Thus PCPs of proximity consider languages which consist of pairs of strings, and therefore are referred to as *pair languages*.

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

**Definition 2 (Relative Hamming distance).** *Let* $z, z' \in \{0,1\}^K$ *be two strings. The* relative Hamming distance *between* $z$ *and* $z'$ *is defined as*

$$\Delta(z, z') \triangleq |\{i \in [K] : z_i \neq z'_i\}|/K.$$

*We say that* $z$ *is* $\delta$-far *from* $z'$ *if* $\Delta(z, z') \geq \delta$. *More generally, let* $S \subseteq \{0,1\}^K$; *we say* $z$ *is* $\delta$-far *from* $S$ *if* $\Delta(z, z') \geq \delta$ *for every* $z' \in S$.

**Definition 3 (PCP Verifiers).** *A* verifier *is a probabilistic polynomial time algorithm* $V$ *that, on an input* $x$, *tosses* $r = r(|x|)$ *random coins and generates a sequence of* $q =$

[2] We actually rely on the recently constructed LDS compiler [6], which in turn is based on OCL compilers.

$q(|x|)$ *queries* $I = (i_1, \ldots, i_q)$ *and a circuit* $D : \{0,1\}^q \to \{0,1\}$ *of size at most* $d(|x|)$.[3]

*We think of* $V$ *as representing a probabilistic oracle machine that queries its oracle* $\pi$ *at positions* $I$*, receives the* $q$ *answer bits* $\pi|_I \triangleq (\pi_{i_1}, \ldots, \pi_{i_q})$*, and accepts if and only if* $D(\pi_I) = 1$*. We write* $(I, D) \leftarrow V(x)$ *to denote the queries and circuit generated by* $V$ *on input* $x$ *(and random coin tosses). We call* $r$ *the* randomness complexity*,* $q$ *the* query complexity*, and* $d$ *the* decision complexity *of* $V$.

**Definition 4** (PCPP **verifier for a pair language [4]).** *For functions* $s, \delta : \mathbb{N} \to [0,1]$*, a verifier* $V$ *is a* probabilistically checkable proof of proximity *(*PCPP*) system for a pair language* $L$ *with* proximity parameter $\delta$ *and* soundness error $s$*, if the following two conditions hold for every pair of strings* $(a, z)$*:*

*Completeness: If* $(a, z) \in L$ *then there exists* $\pi$ *such that* $V(a)$ *accepts oracle* $z \circ \pi$ *with probability* 1*. Formally*

$$\exists \pi \Pr_{(I,D) \leftarrow V(a)} [D((z,\pi)|_I) = 1] = 1.$$

*Soundness: If* $z$ *is* $\delta(|a|)$*-far from* $L(a)$*, then for every* $\pi$*, the verifier* $V(a)$ *accepts oracle* $z \circ \pi$ *with probability strictly less than* $s(|a|)$*. Formally,*

$$\forall \pi \Pr_{(I,D) \leftarrow V(a)} [D((z \circ \pi)|_I) = 1] < s(|a|).$$

**Theorem 2 (Efficient** PCPP **for Pair-language (Theorem 3.3 of [4])).** *Let* $T : \mathcal{N} \to \mathcal{N}$ *be a non-decreasing function, and let* $L$ *be a pair language. If* $L$ *can be decided in time* $T$*,*[4] *then for every constant* $\rho \in (0,1)$ *there exists a PCP of proximity for* $L$ *with randomness complexity* $O(\log T)$*, query complexity* $q = O(1/\rho)$*, perfect completeness, soundness error* $1/2$*, and proximity parameter* $\rho$*.*

We mention that [4] does not discuss the complexity of constructing the PCPP proof, but the efficiency follows by a close inspection of their construction.

# 3 The Tampering Model

## 3.1 Circuits with Memory Gates

Similarly to [37], we consider a circuit model that includes memory gates. Namely, a circuit consists of (the usual) AND, OR, and NOT gates, connected to each other via wires, as well as input wires and output wires. In addition, a circuit may have memory gates. Each memory gate has one (or more) input wires and one (or more) output wires. Each memory gate is initialized with a bit value 0 or 1. This value can be updated during each run of the circuit.

[3] For the sake of simplicity we consider only bit queries.

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

Each time the circuit is run with some input $x$, all the wires obtain a $0/1$ value. The values of the input wires to the memory gates define the way the memory is updated. We allow only two types of updates: delete or unchange. Specifically, if an input wire to a memory gate has the value $0$, then the memory gate is overwritten with the value $0$. If an input wire to a memory gate has the value $1$, then the value of the memory gate remains unchanged. We denote a circuit $C$ initialized with memory $s$ by $C_s$.

### 3.2 Tampering Attacks

We consider adversaries, that can carry out the following attack: The adversary has black-box access to the circuit, and thus can repeatedly run the circuit on inputs of his choice. Each time the adversary runs the circuit with some input $x$, he can tamper with the wires and the memory gates. We consider the following type of faults: Setting a wire (or a memory gate) to $0$ or $1$, or toggling with the value on a wire (or a memory gate).

More specifically, the adversary can adaptively choose an input $x_i$ and a set of tampering instructions (as above), and he receives the output of the tampered circuit on input $x_i$. He can do this adaptively as many times as he wishes. We emphasize that once the memory has been updated, say from $s$ to $s'$, the adversary no longer has access to the original circuit $C_s$, and now only has access to $C_{s'}$. Namely, the memory errors are persistent, while the wire errors are not persistent.

We denote by $\mathsf{TAMP}_{\mathcal{A}}(C_s)$ the output of an adversary $\mathcal{A}$ that carries out the above tampering attack on a circuit $C_s$. We say that an adversary $\mathcal{A}$ is a $\lambda$-tampering adversary if during each run of the circuit he tampers with at most a $\lambda$-fraction of the circuit. Namely, $\mathcal{A}$ can make at most $\lambda \cdot |C_s|$ tampering instructions for each run, where each instruction corresponds either to a wire tampering or to a memory gate tampering.

**Remark.** In this work, we define the size of a circuit $C$, denoted by $|C|$, as the number of wires in $C$ plus the number of memory gates in $C$. Note that this is not the common definition (where usually the size includes also the gates); however, it is equivalent to the common definition up to constant factors.

To define security of a circuit against tampering attacks we use a simulation-based definition, where we compare the real world, where an adversary $\mathcal{A}$ (repeatedly) tampers with a circuit $C_s$ as above, to a simulated world, where a simulator $\mathsf{Sim}$ tries to simulate the output of $\mathcal{A}$, while given only black-box access to the circuit $\mathsf{C}_s$, and without tampering with the circuit at all.

In this work, we give the simulator $\mathsf{Sim}$, in addition to black box access to the circuit $C_s$, the privilege to request $\log T$ bits of information about $s$, where $T$ is the number of times $\mathcal{A}$ runs the tampered circuit. More specifically, the simulator, in addition to black-box access to the circuit $C_s$, is also given oracle access to a leakage oracle that takes as input any leakage request, represented as a boolean circuit $L : \{0,1\}^k \to \{0,1\}$ and returns $L(s)$. The simulator $\mathsf{Sim}$ may query the leakage oracle at most $\log T$ times. We denote the output of $\mathsf{Sim}$ by $\mathsf{LeakBB}_{\mathsf{Sim},\log T}(C_s)$. As noted in the Introduction, this leakage is necessary if we restrict ourselves to deterministic constructions.

**Definition 5.** *We say that a circuit $C_s$ is secure against $\lambda$-tampering adversaries, if for every $\lambda$-tampering adversary $\mathcal{A}$ there exists a simulator $\mathsf{Sim}$, that runs in time*

$\mathrm{poly}(|\mathcal{A}|, |C_s|)$*, such that*

$$\{\mathsf{TAMP}_{\mathcal{A}}(C_s)\}_{k\in\mathbb{N}} \equiv \{\mathsf{LeakBB}_{\mathsf{Sim},\log T}(C_s)\}_{k\in\mathbb{N}},$$

*where $T$ is an upper bound on the number of times that $\mathcal{A}$ runs the (tampered) circuit.*

We give an efficient method for constructing circuits that remain secure against adversaries that tamper with a constant fraction of the wires in the circuit. Namely, we prove that there exists a constant $\lambda > 0$ such that the resulting circuit is secure against $\lambda$-tampering adversaries.

# 4 The Compiler

In this section, we present our efficient compiler $\mathcal{T}$, which takes a circuit $C_s$, with memory containing a secret $s \in \{0,1\}^k$, and transforms it into a tamper-resilient circuit.

We describe our compiler in stages:

- First, we present a compiler that takes as input a circuit $C_s$ and outputs a circuit $C_S^{(1)}$, which we partition into four segments. We prove that $C_S^{(1)}$ is secure against adversaries that tamper with at most a constant fraction of the memory gates, and tamper with at most a constant fraction of the wires in Segment 2, Segment 3, and Segment 4 of the circuit (and may tamper arbitrarily with Segment 1 of the circuit). We refer to such security as security against *local tampering*.
- Then, we show how to make the size of the memory, and the size of Segments 2-4, large enough so that the resulting circuit, denoted by $C_{\tilde{S}}^{(2)}$, is secure against adversaries who tamper with at most a constant fraction of the circuit *overall*. Namely, we prove that there is a constant $\lambda > 0$ such that the resulting circuit, $C_{\tilde{S}}^{(2)}$, is secure against any adversary that for each run of the circuit sends at most $\lambda \cdot |C_{\tilde{S}}^{(2)}|$ tampering instructions, where each tampering instruction is either a wire tampering or a memory tampering.

We note that both constructions have an AND gate (denoted by $G_{\mathsf{cas}}$) which has a large fan-out. In the full version we show how using some additional ideas, one can bypass the need for such a gate.

## 4.1 The Compiler: First Construction

Let $\mathsf{ECC}(\cdot)$ be a binary error correcting code which can efficiently correct a constant fraction of errors $\delta > 0$. We denote the (efficient) decoding procedure by $\mathsf{Dec}$.

In what follows we present a compiler that takes as input a circuit $C_s$ and outputs a circuit $C_S^{(1)}$. The circuit $C_S^{(1)}$ takes as input a string $x \in \{0,1\}^n$ and outputs a bit $b$. In the case of no tampering, we show the correctness property: $C_S^{(1)}(x) = C_s(x)$. Moreover, we prove that the circuit $C_S^{(1)}$ is resilient to *local tampering*.

**Remark.** For simplicity we assume that $s = 0^k$ is "illegal." We note that this assumption is not necessary, and is made to simplify the construction and the analysis.

We next describe the circuit $C_S^{(1)}$, which we partition into four segments:

**Memory: Encoding Secret $s$.** The encoding $S = \mathsf{ECC}(s)$ is placed in the memory of $C^{(1)}$.

**Segment 1:** This segment consists of three sub-segments.

1. **Encoding Input $x$.** The first sub-computation encodes the public input $x$ using $\mathsf{ECC}$. Namely, it takes as input $x$ and outputs $\mathsf{ECC}(x)$.
   We assume without loss of generality that $|\mathsf{ECC}(x)| = |\mathsf{ECC}(s)|$. This is without loss of generality since if, for example, $|x| < |s|$ then the encoding procedure will first artificially increase $x$ by appending zeros to it, and then encode; and similarly if $|s| < |x|$.
2. **Circuit Computation.** The second sub-computation computes the output bit $b = C_s(x)$. Namely, it takes as input $x$ and $S$ and outputs $b = C_s(x)$, where $s = \mathsf{Dec}(S)$.
3. **PCPP Computation.** The third sub-computation computes a PCPP for the following pair language:

   $$\mathcal{L} = \{(b, X \circ S) \mid \exists x \in \{0,1\}^n, s \in \{0,1\}^k \backslash \{0^k\} : X = ECC(x), S = ECC(s), b = C_s(x)\}$$

   Namely, it takes as input the pair $(b, X \circ S)$ and outputs $\pi$, which is a PCP of proximity for the statement $(b, X \circ S) \in \mathcal{L}$.
   The PCPP we use is one with randomness complexity $r = O(\log |C_s|)$, query complexity $q = O(1/\rho)$ for $\rho = \frac{\delta}{6}$,[5] perfect completeness, and soundness $1/2$, where the verifier rejects with probability at least $1/2$ statements $(b, X \circ S)$, for which $X \circ S$ is $\rho$-far from the language $L_b$. The existence of such a PCPP follows from Theorem 2 (see Section 2).

We note that the outputs of each of the above sub-computations are used by many other subcomputations. Thus, each output wire of each of the above sub-computations is split into many output wires. These output wires all belong to Segment 2, except one output wire (which computes the bit $b = C_s(x)$, and belongs to Segment 4).

**Segment 2: PCPP Verification.** This segment consists of $\tau \triangleq 2^r = \mathrm{poly}(|C_s|)$ circuits, $C_{\mathsf{v}_i}$, one for every possible PCPP verifier. Note that each verifier circuit has constant size, which we denote by $\tilde{\sigma}_{\mathsf{v}}$.[6] Each verifier circuit $C_{\mathsf{v}_i}$ takes as input $q$ bits from $(X \circ S, b, \pi)$, and outputs a single bit $\beta_i$. All the output wires of the circuits $C_{\mathsf{v}_i}$ are fed as input to a single AND gate $G_{\mathsf{cas}}$ with fan-in $\tau$. Thus, the output of $G_{\mathsf{cas}}$ is $\bigwedge_{1 \le i \le \tau} \beta_i$.
Let $K$ be the number of bits in $S$, the encoding of the secret input $s$. The AND gate $G_{\mathsf{cas}}$ (from Segment 2) has fan-out $2K$, where the first $K$ of the output wires belong to Segment 3 and the other $K$ output wires belong to Segment 4. We denote the values of the output wires of $G_{\mathsf{cas}}$ by $\{\gamma_j\}_{j=1}^{2K}$.

**Segment 3: Error Cascade.** This segment contains only the first $K$ output wires of $G_{\mathsf{cas}}$, which have values $\{\gamma_j\}_{j=1}^{K}$. For every $j \in [K]$, the $j$'th output wire of $G_{\mathsf{cas}}$ is fed as input to memory gate $j$. Thus, if $\gamma_j = 0$, memory position $j$ is overwritten with a 0. If $\gamma_j = 1$, memory position $j$ is unchanged.

**Segment 4: The Output.** This segment consists of the rest of the circuit: The other $K$ output wires of $G_{\mathsf{cas}}$, which have values $\{\gamma_j\}_{j=K+1}^{2K}$, and a single AND gate $G_{\mathsf{out}}$

[5] Recall that $\delta$ is the fraction of errors that the error-correcting code $\mathsf{ECC}$ can correct.

[6] We note that the amount of tampering we ultimately tolerate depends on $\tilde{\sigma}_{\mathsf{v}}$.

which has fan-in $K+1$ and outputs a single bit. $G_{\mathsf{out}}$ takes as input $\left(b, \{\gamma_j\}_{j=K+1}^{2K}\right)$ (all these input wires are part of this segment), and outputs $\tilde{b} = b \wedge \left(\bigwedge_{K+1 \le j \le 2K} \gamma_j\right)$. We note that Segment 4 also includes the output wire of $G_{\mathsf{out}}$, which is the final output of the circuit $C_S^{(1)}(x)$.

The compiled circuit $C_S^{(1)}$ is depicted in Figure 1.

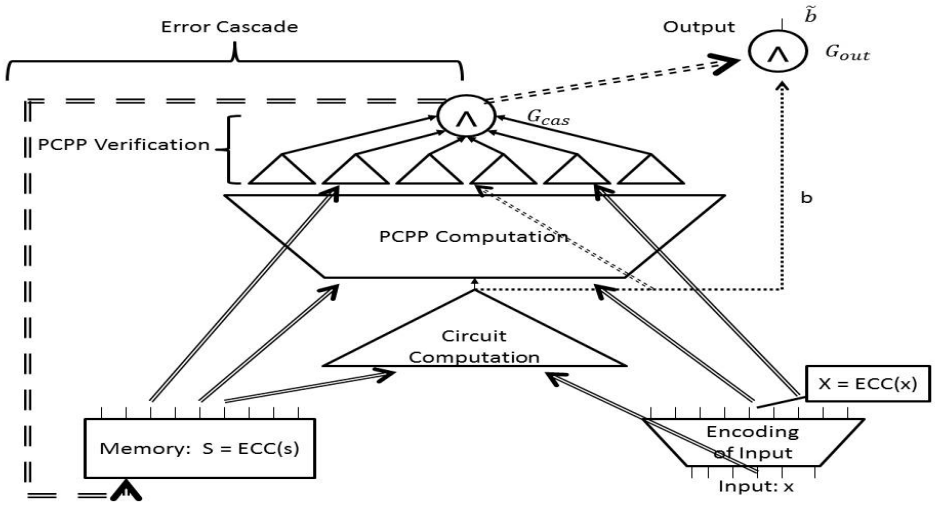


**Fig. 1. The compiled circuit $C_S^{(1)}$.** We show here the Memory and the 4 segments of the compiled circuit. Note that the encoding $S$ and the input $x$ are fed into the Circuit Computation Stage of Segment 1. Additionally, the encodings $S$ and $X$ are fed into the PCPP Computation Stage of Segment 1, and are fed to Segment 2 (the PCPP Verification Stage) along with the bit $b$, the output of the Circuit Computation Stage. The outputs of $G_{\mathsf{cas}}$ feed back into memory (this is Segment 3– Error Cascade) and to the final output gate $G_{\mathsf{out}}$ (in Segment 4) along with the output bit $b$ of the Circuit Computation Stage. The final output of the circuit is $\tilde{b}$.

The circuit $C_S^{(1)}$ is secure against an adversary $\mathcal{A}$ who may tamper as follows:

- $\mathcal{A}$ can tamper with any number of wires in Segment 1 of the circuit (which consists of the encoding computation of the input $x$, the circuit computation $C_s(x)$, and the PCPP computation).
- $\mathcal{A}$ can tamper with at most $\lambda$-fraction of the wires in Segment 2 (PCPP verification), $\lambda$-fraction of the wires in Segment 3 (error cascade), $\lambda$-fraction of the wires in Segment 4 (output segment) and with at most $\lambda$-fraction of the memory gates, where $\lambda = \min\{\frac{1}{3\bar{\sigma_V}}, \frac{\delta}{6}\}$.

We call such an adversary $\lambda$-*locally tampering*, and we denote the output of any such adversary by $\mathsf{LocalTAMP}_{\mathcal{A}}(C_S^{(1)})$.

**Lemma 1.** *Let* $\lambda = \min\{\frac{1}{3\tilde{\sigma_V}}, \frac{\delta}{6}\}$. *Then for any* $\lambda$-locally tampering *adversary* $\mathcal{A}$ *there exists a simulator* $\mathsf{Sim}$*, that runs in time* $\mathrm{poly}(|\mathcal{A}|, |C_s|)$*, such that*

$$\{\mathsf{LocalTAMP}_{\mathcal{A}}(C_S^{(1)})\}_{k\in\mathbb{N}} \equiv \{\mathsf{LeakBB}_{\mathsf{Sim},\log T}(C_s)\}_{k\in\mathbb{N}},$$

*where* $T$ *is an upper bound on the number of times that* $\mathcal{A}$ *runs the (tampered) circuit.*

We give the proof idea of Lemma 1, and defer the formal proof to the full version.

*Proof idea.* Fix any $\lambda$-*locally tampering* adversary $\mathcal{A}$. Let $T$ be an upper bound on the number of times the adversary $\mathcal{A}$ runs a possibly tampered version of the circuit $C_s^{(1)}$. We construct a simulator $\mathsf{Sim}$ with black-box access to $C_s$, which is allowed to request $\log T$ bits of leakage on the secret $s$, and simulates the view of $\mathcal{A}$.

$\mathsf{Sim}$ chooses the leakage function to be the function $L : \{0,1\}^k \to \{0,1\}^{\log T}$, which has the adversary $\mathcal{A}$ hard-wired into it, and on input a secret $s$, outputs the largest index $i^*$, such that in the first $i^* - 1$ runs of the (possibly tampered) circuit by $\mathcal{A}$, all the (possibly tampered) input wires to the gate $G_{\mathsf{cas}}$ have the value 1.

After querying this leakage function and receiving an output $i^*$, $\mathsf{Sim}$ internally emulates $\mathcal{A}$, by emulating the output of each run of the circuit. Each time $\mathcal{A}$ calls the circuit with input $x_i$ and a set of tampering instructions, the simulator $\mathsf{Sim}$ simulates the run of the (possibly tampered) circuit as follows:

- Simulate the output $X_i'$ of the (possibly tampered) Encoding Input Segment. Efficiently decode $X_i'$ to obtain $x_i' = \mathsf{Dec}(X_i')$.
  Note that $X_i'$ may be far from a codeword, in which case $x_i'$ may be $\perp$. Also, note that the simulator $\mathsf{Sim}$ is indeed able to carry out the above computation exactly as in the real computation, since this part of the computation does not use the secret $s$, and since $\mathsf{Sim}$ knows $\mathcal{A}$'s tampering instructions.
- If $i < i^*$, set all the input wires of $G_{\mathsf{cas}}$ to be 1, and set the output of the Circuit Computation Segment to be $b_i' = C_s(x_i')$. As we prove in the full version, the fact that $i < i^*$ implies that it must be the case that $x_i' \neq \perp$, and that the output of the Circuit Computation Segment is indeed $C_s(x_i')$.
  Simulate the output $b_{\mathsf{out}}$ of $G_{\mathsf{out}}$, using the tampering instructions of $\mathcal{A}$, assuming that the values of the $K + 1$ incoming wires (before applying the tampering instructions to these wires) are $(1^K, b_i')$. Return $b_{\mathsf{out}}$.
  Note that it is not necessarily the case that $b_{\mathsf{out}} = b_i'$ since the adversary $\mathcal{A}$ may tamper with some of the incoming wires of $G_{\mathsf{out}}$ and may tamper with the output wire.
- If $i \geq i^*$, return $b_{\mathsf{out}} = 0$, unless $\mathcal{A}$ tampers with the output wire, in which case $\mathsf{Sim}$ returns $b$ if the tamper is "set to $b$", and returns 1 if the tamper is "toggle".
  We will argue that in this case, even if $\mathcal{A}$ tampers with the input wires to $G_{\mathsf{out}}$, the output of $G_{\mathsf{out}}$ is always going to be 0, since he cannot tamper with all the input wires (recall that he can tamper with at most a $\lambda$-fraction of the wires). Thus, the output wire is always 0, unless the output wire itself is tampered with.

We defer to the full version the proof that this simulator indeed simulates the view of $\mathcal{A}$ correctly.

### 4.2 The Compiler: Final Construction

Note that $C_S^{(1)}$ is secure against any adversary that tampers with only a constant fraction of the memory, and a constant fraction of the wires in Segments 2, 3 and 4. We would like to construct a circuit that is secure against adversaries that tamper with a constant fraction of the circuit overall. Namely, we would like our circuit, which will be denoted by $C_{\tilde{S}}^{(2)}$, to have the property that there is a constant $\lambda > 0$ such that $C_{\tilde{S}}^{(2)}$ is secure against any adversary that for each run of the circuit sends at most $\lambda \cdot |C_{\tilde{S}}^{(2)}|$ tampering instructions, where each tampering instruction is either a wire tampering or a memory tampering.

We do this by adding enough memory gates, and enough wires to the critical segments: the PCPP Verification Segment, the Error Cascade Segment and the Output Segment, so that the following two conditions hold.

1. Each of these segments, as well as the memory, remains resilient to a constant fraction of tampering.
2. The total number of wires in each of these segments, and the size of the memory, is a constant fraction of the total number of wires in the entire circuit.

More specifically, let $\sigma_{\mathsf{comp}} = \sigma_{\mathsf{comp}}(k)$ be the size of Segment 1 of the circuit $C_S^{(1)}$. We transform the circuit $C_S^{(1)}$, to ensure that the size $K = K(k)$ of the encoding of $s$ in memory, the size $\sigma_{\mathsf{v}} = \sigma_{\mathsf{v}}(k)$ of Segment 2 (the PCPP Verification Segment), the size $\sigma_{\mathsf{cas}} = \sigma_{\mathsf{cas}}(k)$ of Segment 3 (the Error Cascade Segment), and the size $\sigma_{\mathsf{out}} = \sigma_{\mathsf{out}}(k)$ of Segment 4 (the Output Segment), satisfy

$$\Theta(K(k)) = \Theta(\sigma_{\mathsf{v}}(k)) = \Theta(\sigma_{\mathsf{cas}}(k)) = \Theta(\sigma_{\mathsf{out}}(k)) = \Omega(\sigma_{\mathsf{comp}}(k)),$$

without changing the size $\sigma_{\mathsf{comp}}$ of Segment 1, and while keeping each of the above segments (Segments 2, 3, and 4), and the memory gates, resilient to constant fraction of tampering.

This transformation will allow us to prove security against adversaries who tamper with a constant fraction of the circuit overall. We proceed with the formal construction.

Let

$$M = M(k) = \max\{\sigma_{\mathsf{v}}(k), K(k), \sigma_{\mathsf{comp}}(k)\}.$$

Let $p_1(k) \triangleq M/\sigma_{\mathsf{v}}(k)$, and let $\sigma'_{\mathsf{v}}(k) \triangleq p_1(k) \cdot \sigma_{\mathsf{v}}(k)$. Similarly, let $p_2(k) \triangleq M(k)/K(k)$, and let $K'(k) \triangleq p_2(k) \cdot K(k)$. For the sake of simplicity we assume that $p_1$ and $p_2$ are integers.[7] Note that under this simplifying assumption $K' = \sigma'_{\mathsf{v}} = M$.

**Memory: Encoding Secret $s$.** Let $\widetilde{\mathsf{ECC}}(s) = \mathsf{ECC}(s)^{p_2(k)}$, where by $\mathsf{ECC}(s)^{p_2(k)}$ we denote $p_2(k)$ concatenated copies of $\mathsf{ECC}(s)$. Note that $\widetilde{\mathsf{ECC}}$ is an error-correcting code which has the same distance as $\mathsf{ECC}$, which is at least $2\delta$. (The reason the distant of $\mathsf{ECC}$ is at least $2\delta$ follows from the fact that it can correct up to $\delta$-fraction of errors.) Let $\tilde{S} = \widetilde{\mathsf{ECC}}(s)$. We denote by $\tilde{S}(i,j)$ the $j$-th bit of the $i$-th copy of $S$.

[7] This simplifying assumption makes the analysis somewhat cleaner. Without this assumption, we would need to define $p_1(k) \triangleq \lceil M/\sigma_{\mathsf{v}}(k) \rceil$ and $p_2(k) \triangleq \lceil M(k)/K(k) \rceil$.

We denote by $S$ the first row of the encoding $\tilde{S}$: $(\tilde{S}(1,j))_{j\in[K]}$. The encoding $\tilde{S}$ is placed in the memory of $C^{(2)}$.

**Segment** 1**.** This segment, which includes the encoding of the input $x$, the circuit computation, and the PCPP computation, remains exactly the same as Segment 1 of $C_S^{(1)}$. In order to keep this segment the same, instead of using the entire new memory $\widetilde{\mathsf{ECC}(S)}$, this segment only uses the first row of $\widetilde{\mathsf{ECC}(s)}$, which is exactly $S = \mathsf{ECC}(s)$, the memory content of $C^{(1)}$.

**Segment** 2**: PCPP Verification.** This stage consists of $\tau' \triangleq p_1(k) \cdot \tau$ number of circuits $\{C_{\mathsf{v}_{i,j}}\}_{i\in[\tau],j\in[p_1(k)]}$, where each PCPP verifier from $C_S^{(1)}$ is simply copied $p_1(k)$ times. Namely, for each $i \in [\tau]$ and each $j \in [p_1(k)]$, the circuit $C_{\mathsf{v}_{i,j}}$ is simply a copy of $C_{\mathsf{v}_i}$. We note that each $C_{\mathsf{v}_{i,j}}$ only accesses the first row of memory $S = (\tilde{S}_{1,j})_{j\in[k]}$, and as before, each verifier circuit $C_{\mathsf{v}_{i,j}}$ takes as input a constant number of bits from $(X \circ S, b, \pi)$ and outputs a single bit $\beta_{i,j}$. Note that each verifier circuit still has constant size $\tilde{\sigma}_{\mathsf{v}}$.

All these $\tau'$ output wires of the circuits $C_{\mathsf{v}_{i,j}}$ are inputs to the AND gate $G_{\mathsf{cas}}$. This gate has $K'$ additional input wires that belong to Segment 3 below (where $K'$ is the number of bits in $\tilde{S}$). The gate $G_{\mathsf{cas}}$ has $2K'$ output wires, and we denote the values on these wires by $\{\gamma_{\ell,m}\}_{i\in[p_2(k)],j\in[2K]}$. The first $K'$ output wires (with values $\{\gamma_{\ell,m}\}_{i\in[p_2(k)],j\in[K]}$) belong to Segment 3, and the other $K'$ output wires belong to Segment 4.

**Segment** 3**: Error Cascade.** This segment has two parts:

- A circuit $C_{\mathsf{code}_{i,j}}$ of constant size $\sigma_{\mathsf{code}}$ for each bit $\tilde{S}_{i,j}$ of $\tilde{S}$: $1 \le i \le p_2(k)$, $1 \le j \le K$:
  Input: $\tilde{S}(1,j)$, $\tilde{S}(i,j)$.
  Output: $\psi_{i,j} = \neg(\tilde{S}(1,j) \oplus \tilde{S}(i,j))$.
  All these output wires with values $\psi_{i,j}$ are inputs to $G_{\mathsf{cas}}$. Thus, in total, $G_{\mathsf{cas}}$ has $K' + \tau'$ input wires ($\tau'$ from Segment 2 and $K'$ from Segment 3), and it outputs
  $$\left(\bigwedge_{i\in[\tau],j\in[p_1(k)]} \beta_{i,j}\right) \wedge \left(\bigwedge_{i\in[p_s(k)],j\in[K]} \psi_{i,j}\right)$$
- The first $K'$ output wires of $G_{\mathsf{cas}}$, denoted by $(\gamma_{\ell,m})_{\ell\in[p_2(k)],m\in[K]}$, are fed to the memory gates. More specifically, for every $\ell \in [p_2(k)]$ and $m \in [K]$, the $(\ell,m)$-th output $\gamma_{\ell,m}$ of $G_{\mathsf{cas}}$ is the input to memory gate $(\ell,m)$. Thus, if $\gamma_{\ell,m} = 0$, memory position $(\ell,m)$ is overwritten with a 0. If $\gamma_{\ell,m} = 1$, memory position $(\ell,m)$ is unchanged.

**Segment** 4**: The Output.** This segment of the circuit consists of $K'$ output wires of $G_{\mathsf{cas}}$, which have the values $\{\gamma_{\ell,m}\}_{\ell\in[p_2(k)],m\in[K+1,2K]}$. This segment has a single AND gate $G_{\mathsf{out}}$ which has fan-in $K'+1$. This segment contains all the $K'+1$ input wires to $G_{\mathsf{out}}$: The first $K'$ input wires are exactly the $K'$ output wires of $G_{\mathsf{cas}}$ (with values $\{\gamma_{\ell,m}\}_{\ell\in[p_2(k)],m\in[K+1,2K]}$), and the other input wire of $G_{\mathsf{out}}$ is an output wire of the Circuit Computation in Segment 1, which computes the bit $b = C_s(x)$.

The AND gate $G_{\mathsf{out}}$ has a single output wire which is

$$\tilde{b} = b \wedge \left( \bigwedge_{\ell \in [p_2(k)], m \in [K'+1, 2K']} \gamma_{\ell,m} \right).$$

The final output of the circuit $C^{(2)}_{\tilde{S}}(x)$ is $\tilde{b}$.

Recall that we denote by $\sigma'_{\mathsf{v}} = \sigma'_{\mathsf{v}}(k)$ the size of the PCPP Verification Segment in $C^{(2)}_{\tilde{S}}$, and we denote by $K'$ the number of memory gates in $C^{(2)}_{\tilde{S}}$. We also denote by $\sigma'_{\mathsf{cas}} = \sigma'_{\mathsf{cas}}(k)$ the size of the Error Cascade Segment in $C^{(2)}_{\tilde{S}}$ and denote by $\sigma'_{\mathsf{out}} = \sigma'_{\mathsf{out}}(k)$ be the size of the Output Segment in $C^{(2)}_{\tilde{S}}$. It follows by construction that

$$\Theta(\sigma'_{\mathsf{v}}(k)) = \Theta(\sigma'_{\mathsf{cas}}(k)) = \Theta(\sigma'_{\mathsf{out}}(k)) = \Theta(K'(k)) = \Omega(\sigma_{\mathsf{comp}}(k)).$$

In what follows, we denote by $\sigma'$ the total size of the circuit $C^{(2)}_{\tilde{S}}$. Let

$$\alpha = \alpha(k) = \min\left\{ \frac{\sigma'_{\mathsf{v}}(k)}{\sigma'(k)}, \frac{\sigma'_{\mathsf{cas}}(k)}{\sigma'(k)}, \frac{\sigma'_{\mathsf{out}}(k)}{\sigma'(k)} \right\}.$$

Note that $\alpha(k) = \Theta(1)$.

**Theorem 3.** *Let $\lambda' = \min\{\alpha \frac{1}{3\sigma'_{\mathsf{v}}}, \alpha \frac{\delta}{6\sigma_{\mathsf{code}}}\}$. Then $C^{(2)}_{\tilde{S}}$ is secure against $\lambda'$-tampering adversaries (as defined in Definition 5).*

We defer the proof of Theorem 3 to the full version. We note that there we use the fact that every $\lambda'$-*globally tampering* adversary in $C^{(2)}_{\tilde{S}}$ is also a $\lambda = \min\{\frac{1}{3\sigma'_{\mathsf{v}}}, \frac{\delta}{6\sigma_{\mathsf{code}}}\}$-*locally tampering* adversary in $C^{(2)}_{\tilde{S}}$.

## References


1. Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.
2. Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. Cryptology ePrint Archive, Report 2010/544, 2010. http://eprint.iacr.org/.
3. Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In *EUROCRYPT*, pages 491–506, 2003.
4. Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.
5. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, 1997.
6. Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In *ASIACRYPT*, pages 722–739, 2011.
7. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In *EUROCRYPT*, pages 37–51, 1997.
8. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *IACR Cryptology ePrint Archive*, 2011:277, 2011.

9. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *FOCS*, pages 97–106, 2011.
10. Ran Canetti, Yevgeniy Dodis, Shai Halevi, Eyal Kushilevitz, and Amit Sahai. Exposure-resilient functions and all-or-nothing transforms. In *EUROCRYPT*, pages 453–469, 2000.
11. Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.
12. Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Improved non-committing encryption with applications to adaptively secure protocols. In *ASIACRYPT*, pages 287–302, 2009.
13. Seung Geol Choi, Aggelos Kiayias, and Tal Malkin. Bitr: Built-in tamper resilience. In *ASIACRYPT*, pages 740–758, 2011.
14. Dana Dachman-Soled and Yael Tauman Kalai. Securing circuits against constant-rate tampering, 2012. http://www.cs.columbia.edu/∼dglasner/MyPapers/tamper_pcp.pdf.
15. Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *CRYPTO*, pages 432–450, 2000.
16. R. L. Dobrushin and S. I. Ortyukov. Upper bound for the redundancy of self- correcting arrangements of unreliable functional elements. 13:203–218, 1977.
17. Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC*, pages 361–381, 2010.
18. Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.
19. Yevgeniy Dodis and Krzysztof Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. In *CRYPTO*, pages 21–40, 2010.
20. Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.
21. Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *ICS*, pages 434–452, 2010.
22. W. Evans and L. Schulman. On the maximum tolerable noise of k-input gates for reliable computation by formulas.
23. W. Evans and L. Schulman. Signal propagation and noisy circuits. In *IEEE Trans. Inform. Theory, 45(7)*, pages 2367–2373, 1999.
24. Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343–360, 2010.
25. Sebastian Faust, Krzysztof Pietrzak, and Daniele Venturi. Tamper-proof circuits: How to trade leakage for tamper-resilience. In *ICALP (1)*, pages 391–402, 2011.
26. Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT*, pages 135–156, 2010.
27. T. Feder. Reliable computation by networks in the presence of noise. In *IEEE Trans. Inform. Theory, 35(3)*, pages 569–571, 1989.
28. Péter Gács and Anna Gál. Lower bounds for the complexity of reliable boolean circuits with noisy gates. *IEEE Transactions on Information Theory*, 40(2):579–583, 1994.
29. Anna Gál and Mario Szegedy. Fault tolerant circuits and probabilistically checkable proofs. In *Structure in Complexity Theory Conference*, pages 65–73, 1995.
30. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *CHES*, number Generators, pages 251–261, 2001.
31. Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (atp) security: Theoretical foundations for security against hardware tampering. In *TCC*, pages 258–277, 2004.

32. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.
33. Shafi Goldwasser and Guy Rothblum. How to compute in the presence of leakage. Electronic Colloquium on Computational Complexity, 2012.
34. Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, pages 59–79, 2010.
35. Sudhakar Govindavajhala and Andrew W. Appel. Using memory errors to attack a virtual machine. In *IEEE Symposium on Security and Privacy*, pages 154–165, 2003.
36. Bruce E. Hajek and Timothy Weller. On the maximum tolerable noise for reliable computation by formulas. *IEEE Transactions on Information Theory*, 37(2):388–, 1991.
37. Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits ii: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.
38. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.
39. Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO*, pages 41–58, 2010.
40. Yael Tauman Kalai, Bhavana Kanukurthi, and Amit Sahai. Cryptography with tamperable and leaky memory. In *CRYPTO*, pages 373–390, 2011.
41. Yael Tauman Kalai, Anup Rao, and Allison Lewko. Formulas resilient to short-circuit errors, 2011. Manuscript.
42. Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.
43. John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. In *ESORICS*, pages 97–110, 1998.
44. Daniel J. Kleitman, Frank Thomson Leighton, and Yuan Ma. On the design of reliable boolean circuits that contain partially unreliable gates. In *FOCS*, pages 332–346, 1994.
45. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.
46. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.
47. Markus G. Kuhn and Ross J. Anderson. Soft tempest: Hidden data transmission using electromagnetic emanations. In *Information Hiding*, pages 124–142, 1998.
48. Feng-Hao Liu and Anna Lysyanskaya. Algorithmic tamper-proof security under probing attacks. In *SCN*, pages 106–120, 2010.
49. Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.
50. Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.
51. Krzysztof Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.
52. N. Pippenger. Reliable computation by formulas in the presence of noise. In *IEEE Trans. Inform. Theory, 34(2)*, pages 194–197, 1988.
53. Nicholas Pippenger. On networks of noisy gates. In *FOCS*, pages 30–38. IEEE, 1985.
54. Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, pages 200–210, 2001.
55. Josyula R. Rao and Pankaj Rohatgi. Empowering side-channel attacks. Cryptology ePrint Archive, Report 2001/037, 2001. http://eprint.iacr.org/.
56. J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. 1956.