Securing Circuits and Protocols Against $1/\operatorname{poly}(k)$ Tampering Rate

Dana Dachman-Soled[1] and Yael Tauman Kalai[2]

[1] University of Maryland
danadach@ece.umd.edu
[2] Microsoft Research
yael@microsoft.com

Abstract. In this work we present an efficient compiler that converts any circuit C into one that is resilient to tampering with $1/\operatorname{poly}(k)$ fraction of the wires, where k is a security parameter *independent* of the size of the original circuit $|C|$. Our tampering model is similar to the one proposed by Ishai *et al.* (Eurocrypt, 2006) where a tampering adversary may tamper with any wire in the circuit (as long as the overall number of tampered wires is bounded), by setting it to 0 or 1, or by toggling with it. Our result improves upon that of Ishai *et al.* which only allowed the adversary to tamper with $1/|C|$ fraction of the wires.

Our result is built on a recent result of Dachman-Soled and Kalai (Crypto, 2012), who constructed tamper resilient circuits in this model, tolerating a *constant* tampering rate. However, their tampering adversary may learn logarithmically many bits of sensitive information. In this work, we avoid this leakage of sensitive information, while still allowing leakage rate that is *independent* of the circuit size.

We mention that the result of Dachman-Soled and Kalai (Crypto, 2012) is only for Boolean circuits (that output a single bit), and for circuits that output k bits, their tampering-rate becomes $1/O(k)$. Thus for cryptographic circuits (that output k bits), our result strictly improves over (Dachman-Soled and Kalai, Crypto, 2012).

In this work, we also show how to generalize this result to the setting of two-party protocols, by constructing a general 2-party computation protocol (for any functionality) that is secure against a tampering adversary, who in addition to corrupting a party may tamper with $1/\operatorname{poly}(k)$-fraction of the wires of the computation of the honest party and the bits communicated during the protocol.

Key words: Tamper-resilient circuits, Two-party computation

1 Introduction

Constructing cryptographic schemes that are secure against physical attacks is a fundamental problem which has recently gained much attention in the cryptographic community. Indeed, physical attacks exploiting the implementation (rather than the functionality) of cryptographic schemes such as RSA have been known in theory for several years [41, 8] and recent works have shown that these attacks can be carried out in practice [9, 49]. There are many different types of physical attacks in the literature. For instance, Kocher et al. [42] 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 [41]. Other types of physical attacks include: inducing faults to the computation [7, 8, 42], using electromagnetic radiation [28, 54, 53], and several others [53, 39, 43, 31].

Although these physical attacks have proven to be a significant threat to the practical security of cryptographic devices, until recently cryptographic models did not take such attacks into account. In fact, traditional cryptographic models idealize the parties interaction and implicitly assume that an adversary may only observe an honest partys input-output behavior. Over the past few years, 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, 34, 47, 29, 33, 18, 50, 1, 48, 38, 15, 14, 22, 35, 30]). 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 partys 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 partys memory or circuit. In this work, we focus on constructing schemes that are secure even in the presence of tampering.

1.1 Our Results

We present a compiler that converts any circuit into one that is resilient to (a certain form of) tampering. Then, we generalize this result, and show how to construct a general two-party computation protocol that is secure against such tampering. We consider the tampering model of Ishai et al. [33]. Specifically, we consider a tampering adversary that may tamper with any (bounded) set of wires of the computation.

We note that our compiler that converts any circuit into a "tamper resilient" one, cannot guarantee correctness of the computation in the presence of tampering. This is the case, since the adversary may always tamper with the final output wire of the circuit. Therefore, as in [33], we do not guarantee correctness, but instead ensure privacy. In particular, we consider circuits that are associated with a secret state. We model such circuits as standard circuits (with AND, OR, and NOT gates), with additional secret, persistent memory that contains the secret state. The circuit itself is public and its topology is fully known to the adversary, whereas the memory content is secret. Following the terminology of [33], we refer to such circuits as private circuits. Our notion of security guarantees that the secret state of the circuit is protected even when an adversary may run the circuit on arbitrary inputs while continuously tampering with the wires of the circuit.

There are several fundamental impossibility results for tampering, which any positive result must circumvent. In the following, we discuss some of these limitations.

Class of Tampering Functions. 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. Thus, we must specify a class of allowed tampering functions. As in [33], in this we consider

tampering adversaries who can tamper with individual wires [33, 23, 12] and individual memory gates [11, 29, 19]. 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], where the tampering rate achieved is $1/|C|$, where $|C|$ is the size of the original circuit, we allow the adversary to tamper with any $1/\operatorname{poly}(k)$-fraction of wires and memory gates in the circuit, where k is security parameter and $\operatorname{poly}(k)$ is independent of the size of the original circuit. We note that the recent work of [12] gave a construction that is resilient to constant tampering rate. However, in their construction a tampering adversary may learn logarithmically many bit on the secret state of the circuit, and their guarantee was that such an adversary learns only logarithmically many bits about the secret state. We give the guarantee that a tampering adversary does not learn anything beyond the input/output behavior.

Necessity of Feedback. As noted by [29], 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.

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 [33, 12], 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 [33, 12], 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 adversarys view. Like in [33], we give the simulator only black-box access to the original private circuit with no additional leakage on the secret state. This is in contrast to the work of [12], who achieve a constant tampering rate, but where the simulator requires $O(\log k)$ bits of leakage on the secret state, where k is security parameter, in order to simulate. Thus, our result is meaningful in settings where [12] is not.

For example,[3] consider a setting where the same cryptographic key is placed on several devices, which are all obtained by an adversary. In this case, [12] does not guarantee any privacy for the cryptographic key, since $O(\log(k))$ bits leaked from each of several devices may give enough information to reconstruct the entire cryptographic key. Another example is a setting where secrecy of an algorithm is desired in order to protect intellectual property. In this case, the secret state of the device is the algorithm and the circuit is the universal circuit. Here, the same algorithm is placed on a large number of devices and is marketed. Thus, if $O(\log(k))$ bits are leaked from each device, then it may be possible to recover the entire algorithm.

Finally, we show how one can use our tamper-resilient compiler to achieve tamper-resilient secure two-party computation. We elaborate on this result in Section 1.4, but

[3] The following example, which was the motivating force behind this research, was brought to our attention by Shamir [55].

mention here that the results of [12] do not apply to this regime. Loosely speaking, the reason is that in this setting, the secret state of the circuit consists of the private input and randomness of each party, and (even logarithmic) leakage on the input and randomness of each party may completely compromise security of the two-party computation protocol.

Our Results More Formally. We present a general compiler T that converts a circuit C with a secret state s (denoted by C_s) into a circuit $T(C_s)$. We consider $\mathcal{PPT}$ adversaries $\mathcal{A}$ who receive access to $T(C_s)$ 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 $\lambda = \frac{1}{\mathrm{poly}(k)}$ and σ is the size of the circuit $T(C_s)$. Thus, in each run, $\mathcal{A}$ may tamper with a $1/\mathrm{poly}(k)$-fraction of wires and memory gates.

Theorem 1 (Main Theorem, Informal). *There exists an efficient transformation T which takes as input any circuit C_s with private state s, and outputs a circuit $T(C_s)$ such that the following two conditions hold:*

Correctness: For every input x, $T(C_s)(x) = C_s(x)$.
Tamper-Resilience: For every $\mathcal{PPT}$ adversary $\mathcal{A}$, which may tamper with $\lambda = 1/\mathrm{poly}(k)$-fraction of wires and memory gates in $T(C_s)$ per run, there exists an expected polynomial time simulator Sim*, which can simulate the view of $\mathcal{A}$ given only black-box access to C_s.*

Intuitively, the theorem asserts that adversaries who may observe the input-output behavior of the circuit while tampering with at most a λ-fraction of wires and memory gates in each run, do not get extra knowledge over what they could learn from just input-output access to the circuit.

1.2 Comparison with Ishai *et al.* [33] and Dachman-Soled *et al.* [12]

Our work follows the line of work of [33, 12]. As in our work, both these works 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. In the following, we discuss some similarities and differences among these works.

– In our construction, as in the construction of [33], we require the use of "randomness gates", which output a fresh random bit in each run of the circuit.[4] In contrast, the construction of [12] is deterministic.

[4] Alternatively, [33] can get rid of these randomness gates at the cost of relying on a computational assumption.

- The constructions of [33, 12] provide information-theoretic security, while our construction requires computational assumptions.
- As mentioned previously, [33] constructs tamper resilient circuits that are resilient only to local tampering: To achieve resilience to tampering with t wires per run, the circuit size blows up by a factor of at least t. In contrast, our tamper-resilient circuits are resilient to a $1/\operatorname{poly}(k)$-fraction of tampering, where k is security parameter. Thus, our tampering rate is *independent* of the original circuit size.
- The construction of [12] achieves a constant tampering rate, but requires $O(\log k)$ leakage on the secret state in order to simulate. As discussed above, in some settings the guarantees provided by [12] are too weak, while our construction still guarantees meaningful security.
  Moreover, [12] achieves constant tampering rate only for Boolean circuits that output a *single* bit. For circuits with k bit output, the resulting tampering-resilient circuit is only resilient to $1/k$-fraction of tampering.
- The tampering model of [33] 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 allow "persistent faults" only on memory gates (and not on wires), so if a memory value is modified during one run, it remains modified for all subsequent runs.

1.3 Overview of our Construction

Intuitively, our compiler works by first applying to the circuit C_s the leakage-resilient compiler T_{LR} of Juma and Vahlis [35]. The Juma-Vahlis compiler, T_{LR}, converts the circuit C_s into two subcomputations (or modules), $\mathsf{Mod}^{(1)}$ and $\mathsf{Mod}^{(2)}$, and provides the guarantee that (continual) leakage on the sub-computations $\mathsf{Mod}^{(1)}$ and $\mathsf{Mod}^{(2)}$ leak no information on the secret seed s. We refer the reader to [35] for the precise security guarantee. We emphasize that $T_{\mathsf{LR}}(C_s)$ has no security guarantees against a tampering adversary (rather only against a leaking adversary).

Our next idea is to use the tamper-resilient compiler T_{TR} of [12]. This compiler provides security against a (continual) *tampering* adversary, guaranteeing that the adversary learns at most $\log n$ bits about the secret s. In this work our goal is to remove this leakage from the security guarantee. To this end, we apply the tamper-resilient compiler T_{TR} to each sub-computation separately, each of which is now resilient to leakage.

We note however, that the Juma-Vahlis compiler relies on a secure hardware component. We do not want to rely on any such tamper-proof component. Therefore, we replace the tamper-proof component with a secure implementation. We describe our compiler in stages:

- First, we present a compiler (as above) that takes as input a circuit C_s and outputs a compiled circuit $T^{(1)}(C_s)$ that consists of 4 components. We prove that $T^{(1)}(C_s)$ is secure against adversaries that tamper with at most a $1/\operatorname{poly}(k)$ fraction of wires overall, but do not tamper with any of the wires in the first component, where the first component corresponds to the hardware component in the [35] construction (See Section 3.1).
- Then, we show how to get rid of the tamper-proof component and allow $1/\operatorname{poly}(k)$-fraction tampering overall (See Sections 3.2 and 5).

1.4 Extension to Tamper-Resilient Secure Two-Party Computation.

We consider the two-party computation setting, where in addition to corrupting parties, an adversary may tamper with the circuits of the honest parties and the messages sent by the honest parties. In this setting, we show how to use our construction of tamper-resilient circuits to obtain a general tamper-resilient secure two-party computation protocol, where an adversary may actively corrupt parties and additionally tamper with $1/\operatorname{poly}(k)$-fraction of wires, memory gates, and message bits overall.

To achieve our result, we start with any two-party computation (2-PC) protocol that is secure against malicious corruptions, and where the total number of bits exchanged depends only on security parameter k, and not on the size of the circuit computing the functionality. Such a 2-PC protocol can be constructed from fully homomorphic encryption and (interactive) CS-proofs. In addition we assume that each message sent in the protocol is accompanied with a signature. Then, for each party and each round of the protocol, we consider the private circuit computing the next message function, where the secret state is the party's private input and randomness and the public input is the transcript. We then run (a slight modification of) our tampering compiler on each such next message circuit to obtain a circuit that is resilient to $1/\operatorname{poly}(k)$-fraction of tampering. Since the total number of such circuits is $\operatorname{poly}(k)$, we achieve resilience to a $1/\operatorname{poly}(k)$-fraction of tampering overall. We refer the reader to Section 6 for details.

1.5 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 [13, 52, 51, 25, 20, 32, 26, 21], who considered the same noise model, ultimately showing that any circuit of size σ 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.* [37], Kleitnam *et al.* [40], and Gál and Szegedy [27]. The works of [40] and [37] consider a different model where the only type of faults allowed are short-circuiting gates. [27] consider a model that allows arbitrary faults on gates, and show how to construct tamper-resilient circuits for symmetric Boolean functions. We note that [27] allow a constant fraction δ 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. [27] also give a more general construction for any circuit which relies on PCP's. However, in order for their construction to work, they require an entire PCP proof π 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 (similarly to [12]) 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 [34, 47, 18, 50, 16, 24, 35, 30]. 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 [29, 19, 11, 44, 36, 45, 17] 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. We note that this model of allowing tampering only with memory is very similar to the problem of "related key attacks" (see [4, 2] and references therein). In contrast, in our work, as well as in the works of [33, 23, 12], the focus is on constructing circuits resilient to tampering with both the memory as well as the wires of the circuit.

Faust *et al.* [23] consider a model that is reminiscent to the model of [33, 12] 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 δ. As in [12], they allow the adversary to learn a logarithmic number of bits of information on the secret key. In addition, their result requires the use of small tamper-proof hardware components.

2 The Tampering Model

2.1 Circuits with Memory Gates

Similarly to [33], 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.

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.

2.2 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 $\mathsf{TAMP}_{\mathcal{A}}(T(C_s))$ the output distribution of an adversary $\mathcal{A}$ that carries out the above (continual) tampering attack on a compiled circuit $T(C_s)$. We note that our tampering compiler T is randomized and so the distribution is over the coins of T. We say that an adversary $\mathcal{A}$ is a λ-tampering adversary if during each run of the circuit he tampers with at most a λ-fraction of the circuit. Namely, $\mathcal{A}$ can make at most $\lambda \cdot |T(C_s)|$ 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 $T(C_s)$ as above, to a simulated world, where a simulator Sim tries to simulate the output of $\mathcal{A}$, while given only black-box access to the circuit C_s, and without tampering with the circuit at all. We denote the output distribution of the simulator by Sim^{C_s}.

Definition 1. *We say that a compiler T secures a circuit C_s against $\mathcal{PPT}$ λ-tampering adversaries, if for every $\mathcal{PPT}$ λ-tampering adversary $\mathcal{A}$ there exists a simulator* Sim*, that runs in expected polynomial time (in the runtime of $\mathcal{A}$), such that for sufficiently large k,*

$$\{\mathsf{TAMP}_{\mathcal{A}}(T(C_s))\}_{k\in\mathbb{N}} \overset{c}{\approx} \{\mathsf{Sim}^{C_s}\}_{k\in\mathbb{N}}.$$

In this work we construct such a compiler that takes any circuit and converts it into one that remains secure against adversaries that tamper with $\lambda = 1/\operatorname{poly}(k)$-fraction of the wires in the circuit, where k is the security parameter. Our compiler is uses both the Juma-Vahlis leakage compiler [35] and the recent tampering compiler of [12].

3 The Compiler

3.1 Overview of the First Construction

We start by presenting our first tampering compiler $T^{(1)}$ that takes as input a circuit C_s, and generates a tamper-resilient version of C_s which requires a tamper-proof component. In the case of no tampering, we show the correctness property: $T^{(1)}(C_s)(x) = C_s(x)$. Moreover, we prove that the circuit $T^{(1)}(C_s)$ is resilient to tampering with rate $1/\operatorname{poly}(k)$, where k is the security parameter.

High-level. On a very high-level, $T^{(1)}(C_s)$ works as follows.

1. Apply the Juma-Vahlis compiler T_{LR} to the circuit C_s to obtain a hardware component and two modules $(\mathsf{Mod}^{(1)}, \mathsf{Mod}^{(2)})$. First, $\mathsf{Mod}^{(1)} = \mathsf{Mod}^{(1)}_{\mathsf{PK},\mathsf{Enc}_{\mathsf{PK}}(s)}$ is the sub-computation that takes as input a string x and outputs the homomorphic evaluation of C_s on input x. We refer to this sub-computation as Component 2 of $T^{(1)}(C_s)$ and denote the output of this component by ψ_{comp}. Then a leakage and tamper-resilient hardware is used generate a "fresh" encryption of 0, denoted by ψ_{rand}, which is used to "refresh" the ciphertext ψ_{comp}. We refer to the leakage resilient-hardware outputting encryptions of 0 as Component 1. Component 3 of $T^{(1)}(C_s)$ then takes as input ψ_{comp} and ψ_{rand} and outputs the re-randomized ciphertext $\psi_* = \psi_{\mathsf{comp}} + \psi_{\mathsf{rand}}$. Finally, the second sub-computation of the Juma-Vahlis compiler, $\mathsf{Mod}^{(2)} = \mathsf{Mod}^{(2)}_{\mathsf{SK}}$, takes as input the refreshed ciphertext ψ_* and decrypts it to obtain $b = C_s(x)$. This sub-computation is referred to as Component 4 of $T^{(1)}(C_s)$.
2. The next idea is to apply the tampering compiler of [12], T_{TR}, to each of the components separately. We note that this tampering compiler allows a tampering adversary learn logartihmically many bits about the secret state of the circuit. However, since we apply the compiler to Components 2, 3, 4, which inherit the leakage resilient properties of the Juma-Vahlis compiler and are thus resilient to leakage of logarithmic size, this is not a concern to us.
   Unfortunately, this does not quite work. The reason is that the security definition of the tamper-resilient compiler T_{TR} allows the adversary to tamper with the input. Hence, if we simply take the components described above, then a tampering adversary may tamper with the inputs to each of the components, and may completely ruin the security guarantees of the Juma-Vahlis compiler. In particular, the refreshed ciphertext ψ_*, may no longer be distributed correctly. Instead we do the following:
3. Compute the second component, i.e. the tamper-resilient circuit $T_{\mathsf{TR}}(\mathsf{Mod}^{(1)})$. However, instead of outputting a single ciphertext ψ_{comp}, the circuit $T_{\mathsf{TR}}(\mathsf{Mod}^{(1)})$ will output M copies of ψ_{comp}, where M is a (large enough) parameter that will be specified below. We will argue that for any tampering adversary, either self-destruct occurs or a majority of the copies of ψ_{comp} are exactly correct.
4. Next apply a version of T_{TR} to the third and fourth components, with the guarantee that now an adversary cannot tamper with the input (without causing a self destruct), since the input is replicated M times, and an adversary can only tamper with a small fraction of these wires, and the compiled circuit will check for replicas. This version of T_{TR} turns out to be much simper than T_{TR} since the is size of the third and fourth components depends only on the security parameter, independent of the size of C_s, which turns out to simplify matters significantly.

We defer the details of the construction of $T^{(1)}(C_s)$ to the full version.
We are now ready to state the main theorem of this section:

Theorem 2. *$T^{(1)}(C_s)$ is secure against all $\mathcal{PPT}$ $\lambda = 1/\operatorname{poly}(k)$-tampering adversaries (as defined in Definition 1) who do not tamper with Component 1, assuming semantic security of the underlying encryption scheme $\mathcal{E}_{FH}$.*

We defer the proof of Theorem 2 to the full version.

3.2 Overview of Construction of Component 1

We now show how to construct Component 1, instead of relying on tamer-resilient hardware. Recall that our goal is to compute an encryption of 0 in a robust way so that *even after tampering* the output is statistically close to a fresh encryptions of 0 (assuming the output wires were not tampered with). Unfortunately, we don't quite manage to do this. Instead, we achieve a slightly weaker goal. We construct a circuit component that computes an encryption of 0, so that *even after tampering*, if self destruct did not occur, then the output of the computation is of the form $\psi_{fresh} + \psi_{rest}$, where ψ_{fresh} is a fresh encryption of 0, and ψ_{rest} is a simulatable (not necessarily fresh) encryption of 0 with "good" randomness and which is independent of ψ_{fresh}. Moreover, one can efficiently determine when self destruct occurred. It turns out that such a component has the security guarantees needed in order to replace the hardware component in Sections 3.1.

Clearly, this component will be randomized, since ciphertexts are randomized. We note that this is the first (and only) time randomization is used by the compiled circuit. Note that the time it takes to compute a ciphertext is completely independent of the size of the underlying circuit C_s, and depends only on the security parameter k. Moreover, recall that we allow the adversary to tamper with at most $1/\operatorname{poly}(k)$ wires.

The basic idea is the following: repeat the following sub-computation M times: Compute a fresh ciphertext of 0, along with a non-interactive zero-knowledge proof that it is indeed an encryption of 0 with "good" randomness. We denote the output of the i'th sub-computation by (ψ_i, π_i), where $\psi_i \leftarrow \mathsf{Enc}(0)$ and π_i is the corresponding NIZK. The basic observation is that at least one of these sub-computations will not be tampered with at all (due to the limit on the tampering budget), and hence one of these (untampered) sub-computations can be thought of as a secure hardware component.

Next the idea would be to add all these ciphertext together, to compute the final ciphertext $\psi = \sum_{i=1}^{M} \psi_i$. Note that if we knew that this addition computation was not tampered with, then we would be done. But clearly we do not have such a guarantee. Instead we will add a proof that this sum was computed correctly. However, in order to add a proof we need to identify the underlying language (or what exactly are we proving). Note that it is insufficient to prove that there exist ciphertexts $\psi'_1, \ldots, \psi'_M$, and corresponding proofs $\pi'_1, \ldots, \pi'_M$, such that $\psi = \sum_{i=1}^{M} \psi'_i$. This is insufficient since we will need the guarantee that at least one of these ciphetexts ψ'_i was computed without any tampering, and thus can be thought of as a fresh encryption of 0. To enforce this, we need to prove that these ciphertexts $\psi'_1, \ldots, \psi'_M$ are exactly those computed previously.

To this end, we use a signature scheme, and prove that we know a bunch of signed ciphertexts and corresponding proofs $\{\psi'_i, \sigma'_i, \pi'_i\}_{i=1}^{M}$ such that all the signatures are valid, all the proofs are valid, and $\sum_{i=1}^{M} \psi'_i = \psi$, where ψ is the claimed sum. More specifically, we fix an underlying signature scheme, and store in the memory of this component a pair of keys $(\mathsf{sksig}, \mathsf{vksig})$ for this signature scheme. The M sub-computations now each compute a triplet $(\psi_i, \sigma_i, \pi_i)$, where $\psi_i \leftarrow \mathsf{Enc}(0)$, σ_i is a signature of ψ_i, and π_i is a NIZK proof that indeed ψ_i is an encryption of 0 with "good" randomness. As before the size of each computation of $(\psi_i, \sigma_i, \pi_i)$ depends only on the security parameter and hence we can assume that at least one of these computations is not tampered with.

Once all these triplets $(\psi_i, \sigma_i, \pi_i)$ were computed, we compute $\psi = \sum_{i=1}^{M} \psi_i$ together with a succinct proof-of-knowledge that we know M triplets $(\psi_i, \sigma_i, \pi_i)$ such

that $\psi = \sum_{i=1}^{M} \psi_i$, each signature σ_i is a valid signature of ψ_i, and each proof π_i is a valid proof that ψ_i is an encryption of 0 with "good" randomness. We note that this part of the computation takes as input only the outputs of the previous M subcomputations, the verification key vksig, and the CRS. Intuitively, security seems to follow from the security of the signature scheme: Since the adversary is not given the secret key sksig during this computation, he cannot forge a signature on a new message, and hence must use the M ciphertexts output by the M sub-computations.

Unfortunately, this intuition is misleading, and there is a problem with this approach that complicates our construction. The problem is that some of the subcomputations that supposedly output a triplet $(\psi_i, \sigma_i, \pi_i)$ can be completely corrupted, and instead of outputting a signature σ_i may output the secret key sksig (or an arbitrary function of sksig). In such a case, during the proof that $\psi = \sum \psi'_i$, a tampering adversary, may choose the ciphertext ψ'_i arbitrarily (and in particular, depending on the *untampered* ciphertext) and forge a signature. We get around such an attack by using a very specific (one-time) *information-theoretically secure* signature scheme.

The signature scheme we use is an information-theoretical one-time (symmetric) version Lamport's signature scheme, where there is no verification key (only a secret key which is used both for verifying and computing signatures). Recall that the secret key in Lamport's scheme consists of $2k$ random strings: $\mathsf{sksig} = (x_{1,0}, x_{1,1}, \ldots, x_{k,0}, x_{k,1})$. A valid encryption of a message $m = (m_1, \ldots, m_k)$ is the tuple $(x_{1,m_1}, \ldots, x_{k,m_k})$. The reason we use this specific signature scheme is that it has an important feature, described below.

In our M subcomputations we use M independent secret keys. Namely, we store M independently generated keys $(\mathsf{sksig}^i)_{i=1}^M$ in memory, where each $\mathsf{sksig}^i = (x^i_{1,0}, x^i_{1,1}, \ldots, x^i_{k,0}, x^i_{k,1})$. During the i'th subcomputation, where supposedly the triplet $(\psi_i, \sigma_i, \pi_i)$ is computed, we use only sksig^i.

Our signature scheme has the following desired property: Consider a tampering adversary, who may completely tamper with the wires of subcomputation i, and thus can set σ_i to be an arbitrary function of the secret key sksig^i. Our signature scheme has the guarantee that this *arbitrary* string σ_i can (information-theoretically) be used to sign at most one message, and this message is determined by σ_i. Thus, we have the guarantee that the witness $\{(\psi'_i, \sigma'_i, \pi'_i)\}_{i=1}^M$ extracted from the proof-of-knowledge has the property that if the signatures and proofs are valid and $\psi = \sum \psi'_i$, then (with overwhelming probability) the signed ciphertexts $\{\psi'_i\}$ were generated *independently* of the untampered ciphertext, and are all "good" encryptions of 0.

The proof system we use must be a *succinct* proof-of-knowledge. The reason is that we will run the verification circuit M times, and argue that most of the verification circuits cannot be tampered with. However, to argue this we use the fact that the size of each verification circuit is of size $\mathrm{poly}(k)$, independent of the original circuit size. To ensure that each verification circuit is indeed of size $\mathrm{poly}(k)$ (independent of M) we need to use succinctness, since the verification circuit depends on the proof length.

The actual succinct proof-of-knowledge we use is *universal arguments* [3], which is an interactive version of CS-proofs. Universal arguments consist of 4 messages, which we denote by $(\alpha, \beta, \gamma, \delta)$. The verifier's messages α and γ (which are random) are

stored in the memory, and the prover's messages (β and δ) are computed during the computation of the circuit.

There are still some technical difficulties that remain. First, everything in memory must be stored in a tamper-resilient way, with the guarantee that if something in memory is corrupted then self-destruct occurs. To this end, we store M copies of the CRS and M copies of the public key of the encryption scheme. As done in previous components, we check that all the copies are the same, and if not the component self-destructs (i.e., the memory is overwritten with zeros). We also need to store the secret keys $\mathsf{sksig}^1, \ldots, \mathsf{sksig}^M$ in a robust manner, but note that since there are M such keys, simply storing M copies of each secret key is not good enough, since we allow $\mathrm{poly}(k)$ fraction of the memory gates to be tampered with, and in particular all of the repetitions of a single secret key sksig^i can be tampered with. Instead, we compute the hash value $h(\mathsf{sksig}) = h(\mathsf{sksig}^1, \ldots, \mathsf{sksig}^M)$, where h is a collision resistant hash function, and we store M copies of $h(\mathsf{sksig})$.

In the proof-of-knowledge, the statement is the tuple $(\psi, \mathsf{CRS}, \mathrm{PK}, h(\mathsf{sksig}))$, and we prove that we know a witness $\{\psi_i, \sigma_i, \pi_i, \mathsf{sksig}^i\}_{i\in[M]}$ such that $\psi = \sum_{i=1}^{M} \psi_i$, all the proofs π_i are accepted (with respect to CRS), all the signatures σ_i are valid (with respect to sksig^i), and $h(\mathsf{sksig}^1, \ldots, \mathsf{sksig}^M) = h(\mathsf{sksig})$. Unfortuantely, using a symmetric (information theoretical) signature scheme, introduces a new problem: This computation now does use the secret key, and hence a new signature may be forged during this computation.

We solve this problem by adding another proof-of-knowledge before this proof-of-knowledge, which ties the hands of the adversary, and causes him to "commit" to these signatures (without knowing the secret keys). More specifically, after the initial M sub-computations, we compute $h(\sigma) \triangleq h(\sigma_1, \ldots, \sigma_M)$ and add a universal argument that we know $(\sigma_1, \ldots, \sigma_M)$ such that $h(\sigma_1, \ldots, \sigma_M) = h(\sigma)$. Note that this computation does *not* use the secret keys $(\mathsf{sksig}^1, \ldots, \mathsf{sksig}^M)$. We think of $h(\sigma)$ as a commitment to the signatures.

Then in the next proof-of-knowledge, the instance is $(\psi, \mathsf{CRS}, \mathrm{PK}, h(\mathsf{sksig}), h(\sigma))$, and we prove that we know a witness $(\psi_i, \sigma_i, \pi_i, \mathsf{sksig}^i)$ such that $\psi = \sum_{i=1}^{M} \psi_i$, all the proofs π_i are accepted (with respect to CRS), all the signatures σ_i are valid (with respect to sksig^i), $h(\mathsf{sksig}^1, \ldots, \mathsf{sksig}^M) = h(\mathsf{sksig})$, and $h(\sigma_1, \ldots, \sigma_M) = h(\sigma)$. We use the fact that h is collision resistant to argue that even if the adversary uses the secret key here to forge signatures of new messages, these new signatures cannot hash to $h(\sigma)$ assuming the adversary cannot find collisions in h.

We now present the details of the construction and security proof for Component 1.

4 Component 1:

4.1 Universal Arguments

In what follows we give the properties of the universal argument that will be useful for us. We note that the definition below slightly differs from its original form in [3]. First, we define universal arguments for any language in $\mathsf{NTIME}(T)$ (i.e., any language computable by a non-deterministic Turing machine running in time T), for any T :

$\mathbb{N} \to \mathbb{N}$, whereas Barak and Goldreich (following Micali [46]) define it for a universal non-deterministic language. Second, our proof-of-knowledge property slightly differs from the one presented in [3], but easily follows from their original formulation.

Definition 2. *Let $T : \mathbb{N} \to \mathbb{N}$, and let L be any language in $\mathsf{NTIME}(T)$. A universal argument for L is a 4-round argument system (P, V) with the following properties:*

1. **Efficiency.** *There exists a polynomial p,*[5] *such that for any instance $x \in \{0,1\}^k$ the time complexity of $V(x)$ is $p(k)$, independent of T. In particular the communication complexity is at most $p(k)$ as well. Moreover, if $x \in L$ then for any valid witness w, the runtime*[6] *of $P(x, w)$ is at most $T(k) \cdot \mathrm{polylog(T(k))}$.*
2. **Completeness.** *For every $x \in L$ and for any corresponding witness w,*

$$\Pr[(P(x, w), V(x)) = 1] = 1.$$

3. **Computational Soundness.** *For every polynomial size circuit family $\{P_k^*\}$ and for every $x \in \{0,1\}^k \setminus L$,*

$$\Pr[(P_k^*(x), V(x)) = 1] = \mathrm{neg}(k).$$

4. **Proof-of-Knowledge Property.** *There exists a a polynomial q and a probabilistic algorithm E (an extractor) such that for every poly-size circuit family $\{P_k^*\}$ and for every $x \in \{0,1\}^k$, if $\Pr[(P_k^*(x), V(x)) = 1] \geq \epsilon$ then*

   $\Pr[E^{P_n^*}(x)$ *outputs a valid witness after running in time $q(1/\epsilon, T(k))$]* $= 1 - \mathrm{neg}(n)$.

   In particular, if P^* succeeds in proving that $x \in \{0,1\}^k \cap L$ with non-negligible probability, then E can extract a corresponding witness in expected polynomial time in $T(k)$.

4.2 A Formal Description of Component 1

We first describe the cryptographic ingredients used by Component 1.

– A one-time symmetric signature scheme $\Pi_{\mathsf{Sign}} = (\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Verify})$, defined as follows:

  $\mathsf{SigGen}(1^k)$: SigGen outputs a random string sksig which consists of k pairs of random strings

$$(x_{1,0}, x_{1,1}), \ldots, (x_{k,0}, x_{k,1}),$$

  where each $x_{\ell,b} \in_R \{0,1\}^{2k}$ is of length $2k$.

  $\mathsf{Sign}(\mathsf{sksig}, m)$, where $|m| = k$: Let $m = m[1], \ldots, m[k]$ be the bit representation of m. Sign outputs $\sigma = (x_{1,m[1]}, m[1]), \ldots, (x_{k,m[k]}, m[k])$.

[5] This polynomial is a universal polynomial that does not depend on the language L.

[6] We note that this is not the complexity guarantee given in the work of [3]. However, this complexity can be achieved by instantiating the universal argument using the recent efficient PCP construction of [5].

$\mathsf{Verify}_{\mathsf{Sign}}(\mathsf{sksig}, \sigma, m)$: $\mathsf{Verify}_{\mathsf{Sign}}$ parses $\sigma = (y_1, b[1]), \ldots, (y_k, b[k])$ and checks that for every $j \in [k]$ it holds that $b[j] = m[j]$ and $y_j = x_{j,m[j]}$. If yes, it outputs 1, and otherwise it outputs 0.

- A family of collision resistant hash functions $\mathcal{H} = \{h_{\mathsf{key}}\}$, where $h_{\mathsf{key}} : \{0,1\}^* \to \{0,1\}^k$.
- A non-interactive zero-knowledge (NIZK) proof system Π_{NIZK}.
- Universal arguments, which is an interactive variant of the CS proof system. Universal arguments consist of 4 messages, which we denote by $(\alpha, \beta, \gamma, \delta)$. The messages α and γ are sent by the verifier and are uniformly random strings.

We now describe Component 1. In what follows M is a parameter chosen as in Section 3.

Remark. For the sake of simplicity (and in an effort to focus on the new and interesting aspects of our component), in our formal description below, we do not formally define the notion of a ciphertext with "good" randomness. Intuitively, by "good" randomness we mean randomness r for which the error term in the ciphertext $\mathsf{Enc}_{\mathsf{PK}}(0; r)$ is not too big, so that one can perform homomorphic operations on it (that can later be decrypted using the secret key). We use the fact that a random string r is "good" with overwhelming probability.

In what follows, we use this notion of "good" randomness in a hand-wavy manner and assume that the sum of M ciphertext with "good" randomness is a ciphertext with "good" randomness (an assumption which of course does not hold inductively).

Memory: Encoding the Memory. Generate M secret keys $\mathsf{sksig}^1, \ldots, \mathsf{sksig}^M \leftarrow \mathsf{SigGen}(1^k)$ for the signature scheme, and place in memory. Recall that for each i, the key sksig^i consists of k pairs of random values which we denote by $(x^i_{1,0}, x^i_{1,1})$, $\ldots$, $(x^i_{k,0}, x^i_{k,1})$. Let $\mathsf{sksig} = \mathsf{sksig}^1 || \cdots || \mathsf{sksig}^M$.

In what follows, for any random variable x, we let $\tilde{x} = x^M$ denote M concatenated copies of x.

Compute the following encodings and place in memory:

1. Place $\widetilde{\mathrm{PK}}$ in memory, where PK is the public-key of the underlying (homomorphic) encryption scheme.
2. Choose a random function h_{key} from the collision resistant family $\mathcal{H}$, and place $\widetilde{\mathsf{key}}$ in memory.
3. Compute $\mathsf{h}(\mathsf{sksig}) = h_{\mathsf{key}}(\mathsf{sksig})$ and place $\widetilde{h(\mathsf{sksig})}$ in memory.
4. Choose a common reference string CRS for the NIZK proof system Π_{NIZK} and place $\widetilde{\mathsf{CRS}}$ in memory.
5. Choose random strings (α_1, γ_1) to be the random coins of the verifier in the first universal argument, and (α_2, γ_2) to be the random coins of the verifier in the second universal argument. Place $\widetilde{\alpha}_1$, $\widetilde{\gamma}_1$, $\widetilde{\alpha}_2$, $\widetilde{\gamma}_2$ in memory.

In what follows, when the circuit computation accesses one of the stored values $x \in \{\mathrm{PK}, \mathsf{key}, \mathsf{CRS}, \mathsf{h}(\mathsf{sksig}), \alpha_1, \gamma_1, \alpha_2, \gamma_2\}$, we always assume that it is accessing the first column of $\tilde{x}$.

Segment 1**.**

1. The first part of the computation takes randomness of length $M \cdot \mathrm{poly}(k)$ as input and performs M parallel subcomputations. We refer to each subcomputation as a block and denote the M blocks by $\mathsf{B}_1, \ldots, \mathsf{B}_M$. For $1 \leq i \leq M$, a random string $r_i = r_i^1 || r_i^2 \in \{0,1\}^{\mathrm{poly}(k)}$ is generated by hardware randomness gates. Each block B_i receives the corresponding $r_i^1 || r_i^2$ as input and performs the following computation:
   - On input r_i^1, compute $\psi_i = \mathsf{Enc}_{\mathrm{PK}_{FHE}}(0; r_i^1)$. Each bit of the output $\psi_i[1], \ldots, \psi_i[k]$ is split into 4 wires which are used later on, as specified.
   - On input ψ_i, sksig^i, compute $\sigma_i = \mathsf{Sign}(\mathsf{sksig}^i, \psi_i)$. Each bit of the output $\sigma_i[1], \ldots, \sigma_i[k]$ is split into 4 wires which are used later on, as specified.
   - On input r_i^1, r_i^2, CRS, compute a NIZK proof π_i, using proof system Π_{NIZK} with CRS and randomness r_i^2, that there exists "good" randomness r_i^1 such that $\mathsf{Enc}_{\mathrm{PK}_{FHE}}(0; r_i^1) = \psi_i$. Each bit of the output $\pi_i[1], \ldots, \pi[\mathrm{poly}(k)]$ is split into 2 wires which are used later on, as specified.
2. The next part of the computation takes as input $\psi_1, \ldots, \psi_M$ and outputs $\psi_{rand} = \sum_{i=1}^{M} \psi_i$. Each of the k output wires corresponding to the bits of $\psi_{rand} = \psi_{rand}[1], \ldots, \psi_{rand}[k]$ will be split into $M+2$ wires, which are used later on, as specified.
3. This part of the computation takes as input $\sigma_1, \ldots, \sigma_M$ and key, and computes $\mathsf{h}(\sigma) = h_{\mathsf{key}}(\sigma_1, \ldots, \sigma_M)$. Each of the k output wires corresponding to the bits of $\mathsf{h}(\sigma) = \mathsf{h}(\sigma)[1], \ldots, \mathsf{h}(\sigma)[k]$ is split into $2M+4$ wires which are used later on, as specified.
4. This part of the circuit computes a universal argument that proves knowledge of signatures $\sigma_1, \ldots, \sigma_M$ that hash to $\mathsf{h}(\sigma)$. More specifically, this part of the computation takes as input a witness $\sigma_1, \ldots, \sigma_M$ and the tuple $(\mathsf{key}, h(\sigma), \alpha_1, \gamma_1)$, and does the following:
   - Take α_1 to be the verifier's first message. Compute the second message β_1 of the universal argument for the following language:
   $$\mathcal{L}_1 = \{(\mathsf{h}(\sigma), \mathsf{key}) \mid \exists \sigma'_1, \ldots, \sigma'_M : h_{\mathsf{key}}(\sigma'_1, \ldots, \sigma'_M) = \mathsf{h}(\sigma)\}.$$
   Each bit of the output $\beta_1 = \beta_1[1], \ldots, \beta_1[k]$ is split into M wires which are used later on, as specified. This part of the computation also outputs a state STATE_1 which is passed to the next part of the computation, below.
   - The next part of the computation takes as input STATE_1 and γ_1, where γ_1 is the third message of the verifier. Compute the fourth message δ_1 for the language $\mathcal{L}_1$ and statement $(\mathsf{h}(\sigma), \mathsf{key})$. Each bit of the output $\delta_1 = \delta_1[1], \ldots, \delta_1[\mathrm{poly}(k)]$ is split into M wires which are used later on, as specified.
5. This part of the circuit computes a universal argument that ψ_{rand} was computed "correctly". More specifically, this part of the computation takes as input a witness
   $$((\psi_1, \sigma_1, \pi_1, \mathsf{sksig}_1), \ldots, (\psi_M, \sigma_M, \pi_M, \mathsf{sksig}_M))$$
   and the tuple $(\psi_{rand}, \mathsf{key}, h(\sigma), \mathsf{h}(\mathsf{sksig}), \mathsf{CRS}, \alpha_2, \gamma_2)$ and does the following:

– Take α_2 to be the verifier's first message and compute the second message β_2 of the universal argument for the following language:

$$\begin{aligned}\mathcal{L}_2 = \{\psi_{rand}, \mathsf{h}(\sigma), \mathsf{key}, \mathsf{CRS} \mid \exists(\psi'_1, \sigma'_1, \mathsf{sksig}'_1, \pi'_1), \ldots, (\psi'_M, \sigma'_M, \mathsf{sksig}'_M, \pi'_M) : \\ \sum_{i=1}^{M} \psi'_i = \psi_{rand}; \\ \wedge\ h_{\mathsf{key}}(\sigma'_1, \ldots, \sigma'_M) = \mathsf{h}(\sigma); \\ \wedge\ \text{ for } 1 \le i \le M, \mathsf{Verify}_{\mathsf{Sign}}(\mathsf{sksig}'_i, \psi'_i, \sigma'_i) = 1 \\ \wedge\ \text{ for } 1 \le i \le M, \mathsf{Verify}_{\Pi}(\mathsf{CRS}, \psi'_i, \pi'_i) = 1 \\ \wedge\ h_{\mathsf{key}}(\mathsf{sksig}'_1, \ldots, \mathsf{sksig}'_M) = \mathsf{h}(\mathsf{sksig})\}\end{aligned}$$

Each bit of the output $\beta_2 = \beta_2[1], \ldots, \beta_2[k]$ is split into M wires which are used later on, as specified. This part of the computation also outputs a state STATE_2 which is passed to the next part of the computation, below.

– The next part of the computation takes γ_2 to be the third message of the verifier, and uses STATE_2 to compute the fourth message δ_2 for the language $\mathcal{L}_2$ and statement $(\psi_{\mathsf{rand}}, \mathsf{h}(\sigma), \mathsf{key}, \mathsf{CRS})$. Each bit of the output $\delta_2 = \delta_2[1], \ldots, \delta_2[\text{poly}(k)]$ is split into M wires which are used later on, as specified.

Segment 2: Universal Argument Verification. This part consists of two sub-computations:

Verification of the computation of $\mathsf{h}(\sigma)$. This part consists of M copies of the verifier circuit for the universal argument for language $\mathcal{L}_1$ which takes as input the statement $(\mathsf{h}(\sigma), \mathsf{key})$, first message α_1, second message β_1 third message γ_1, and fourth message δ_1. We denote the i-th verifier circuit for $1 \le i \le M$ by Verify_i^1 and its output by λ_i^1.

Verification of the computation of ψ_{rand}. This part consists of M copies of the verifier circuit for the universal argument for language $\mathcal{L}_2$ which takes as input the statement $(\psi_{rand}, \mathsf{h}(\sigma), \mathsf{key}, \mathsf{CRS})$, first message α_2, second message β_2 third message γ_2, and fourth message δ_2. We denote the i-th verifier circuit for $1 \le i \le M$ by Verify_i^2 and its output by λ_i^2.

All these $2M$ output wires are inputs to the AND gate G_{cas}. This gate has $7k \cdot M$ additional input wires that belong to Segment 3 below. The gate G_{cas} has $K'' + k \cdot M$ output wires, where K'' is the size of the entire memory of of the circuit (of all components). We denote the values on these wires by $\{\mu_i\}_{i \in [K'' + k \cdot M]}$. The first K'' output wires (with values $\{\mu_i\}_{i \in [K'']}$) belong to Segment 3, and the other $k \cdot M$ output wires belong to Segment 4.

Segment 3: Error Cascade. This part is split into two subcomputations. The first subcomputation checks that all of the encodings $\tilde{x}$ that were placed in memory are uncorrupted. The second part propagates errors and overwrites memory.

– A circuit $C^{\widetilde{\mathsf{key}}}_{\mathsf{code}_{i,j}}$ of constant size σ_{code} for $1 \le i \le M$, $1 \le j \le k$:
Input: $\widetilde{\mathsf{key}}$.

Output:

$$\omega^1_{i,j} = \neg(\widetilde{\mathsf{key}}(1,j) \oplus \widetilde{\mathsf{key}}(i,j))$$

Similar subcircuits are constructed for the remaining encodings $\widetilde{\mathrm{PK}}$, $\widetilde{h(\mathsf{sksig})}$, $\widetilde{\mathsf{CRS}}$, $\widetilde{\alpha}_1, \widetilde{\gamma}_1, \widetilde{\alpha}_2, \widetilde{\gamma}_2$ with corresponding output wires $[\omega^2_{i,j}]_{i\in[M],j\in[k]}$, $[\omega^3_{i,j}]_{i\in[M],j\in[k]}$, $[\omega^4_{i,j}]_{i\in[M],j\in[k]}$, $[\omega^5_{i,j}]_{i\in[M],j\in[k]}$, $[\omega^6_{i,j}]_{i\in[M],j\in[k]}$, $[\omega^7_{i,j}]_{i\in[M],j\in[k]}$, $[\omega^8_{i,j}]_{i\in[M],j\in[k]}$. All these output wires are inputs to G_{cas}. Thus, in total, G_{cas} has $8kM + 2M$ input wires (M from Segment 1, M from Segment 2 and $8k \cdot M$ from Segment 3), and outputs:

$$\left(\bigwedge_{i\in[M]} \lambda^1_i\right) \wedge \left(\bigwedge_{i\in[M]} \lambda^2_i\right) \wedge \left(\bigwedge_{i\in[M],j\in[k],\ell\in[7]} \psi^\ell_{i,j}\right)$$

– The first K'' output wires of G_{cas} are fed to all the memory gates. If the output of G_{cas} is 0, then the memory gates are set to 0. Otherwise, the memory gates remain unchanged.

Segment 4: The Output of Component 1. This segment has k AND gates $G_{\mathsf{out},1}, \ldots, G_{\mathsf{out},k}$, each with fan-in $M+1$. This segment contains all the $k \cdot M + k$ input wires to $G_{\mathsf{out},1}, \ldots, G_{\mathsf{out},k}$: The first M input wires to each gate $G_{\mathsf{out},j}$ come from the output wires of G_{cas} (with values $\{\mu_i\}_{i=K''+(j-1)\cdot M+1}^{K''+j\cdot M}$), and the other input wire of $G_{\mathsf{out},j}$ is the j-th output wire of the Circuit Computation in Segment 1, which computes the encryption ψ_{rand}. Each AND gate $G_{\mathsf{out},j}$ has fan-out M, where the M output wires of $G_{\mathsf{out},j}$ are set to:

$$\psi^*_{rand}[j] = \psi_{rand}[j] \wedge \left(\bigwedge_{K'+(j-1)\cdot M+1\le i\le K'+j\cdot M} \mu_i\right).$$

The final output of Component 1 is denoted by ψ^*_{rand}.

Remark. We note that the size of Component 1 is of order $M \cdot \mathrm{poly}(k) \cdot \mathrm{polylog}(\mathrm{M} \cdot \mathrm{poly}(\mathrm{k}))$, which can be written as $M \cdot \mathrm{poly}(k)$ (since M is poly-sized and so $\mathrm{polylog}(\mathrm{M})$ is smaller than k), due to the fact that we use the recent efficient PCP construction of Ben-Sasson et al. [5] to construct our universal arguments. We note that this implies that a $1/\mathrm{poly}(k)$-tampering adversary cannot tamper with each of the M subcomputations at the beginning of Component 1. An important assumption throughout the analysis will be that at least one of the M subcomputations is untampered.

Notation. In the following theorem, for any λ-tampering adversary $\mathcal{A}$ (as defined in Definition 1), we denote by t the maximum number of times $\mathcal{A}$ runs the tampered circuit. For each run $i \in [t]$, we denote by $\psi_{rand,i}$ the ciphertext in the statement of the second universal argument. For each run $i \in [t]$, we denote by $\psi_{fresh,i}$ the ciphertext outputted by the untampered subcomputation, and denote by $\psi_{rest,i} = \psi_{rand,i} - \psi_{fresh,i}$.

We denote by

$$(i^*, (\psi_{fresh,1}, \psi_{rest,1}), \ldots, (\psi_{fresh,i^*-1}, \psi_{rest,i^*-1}), \psi_{fresh,i^*}) \leftarrow \mathsf{REAL}_{\mathcal{A}},$$

where i^* is the first round where self destruct occurs in the executions with the tampering instructions of $\mathcal{A}$. If self destruct does not occur (i.e. $i^* = t+1$) then set $\psi_{fresh,i^*} = \perp$.

Theorem 3. *Assume the soundness of the underlying universal argument, the security (existential security against adaptive chosen message attacks) of the underlying signature scheme Π_{Sign} , the semantic security of the underlying encryption scheme $\mathcal{E}_{FH}$, the security of the underlying collision resistant hash family $\mathcal{H}$, and the soundness and security of the underlying NIZK proof system Π_{NIZK}. Let $\lambda = 1/\operatorname{poly}(k)$.*

Then for any $\mathcal{PPT}$ λ-tampering adversary $\mathcal{A}$ there exists a simulator $S = (S_1, S_2)$ running in expected polynomial time, such that

$$(i', \psi'_{rest,1}, \ldots, \psi'_{rest,i'-1}, \psi'_{fresh}, \text{STATE}) \leftarrow S_1(1^M, \text{PK}),$$

and for $(\psi_{fresh,1}, \ldots, \psi_{fresh,t}) \leftarrow \mathsf{Enc}_{\text{PK}}(0)$ fresh encryptions of 0,

$$j' \leftarrow S_2(1^M, \text{PK}, \text{STATE}, \psi_{fresh,1}, \ldots, \psi_{fresh,t}),$$

such that

1. *$\psi'_{rest,1}, \ldots, \psi'_{rest,i'-1}$ are (simulatable) encryptions of 0 with "good" randomness.*
2. *$j' \leq i'$.*
3. *$\mathsf{REAL}_{\mathcal{A}} \equiv (j', (\psi_{fresh,1}, \psi'_{rest,1}), \ldots, (\psi_{fresh,i''-1}, \psi'_{rest,j'-1}), \tilde{\psi}_{fresh})$, where $\tilde{\psi}_{fresh} = \psi_{fresh,j'}$ for $j' < i'$, and $\tilde{\psi}_{fresh} = \psi'_{fresh}$ for $j' = i' \leq t$, and $\tilde{\psi}_{fresh} = \perp$ for $j' = i' = t+1$.*

We defer the proof of Theorem 3 to the full version.

5 The Final Construction

Let $T^{(2)}(C_s)$ be our original compiled circuit $T^{(1)}(C_s)$, described in Sections 3.1, where Component 1 of $T^{(1)}(C_s)$, which was implemented by tamper-resilient hardware, is replaced with the Component 1 described in Sections 3.2 and 4.

We are now ready to state our main theorem.

Theorem 4. *Assume the soundness of the underlying universal argument, the security (existential security against adaptive chosen message attacks) of the underlying signature scheme Π_{Sign} , the semantic security of the underlying encryption scheme $\mathcal{E}_{FH}$, the security of the underlying collision resistant hash family $\mathcal{H}$, and the soundness and security of the underlying NIZK proof system Π_{NIZK}. Let $\lambda = 1/\operatorname{poly}(k)$.*

Then $T^{(2)}(C_s)$ is secure against ppt λ-tampering adversaries (as defined in Definition 1). Note that the adversary may tamper with all components, including Component 1.

Overview of Proof of Theorem 4. First, we consider a Component 1 which provides weaker guarantees than the idealized hardware component described in Section 3.1. We call this hardware component WeakComp1. Next, we show that with small modifications, we can reprove Theorem 2 when the idealized hardware component is replaced with the hardware component WeakComp1. Finally, we use Theorem 3 to show that the construction of Component 1 given in Sections 3.2 and 4 is an implementation of WeakComp1, which is secure against λ-tampering adversaries.

The WeakComp1 Hardware Component. We assume the existence of a hardware component WeakComp1, which computes ciphertexts $\psi_{\mathsf{fresh}}, \psi_{\mathsf{rest}}$ and outputs M copies of $\psi_{\mathsf{rand}} = \psi_{\mathsf{rest}} \oplus \psi_{\mathsf{fresh}}$, where ψ_{rest} is an arbitrary "good" encryption of 0 and ψ_{fresh} is a randomly generated encryption of 0 independent of ψ_{rest}.

Plugging in WeakComp1. We state the following lemma, which uses the component WeakComp1 defined above in order to obtain a fully tamper-resilient circuit. We defer the proof to the full version.

Lemma 1. *Replace Component* 1 *of* $T^{(1)}(C_s)$ *with* WeakComp1 *described above, yielding* $\tilde{T}^{(1)}(C_s)$*. Then* $\tilde{T}^{(1)}(C_s)$ *is secure against ppt* $\lambda = 1/\operatorname{poly}(k)$*-tampering adversaries (as defined in Definition 1), that do not tamper with* WeakComp1*, assuming semantic security of the underlying encryption scheme* $\mathcal{E}_{FH}$*.*

Putting it all Together. We now argue that our construction remains secure when we replace WeakComp1 in $\tilde{T}^{(1)}(C_s)$ with Component 1 described in Section 4 to yield $T^{(2)}(C_s)$.

Fix any ppt λ-tampering adversary $\mathcal{A}$. Now, consider the adversaries $\mathcal{A}^1$, which is the adversary $\mathcal{A}$, restricted to tampering with and running only the first component (note that we simulate the final output of the circuit—assuming self-destruct does not occur—for $\mathcal{A}^1$ in order to obtain the correct tampering function in each run). By Theorem 3, we have that there exists a simulator $S = (S_1, S_2)$ for $\mathcal{A}^1$ running in expected polynomial time, such that on input $(1^M, \text{PK})$, S_1 outputs $(i', \psi'_{rest,1}, \ldots, \psi'_{rest,i'-1}, \psi'_{fresh}, \text{STATE})$ and on input $(1^M, \text{PK}, \text{STATE}, \psi_{fresh,1}, \ldots, \psi_{fresh,t})$, where $\psi_{fresh,1}, \ldots, \psi_{fresh,t}$ are fresh encryptions of 0, S_2 outputs j', where j' is the index of the first run of Component 1, where some wire to G^1_{cas} is set to 0. Note that by combining the inputs and outputs of S_1, S_2 we obtain $\psi_{\mathsf{rand},1} = \psi_{\mathsf{fresh},1} + \psi_{\mathsf{rest},1}, \ldots, \psi_{\mathsf{rand},j'-1} = \psi_{\mathsf{fresh},j'-1} + \psi_{\mathsf{rest},j'-1}$. Let this sequence of ciphertexts define the input-output behavior of the hardware component WeakComp1.

Now, note that by the security properties of Component 1 (See Theorem 3 in Section 4), we are guaranteed that the following two distributions are statistically close: $\mathsf{REAL}_{\mathcal{A}} \equiv (j', (\psi_{fresh,1}, \psi'_{rest,1}), \ldots, (\psi_{fresh,j'-1}, \psi'_{rest,j'-1}), \tilde{\psi}_{fresh})$ (where, loosely speaking, a draw from $\mathsf{REAL}_{\mathcal{A}}$ corresponds to a setting of the above random variables in a real execution).

To simulate the view of $\mathcal{A}$, we distinguish between two cases: Simulating runs of the circuit when $i < j'$ and simulating runs of the circuit when $i \geq j'$. Consider the adversaries $\mathcal{A}^{2,3,4}$, which is the adversary $\mathcal{A}$, restricted to tampering with only the second, third, and fourth component and interacts with $\tilde{T}^{(1)}(C_s)$. As noted above, for runs $i < j'$, the input-output behavior of $T^{(2)}(C_s)$ in the presence of $\mathcal{A}$ is identical to

the input-output behavior of $\tilde{T}^{(1)}(C_s)$ in the presence of $\mathcal{A}^{2,3,4}$, where $\mathsf{WeakComp1}$ is defined as above. Therefore, by the security of $\tilde{T}^{(1)}(C_s)$ (see Lemma 1) we have that there exists a simulator Sim for runs $i < j'$ that simulates the view of $\mathcal{A}$.

Finally, for runs $i \geq j'$, we have that "self-destruct" already occurred and so we can perfectly simulate the view of $\mathcal{A}$ as follows: Return 0 unless $\mathcal{A}$ tampers with the output wire, in which case the circuit returns b if the tamper is "set to b", and returns 1 if the tamper is "toggle". This concludes the proof of Theorem 4.

6 Extension to Tamper-Resilient Two Party Computation

In this section, we consider a two-party computation setting, where in addition to corrupting parties, an adversary may tamper with the circuits of the honest parties and the messages sent by the honest parties. As usual, we restrict the adversary to tampering with a $\lambda = 1/\operatorname{poly}(k)$-fraction of wires, memory gates, and message bits overall.

Our security definition follows the standard ideal/real paradigm, which requires that the view of the (real world) adversary, who may tamper with λ-fraction of wires, memory gates and message bits, can be simulated by a simulator in the ideal world *without tampering*. We emphasize that the ideal world we consider is the "standard" ideal world, whereas in the real world we allow the adversary tampering power.

We note that we allow both parties a tamper-free input-dependent preprocessing phase, which does not require interaction and can be done individually, offline by each party. This phase allows the parties to prepare their tamper-resilient circuits and place their private inputs in memory, while no tampering occurs.

Our approach is quite simple. We begin with any two-party computation (2-PC) protocol secure against malicious corruptions, where the communication complexity depends only on security parameter, k, and not on the size of the circuit computing the functionality. Such a 2-PC protocol can be constructed from any fully homomorphic encryption scheme and succinct argument system (such as universal arguments [46, 3]).

For each party P_b, $b \in \{0, 1\}$ and each round i of the protocol, we consider the circuit $\mathsf{Next}^i_{x_b, r_b}$, which has the (secret) values x_b and r_b hardwired into it (corresponding to the input and the random coins of Party P_b). It takes as input the current transcript TRANS and it outputs the next message for party P_b. We run (a slight modification of) our tampering compiler $T^{(2)}(\mathsf{Next}^i_{x_b, r_b})$ on each such circuit to obtain a circuit which outputs the $\operatorname{poly}(k)$-bit next message for party P_b at round i. By the security guarantees of $T^{(2)}$ (see Theorem 4), the compiled circuit $T^{(2)}(\mathsf{Next}^i_{x_b, r_b})$ is resilient to $1/\operatorname{poly}(k)$-fraction of tampering. Since the total number of such circuits is $\operatorname{poly}(k)$, we are ultimately resilient to a $\lambda = 1/\operatorname{poly}(k)$-fraction of tampering.

This idea does not quite work, since the adversary may tamper with the messages sent between the two parties, which may render the resulting protocol insecure. To get around this, we add signatures to our protocol. Namely, we assume each player is associated with a verification key. This key can be transmitted via an error-correcting code in the beginning of the protocol, and we require that the length of this key be a large enough $\operatorname{poly}(k)$ so that an adversary cannot cause this message to decode to a different key (using his tampering budget). Each time a player sends a message, he will sign his message together with the entire transcript so far. Intuitively, each party must

sign the entire transcript to protect against a tampering adversary who gets signatures $\sigma_1, \ldots, \sigma_z$ on z protocol messages $m_1, \ldots, m_z$, and then forwards a transcript to an honest party which is a permutation of the z messages $m_1, \ldots, m_z$.

6.1 Overview of The Model: Tamper-Resilient 2-PC

We consider the setting where two parties P_0, with input x_0, and P_1, with input x_1 interact to compute a functionality $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_0, f_1)$. P_0 wishes to obtain $f_0(x_0, x_1)$ and P_1 wishes to obtain $f_1(x_0, x_1)$. In what follows, for the sake of simplicity of notation we assume that $f_0 = f_1 = f$, though our results extend trivially to the case where f_0 and f_1 differ.

Our security definition follows the ideal/real paradigm. We emphasize that our ideal model is identical to the standard ideal model, while our real model is stronger that the standard ideal model since we consider adversaries $\mathcal{A}$ who may corrupt one or more parties P_0, P_1 and may also behave as a λ-tampering adversary on the honest parties' circuits (which the honest party may prepare via input-dependent pre-processing).

The random variable $\mathsf{IDEAL}_{f,\mathsf{Sim}}(x_0, x_1)$ is defined as the output of both parties in the ideal execution computing functionality f (where Sim controls the malicious party and chooses its output). If both parties are honest, then $\mathsf{IDEAL}_{f,\mathsf{Sim}}(x_0, x_1)$ is defined as the output of both parties in the above ideal execution along with the output of Sim.

The random variable $\mathsf{REAL}_{\Pi_{\mathsf{TAMP}},\mathcal{A}}(x_0, x_1)$ is defined as the output of both parties after running Π_{TAMP} with inputs (x_0, x_1), where the honest party outputs the output of the protocol, and the malicious party controlled by $\mathcal{A}$ may output an arbitrary function of its view. If both parties are honest then $\mathsf{REAL}_{\Pi_{\mathsf{TAMP}},\mathcal{A}}(x_0, x_1)$ is defined as the output of both honest parties, together with the output of $\mathcal{A}$, which may be an arbitrary function of its view (i.e., of the transcript).

Definition 3. *(secure tamper-resilient two-party computation): Let f and Π_{TAMP} be as above. Protocol Π_{TAMP} is said to* securely compute f *(in the malicious model and in the presence of a λ-tampering adversary) if for every probabilistic polynomial-time real-world adversary $\mathcal{A}$, who may corrupt one of the parties (or both), and may also behave as a λ-tampering adversary on the honest parties' circuits, there exists an expected polynomial-time simulator Sim in the ideal-world, such that*

$$\{\mathsf{IDEAL}_{f,\mathsf{Sim}}(x_0, x_1)\} \stackrel{c}{\approx} \{\mathsf{REAL}_{\Pi_{\mathsf{TAMP}},\mathcal{A}}(x_0, x_1)\}\,.$$

6.2 Achieving Tamper-Resilient 2-PC

Fix any two-party functionality f. We assume the existence of a secure (against active corruptions) two-party protocol $\Pi_{\mathsf{MPC}}(f)$ for computing f, where the total communication complexity is $\ell(k)$, where k is security parameter, and $\ell(\cdot)$ is a fixed polynomial, independent of the size of the circuit which computes the functionality f. It is well-known that such a two-party protocol can be constructed from fully homomorphic encryption and CS-proofs.

To simplify our exposition, we construct our protocol in the public key model. Here, each party P_0, P_1 publishes a verification key vksig_0, vksig_1 for a digital signature scheme $\Pi_{\mathsf{Sign}} = (\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Verify})$, while storing the corresponding secret

key $\mathsf{sksig}_0, \mathsf{sksig}_1$. We note that such a protocol in the public key model can easily be converted to a protocol in the standard model. We defer the details to the full version.

Let r denote the number of rounds in the two-party protocol Π_{MPC} described above. For $i \in [r]$ let $\mathsf{Next}^{0,i}_{\text{STATE}}$ denote the circuit that has the secret state $\text{STATE} = (x_0, r_0, \mathsf{vksig}_0, \mathsf{vksig}_1, \mathsf{sksig}_0)$ hardwired into it, where x_0 and r_0 are the input and randomness of party P_0, $(\mathsf{vksig}_0, \mathsf{sksig}_0)$ are the verification and signing keys of P_0, and vksig_1 and the verification key of P_1. The circuit $\mathsf{Next}^{0,i}_{\text{STATE}}$ computes the next message function of party P_0 in the i'th round of the resulting tamper-resilient protocol.

The tamper-resilient protocol emulates Π_{MPC}. Each message of the tamper-resilient protocol consists of all the messages sent so far in Π_{MPC}, along with signatures. More formally, $\mathsf{Next}^{0,i}_{\text{STATE}}$ takes as input a message TRANS_{i-1}, which consists of all the $i-1$ pairs of messages sent in Π_{MPC} during the first $i-1$ rounds, where each message is accompanied by a signature of the entire transcript thus far. The circuit $\mathsf{Next}^{0,i}_{\text{STATE}}$, on input TRANS_{i-1}, does the following (the circuits $\mathsf{Next}^{1,i}_{\text{STATE}}$ are defined analogously):

1. Parse TRANS_{i-1}, as $2(i-1)$ message-signature pairs $(m_{b,j}, \sigma_{b,j})_{b\in\{0,1\},j\in[i-1]}$. Check that $\sigma_{1,i-1}$ is a valid signature of the message $\mathsf{MSG}_{i-1} = (m_{0,j}, m_{1,j})_{j\in[i-1]}$ w.r.t. vksig_1, and check that $\sigma_{0,i-1}$ is a valid signature for $(\mathsf{MSG}_{i-2}, m_{0,i-1})$, where $\mathsf{MSG}_{i-2} = (m_{0,j}, m_{1,j})_{j\in[i-2]}$. If either does not verify, output 0.
2. Otherwise, compute the next message $m_{0,i}$ of party P_0 in protocol Π_{MPC} given transcript MSG_{i-1}, randomness r_0 and input x_0.
3. Compute a signature $\sigma_{0,i}$ corresponding to the message $(\mathsf{MSG}_{i-1}, m_{0,i})$.
4. Output $(\mathsf{TRANS}_{i-1}, m_{0,i}, \sigma_{0,i})$

The tamper-resilient protocol Π_{TAMP} is depicted in Figure 1.

Theorem 5. *For every two-party functionality f, Π_{TAMP} securely computes f in the malicious model and in the presence of a $1/\operatorname{poly}(k)$-tampering adversary, where k is the security parameter, and* poly *is a fixed polynomial independent of the size of the circuit computing the functionality f.*

We defer the proof of Theorem 5 to the full version.

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. Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM J. Comput.*, 38(5):1661–1694, 2008.
4. Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In *EUROCRYPT*, pages 491–506, 2003.
5. Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *STOC*, pages 585–594, 2013.
6. 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.

Protocol Π_{TAMP} for computing functionality f

Public and Private keys: Party P_b generates a pair of verification and signing keys $(\mathsf{vksig}_b, \mathsf{sksig}_b) \leftarrow \mathsf{SigGen}(1^k)$. It publishes the verification key vksig_b, while keeping private the corresponding signing key sksig_b.

Inputs: Party P_b has an input x_b and a random tape r_b.

Preprocessing for Party P_b: Construct $r+1$ tamper-resilient subcircuits:

$$T^{(2)}(\mathsf{Next}^{b,1}_{\text{STATE}}), \ldots, T^{(2)}(\mathsf{Next}^{b,r}_{\text{STATE}}), T^{(2)}(\mathsf{Out}^{b}_{\text{STATE}}),$$

where $\text{STATE} = (x_b, r_b, \mathsf{vksig}_0, \mathsf{vksig}_1, \mathsf{sksig}_b)$. We emphasize that the compiler is run independently $r+1$ times, each time with fresh randomness.
For each $i \in [r]$, the circuit $\mathsf{Next}^{0,i}_{\text{STATE}}$, is defined as follows ($\mathsf{Next}^{1,i}_{\text{STATE}}$ is defined analogously):

It takes as input a partial transcript TRANS_{i-1}, and does the following:

1. Parse TRANS_{i-1} as $2(i-1)$ signed messages. Consider the last two messages denoted by $(m_{0,i-1}, \sigma_{0,i-1})$ and $(m_{1,i-1}, \sigma_{1,i-1})$. Let $\mathsf{MSG}_{i-2} = (m_{b,j})_{b\in\{0,1\}, j\in[i-2]}$ be the first $i-2$ pairs of messages sent in TRANS_{i-1}. Check that $\sigma_{0,i-1}$ is a valid signature of $(\mathsf{MSG}_{i-2}, m_{0,i-1})$ w.r.t. verification key vksig_0, and that $\sigma_{1,i-1}$ is a valid signature of $\mathsf{MSG}_{i-1} = (\mathsf{MSG}_{i-2}, m_{0,i-1}, m_{1,i-1})$ w.r.t. verification key vksig_1. If either fails, send $\perp$ to P_1 and abort.
2. Otherwise, run the next message function of protocol Π with partial transcript MSG_{i-1}, to obtain the next message $m_{0,i}$.
3. Compute $\sigma_{0,i}$, a signature of $(\mathsf{MSG}_{i-1}, m_{0,i})$ w.r.t. secret key sksig_0.
4. Output the partial (signed) transcript $(\mathsf{TRANS}_{i-1}, m_{0,i}, \sigma_{0,i})$.

The circuit $\mathsf{Out}^{0}_{\text{STATE}}$ takes as input a transcript TRANS_r, and does the following ($\mathsf{Out}^{1}_{\text{STATE}}$ is defined analogously):

1. Parse TRANS_r as $2r$ signed messages. Check the validity and consistency of the last two signatures If either fails output $\perp$.
2. Let MSG_r denote the $2r$ messages in TRANS_r.
3. Compute the output y_0 of protocol Π, assuming the messages exchanged in Π were MSG_r.
4. Output y_0.

Protocol Execution:

At round $i \in [r]$, party P_0, upon receiving a message TRANS_{i-1}, runs $T^{(2)}(\mathsf{Next}^{0,i}_{\text{STATE}})$ on input TRANS_{i-1}, and sends the output (which is of the form $(\mathsf{TRANS}_{i-1}, m_{0,i}, \sigma_{0,i})$) to P_1.
Analogously, party P_1, upon receiving a message $(\mathsf{TRANS}_{i-1}, m_{0,i}, \sigma_{0,i})$ from P_0, runs $T^{(2)}(\mathsf{Next}^{1,i}_{\text{STATE}})$ on input $(\mathsf{TRANS}_{i-1}, m_{0,i}, \sigma_{0,i})$, and sends the output to P_0.

Output:

1. Upon receiving the last message of the protocol, each party P_b runs $T^{(2)}(\mathsf{Out}^{b}_{\text{STATE}})$ on this last message to compute the output y_b.
2. Output y_b.

Fig. 1. Tamper-resilient, secure two-party computation protocol of functionality f.

7. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, pages 335–359, 1997.
8. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In *EUROCRYPT*, pages 37–51, 1997.
9. David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.
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. Seung Geol Choi, Aggelos Kiayias, and Tal Malkin. Bitr: Built-in tamper resilience. In *ASIACRYPT*, pages 740–758, 2011.
12. Dana Dachman-Soled and Yael Tauman Kalai. Securing circuits against constant-rate tampering. In *CRYPTO*, pages 533–551, 2012.
13. R. L. Dobrushin and S. I. Ortyukov. Upper bound for the redundancy of self- correcting arrangements of unreliable functional elements. 13:203–218, 1977.
14. 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.
15. Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.
16. Yevgeniy Dodis and Krzysztof Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. In *CRYPTO*, pages 21–40, 2010.
17. Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Non-malleable codes from two-source extractors. In *CRYPTO (2)*, pages 239–257, 2013.
18. Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.
19. Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *ICS*, pages 434–452, 2010.
20. W. Evans and L. Schulman. Signal propagation and noisy circuits. In *IEEE Trans. Inform. Theory, 45(7)*, pages 2367–2373, 1999.
21. W. Evans and L. Schulman. On the maximum tolerable noise of k-input gates for reliable computation by formulas. In *IEEE Trans. Inform. Theory, 49(11)*, pages 3094–3098, 2003.
22. Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343–360, 2010.
23. Sebastian Faust, Krzysztof Pietrzak, and Daniele Venturi. Tamper-proof circuits: How to trade leakage for tamper-resilience. In *ICALP (1)*, pages 391–402, 2011.
24. 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.
25. T. Feder. Reliable computation by networks in the presence of noise. In *IEEE Trans. Inform. Theory, 35(3)*, pages 569–571, 1989.
26. 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.
27. Anna Gál and Mario Szegedy. Fault tolerant circuits and probabilistically checkable proofs. In *Structure in Complexity Theory Conference*, pages 65–73, 1995.
28. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *CHES*, number Generators, pages 251–261, 2001.
29. 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.
30. Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, pages 59–79, 2010.

31. 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.
32. 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.
33. Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits ii: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.
34. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.
35. Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO*, pages 41–58, 2010.
36. Yael Tauman Kalai, Bhavana Kanukurthi, and Amit Sahai. Cryptography with tamperable and leaky memory. In *CRYPTO*, pages 373–390, 2011.
37. Yael Tauman Kalai, Anup Rao, and Allison Lewko. Formulas resilient to short-circuit errors, 2011. Manuscript.
38. Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.
39. John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. In *ESORICS*, pages 97–110, 1998.
40. 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.
41. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.
42. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.
43. Markus G. Kuhn and Ross J. Anderson. Soft tempest: Hidden data transmission using electromagnetic emanations. In *Information Hiding*, pages 124–142, 1998.
44. Feng-Hao Liu and Anna Lysyanskaya. Algorithmic tamper-proof security under probing attacks. In *SCN*, pages 106–120, 2010.
45. Feng-Hao Liu and Anna Lysyanskaya. Tamper and leakage resilience in the split-state model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 517–532. Springer, 2012.
46. Silvio Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453. IEEE Computer Society, 1994.
47. Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.
48. Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.
49. Andrea Pellegrini, Valeria Bertacco, and Todd M. Austin. Fault-based attack of rsa authentication. In *DATE*, pages 855–860, 2010.
50. Krzysztof Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.
51. N. Pippenger. Reliable computation by formulas in the presence of noise. In *IEEE Trans. Inform. Theory, 34(2)*, pages 194–197, 1988.
52. Nicholas Pippenger. On networks of noisy gates. In *FOCS*, pages 30–38. IEEE, 1985.
53. Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, pages 200–210, 2001.
54. Josyula R. Rao and Pankaj Rohatgi. Empowering side-channel attacks. Cryptology ePrint Archive, Report 2001/037, 2001. http://eprint.iacr.org/.
55. Adi Shamir. Personal communication, 2012.
56. J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. 1956.