Private Circuits II: Keeping Secrets in Tamperable Circuits

Yuval Ishai*, Manoj Prabhakaran**, Amit Sahai***, and David Wagner†

Abstract. Motivated by the problem of protecting cryptographic hardware, we continue the investigation of *private circuits* initiated in [16]. In this work, our aim is to construct circuits that should protect the secrecy of their internal state against an adversary who may modify the values of an *unbounded* number of wires, *anywhere in the circuit.* In contrast, all previous works on protecting cryptographic hardware relied on an assumption that some portion of the circuit must remain *completely free* from tampering.

We obtain the first feasibility results for such private circuits. Our main result is an efficient transformation of a circuit C, realizing an arbitrary (reactive) functionality, into a private circuit C' realizing the same functionality. The transformed circuit can successfully detect any serious tampering and erase all data in the memory. In terms of the information available to the adversary, even in the presence of an unbounded number of adaptive wire faults, the circuit C' emulates a black-box access to C.

1 Introduction

Can you keep a secret when your brain is being tampered with? In this paper we study the seemingly paradoxical problem of constructing a circuit such that *all parts* of the circuit are open to tampering at the level of logic gates and wires, and yet the circuit can maintain the secrecy of contents of memory. We construct *private circuits* which, even as they are being tampered with, can detect such tampering and, if necessary, "self-destruct" to prevent leaking their secrets. We consider security against a powerful inquisitor who may adaptively query the circuit while tampering with an arbitrary subset of wires within the circuit, *including the part of the circuit that is designed to detect tampering.*

The above question is motivated by the goal of designing secure cryptographic hardware. While the traditional focus of cryptography is on analyzing *algorithms*, in recent years there have been growing concerns about *physical* attacks that exploit the *implementations* (rather than the functionality) of cryptographic

* Technion. This research was supported by grant 2004361 from the United States-Israel Binational Science Foundation (BSF) and grant 36/03 from the Israel Science Foundation.

** U.I.U.C.

*** U.C.L.A. This research was supported by grants from the NSF ITR and Cybertrust programs, a grant from BSF, an Alfred P. Sloan Foundation Fellowship, and a generous equipment grant from Intel.

† U.C. Berkeley.

primitives. For instance, it is in some cases possible to learn the secret key of an encryption scheme by measuring the power consumed during an encryption operation or the time it takes for the operation to complete [19, 20]. Other types of physical attacks rely on inducing faults [6, 5, 20], electromagnetic radiation [28, 11, 29], magnetic fields [27], cache hit ratios [18, 24], probing wires using a metal needle [1], and others [17, 31, 32, 30, 2, 30]. In general, attacks of this type have proven to be a significant threat to the practical security of embedded cryptographic devices.

One possible approach for defeating the above type of attacks is by designing specific hardware countermeasures, such as adding large capacitors to hide the power consumption. Many such countermeasures have been proposed in the literature. An inherent limitation of these approaches is that each such countermeasure must be specially tailored for the set of *specific physical attacks* it is intended to defeat. For example, one might design physical protection against attacks based on electro-magnetic radiation, but still be vulnerable to attacks based on physical probes.

A different approach is to tackle the problem at the *logical* level, namely by designing algorithms that, when implemented, will be robust against a wide class of physical attacks. Here, we would want to classify attacks not based on the physical mechanism of the attack, but rather on the *logical effect of the attack* – for instance, can we defend against *all* physical attacks that toggle the value on a wire? Several ad-hoc approaches have been suggested (e.g., [10, 21, 15]) with some subsequently broken [7, 9]. Recently, a more general and theoretically sound study of physical security has been initiated in [16, 22, 12] (see Section 1.4 for an account of this related work).

The current paper continues this line of work, but departs from all previous work in the following fundamental way. All types of attacks that were previously considered from a theoretical perspective are either (1) in some sense *spatially limited*, and in particular cannot be applied to the entire circuitry on which they are mounted [12]; or (2) deal with observation rather than faults [16, 22]. The question that motivates our work is the intriguing possibility of offering protection even against adversaries that can tamper with the *entire* circuit. This goal might sound too ambitious. For instance, the adversary can easily modify the *functionality* of the circuit by simply destroying it completely. However, this does not rule out the possibility of preventing the adversary from learning the *secret information*, say a cryptographic key, stored in the circuit. Once the device is already in the hands of the adversary, secrecy is the primary relevant concern.

The above question is captured by our notion of a *private circuit*, which we also call a *self-destructing circuit*. Informally, such a circuit should carry out some specified functionality (say, encryption) while protecting the secrecy of its internal state (a key) even against an *unbounded* number of adversarial faults. A natural way for achieving this goal is to build a tamper detection mechanism which can detect faults and trigger a "self-destruction" mechanism to erase all internal state. (This is akin to a prisoner of war taking a suicide pill.) The central problem with implementing this approach in our setting is that *such a tamper detection circuitry as well as the self-destruction mechanism itself can be* *attacked and disabled by the adversary.* Thus, it is tempting to conjecture that such self-destructing circuits simply cannot exist.

In this paper, we obtain the first positive results establishing the feasibility of private circuits in the presence of adversarial faults that can affect *any* wire inside the circuit. Before describing our results, we give some further motivating discussion, and a more detailed account of the types of circuits and the fault model we consider.

1.1 Discussion of Difficulties

We briefly discuss some natural ideas to prevent loss of privacy due to faults and why they don't appear to work, as well as some inherent limitations to our model.

Natural Approaches. First, one can consider standard techniques for fault-tolerant circuits based on error-correcting codes or redundancy (see [26] and references therein). However, such approaches are limited to tolerating only a *bounded* number of total adversarial faults, whereas we are interested in the case where the adversary can induce, over time, an *unbounded* number of faults, eventually even faulting every wire in the circuit!

Next, one may think of using signature schemes or related techniques, which would work as follows at a high level: hard-wire into the circuit a signature on the circuit, and then verify the correctness of the signature before executing the original functionality, otherwise cause a "self-destruct" (*c.f.* [12]). In our context, this fails for a simple reason: the adversary can fault the wire in the circuit that contains the "Correct"/"Incorrect" output of the signature verification algorithm, so that this wire always reads "Correct", regardless of whether the signature verification succeeded or not.

Similarly, one may think of directly applying multi-party computing techniques [14, 4, 8] providing security against mobile Byzantine faults [23]. However, here we cannot rely on an "honest majority", since there is an unbounded number of faults, and every part of the circuit is susceptible to attacks. In protocols for multi-party computation with no honest majority, each party executes a large set of instructions, which invariably includes a verification step. In our model, the adversary can fault just this verification portion of each party's computation in order to fool the party into thinking that the verification always succeeds. Thus, whatever approach we take, we must somehow prevent this kind of attack.

Another idea that seems to immediately help is to use randomization: perhaps if we randomly encode "Correct" or "Incorrect" as 0 or 1, then we can prevent the above problems. But even this is problematic, because the adversary can, as its first set of actions, create faults that set all wires that should contain random values to 0, thereby eliminating the randomization. (In our approach, we are able to combine randomization ideas with other redundant encodings in order to fault-tolerantly detect such behavior.)

Limitations. To motivate the type of fault model we consider, it is instructive to address some inherent limitations on the type of adversarial behavior one could hope to resist, and still obtain a general transformation result that holds for all circuits. These limitations follow from the impossibility of program obfuscation [3]: The most immediate limitation is observed in [16], that it is impossible to protect against an attacker which can simultaneously *read* the values of *all* wires in the circuit. However, a number of natural *fault* models are also equivalent to program obfuscation. For instance, allowing the adversary to cause arbitrary immediate changes to the entire circuit trivially allows it to replace the entire circuit with one that outputs the contents of all memory, thus making the problem equivalent to program obfuscation. Similarly, if the adversary is allowed to insert or replace wires it can mount the same type of attack by undetectably adding wires from all memory cells to the output.

These limitations mean that we must consider attack models in which the adversary is more restricted. We concentrate on models in which the adversary can cause an *unbounded* number of faults over time, where these faults are localized to individual wires *anywhere* in the circuit.

1.2 The Model

We consider reactive functionalities, i.e., functions with an internal state that may be updated at each invocation. Such a functionality can be realized by a stateful boolean circuit C that, given an external input and its current internal state, produces an external output and a new internal state. (The state corresponds to some secret data stored in the circuit's memory.) We think of the interaction of such a circuit with the environment as being paced by *clock cycles*, where in each cycle the circuit receives an input, produces an output, and updates its internal state.[1] We would like to protect the secrecy of the internal state against an adversary that can induce faults in an unbounded number of wires. That is, in each clock cycle (or epoch) the adversary can adaptively choose t wires and permanently "set" (to 1), "reset" (to 0), or "toggle" the value of each wire. Then, the adversary can feed a new input to the modified circuit and observe the resulting output. By inducing such faults, the adversary's hope is to extract more information about the circuit's internal state than is possible via black-box access to the circuit. For instance, if the circuit implements an encryption or a signature scheme, the adversary may try to learn some nontrivial information about the secret key.

Our goal is to prevent the adversary from gaining any advantage by mounting the above type of attack. We formalize this requirement via a simulation-based definition (in the spirit of similar definitions from [16, 12]). Specifically, we say that a stateful circuit C' is a secure private (or self-destructing) implementation of C if there is a (randomized, efficient) transformation from an initial state s_0 and circuit C to an initial state s'_0 and circuit C' such that:

[1] An attacker may also try to tamper with the clock. To counter such attacks, we envision the use of volatile memory, such as DRAM, to implement memory cells whose contents fade over time if not refreshed regularly. In our main result, we need to assume that the attacker cannot induce too many faults within a single "epoch" defined by the amount of time it takes for the volatile memory to lose its value. If the adversary is tampering with the clock, then we define a clock cycle to the lesser of the actual clock cycle and one epoch.

1. $C'[s'_0]$ realizes the same functionality as $C[s_0]$.
2. Whatever the adversary can observe by interacting with $C'[s'_0]$ and adaptively inducing an unbounded number of wire faults, can be simulated by only making a *black-box* use of $C[s_0]$, without inducing any faults.

1.3 Our Contribution

We present general feasibility results, showing how to efficiently transform an arbitrary (reactive, stateful) circuit C into an equivalent self-destructing circuit C'. Specifically:

1. In the case of *reset only* wire faults (set wires or memory cells to 0), the circuit C' is secure against an unbounded number of adaptive faults. Security is either statistical, if C' is allowed to produce fresh randomness in each cycle, or is computational otherwise.
2. In the case of *arbitrary* wire faults (set wires or memory cells to 1, set to 0, or toggle the value), we can get the same results as above except that we limit the adversary to performing only a bounded number of faults per clock cycle.[2] Since the adversary can choose the faults to be permanent, the overall number of faults is still unbounded.

In all cases, the circuit C' is proven secure under the conservative simulation-based definition outlined above. Our techniques in both constructions can also yield privacy against a bounded number of probing attacks per cycle as per [16].

Our techniques. A central high-level idea behind our constructions is the following. Given a circuit C, we compile it into a similar circuit C' which can be viewed as a "randomized minefield". As long as C' is not tampered with, it has the same functionality as C. However, any tampering with C' will lead with high probability to "exploding a mine", triggering an automatic self-destruction and rendering C' useless to the adversary.

Implementing the above approach is far from being straightforward. One problem that needs to be dealt with is preventing the adversary from learning some useful partial information by merely getting "lucky" enough to not land on a mine. This problem is aggravated by the fact that the adversary may possess partial information about the values of internal circuit wires, implied by the observed inputs and outputs. Another problem, already discussed above, is that of protecting the self-destruction mechanism itself from being destroyed. This difficulty is overcome through a novel distributed and randomized self-destruction mechanism.

Combining the techniques in this paper with the results in [16], one can construct self-destructing circuits which simultaneously resist probing attacks in

[2] The complexity of the constructions depends linearly on the parameter t bounding the number of faults. In fact, our constructions resist attacks that can involve an *arbitrary* number of simultaneous faults, provided that no more than t faults are (simultaneously) concentrated in the same area. Thus, the task of mounting such a coordinated attack within a single clock cycle does not become easier as the size of the circuit grows.

addition to fault attacks. (As in [16], and as discussed above, we need to consider a limited number of probes in each clock cycle.)

1.4 Related Work

As noted above, negative results for program obfuscation [3] rule out the possibility of defeating adversaries who can observe *all* values propagating through the circuit, for all circuits. This observation motivated the study of "private circuits", withstanding a *limited* number of such probing attacks [16]. The results of [16] do not consider active faults of the type considered here, yet are used as an essential building block in our main constructions.

A more general study of security against passive attacks was taken in [22] under an elegant framework of "physically observable cryptography". In contrast to [16], the focus of [22] is on obtaining model-independent *reductions* between physically secure primitives rather than implement them with respect to a specific attack model.

Most relevant to our work is the work of Gennaro et al. [12], who considered the problem of achieving security when an adversary can tamper with hardware. In contrast to the current work, they make the (seemingly necessary) assumption that there are parts of the circuitry that are *totally tamper-proof*. Indeed, as discussed above, the typical use in [12] is to have a signature stored in memory that is verified by the tamper-proof hardware. We stress that in our model, *no part of the circuitry* is free from tampering. In particular, all wires and internal memory cells can be affected. Thus, if an approach like the above is attempted, the adversary can tamper with the signature-checking portion of the circuitry (e.g., permanently fixing the output bit of the signature checker to indicate success). To the best of our knowledge, our work is the first that allows every portion of the hardware to be tampered with, at the level of individual wires between logical gates.

We note that [12] consider a more general type of tampering attack, albeit in a more restricted setting, in which the adversary can apply an arbitrary polynomial-time computable function to the contents of the memory. Defending against this general type of attacks is, in general, impossible in our setting (where no wire is free from tampering). Indeed, if the attacker could simply set the value of a wire to some arbitrary function of the other wires, then the impossibility result based on program obfuscation [3] would still hold.

Finally, it is instructive to contrast the positive results we achieve with a negative result from [12]. In the model of [12] it is shown that an attacker can recover the secret information stored in the memory, say a signature key, by sequentially setting or resetting bits of the memory and observing the effects of these changes on the output. Our model gets around this impossibility by allowing to feed values back into the memory. This form of feedback, which prevails in real-world computing devices, is essential for realizing the strong notion of privacy considered in this work.

1.5 Future Work

In this work we initiate the study of a fascinating question — can a circuit keep a secret even when *all* parts of the circuit are open to tampering? We give the first positive results, for an unbounded number of individual wire faults to any set of wires in the circuit. We believe the theory of private circuits, and private cryptographic implementations more generally, is still in its infancy and there are many more questions to address. Most notably, what other fault models allow for general positive results? As discussed above, negative results on obfuscation [3] give rise to severe restrictions on such fault models.

2 Preliminaries

Physical Model. We consider clocked circuits with memory gates. Specifically, our model is as follows:

– A memory gate has one input wire and one output wire: in each clock cycle, the output value of the memory gate becomes the input value from the previous clock cycle. The memory can be initialized with some data, which gets updated in each clock cycle. We shall denote a circuit C initialized with data D by $C[D]$.
– In addition to the memory gates, the circuit can have AND, OR and NOT gates, as well as input wires and output wires.
– The adversary can set each input wire to 0 or 1, and can read output wires.
– The adversary can also cause *faults* in the circuit. We consider the following kinds of faults: (1) setting a wire to 1 (which we call a "set" attack), (2) setting a wire to 0 (which we call a "reset" attack), or (3) toggling the value on a wire.
– We assume that wires are conducting: that is, with a single fault on a wire the adversary simultaneously causes faults everywhere that wire goes. In our construction in Section 5 we use NOT gates which are reversible (see e.g. [33]), so that faults on the output side of a NOT gate propagate to the input side. For AND and OR gates (as well as NOT gates in the construction in Section 4), faults can be introduced on input and output wires independently of each other.

Circuit Transformations. We shall refer to transformations which take a (circuit, data) pair to another (circuit, data) pair. It will always be the case that these are two separate transformations carried out independently of each other, one for the circuit and one for the data. However, for convenience and brevity we shall use a single transformation to denote these two transformations.

Definition 1 *A transformation between (circuit, data) pairs $T^{(k)}$ is called* functionality preserving *if for any pair (C, D), if $T^{(k)}(C, D) \mapsto (C_1, D_1)$ then $C[D]$ and $C_1[D_1]$ have the same input-output behavior.*

ISW Transformation and Security Definition. The starting point for our constructions is a transformation $T^{(k)}_{\text{ISW}}$ from [16]. The transformation yields a circuit which uses standard gates and some randomness gates (which output fresh random bits in each clock cycle). $T_{\text{ISW}}^{(k)}$ ensures that *reading* (but not tampering with) "a few wires" of the circuit in each clock cycle does not leak any information about the initial data in the memory (beyond what the output reveals). This is achieved using a (proactive) secret-sharing scheme, which shares each bit among k or more wires. Here we will not need any particulars of that construction, beyond the properties summarized below.

$T_{\text{ISW}}^{(k)}(C, D) \mapsto (C', D')$, is a functionality preserving transformation where each wire in C' is assigned at most two indices from $[k]$. To define the security guarantees of $T_{\text{ISW}}^{(k)}$ we define two adversaries: an "ideal" adversary which has only black-box access to C and a "real" adversary which can probe the internals of C'. For future reference we define these classes of adversaries $\mathcal{A}_{\text{IDEAL}}$ and $\mathcal{A}_{\text{ISW}}^{(k)}$ more formally, below.

- If $\mathcal{A} \in \mathcal{A}_{\text{IDEAL}}$ is given a circuit $C[D]$, then in every clock cycle $\mathcal{A}$ can feed inputs to the circuit and observe the outputs. This kind of access to the circuit is considered legitimate (ideal).
- If $\mathcal{A} \in \mathcal{A}_{\text{ISW}}^{(k)}$ is given a circuit $C'[D']$ with wires indexed from $[k]$, then in each cycle it can feed inputs to the circuit, read the outputs *and* probe wires in the circuit such that no more than $k-1$ indices are covered by the wires probed in that clock cycle.[3]

Without loss of generality, all adversaries are considered to output a single bit at the end of the interaction with the circuit.

Lemma 1 (Properties of the ISW Transformation). *[16] There exists a functionality preserving transformation $T_{\text{ISW}}^{(k)}(C, D) \mapsto (C', D')$, where C' uses AND gates, XOR gates, NOT gates and "randomness gates," and each wire is assigned at most two indices from $[k]$, such that the following hold:*

1. *Values on any $k-1$ wires in C' (excluding wires in the input and output phases), such that no two wires share an index, are distributed so that the following condition holds (distribution being as determined by the distribution of the outputs of the randomness gates during that clock cycle): any bit, even conditioned on all other $k-2$ bits and all other information obtained by any $\mathcal{A}' \in \mathcal{A}_{\text{ISW}}^{(k)}$ in previous clock cycles, has entropy at least c for a fixed $c > 0$.*
2. *$\forall C$, $\exists \mathcal{S}_{\text{ISW}}$ (a universal simulator), such that $\forall D$, $\forall \mathcal{A}' \in \mathcal{A}_{\text{ISW}}^{(k)}$, we have $\mathcal{S}' = \mathcal{S}_{\text{ISW}}^{\mathcal{A}'} \in \mathcal{A}_{\text{IDEAL}}$, and $\mathcal{S}'$ after interacting with $C[D]$ outputs 1 with almost the same probability as $\mathcal{A}'$ outputs 1 after interacting with C' $[D']$ (the difference in probabilities being negligible in the security parameter k).*

We remark that in [16] these properties are not explicitly stated in this form. In particular in the second property above, [16] is interested only in restricting

[3] To be precise about the counting, we should consider the values on the wires that go into the memory at a clock cycle same as the values that come out of the memory at the next clock cycle. Thus probing one of these wires in one clock cycle counts towards probes in both clock cycles.

$\mathcal{A}'$ to probing at most $(k-1)/2$ wires. However to employ $T_{\mathrm{ISW}}^{(k)}$ within our transformations we shall use the fact that the construction allows $\mathcal{A}'$ to probe any number of wires as long as they cover at most $k-1$ indices.

3 Security When Circuits are Completely Tamperable

In the next two sections we present our constructions which do not require any untamperable components (except the topology of the circuit and the atomic gates (AND, NOT, OR)). The adversary is allowed to change the values in any of the wires in the circuit. We give constructions for two scenarios:

1. **Tamper-resistance against "reset" attacks:** In this case the only kind of faults that the adversary can introduce into the circuit are "resets." That is, it can change the value of any wire to zero (but not to one). In each clock cycle, the adversary can set the input values, reset any number of wires of its choice and observe the outputs. We call this class of adversaries $\mathcal{A}_{\mathrm{RESET}}$.
2. **Tamper-resistance against "set, reset and toggle" attacks:** Here the adversary is allowed to set or reset the wires. That is, it can change the value of any wire to one or zero. Also, it can toggle the value in a wire (if the value prior to attack is zero, change it to one, and vice versa). There is an *a priori* bound on the number of new wires it can attack (set, reset or toggle) at each clock cycle. However, it is allowed to introduce *persistent (or permanent) faults* to any wire it attacks (such a fault will not be counted as a new fault in every cycle). Hence, after multiple clock cycles, the adversary can potentially have faults in *all* the wires in the circuit simultaneously. We call this class of adversaries $\mathcal{A}_{\mathrm{TAMPER}}^{(t)}$, where t is the bound on the number of wires the adversary can attack in each clock cycle.

The two constructions use similar techniques. First we introduce our basic construction techniques and proof ideas for the reset-only case, and then explain the extensions used to make the construction work for the general case.

4 Tamper-Resistance against Reset Attacks

We present our construction as two transformations $T_1^{(k)}$ and $T_2^{(k)}$. The complete transformation consists of applying $T_1^{(k)}$ followed by $T_2^{(k)}$. The first transformation converts any given circuit to a private circuit which uses "encoded randomness gates" (which output fresh random bits in every cycle, but each bit of the output is encoded into a pair of bits as explained later). The second transformation converts the resulting circuit to a standard deterministic circuit (using only AND, NOT and OR gates), while preserving the security property. The formal security statements for $T_1^{(k)}$ and $T_2^{(k)}$ follow.

Lemma 2. *There is a polynomial time (in input size and security parameter k) functionality preserving transformation $T_1^{(k)}(C, D) \mapsto (C_1, D_1)$, where C_1 uses* *"encoded randomness gates," such that* $\forall C$, $\exists \mathcal{S}_1$ *(a universal simulator), such that* $\forall D$, $\forall \mathcal{A}_1 \in \mathcal{A}_{\text{RESET}}$, *we have* $\mathcal{S} = \mathcal{S}_1^{\mathcal{A}_1} \in \mathcal{A}_{\text{IDEAL}}$ *and the following two experiments output 1 one with almost the same probability (the difference in probabilities being negligible in the security parameter* k*):*

- Experiment A: $\mathcal{S}$ *outputs a bit after interacting with* $C[D]$.
- Experiment B: $\mathcal{A}_1$ *outputs a bit after interacting with* $C_1[D_1]$.

Lemma 3. *There is a polynomial time (in input size and security parameter* k*) functionality preserving transformation* $T_2^{(k)}(C_1, D_1) \mapsto (C_2, D_2)$*, where* C_1 *may use encoded randomness gates, such that* $\forall C_1$, $\exists \mathcal{S}_2$ *(a universal simulator), such that* $\forall D_1$, $\forall \mathcal{A} \in \mathcal{A}_{\text{RESET}}$, *we have* $\mathcal{A}_1 = \mathcal{S}_2^{\mathcal{A}} \in \mathcal{A}_{\text{RESET}}$ *and the following two experiments output 1 with almost the same probability (the difference in probabilities being negligible in the security parameter* k*):*

- Experiment B: $\mathcal{A}_1$ *outputs a bit after interacting with* $C_1[D_1]$.
- Experiment C: $\mathcal{A}$ *outputs a bit after interacting with* $C_2[D_2]$.

Theorem 1. *There is a polynomial time (in input size and security parameter* k*) functionality preserving transformation* $T_{\text{RESET}}^{(k)}(C, D) \mapsto (C_2, D_2)$*, such that* $\forall C$, $\exists \mathcal{S}_0$ *(a universal simulator), such that* $\forall D$, $\forall \mathcal{A} \in \mathcal{A}_{\text{RESET}}$, *we have* $\mathcal{S} = \mathcal{S}_0^{\mathcal{A}} \in \mathcal{A}_{\text{IDEAL}}$ *and experiment A and experiment C output 1 with almost the same probability (the difference in probabilities being negligible in the security parameter* k*).*

Proof. This follows from the above two lemmas, by setting $T_{\text{RESET}}^{(k)}(C, D) = T_2^{(k)}(T_1^{(k)}(C, D))$ and $\mathcal{S}_0^{\mathcal{A}} = \mathcal{S}_1^{\mathcal{S}_2^{\mathcal{A}}}$.

4.1 Proof of Lemma 2

As proof of Lemma 2 we first present the transformation $T_1^{(k)}$. We then will demonstrate a universal simulator as required in the Lemma and show the correctness of simulation.

The Transformation $T_1^{(k)}$ The transformation $T_1^{(k)}$ is carried out in two stages. In the first step, we apply the transformation $T_{\text{ISW}}^{(k)}$ from [16] to (C, D) to obtain (C', D').

Next we shall transform (C', D') further so that the following "encoding" gets applied to all the data: the bit 0 is mapped to a pair of bits 01 and the bit 1 is mapped to 10. We shall refer to this encoding as the *Manchester* encoding. Encoding D' to get D_1 is straight-forward: we simply replace 0 and 1 by 01 and 10 respectively (thereby doubling the size of the memory and doubling the number of wires connecting the memory to the circuit). C' is transformed to get C_1 as follows:

1. The input data is passed through a simple encoding gadget, which converts 0 to 01 and 1 to 10. The encoding simply involves fanning out the signal into two: the first output wire and an input to a NOT gate whose output is the second output wire.

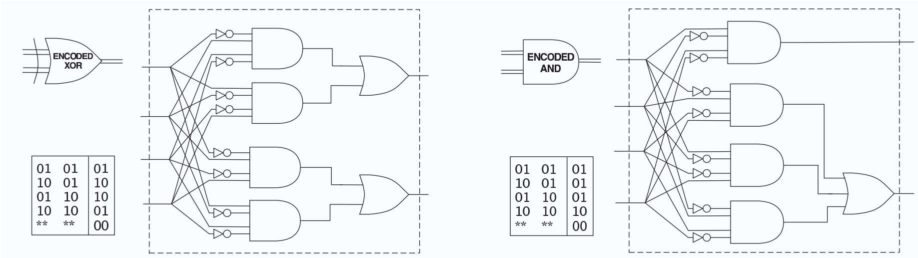

Fig. 1. XOR and AND Gadgets used by $T_1^{(k)}$. Note that the outputs of the gadgets are implemented as OR of ANDs of input wires and their NOTs. It is important that the gadgets do not have NOT gates except at the input side (to maintain the invariant that the encoding 11 does not appear in the circuit).

2. The "core" of the circuit C_1 is derived from C' as follows: every wire in C' is replaced by a pair of wires. Then the input wire pairs are connected to the outputs from the encoding gates (described above), and the output wire pairs are fed into the decoding phase (below). The gates to which the wires are connected are modified as follows:
   (a) Each randomness gate is replaced by an encoded randomness gate.
   (b) XOR and AND gates in C' are replaced by the gadgets shown in Figure 1. NOT gates are replaced by a gadget which simply swaps the two wires in the pair.
3. An "error cascade" stage (described below) is added before the output stage (including the output from the circuit to the memory).
4. A simple decoding stage is added just before the final output wires (excluding the wires going into the memory): the decoding is done by simply ignoring the second wire in the encoding of each signal.

Error Cascading The circuit will be designed to "detect" reset attacks, and if an attack is detected, to erase all the data in the memory. (Such self-destruction is not *required* by Lemma 2, but it is a desirable property that is achieved by our construction.) The basic step in this is to ensure that if a detectable error is produced at some point in the circuit, it is propagated all the way to an "error cascading stage" (such an error propagation will be ensured by the gadgets in Figure 1). Then, the cascading stage will ensure that all the data in the memory and output is erased.

The detectable error referred to above is the invalid encoding 00 in a pair of wires corresponding to a single wire in C'. Recall that the only valid encodings are 01 and 10. We shall denote the encoding 00 by the symbolic value $\perp$. (We will show that the transformed circuit is so designed that the other invalid encoding, namely 11, will never occur in the circuit, even after any number of reset attacks.) As sketched in Figure 2, the error cascading phase is built using small cascade gadgets which take two encoded inputs and convert them both to $\perp$ if either of them is $\perp$. (These gadgets are implemented similar to the gadgets in Figure 1,

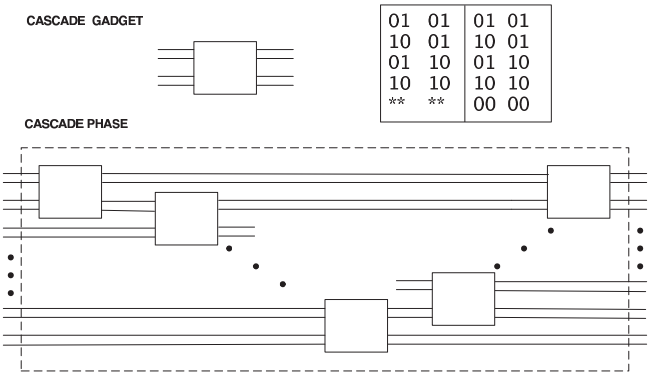

Fig. 2. The error-cascade phase and the truth table of the cascade gadget used by $T_1^{(k)}$.

with NOT gates only at the input side.) It is easy to see that if any of the input pairs has the value $\perp$, then after the cascade phase all output pairs will encode $\perp$.

All the wire pairs going to the output or back to the memory are passed through the cascade phase. In addition, for simplicity, we shall have all the wire pairs coming from the input and from the memory also go into the cascade phase. This will ensure that if a $\perp$ value appears anywhere in the memory or input, the entire memory is erased in the next round (even if the wire pair where $\perp$ originally appears does not influence the output or memory otherwise).

Note that in Figure 2 the inputs to the cascade stage are ordered top to bottom. The wire pairs corresponding to the output signals are fed into the cascade phase first and the other signals are fed into the cascade phase below them. This will ensure that even if a $\perp$ is introduced *within* the cascade phase, but in one of the wires going into the memory, the entire output is erased in the same clock cycle (and the memory contents will get erased in the next clock cycle). We shall use the convention that the wires within the cascade phase, except those corresponding to the output wires, are considered as part of the core of the circuit.

The Simulator for $T_1^{(k)}$ The universal simulator $\mathcal{S}_1$ is constructed as follows: $\mathcal{S}_1$ runs $\mathcal{A}_1$, simulating to it $C_1[D_1]$. This means $\mathcal{A}_1$ can request a particular input value for C_1, and expect to be provided the output from C_1 for that input. Further $\mathcal{A}_1$ can request that a particular reset attack be applied to C_1. The output of the simulated C_1 that $\mathcal{S}_1$ provides to $\mathcal{A}_1$ should be indistinguishable from what $\mathcal{A}_1$ would have seen had it been actually working with $C_1[D_1]$. However, $\mathcal{S}_1$ has to do this with only black-box access to (the functionally equivalent) $C[D]$.

$\mathcal{S}_1$ internally runs the simulator $\mathcal{S}_{\text{ISW}}$ corresponding to the transformation $T_{\text{ISW}}^{(k)}$ which takes C to C'. It gives $\mathcal{S}_{\text{ISW}}$ blackbox access to C, which can then simulate (non-blackbox, probing) access to C'. So $\mathcal{S}_1$ can work as if it had (probing) access to C'. It then requests $\mathcal{S}_{\text{ISW}}$ to set the inputs to (the simulated) C'

to the same as the input $\mathcal{A}_1$ requests for C_1. It provides $\mathcal{A}_1$ with the output values of the simulated C_1 which are the same as that given by $\mathcal{S}_{\text{ISW}}$. We need to describe how the reset attacks launched by $\mathcal{A}_1$ on C_1 are translated by $\mathcal{S}_1$ to probing attacks on C'.

Recall that C_1 is composed of an encoding of C', and a few extra phases (input encoding, error cascading and output decoding phases). $\mathcal{S}_1$ maintains a flag called `destroyed` which is initially set to false. While the flag remains set to false, at each clock cycle, $\mathcal{S}_1$ receives an attack pattern (i.e., a collection of reset attacks) from $\mathcal{A}_1$ and applies it to the simulated circuit as follows.

1. *For attacked wires that belong to the core of the circuit:* $\mathcal{S}_1$ will determine a set of indices $I \subset [k]$, as follows. I is initialized to the empty set. Then each attacked wire is considered as below:
   (a) *If the attacked wire is outside of the gadgets in Figure 1:* Then the wire is one of a wire pair in C_1 which encodes a single a wire in C'. Recall that all the wires in C' are indexed by values in $[k]$. $\mathcal{S}_1$ will add the index of the wire corresponding to the wire that is reset, to I. (If the wire has two index values, both are added to I.)[4]
   (b) *If the attacked wire is inside a gadget in Figure 1:* The gadget corresponds to a gate (AND, XOR or NOT) in C'. For each input wire to this gate in C', $\mathcal{S}_1$ checks if that wire has an *influence* on whether the attack creates a $\perp$ value or not. A wire is said to have such an influence if there are two settings of the values of the input to the gate such that in one a $\perp$ value is created and in the other it is not. $\mathcal{S}_1$ adds the indices of the wires with influence to I.

   Once I is determined by considering all attacks in the core of the circuit, $\mathcal{S}_1$ (acting as the adversary $\mathcal{A}'$) will make probes into the circuit simulated by $\mathcal{S}_{\text{ISW}}$ on all wires with indices in I. From the values obtained from this probe, it can check if the reset attacks produce a $\perp$. If they do, then $\mathcal{S}_1$ will set the flag `destroyed`.

   Note that $\mathcal{S}_{\text{ISW}}$ allows I to have at most $k-1$ indices. If I is of size k (i.e., $I = [k]$), then $\mathcal{S}_1$ sets the flag `destroyed` (without querying $\mathcal{S}_{\text{ISW}}$).
2. *For attacked wires that belong to the encoding phase:* Such a wire is an input to an encoding gate (the outputs from an encoding gate are considered part of the core of the circuit), or equivalently an input to the simulated C_1. In this case the corresponding input to C' is set to zero.
3. *For attacked wires that belong to the cascade phase:* The cascade phase consists of pairs of wire which correspond to wires going to the output phase in C'. (The wires in the cascade phase that correspond to the wires going to the memory are considered part of the core of the circuit). $\mathcal{S}_1$ obtains the values on these from the simulated C_1 and determines how the attack affects the output from the cascade phase.

[4] C' has input encoding and output decoding phases. Recall that the wires in these phases are not indexed by values in $[k]$ and the Manchester encodings of these wires are not considered to belong to the core of the transformed circuit.

4. *For attacked wires is in the decode phase:* A reset attack on the second wire in a pair does not have any influence on the output while a reset attack on the first wire causes the corresponding output to be reset.

Once the flag `destroyed` is set $\mathcal{S}_1$ simply produces the all zero output in every round.

The Proof We start by observing what reset attacks can do in C_1. An invariant maintained in C_1 is that no wire pair carries the value 11: this is true for the data in the memory and also for the inputs coming out of the encoding stage; a reset attack on a wire which is 00, 10 or 01 cannot generate 11; further each gadget ensures that the invariant is maintained from inputs to outputs even when the internal wires of the gadget are attacked. (This follows from an analysis of the gadgets of the form of "OR of ANDs of signals and their NOTs.") Not having 11 in the wires has the following consequences:

- *Impossibility of changing a signal to a non-$\perp$ value:* Reset attacks can either leave a wire pair unchanged, or convert it to a $\perp$, but not generate a new non-$\perp$ value.
- $\perp$ *Propagation and Self-destruction:* The gadgets shown in Figure 1 are "$\perp$-propagating." That is, if any input wire pair encodes $\perp$ the output will be $\perp$ too. Thus any $\perp$ introduced by an attack in the core of the circuit will reach the cascade stage, which will ensure that even a single $\perp$ will result in the entire memory being erased and the outputs zeroed out.
  Thus, the output of the circuit will either be correct (or contain resets introduced after the cascade stage), or will be all zeroes. If a $\perp$ is introduced it will result in the entire memory being erased as well. If the $\perp$ is introduced after the cascade phase, this will happen in the next round.

Now we turn to the simulation by $\mathcal{S}_1$. $\mathcal{S}_1$ simply uses $\mathcal{S}_{\text{ISW}}$ to get the outputs and also to check if a $\perp$ is created by the resets.

First, suppose that the set of indices I determined by $\mathcal{S}_1$ is of size at most $k-1$ in each round. In this case we observe the simulation by $\mathcal{S}_1$ is perfect. This is because, in the simulation, C_1 as simulated by $\mathcal{S}_1$ can be considered to encode the values in C' as simulated by $\mathcal{S}_{\text{ISW}}$. Since in this case for each reset $\mathcal{S}_{\text{ISW}}$ allows all the indices to be queried, $\mathcal{S}_1$ can precisely determine if a $\perp$ value is created or not in the core of C_1. When $\perp$ is not created, the output is simply the correct output (with any modifications caused by reset attacks in or after the cascade phase). When $\perp$ is created in the core, the output will be all zeroes in all subsequent clock cycles. Note that if a $\perp$ is created in the cascade phase in a signal going to the memory (which is considered part of the core), though the output is zeroed out in the same clock cycle, the memory may be zeroed out only in the next clock cycle.

Now we consider the case when $I = [k]$ in some round. $\mathcal{S}_1$ sets the flag `destroyed` but it is possible that in C_1 corresponding to C' as simulated by $\mathcal{S}_{\text{ISW}}$, $\perp$ is not produced. However the probability of the latter happening is negligible in k. To see this, note that in building I, whenever a reset attack causes new indices to be added to I, there is a constant probability (independent of values of wires of indices already added to I) that a $\perp$ is produced by that attack. Further for each attack at most four indices are added (at most two inputs to a gate, each with at most two indices). Thus having added indices for $\Omega(k)$ attacks, the probability that none of the attacks produce a $\perp$ is exponentially small in k.

Thus in either case the simulation is good.

4.2 Proof of Lemma 3

The transformation $T_2^{(k)}$ removes the "encoded randomness gates" from a circuit. If $(C_2, D_2) = T_2^{(k)}(C_1, D_1)$ we need to show that an adversary $\mathcal{A}$ cannot gain any advantage in Experiment C in Lemma 3 than it will when employed by a simulator $\mathcal{S}_2$ in Experiment B.

The plan is to replace the encoded randomness gates with some sort of a pseudorandom generator (PRG) circuit, with an initial seed built into the memory. However, since the PRG circuit itself is open to attack from the adversary, it needs to be somehow protected. First we introduce a transformation which gives a weak protection. Then we show how multiple PRG units protected by such a transformation can be put together to obtain a PRG implementation which will also be secure against the reset attacks.

Lemma 4. Weak Protection Against Reset Attacks: *There is a polynomial time (in input size and security parameter k) transformation $T_{\text{WEAK}}^{(k)}(C_P, D_P) \mapsto (C_Q, D_Q)$, such that the following properties hold for all C_P and D_P:*

- *$C_Q[D_Q]$ is functionally equivalent to $C_P[D_P]$, except that the output of C_Q is Manchester encoded.*
- *Consider any adversary $\mathcal{A} \in \mathcal{A}_{\text{RESET}}$ interacting with $C_Q[D_Q]$. If it resets even one wire inside C_Q (not an input or output wire), with probability at least q (for some constant $q > 0$), at least one of the output signals of C_Q becomes $\perp$.*

$T_{\text{WEAK}}^{(k)}$ differs from $T_1^{(k)}$ in that the resulting circuit (C_Q, above) does not contain any encoded randomness gates. It is just a conventional deterministic circuit. On the other hand, the guarantee given by the transformation is much weaker: it guarantees introducing a $\perp$ into the output only with some positive probability.

The basic idea behind the construction is to randomize all the signals in the circuit, so that a reset attack has a constant probability of causing a $\perp$. The construction is essentially the same as $T_1^{(2)}$ (i.e., with security parameter 2), but without using randomness gates. Instead we use built-in randomness (i.e., it is stored in the memory). This will be sufficient to guarantee that the first time the circuit is attacked, there is a constant probability of producing a $\perp$. Also for this transformation we do not need the cascade stage and the output decoding stage of $T_1^{(2)}$.

Transformation $T_2^{(k)}$. Now we are ready to describe $T_2^{(k)}$. Suppose the input circuit requires n encoded random bits. Let C_P be a PRG circuit, which at every round, outputs n freshly generated pseudorandom bits, as well as refreshes its seed (kept in the memory). Consider k such circuits $C_P[D_P^i]$, D_P^i being a uniformly and independently drawn seed for the PRG. Let $(C_Q, D_Q^i) = T_{\text{WEAK}}^{(k)}(C_P, D_P^i)$. $T_2^{(k)}$ replaces the collection of all n encoded randomness gates by the following: the outputs of $C_Q[D_Q^i]$ $(i = 1, \ldots, k)$, are XOR-ed together using $k-1$ encoded XOR gadgets (from Figure 1).

The proof that the above transformation indeed satisfies the properties required in Lemma 3 is included in the full version of this paper. The proof depends on the fact that as long as the adversary has attacked fewer than k of $C_Q[D_Q^i]$ in C_2, a careful simulation can reproduce the effect of this attack in C_1. On the other hand, if the adversary attacks all k of $C_Q[D_Q^i]$, then due to constant probability of each attack resulting in a $\perp$, except with negligible probability at least one $\perp$ value will indeed be generated which will propagate to the cascade stage and the output of the circuit (and hence can be easily simulated).

5 General Attacks on Wires

Next we turn to more general attacks in which the adversary can set the values in the wires to 1 or 0, as well as toggle the values in the wires. We shall impose a bound on the number of wires it can attack at each cycle, but allow the attacks to be persistent. That is, the wires set or reset in any one cycle are stuck at that value until explicitly released by the adversary; similarly toggled wires retain the toggling fault until released. There is no limit on the number of wires the adversary can release at any cycle.

Theorem 2. *There is a polynomial time (in input size and security parameter k) functionality preserving transformation $T_{\text{FULL}}^{(k)}(C, D) \mapsto (C^*, D^*)$, such that $\forall C$, $\exists \mathcal{S}_0$ (a universal simulator), such that $\forall D$, $\forall \mathcal{A} \in \mathcal{A}_{\text{TAMPER}}^{(t)}$, we have $\mathcal{S}_0^{\mathcal{A}} \in \mathcal{A}_{\text{IDEAL}}$ and the following two experiments output 1 with almost the same probability (the difference in probabilities being negligible in the security parameter k):*

- *Experiment A:* $\mathcal{S}_0^{\mathcal{A}}$ *outputs a bit after interacting with C.*
- *Experiment B:* $\mathcal{A}$ *outputs a bit after interacting with C^*.*

5.1 Proof Sketch of Theorem 2

The construction of $T_{\text{FULL}}^{(k)}$, the simulation and proof of simulation roughly follow that in the reset-only case. The construction first applies the transformation $T_{\text{ISW}}^{(k)}$, then changes the circuit to use some sort of encoding for each bit, adds an error cascade stage, and finally replaces all encoded randomness gates by a psuedo-randomness generator (PRG) circuit.

In the construction for reset attacks, it was crucial that the adversary cannot set a wire to 1, thereby being unable to change an encoded wire pair to anything but $\perp$. Here, however, the adversary is allowed to set as well as reset the wires. Nevertheless, using the fact that it can launch only t attacks per cycle, and using a longer encoding (instead of using a pair of wires) to encode each bit, we can ensure that if the adversary attacks any encoding, it will either leave it unchanged or change it to an invalid encoding. Below we sketch the details of this.

0^{2kt}	0^{2kt}	0^{2kt}
0^{2kt}	1^{2kt}	0^{2kt}
1^{2kt}	0^{2kt}	0^{2kt}
1^{2kt}	1^{2kt}	1^{2kt}
*	*	$0^{kt}1^{kt}$

AND Gadget

0^{2kt}	0^{2kt}	0^{2kt}
0^{2kt}	1^{2kt}	1^{2kt}
1^{2kt}	0^{2kt}	1^{2kt}
1^{2kt}	1^{2kt}	0^{2kt}
*	*	$0^{kt}1^{kt}$

XOR Gadget

0^{2kt}	1^{2kt}
1^{2kt}	0^{2kt}
*	$0^{kt}1^{kt}$

NOT Gadget

0^{2kt}	0^{2kt}	0^{2kt}	0^{2kt}
0^{2kt}	1^{2kt}	0^{2kt}	1^{2kt}
1^{2kt}	0^{2kt}	1^{2kt}	0^{2kt}
1^{2kt}	1^{2kt}	1^{2kt}	1^{2kt}
*	*	$0^{kt}1^{kt}$	$0^{kt}1^{kt}$

Cascade Gadget

Fig. 3. Truthtables for the gadgets used by $T^{(k)}_{\text{FULL}}$. The gadgets can be implemented using atomic gates: AND, OR and NOT gates. The AND gates used have $4kt$ input wires and the NOT gates are reversible (see below).

Encoding: Each bit is encoded by $2kt$ wires, where k is the security parameter and t is the bound on the number of attacks that the adversary can make per cycle. 0 is encoded as 0^{2kt} and 1 as 1^{2kt}. All other values are invalid ($\perp$); a special value $\perp^*$ is defined as $0^{kt}1^{kt}$.
Transformation: First $T^{(k)}_{\text{ISW}}$ is applied to get a circuit using randomness gates. Then the circuit is modified to use the encoded values. The core of the circuit is derived by replacing each wire by $2kt$ wires, and each of the atomic gates (AND, XOR and NOT) by gates shown in Figure 3. Input encoding and output decoding are straightforward. The error cascading stage is the same as shown in Figure 2, but using the cascade gadget from Figure 3. In implementing these gadgets, each bit of the output is generated by a circuit of the form OR of ANDs of input wires or their NOTs, or NOT of such a circuit. The AND gates involved have $4kt$ input wires. (These AND gates are the only components in this work which are not of constant size.) Note that to keep the circuit size polynomial it is important to allow the output to be of the form NOT of OR of ANDs, as well as OR of ANDs. In contrast, in the reset-only case it was important that the gadgets did not have NOT gates except at the input side. However, with

general faults such a restriction is not helpful, nor used. The NOT gates used are reversible, so that faults on the output side of a NOT gate propagate to the input side. (This has the effect that NOT gates appearing immediately before the input to the AND gates can be considered part of the atomic AND gates.)

Finally, the encoded randomness gates can be replaced by a PRG circuit similar to that in Section 4.2.

Simulation: We sketch the simulator $\mathcal{S}_1$ (analogous to the simulator described in Section 4.1) for the part of the transformation before replacing the randomness gates by the PRG circuit. (The details of the simulation for the latter part can be found in the full version.) As in Section 4.1, $\mathcal{S}_1$ will internally run the simulator $\mathcal{S}_{\text{ISW}}$ corresponding to the transformation $T^{(k)}_{\text{ISW}}$ and translates attacks on C_1 to probing attacks on C'. However now the attacks are not just reset attacks but set, reset or toggle attacks. Further, each wire in C' is represented by a bundle of $2kt$ wires in C_1 (instead of just a pair of wires). $\mathcal{S}_1$ maintains the flag `destroyed` and calculates the set of indices I as before. The one additional action now is that *at any clock cycle if kt or more wires in any bundle are subject to attack, then the flag* `destroyed` *is set*. Here, attacking a wire inside any of the gadgets of Figure 3 (i.e., output of any of the AND or OR gates inside a gadget) is considered as an attack on the unique wire in the output of the gadget affected by the attack. Another difference is that, even after the flag `destroyed` is set, the simulator here continues to output non-zero values, but these values can be determined just from the attacks (in the cascade and output phases).

To analyze this new simulator we follow the same arguments as in Section 4.1, but with the following modifications.

- *Impossibility of changing a signal to a non-$\perp$ value:* We claim that as long as the flag `destroyed` is not set, an attack can either leave the values in a bundle corresponding to a signal in C' unchanged or convert it to a $\perp$. To see this note that to produce a new non-$\perp$ signal the adversary must have attacked at least kt wires in a bundle. (These attacks may be direct attacks on the wires or attacks inside a gadget from which the wire emanates.) This is because the minimum distance between the signals that occur in an unattacked circuit (namely $0^{2kt}, 1^{2kt}$ and $\perp^* = 0^{kt}1^{kt}$), and each valid signal is kt. But when the adversary attacks kt or more wires in a single bundle (directly or by attacking a wire inside a gadget), then the simulator sets the flag `destroyed`.
- $\perp$ *Propagation and Self-destruction:* If a $\perp$ (an encoding which is not 0^{2kt} or 1^{2kt}) is produced in any input signal to (the gadget corresponding to) a gate, it results in a $\perp^*$ being generated by the gate. Since $\perp^*$ is far from a valid encoding, the adversary cannot change it to a valid signal in the same cycle. So the $\perp$ value will reach the cascade stage, which ensures that all information in the circuit is lost. (If the $\perp$ is introduced after the cascade phase, this will happen in the next round.)

Now to prove that the simulation is good, first we observe that the probability that `destroyed` is set due to at least kt wires in a bundle being under attack is negligible. This is because at any cycle, the adversary can set/reset at most t wires, and so it will need at least k cycles to attack kt wires in a bundle. But during these cycles if the signal value in that bundle changes, then a $\perp$ is necessarily produced (resulting in the flag `destroyed` being set). Since each signal value is randomized by $T_{\mathrm{ISW}}^{(k)}$ (Lemma 1), except with probability $2^{-\Omega(k)}$ this will indeed happen. The argument extends to the case when some of the attacks are on wires inside the gadgets as well, by observing that all the internal wires have influence on the output of the gadget, and the randomization ensures that with constant probability the input signals to the gadget will take values causing the attack to influence the output wire of the gadget.Here by internal wires in a gadget we refer to the outputs of the AND gates used in the gadgets; the only other wires inside a gadget are all connected to wires external to the gadget either directly or through a *reversible* NOT gate, and as such are accounted for by attacks on the wires external to the gadget. (This is where we require that the NOT gates be reversible; attacks on either side of a NOT gate propagates to the other side as well.)

Given this, the rest of the analysis of this simulator follows that in Section 5.1.

References

1. R. Anderson, M. Kuhn, "Tamper Resistance—A Cautionary Note," *USENIX E-Commerce Workshop*, USENIX Press, 1996, pp.1–11.
2. R. Anderson, M. Kuhn, "Soft Tempest: Hidden Data Transmission Using Electromagnetic Emanations," *Proc. 2nd Workshop on Information Hiding*, Springer, 1998.
3. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. CRYPTO 2001, 2001.
4. M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. of 20th STOC*, 1988.
5. E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," *CRYPTO '97*.
6. D. Boneh, R.A. Demillo, R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *EUROCRYPT'97*, Springer-Verlag, 1997, pp.37–51.
7. S. Chari, C.S. Jutla, J.R. Rao, P. Rohatgi, "Towards Sound Approaches to Counteract Power-Analysis Attacks," *CRYPTO'99*, Springer-Verlag, 1999, pp.398–412.
8. D. Chaum, C. Crepeau, and I. Damgård. Multiparty unconditional secure protocols. In *Proc. of 20th STOC*, 1988.
9. J.-S. Coron, L. Goubin, "On Boolean and Arithmetic Masking against Differential Power Analysis," *CHES'00*, Springer-Verlag, pp.231–237.
10. J. Daemen, V. Rijmen, "Resistance Against Implementation Attacks: A Comparative Study of the AES Proposals," *AES'99*, Mar. 1999.
11. K. Gandolfi, C. Mourtel, F. Olivier, "Electromagnetic Analysis: Concrete Results," *CHES'01*, LNCS 2162, Springer-Verlag, 2001.
12. R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, and T. Rabin. Algorithmic Tamper-Proof (ATP) Security: Theoretical Foundations for Security against Hardware Tampering. Proceedings of *Theory of Cryptography Conference,* 2004.
13. O. Goldreich. *Foundations of Cryptography: Basic Applications.* Cambridge University Press, 2004.

14. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game (extended abstract). In *Proc. of 19th STOC*, 1987.
15. L. Goubin, J. Patarin, "DES and Differential Power Analysis—The Duplication Method," *CHES'99*, Springer-Verlag, 1999, pp.158–172.
16. Y. Ishai, A. Sahai, and D. Wagner, "Private Circuits: Protecting Hardware against Probing Attacks," *Proceedings of Crypto '03*, pages 462-479, 2003.
17. D. Kahn, *The Codebreakers*, The MacMillan Company, 1967.
18. J. Kelsey, B. Schneier, D. Wagner, "Side Channel Cryptanalysis of Product Ciphers," *ESORICS'98*, LNCS 1485, Springer-Verlag, 1998.
19. P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," *CRYPTO'96*, Springer-Verlag, 1996, pp.104–113.
20. P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis," *CRYPTO'99*, Springer-Verlag, 1999, pp.388–397.
21. T.S. Messerges, "Securing the AES Finalists Against Power Analysis Attacks," *FSE'00*, Springer-Verlag, 2000.
22. S. Micali and L. Reyzin. Physically Observable Cryptography. In *Proc. of TCC '04*, pages 278-286, 2004.
23. R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks (Extended Abstract). In *Proc. of PODC '91*, pages 51-59, 1991.
24. D. Page, "Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel," Tech. report CSTR-02-003, Computer Science Dept., Univ. of Bristol, June 2002.
25. B. Pfitzmann, M. Schunter and M. Waidner, "Secure Reactive Systems", IBM Technical report RZ 3206 (93252), May 2000.
26. N. Pippenger, "On Networks of Noisy Gates," in *Proc. of FOCS '85*, pages 30-38.
27. J.-J. Quisquater, D. Samyde, "Eddy current for Magnetic Analysis with Active Sensor," *Esmart 2002*, Sept. 2002.
28. J.-J. Quisquater, D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards," *Esmart 2001*, LNCS 2140, Springer-Verlag, 2001.
29. J.R. Rao, P. Rohatgi, "EMpowering Side-Channel Attacks," IACR ePrint 2001/037.
30. US Air Force, *Air Force Systems Security Memorandum 7011—Emission Security Countermeasures Review*, May 1, 1998.
31. W. van Eck, "Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk," *Computers & Security*, v.4, 1985, pp.269–286.
32. D. Wright, *Spycatcher*, Viking Penguin Inc., 1987.
33. S.G. Younis and T. F. Knight, Jr. Asymptotically Zero Energy Split-Level Charge Recovery Logic. *Proceedings of 1994 International Workshop on Low Power Design*, Napa, CA, 1994.