Algorithmic Tamper-Proof Security under Probing Attacks

Feng-Hao Liu* and Anna Lysyanskaya*

Department of Computer Science, Brown University
{fenghao,anna}@cs.brown.edu

Abstract. Gennaro et al. initiated the study of algorithmic tamper proof (ATP) cryptography: cryptographic hardware that remains secure even in the presence of an adversary who can tamper with the memory content of a hardware device. In this paper, we solve an open problem stated in their paper, and also consider whether a device can be secured against an adversary who can both tamper with its memory and probe a few memory locations or wires at a time. Our results are as follows:

- It is impossible to realize a secure cryptographic functionality with a personal identification number (PIN) where a user is allowed to make up to ℓ incorrect consecutive attempts to enter her PIN, with no total limit on incorrect PIN attempts. (This was left as an open problem by Gennaro et al.)
- It is impossible to secure a deterministic cryptographic device against an adversary who is allowed to both tamper with the memory of the device and probe a memory location; it is also essentially infeasible to secure it if the adversary's probing power is restricted to internal wires; it is impossible to secure it against an adversary whose probing power is restricted to internal wires, but who is also allowed to tamper with a few internal wires.
- By extending the results of Ishai et al., we show that a cryptographic device with a true source of randomness can withstand tampering and limited probing attacks at the same time.

1 Introduction

In cryptography, we typically assume that an adversary launching an attack can neither probe bits of a secret key, nor tamper with it; the adversary may only obtain input/output (i.e. black-box) access to a functionality it is trying to attack. However, in practice, adversaries may attack a cryptographic device through other means. For example, in a side-channel attack [AK96, AK97], an adversary can measure the power consumption of the device [KJJ99, CRR03], timing of operations [Koc96], electromagnetic radiation [AARR03], etc. Additionally, an adversary may tamper with the device's memory [BS97] or circuitry [SA03] and check the effect this might have on the device's computation.

There are several lines of work that place these attacks on theoretical foundations. Gennaro et al. [GLM+04] defined security of a cryptographic functionality against an

* Supported by NSF grant CNS-0347661 and CNS-0831293.

J.A. Garay and R. De Prisco (Eds.): SCN 2010, LNCS 6280, pp. 106–120, 2010.

adversary who can tamper with the contents of a device's memory, and showed how to satisfy their definition. Ishai et al. [ISW03], in contrast, defined and realized security for devices where an adversary can probe the memory of the device (more precisely, wires of its circuit), and even fix some of the wires in a circuit under attack.

In this paper, we examine security under a combination of attacks described in these previous papers. Intuitively, it would seem that it should be possible to combine the positive results of the two lines of work and design a device that would withstand an adversary who can both tamper with its memory and probe a few memory locations at a time. Surprisingly, we show that a cryptographic functionality cannot withstand such a combined attack, unless augmented with a true source of randomness. We give an adversary who, given the ability to only probe one memory location at a time, and the ability to tamper with the memory of a deterministic device, retrieves the entire secret content of the device's memory, even though the device may continuously update its secret content (so the trivial attack that just probes memory locations one by one would not work).

Related work. Gennaro et al. [GLM$^+$04] considered the adversary who, in addition to black-box access to a functionality he is trying to attack, also has the ability to tamper with the memory of this device. Their work was motivated by that of Biham and Shamir [BS97] who showed how to break cryptographic schemes by having the memory of a device modified in a certain way. Gennaro et al. gave a definition of algorithmic tamper-proof (ATP) security: this means that the device is programmed in such a way that the underlying cryptographic functionality (e.g., a signature scheme) will remain secure (e.g., existentially unforgeable) even in the presence of such an adversary. They then showed that, unless a device has a self-destruct capability and can be initialized via a trusted setup phase, this notion of security is unattainable. However, they also showed that using self-destruct and trusted setup, it is possible to achieve ATP security.

Ishai et al. [ISW03] considered the adversary who, in addition to black-box access to a circuit implementing a cryptographic functionality, could also probe individual wires of this circuit (we call this a "memory probing adversary"). They showed, surprisingly, that one could tolerate an adversary that probes up to some constant t wires at a time using a transformed circuit where corresponding to every wire of the original circuit, there are $\Theta(t)$ wires, each carrying a share of value of the original wire. Moreover, every time such a circuit is invoked it can redistribute these shares, and so it can be probed again, so over the lifetime of the circuit, the adversary can probe each wire several times. This resharing does not require a continuous source of true randomness: it can be done using a pseudorandom generator seeded by a random string that resides in the circuit's memory and also gets updated at every invocation. In a follow-up paper, Ishai et al. [IPSW06] further extended this model to allow the adversary to tamper with another c wires: to fix them temporarily or permanently. They showed that it was still possible to have a circuit that withstood such an attack.

Micali and Reyzin [MR04] defined security for cryptographic primitives under side channel attacks and show how to use these primitives; their side channel attack is much more general than attacks that are known in the real world [AK96, Koc96, KJJ99, AARR03, CRR03], and also more general than the probing attack of Ishai et al. [ISW03]. In fact, the model of Ishai et al. [ISW03] is a special case of the

Micali-Reyzin model. Micali and Reyzin do not, however, consider an adversary that tampers with the device.

Recently Dziembowski, Pietrzak, and Wichs [DPW10], they also consider security against the algorithmic tamper and probing adversaries. Their main technique is to construct "non-malleable codes" against a certain class of tampering attacks. In the plain model, their positive result holds for a smaller class of tampering functions that do modifications effecting each bit of the memory but independent of other bits. With a random oracle, they are able to extend the results against a broader class of tampering functions, yet the random oracle model is less desirable. We remark this does not contradict our impossibility results since we consider the stronger adversaries who can perform any polynomial-time computable tampering attacks in the plain model, where it is still open that one can extend their positive results in this case.

Our contribution. Our first contribution in the ATP model is to resolve, in the negative, the problem left open by Gennaro et al. of whether it was possible to realize a secure cryptographic functionality with a personal identification number (PIN) where a user is allowed to make up to ℓ incorrect consecutive attempts to enter her PIN, with no total limit on incorrect PIN attempts. (In contrast, Gennaro et al. showed that it was possible to limit the *total* number of incorrectly entered PINs). Along the way, we also showed that no ATP secure functionality can allow a user to change her PIN.

Next, we address the natural question of whether it is possible to achieve ATP security even in the presence of a memory-probing adversary. Here we remark that suppose the adversary can read all the contents in the memory by probing at one shot, then no security can be achieved. Thus in our model of memory-probing adversary, we consider a relaxation of the adversary's power by restricting the number of bits she can probe in a time. However, we do not limit the total number of bits (information) she can gather over time. This approach is similar to the key leakage model where the leakage is bounded at any moment but not over time.

Then, we give a definition of security for this scenario; our definition is a natural generalization of the definition of Gennaro et al. Next, we show that no deterministic circuit can achieve this notion of security: a memory-probing adversary who can also tamper with the memory can retrieve the secret content of the device's memory, even if she can only probe a constant number (very small fraction of the memory) in any moment.

Note that this impossibility applies to the circuit constructed by Ishai et al.: even though their construction uses randomness, ultimately it is the pseudorandom generator supplying it using a random seed in a deterministic fashion, hence their overall circuit is deterministic. The difference is that they only allow up to a certain number of wires to be tampered, while we consider the much more powerful tampering adversary of Gennaro et al., who may apply any polynomial-time computable transformation to the contents of a circuit's memory.

We also consider a variation of the memory probing adversary: one who may not probe memory cells, but only intermediate wires of the circuit. This is to model the idea that perhaps memory cells can be invulnerable to probing. It turns out that such an adversary is almost equally powerful: even though he is only explicitly allowed to read up to a constant t wires of the circuit at a time, he can cause any deterministic

circuit to behave in such a way that the contents of every wire in a particular invocation of the circuit (other than the protected memory cells) will become exposed to the adversary, i.e. the adversary can read all the wires at once. Due to impossibility of obfuscation [BGI+01], this leads to insecurity. (On the other hand, since we cannot reverse-engineer a circuit either, it does not necessarily imply that the secret content of the circuit can be computed from this information.)

Finally, we also consider the adversary who is allowed to tamper with wires of a circuit in addition to tampering with its memory and probing additional wires. Here, even if we do not allow the adversary to probe memory cells, the adversary can still retrieve the secret content of any deterministic circuit. Moreover, he can do it even if he chooses the set of wires to probe and tamper with *non-adaptively*.

On the positive side, we show that the Ishai et al.'s randomized construction (i.e. the one that uses true randomness, not a pseudorandom generator), in combination with the Gennaro et al.'s construction, achieves ATP security in the face of the circuit probing and tampering attack (but not memory probing). This is the best positive result we could get: for any other scenario we showed negative results!

Having investigated into the models in both paper, we briefly describe the distinction between those two: for the [GLM+04] model, the adversary can tamper with the whole memory, but cannot do with the circuit. In [IPSW06] model, the adversary can partially tamper and probe every part of the circuit, but cannot tamper with the whole memory in a single clock cycle. Both models have positive results. It is natural to consider if we can combine those models, to give the adversary more power, to see if positive results still remain or where they get stuck.

We show, mostly, that security cannot be achieved under a combination of attacks, for circuits without a source of true randomness. Under some conditions, the circuit with such source can apply the previous techniques to achieve security against the combined attacks. This is a separation for the models that shows a circuit with its randomness stored vulnerably is strictly less secure than that with a source of true randomness.

2 Definitions

2.1 ATP Models

Following Gennaro et al., we consider a system with two components: (1) secret content, sc (containing some secret key, sk, randomness, and possibly state information), and (2) a cryptographic algorithm implemented by a circuit C which uses the secret content.

We say that the system implements a certain function F, if for any input a,$C(sc, a) = F(a)$. We say that C implements a keyed cryptographic functionality $F(\cdot, \cdot)$, if for every key sk (from the appropriate domain) there exists a setting sc_{sk} of the secret data, such that the system (C, sc_{sk}) implements the function $F(\mathsf{sk}, \cdot)$. An algorithm computing sc_{sk} will be called a *software setup algorithm*. Finally, a device setup protocol implementing $F(\cdot, \cdot)$ is a pair of algorithms. The first generates the algorithm C, possibly with some additional state information to be passed to the second algorithm. The second is a software setup algorithm: given input sk and C, and possibly an additional state

information input, the algorithm generates an appropriate sc_{sk}. If the software setup algorithm is stateful, we say that the device uses public parameters. We will consider devices with efficient setup algorithms, and all the functionalities the devices compute are polynomially-computable.

Consider C which implements some $F(\cdot,\cdot)$ (e.g., a signature algorithm). Gennaro et al. defined a *tampering adversary* who can request two commands to be carried out: $\mathsf{Run}(\cdot)$ and $\mathsf{Apply}(\cdot)$, and Setup.

- The command $\mathsf{Run}(a)$, invokes the cryptographic computation C using the software content sc on input a. The output is the output of such computation, i.e., $C(sc, a)$. For example, if the cryptographic algorithm is a signature then the output is a signature on the message a using the secret key stored in sc.
- The command $\mathsf{Apply}(f)$ takes as input a function f, and modifies the software content sc to $f(sc)$. From this point on, until a new $\mathsf{Apply}(f)$ is requested, all $\mathsf{Run}(a)$ operations will use $f(sc)$ as the new software content. f can be a probabilistic function. Note that the next invocation of $\mathsf{Apply}(f')$ would change $f(sc)$ to $f'(f(sc))$, i.e. it does not apply f' to the original sc. There is no output for this command.
- The command $\mathsf{Setup}(\mathsf{sk})$ invokes the software setup algorithm, outputting sc such that the device $C(sc,\cdot)$ implements the function $F(\mathsf{sk},\cdot)$.

The device may also have a *self-destruct* capability, called by the circuit C. If this happens, every Run command from then on will always output $\perp$.

As mentioned above, security of smartcards and other portable devices is one of the motivations for considering this model. For convenience, throughout this paper we refer to the system interchangeably as a "card" or a "device".

In the tampering adversary model (referred to in the sequel as the ATP model and the [GLM+04] model), the adversary only applies a polynomial-time computable transformation on the secret memory content sc without reading it directly. On the other hand, the underlying hardware circuit C cannot be tampered with, and results of intermediate computation steps cannot be eavesdropped.

In the following sections, we extend the [GLM+04] model to allow the adversary to not only tamper with the circuit's memory, but also to probe the circuit's wires and gates while the computation is going on, and to tamper with the individual wires in the circuit. We get adversaries of different strengths by allowing various combinations of these attacks. The memory probing adversary is allowed to read one bit at a time of the secret content sc, in addition to being able to tamper with it through the Apply command. The circuit probing adversary will be allowed to retrieve the contents of a wire in the circuit during the execution of a Run command, in addition to being able to issue Apply commands. The wire fixing adversary is allowed to fix a particular wire of the circuit so that for the duration of the Run query it carries a particular bit. We will formalize the definitions of these additional adversarial behaviors in the following sections.

2.2 Memory-Probing Models

In this section, we consider the adversary by allowing the probing attacks on the memory. Besides Run, the adversary can probe several (a constant number of) cells in the

memory once, after Run is finished. To formalize that, we make available to the adversary the following capability: Let the memory content $sc \in \{0,1\}^m$ be an m-bit string, and T be a subset of $\{1, 2, \ldots, m\}$ The command ProbeMem(T) returns the i^{th} bit of the secret content, sc_i for any $i \in T$.

If $|T| = m$, then we could never achieve security.Therefore, it is natural to limit the size of probing by allowing $|T| = t$ for a constant parameter (that does not grow with m). The command ProbeMem can be executed at most once following an execution of the Run command. We allow the adversary to change the set of indices it queries, T, adaptively.

2.3 Circuit-Probing and Circuit-Tampering Models

In this section, we consider another type of attacks: the adversary can tamper or probe the circuit's wires when Run is operating. To formalize that, we let the wires in the circuit be labeled by $W = \{w_1, w_2, \ldots, w_\ell\}$ for some ℓ, and T be a subset of $\{w_1, w_2, \ldots, w_\ell\}$.

For the Circuit-Probing model, the adversary may issue the following command:

- The commands ProbeWire(T) returns the values of the wires $w_i \in T$.

For the Circuit-Tampering model, the adversary may issue the following commands:

- The commands ChangeWire(T, val) returns nothing but replaces the value in the wire $w_i \in T$ with val_i temporarily.
- The commands FixWire(T, val) returns nothing but replaces the value in the wire $w_i \in T$ with val_i permanently.

The adversary is able to apply any and only one of these commands per clock cycle when the circuit is operating (Run is called.) Since this model inherits the results of [IPSW06], it is reasonable for us to limit the size of T by setting $|T| = t$ for some constant parameter, as they did.

2.4 Combined ATP, Memory-Probing, Circuit-Tampering, Circuit-Probing Models

In the following sections, we will consider a variety of combination of models. In summary, section 4 considers the combination of ATP and Memory-Probing models; section 5 considers the combination of ATP, Circuit-Probing, and Circuit-Tampering models. The details will be explained in the sections respectively.

2.5 Security Definition

Here we give a general definition for the security of the circuit. This definition is an extension of the [GLM+04] definition: it gives the adversary a broader set of capabilities.

Definition 1. *Define $\mathcal{A}_{\mathsf{Ideal}}$ be the set of adversaries that can only obtain the input-output behavior of the device, and $\mathcal{A}_{\mathsf{Model}}$ be the set of adversaries that can perform any attack defined in a particular* Model *(this* Model *can be a combination of attack capabilities described above). Let C be a circuit that implements some functionality.*

We say C is Model*-secure if there exists a probabilistic polynomial time simulator S such that for any $A \in \mathcal{A}_{\mathsf{Model}}$, the following two experiments are computationally indistinguishable:*

1. *$S^A \in \mathcal{A}_{\mathsf{Ideal}}$ outputs a bit after interacting with C.*
2. *$A \in \mathcal{A}_{\mathsf{Model}}$ outputs a bit after interacting with C.*

In the following sections, if we don't specify the Model, we are referring to the model discussed in that section.

3 New Impossibility Result in the ATP Model

Consider the following functionality for a signature device with a personal identification number (PIN). The device has a public key pk, and its secret content sc contains the corresponding sk and a personal identification code pin that must be entered for the device to run properly. The idea is that a user Alice of the device would need to remember a short PIN; if she loses the device and it falls into the adversary's hands, the adversary will still have to correctly guess the PIN before he can use it. We want a device that tolerates a few incorrect PIN attempts (since Alice may occasionally mistype the PIN), but stops working after the number of incorrectly entered PINs exceeds a certain threshold α (a constant that is much smaller than all possible PINs). Gennaro et al. showed that this is possible if we want to tolerate α as the *total* number of incorrectly entered PINs, but left as an open problem the question of whether it was possible to have a functionality that allowed any number of incorrectly entered PINs over the life of the device, but would stop working if the number of *consecutive* incorrect PINs exceeds the threshold α. Here we show that this functionality (referred to in the sequel as "signature with consecutive PIN protection") cannot be ATP-secure. We also show that we cannot achieve ATP security for the functionality that allows Alice to change her PIN (referred to in the sequel as "signature with user changeable PIN").

In the following theorems, we assume that the device computes a polynomial-time function that on input PIN and the secret component outputs $1/0$, denoting the validity of the PIN. Also we assume that the PIN has a polynomial-size support.

Theorem 1. *The signature with user changeable PIN functionality cannot be* ATP *secure, even if a circuit can self-destruct, assuming the device implements a polynomial-time change-pin function $f_{ChangePIN}$ that on input $(sc, NewPIN, OldPIN)$ outputs a new valid secret component sc', and the devices calls $f_{ChangePIN}$ when the user changes her PIN.*

Proof. The adversary will take advantage of the existence of this function $f_{ChangePIN}$ in order to break the ATP security of the device. Recall that the adversary may specify, as input to the Apply command, a polynomial-time computable function f. As a result of Apply(f), our adversary will succeed in replacing the old PIN (which he does not know) with a new PIN. For simplicity, the new PIN will be the all-zero string 0^ℓ where ℓ is the length of the PIN. As a result of Apply(f), the adversary will be able to use the device from now on.

This function f works as follows: for every possible PIN p, it runs the following function f_p: On input secret component sc of the device, f_p first checks whether p is the correct PIN, then it returns $sc' = f_{ChangePIN}(sc, 0^\ell, p)$. Else f_p returns sc. Since f does this for every possible PIN p, we guarantee that in the end, sc will be modified in the same way as if the user changed her PIN to 0^ℓ. f is polynomial-time, because the PIN is a memorizable short number, for which all possibilities can be enumerated in polynomial time (from the assumption). □

Theorem 2. *The signature with consecutive PIN protection functionality cannot be* ATP *secure, even if a circuit can self-destruct, assuming the device implements a polynomial-time reset function $f_{ResetPIN}$ that on input $sc, Input, pin$ outputs a valid sc' for the correct PIN, and every time every time the PIN is correctly entered, the counter of consecutive errors is reset.*

Proof. Our adversary will take advantage of the existence of this function $f_{ResetPIN}$ in order to come up with the function f to give as argument to the Apply command. As a result of $\mathsf{Apply}(f)$, the counter for incorrect consecutive PIN attempts will be reset, even though the adversary has not issued $\mathsf{Run}(Input, pin)$ for the correct PIN pin.

f will work as follows: for all possible PINs p, it will run the function f_p. f_p (similarly to the proof of Theorem 1) works like this: on input $(sc, Input)$, where $Input$ is any message in the message space of the signature scheme — for simplicity, let $Input$ be the all-zero message 0^n. it first checks whether p is the correct PIN; if so, it returns $sc' = f_{ResetPIN}(sc, Input, p)$; else, it returns sc.

Once again, since PIN is a memorizable short number, f can call every possible f_p in polynomial time. After $\mathsf{Apply}(f)$ is executed, the secret content is whatever it would be after $\mathsf{Run}(Input, pin)$ is queried with the valid PIN pin. □

4 Impossibility of Deterministic Circuits in the ATP-Memory-Probing Model

Suppose that, after the circuit C executes a Run command, the secret contents sc always remains unchanged, i.e. the same as before the Run command was executed. Then the memory probing adversary can trivially learn the entire string sc by simply probing each bit of sc one by one. Here we show that even if the circuit C updates sc before completing the execution of Run, the memory probing adversary can still compute a candidate sc' that would correspond to the secret contents of the device for some time period.

Let C_{mem} be the function that, on input sc and a outputs the updated version of sc, the secret contents of the device's memory left behind after $\mathsf{Run}(a)$ is executed. For a particular a, let $X_0 = sc$, $X_{i+1} := Next_a(X_i)$ be shorthand for $C_{mem}(X_i, a)$. Let sc and a be given; for $i > 0$, if the circuit is deterministic, each X_i is well-defined.

Theorem 3. *A deterministic signature functionality cannot be* ATP-Memory-Probing *secure, even if the circuit can self-destruct: there exists a polynomial-time adversary that outputs X_i for some i.*

Proof. We prove the theorem by giving an adversary that attacks any device that implements the signature functionality. The adversary will get some "useful" information by probing some bits in the memory. The intuition is: the adversary takes advantage of a polynomial-time computable function f that first identifies a good location to probe, signals this location back to the adversary, and then conveys the secret content in that particular location in the memory. To be more specific: for the memory contents $X_0, X_1, \ldots, X_\ell$ for some ℓ, there is either a simple cycle or at least one bit of the memory that has changes with enough regularity. For the former case, the adversary can always fix the memory content to be the same and then probe it bit by bit. For the latter one, the adversary can obtain this location (having changes with enough regularity) and then transmit one X_t for some t through probing at this location.

Let us explain how this function conveys information and how the adversary receives it with the following algorithms. Algorithm 5 describes the function f that the adversary is going to use: f is parameterized by (r, aux, a). r is an integer that depends on how many times the adversary has already modified the memory. aux is a string that depends on what she is trying to do at this stage. a is the index of $Next_a(sc) = C_{mem}(sc, a)$. With the algorithm, we develop the following lemmas for the theorem.

Lemma 1. *The adversarial function is a polynomial-time computable function.*

Proof. Every step in the algorithm is clearly polynomial-time computable. We put a more detailed proof in the full version of this paper.

Lemma 2. *The adversary will find a sc' that $C(sc', \cdot)$ also implements a valid signature function as $C(sc, \cdot)$ does.*

Proof. We consider two cases, (1) there exists a cycle with length no greater than $m^3 + 3m$ on $X_0, X_1, \ldots, X_\ell$, where $X_0 = sc$, $X_i = Next_a(X_{i-1})$, and $\ell = m^3 + 4m - r$, for some $r \in [m]$. (2) there doesn't. For the first case, we let X_0 be the start of the cycle; otherwise the function will first return X_j, where $Next_a(X_j)$ is the start of the cycle, and then we go back to the case where X_0 is the start of the cycle.

1. Suppose there exists a $j < m^3 + 4m < \ell$ such that $X_j = X_0 = sc$, the adversarial function will move the memory to X_{j-1}. After the device runs, it updates the memory from X_{j-1} to $X_j = X_0 = sc$, so the adversary will probe with the same memory contents (sc) in the first m rounds. Also, since $m^3 + 3m \leq \ell = m^3 + 4m - r$ for $r \in [m]$, the adversarial function will always find this cycle, and the memory will always be X_0 when the adversary probes it. Therefore the adversary will find sc and construct $C(sc, \cdot)$ as desired.
2. Suppose there doesn't exist such small cycle, then the adversary will most likely not get a good candidate after step 1. (*Note: if she still does get one by luck, then she will be very happy and doesn't need the following procedure. The proof is done.*) Now, she is going to query "Is location k a good place to probe?" for every bit. Since there doesn't exist a cycle or a small cycle in $X_0, X_1, \ldots, X_l$, we assume $X_0, X_1, \ldots, X_{m^3+3m}$ are distinct elements without loss of generality. Then the adversary is going to ask which location is a good place to probe. *Note: a good location is the place which contains a lot of* $0, 1$ *alternations.* So the adversarial

function can use those $0/1$'s to convey sc' when the adversary is probing such location.

3. Now we want to prove *there must be a good location to probe:* if location i is not the place to be probed, then the adversarial function will at most move from X_0 to X_2 after "Give me 0" and "Give me 1", since we can always find a two-bit string in $X_0(i), X_1(i), X_2(i)$ that violates "01" (in any one of the eight combinations of those three bits.) Thus, at each time when the third step of the function is run, there are at least m^3 distinct elements (i.e. $X_0, X_1, \ldots X_{m^3}$,) (for a bad location, we waste at most three distinct elements. Thus, every time we have at least $m^3 + 3m - 3m$ distinct elements.)

   Since those elements are distinct, we have $\sum_{j=1}^{j=m} |X_i(j) - X_{i+1}(j)| \geq 1$ for any $i = 1....m^3 - 1$. This implies $\sum_{i=0}^{m^3-1} \sum_{j=1}^{j=m} |X_i(j) - X_{i+1}(j)| \geq m^3$. This is a finite summation and i, j are independent, so we can change the summation order to get: $\sum_{j=1}^{j=m} \sum_{i=0}^{m^3-1} |X_i(j) - X_{i+1}(j)| \geq m^3$. According to the pigeon hole principle, we must have some k such that $\sum_{i=0}^{m^3-1} |X_i(k) - X_{i+1}(k)| \geq m^3/m = m^2 > 5m + 2$. *Note:* $diff_k = \sum_{i=0}^{m^3+4m-r} |X_i(k) - X_{i+1}(k)| \geq \sum_{i=0}^{m^3-1} |X_i(k) - X_{i+1}(k)| \geq 5m + 2 - r$. Thus we must have some k such that $diff_k > 5m + 2 - r$ for $r = 0$ or 1.

   This implies in this case, there must exist a good location to probe. And the adversary will get this one from the function f. After this location is obtained, there are $5m$ alternations of $0/1$ on this bit, and the adversarial function can easily convey the message about sc' through this bit. The remaining argument follows straightforwardly with the algorithm □

Remark 1. A natural question is: what can this attack do to a functionality with a PIN? In such a functionality, the adversary must enter the correct PIN pin to run $\mathsf{Run}(a, pin)$. Recall that we require the PIN to be an easily memorizable string, and so the number of possible choices is not large. Therefore the adversary has a non-negligible probability of guessing it correctly. Once she guesses the correct PIN, she can find out the secret content sc' using the attack above.

Remark 2. This result is not limited to the signature functionality; we used signatures for concreteness. In fact, no deterministic testable (as defined by Gennaro et al.) functionality can be ATP secure in the memory probing model.

5 Impossibility of Deterministic Circuits in the ATP-Circuit-Probing-Tampering-Model

In this section, we are going to consider the model where the adversary can do the probing attacks and tampering attacks on the wires. From the previous section, we have already shown that if the adversary is able to read directly the memory cell (or read from the wires that carry the content into the circuitry) then the deterministic circuit can not achieve ATP security. Those impossibilities are still inherited here. Therefore, we are going to consider further restrictions on the adversary.

Algorithm 1. Description of the Adversary (Theorem 3)

1. Pick an arbitrary a from the message space in the signature scheme.
   for $i = 1$ to m **do**
   Let $r = i$, $aux = \epsilon$, $sc' = \epsilon$.
   Run consecutively $\mathsf{Apply}(f_{r,aux,a})$, $\mathsf{Run}(a)$, and $sc' = sc' \circ \mathsf{ProbeMem}(\{i\})$. I.e. probe location i of the memory and then concatenate the outcome with sc'.
   end for
   Then we have a candidate sc' from the bits we've probed.
   Construct a circuit $C(sc', \cdot)$ and check if this circuit outputs validly for the signature scheme as $C(sc, \cdot)$ does.
2. **if** the constructed circuit does **then**
   Output sc'
   else
   for $i = 1$ to m **do**
   Let $r = 0$, $aux =$"Is location i a good place to probe?"$\circ$"Give me 0"
   Run $\mathsf{Apply}(f_{r,aux,a})$, $\mathsf{Run}(a)$ and then $\mathsf{ProbeMem}(\{i\})$. .
   Let $r = 1$, $aux =$"Is location i a good place to probe?"$\circ$"Give me 1"
   Run $\mathsf{Apply}(f_{r,aux,a})$, $\mathsf{Run}(a)$ and then $\mathsf{ProbeMem}(\{i\})$.
   If the outcomes of two consecutive probes are anything other than 01, then the adversary knows this is not a good location to probe, so it continues. Otherwise, exit **for** and let bit $pb = i$ be the location to be probed.
   end for
   end if
3. Let $str = \epsilon$
   for $i = 1$ to m **do**
   Let $r = i$, $aux =$"Location pb will be probed."$\circ$"I want bit i of the secret." $\circ$"Bits $1, 2, \ldots, i-1$ of the secret are str"
   Run $\mathsf{Apply}(f_{r,aux,a})$, $\mathsf{Run}(a)$, and $b = \mathsf{ProbeMem}(\{pb\})$. (Probe location pb, and get the outcome b.)
   Let $str = str \circ b$ (a concatenation.)
   end for
   Output str.

Before stating them, we first consider some motivations for intuitively understanding. Suppose the adversary has some nano needles that can perform the probing and tampering attacks on wires, but each needle occupies some areas and after placing the needle, the adversary cannot change its position without damaging the original circuit. Thus she should choose a small set of wires which she is going to attack in advance and cannot change them adaptively. In this section, we show that even with the restrictions, the adversary can destroy the ATP security. As a consequence, the adversary with even stronger power that can attack wires adaptively can certainly destroy the ATP security.

Now we state the restrictions explicitly: the adversary needs to select a set of wires to attack before the operation of the device. Note: every wire can be included in this set, and once it is chosen, the adversary can only tamper or probe the wires in this set. Also, after this set has been chosen, the adversary cannot change it. This is called non-adaptive attacks.

Algorithm 2. The adversarial function $f_{r,aux,a}$ (Theorem 3)

On input sc do:

1. Compute $X_0, X_1, \ldots, X_\ell$, for $\ell = m^3 + 4m - r$. *Note: recall* $X_0 = sc, X_i = Next_a(X_{i-1})$ *as defined in the beginning in this section.*
2. If $aux = \epsilon$, then try to determine if the sequence of values $\{X_i\}$ contains a cycle:

   If $aux \neq \epsilon$, goto Step 3: *that is, the adversary already knows that there are no small cycles.*

   Else, check for cycles with its length no greater than $m^3 + 3m$: does there exist an $0 \leq i < j \leq \ell$ such that $X_i = X_j$, and $j - i < m^3 + 3m$, and $X_i, X_{i+1}, \ldots, X_j$ are distinct.

   if NO (i.e. no cycle or there exists a cycle but the length is too large) **then**
   Output X_0.
   else
   consider two cases: (a) $i > 0$: output X_i (b) $i = 0$: output X_{j-1}.
   end if
3. **if** aux contains the string "Location k will be probed." **then**
   go to Step 4. *The adversary already knows which location to probe in the memory to get useful information.*
   else
   aux must contain the string "Is location k a good place to probe?" A good location to probe is one where, as the value of sc changes over time, the bit stored at this memory location keeps changing. Thus, if we want to communicate a bit b to the adversary, we can do so by setting $sc = X_i$ for some X_i whose k^{th} bit is b.
   Let S be the string obtained by concatenating the following bits: $S = X_1(k) \circ X_2(k) \circ \cdots \circ X_l(k)$ where $X_i(j)$ means the j-th bit of X_i. Let $diff_k = \sum_{j=2}^{\ell} |X_j(k) - X_{j-1}(k)|$. I.e., $diff_k$ measures how many times the value stored at the k^{th} memory location changes as sc changes over time.
   if $diff_k > 5m + 2 - r$ **then**
   This is a good location, because $diff_k$ is high. This needs to be communicated back to the adversary. We know that the adversary will be probing the k^{th} memory location to get the answer to this question, and therefore we do as follows:
   consider the two cases:
   (a) aux contains "Give me a 0" then let $t + 1$ be the smallest integer such that $X_{t+1}(k) = 0$. Output X_t.
   (b) aux contains "Give me a 1" then let $t + 1$ be the smallest integer such that $X_{t+1}(k) = 1$. Output X_t.
   else
   k is a bad location.
   consider the two cases:
   (a) aux contains the string "Give me a 0" then if $X_1(k) = 1$ output X_0. If $X_1(k) = 0$, and $X_2(k) = 1$ output X_1. Else if $X_1(k) = 0$,and $X_2(k) = 0$ output X_0.
   (b) aux contains the string "Give me a 1" then output X_0.
   end if
   end if
4. The adversary will probe location k. Among the ℓ possibilities for sc, $X_0, \ldots, X_\ell$, find X_t for a sufficiently large t, consistent with what the adversary already knows, and communicate a bit of X_t by making sure that this bit will be read off from location k. More precisely:

   aux must contain "Location k will be probed", and "I want bit j of the secret.", and "Bits $1, 2, \ldots, j-1$ of the secret are $s_1, s_2, \ldots, s_{j-1}$."
   Find the least $t \geq m^5 - m^3 r$ such that the first $j-1$ bits of X_t are $s_1, s_2, \ldots, s_{j-1}$. Find the least u such that $X_{u+1}(k) = X_t(j)$. Output X_u.

In the following, we are going to show the adversary only needs to attack a small set of wires to destroy ATP security. Since the construction of the adversary and proofs are similar to theorem 3, we only state the theorem here and leave the details including the formalization of the model and proofs the full version of this paper for the curious readers.

Theorem 4. *A deterministic signature functionality cannot be* ATP-Circuit-Probing-Tampering *secure in the non-adaptive model. That is: the adversary first sets the attack range on the output wires of C_{check} and then will either disable the self-destruct function or find out some valid sc'.*

Note: C_{check} is one part of the components in the circuit, which checks if the memory is valid. The functionality is necessary for every cryptographic device. The precise model can be found in the the full version of this paper.

Remark 3. Gennaro et al. showed there is no ATP secure achieved without self-destruct functionality. Thus, if the adversary disables such functionality, she can retrieve the secret content as the authors did in [GLM$^+$04].

Remark 4. Since the signature functionality cannot be ATP secure under non-adaptive model, it is clearly that it cannot withstand a stronger adversary which can do the adaptive attacks.

6 ATP-Circuit-Probing-Tampering Security from Encoded Randomness Gates

In the previous sections, we see the limitations of deterministic circuits. Thus it seems that the signals in the wires should be made independent of the memory content to defend against probing attacks. And this is where randomness comes in handy. Intuitively, one can think that randomness provides an independent and unpredictable shield that hides each signal (using a secret sharing scheme [ISW03]) which the adversary cannot manipulate by merely tampering with the memory content.

In this section, we consider circuits with a source of true randomness. For this model, the previous results in [IPSW06, ISW03, GLM$^+$04] work. After we rule out yet another class of attacks that makes the circuit entirely vulnerable, we can combine the techniques in those works to achieve ATP security in this new model.

Definition 2 (Randomness gate). *A randomness gate is a gate with no input and one output wire that emits a truly random bit each clock cycle.*

Lemma 3. *In the* ATP-Circuit-Probing-Circuit-Tampering *model, there exists an adversary who, for any keyed cryptographic functionality, either discovers a valid secret sc', or determines all the values of all the internal wires corresponding to the execution of the* Run*() command, even for circuits with randomness gates.*

Proof (sketch). Let $RG = \{rg_1, rg_2, \ldots, rg_r\}$ be the set of randomness gates used by the circuit. Since the adversary can tamper with any internal wire, he can fix the output

of every randomness gate. We must make sure that this does not cause the device to self-destruct (for example, a device that remembers the randomness used in previous invocation might detect that something suspicious is going on). To do that, once the output of a randomness gate is fixed, the adversary must run the Apply() command that will make a device that can store m bits of memory "fast-forward" far enough into the future, using true randomness, so that it would no longer remember the fixed randomness. Now the circuit becomes deterministic and we can use a similar attack in the previous section. A formal description is deferred to the full version of this paper.

We see that if all randomness gates are vulnerable under tampering attacks, then the circuit can be made deterministic. Thus, to defend against tampering attacks, we need a more complex gadget: "encoded randomness gate," as proposed in [IPSW06]. Let the encoded randomness gate ERG^k be an element that takes no input and produces a string of output a k-bit string per clock cycle, 1^k representing 1, 0^k representing 0, and others representing the invalid signal. The output distribution is $Pr[ERG^k = 1^k] = Pr[ERG^k = 0^k] = 1/2$. The intuition for this gadget is that the adversary has little probability to fix the entire output of a gadget before causing an invalid signal. From the techniques in [IPSW06], we can design an implementation that if an invalid signal is caused, then it will be passed to the whole circuit and erase the whole output and memory content.

Theorem 5 (main result in [GLM$^+$04]). *Under the assumption of the existence of strong universal unforgeable signature scheme, there exists unforgeable signature scheme that achieves ATP security. That is, there exists a circuit $C(sc, \cdot)$ that implements a signature functionality with secret content sc stored in the **Memory** and is ATP secure.*

The main idea here is let $sc = sc' \circ \sigma_{\Pi}(sc)$ where σ is a universal unforgeable signature scheme and Π is the secret signing key of the card manufacturer, and sc' contains the signing key of the signature device. In brief, since $\sigma_{\Pi}(sc)$ can be only issued by the manufacturer, the adversary is not able to produce it by himself, and thus she cannot produce any other valid sc' that will pass the verification process. The formal reduction proof can be found in [GLM$^+$04].

Next, we recall the main result of Ishai et al. By "registers" we mean a special component of the circuit into which a portion of memory (and intermediate results of computation steps) can be loaded at execution time.

Theorem 6 (main result in [IPSW06]). *There exists a circuit $C(\cdot)$, using AND, OR, NOT, and "encoded randomness gates," with sc stored in its **registers** that implements a signature functionality and achieves* Circuit-Probing-Circuit-Tampering *security.*

Theorem 7 (combined result). *Let m be the length of the secret content sc. There exists a circuit $C(\cdot)$, using AND, OR, NOT, "encoded randomness gates," and $\Theta(m)$ "robust wires" which are invulnerable to probing attacks, with sc stored in its **memory** that implements a signature functionality and achieves* ATP-Circuit-Probing-Tampering *security.*

The idea here is that the circuit first uses the robust wires to load the memory content to the registers. Then during the execution, the device only uses the registers in the circuit

for the memorization of internal states, etc. Finally, the circuit updates the memory through the robust wires. Then Theorem 5 and Theorem 6 combine perfectly.

References

[AARR03] Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side-channel(s). In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 29–45. Springer, Heidelberg (2003)

[AK96] Anderson, R., Kuhn, M.: Tamper Resistance - a Cautionary Note. In: Proceedings of the Second Usenix Workshop on Electronic Commerce, pp. 1–11 (November 1996)

[AK97] Anderson, R., Kuhn, M.: Low cost attacks on tamper resistant devices. In: Lomas, M. (ed.) Security Protocols 1996. LNCS, vol. 1189. Springer, Heidelberg (1997)

[BGI+01] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001)

[BS97] Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)

[CRR03] Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)

[DPW10] Dziembowski, S., Pietrzak, K., Wichs, D.: Non-malleable codes. In: ICS (2010)

[GLM+04] Gennaro, R., Lysyanskaya, A., Malkin, T., Micali, S., Rabin, T.: Algorithmic tamper-proof (atp) security: Theoretical foundations for security against hardware tampering. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 258–277. Springer, Heidelberg (2004)

[IPSW06] Ishai, Y., Prabhakaran, M., Sahai, A., Wagner, D.: Private circuits ii: Keeping secrets in tamperable circuits. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 308–327. Springer, Heidelberg (2006)

[ISW03] Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)

[KJJ99] Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)

[Koc96] Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)

[MR04] Micali, S., Reyzin, L.: Physically observable cryptography. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 278–296. Springer, Heidelberg (2004)

[SA03] Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 2–12. Springer, Heidelberg (2003)