Algorithmic Tamper-Proof (ATP) Security: Theoretical Foundations for Security Against Hardware Tampering

Rosario Gennaro[1], Anna Lysyanskaya[2], Tal Malkin[3], Silvio Micali[4], and Tal Rabin[1]

[1] IBM T.J. Watson Research Center
{rosario,talr}@watson.ibm.com
[2] Department of Computer Science, Brown University
anna@cs.brown.edu
[3] Department of Computer Science, Columbia University
tal@cs.columbia.edu
[4] M.I.T. Laboratory for Computer Science

Abstract. Traditionally, secure cryptographic algorithms provide security against an adversary who has only *black-box* access to the secret information of honest parties. However, such models are not always adequate. In particular, the security of these algorithms may completely break under (feasible) attacks that tamper with the secret key.

In this paper we propose a theoretical framework to investigate the algorithmic aspects related to tamper-proof security. In particular, we define a model of security against an adversary who is allowed to apply arbitrary feasible functions f to the secret key sk, and obtain the result of the cryptographic algorithms using the new secret key $f(sk)$.
We prove that in the most general setting it is impossible to achieve this strong notion of security. We then show minimal additions to the model, which are needed in order to obtain provable security. We prove that these additions are necessary and also sufficient for most common cryptographic primitives, such as encryption and signature schemes.
We discuss the applications to portable devices protected by PINs and show how to integrate PIN security into the generic security design.
Finally we investigate restrictions of the model in which the tampering powers of the adversary are limited. These restrictions model realistic attacks (like differential fault analysis) that have been demonstrated in practice. In these settings we show security solutions that work even without the additions mentioned above.

1 Introduction

Motivation and Our Main Questions. Traditionally, cryptographic algorithms have been designed to provide security against an adversary who has only *black-box* access to the secret information of honest parties. That is, the adversary can query the cryptographic algorithm on inputs of his choice and analyze the responses, which are always computed according to the correct original secret information. By now, cryptographic design has become so advanced that all the major cryptographic primitives can be proven secure against black-box attacks under very weak complexity assumptions. Proofs of security for such cryptographic algorithms assume (as an abstraction) that there is some secure hardware in which the algorithm and secret key of the honest parties are stored, thus denying the adversary any form of access to this data other than exchanging messages. If this assumption is violated, all guarantees are off.

At a closer analysis, the secure hardware assumption encompasses two different components, informally: (1) *Read-proof hardware;* that is, hardware that prevents an enemy from reading anything about the data stored within it; and (2) *Tamper-proof hardware;* that is, hardware that prevents an enemy from changing anything in the data stored within it.

In particular, traditional cryptographic schemes consist of an algorithm which the adversary knows, but cannot change (i.e., stored in tamper-proof hardware), and a secret key, which the adversary does not know and cannot change (i.e., stored in hardware which is both read-proof and tamper-proof).

It is clear that each of these components is necessary, at least to some extent, in order to achieve security of a cryptographic algorithm. If the adversary can read all information belonging to an honest party, he can also perform all the same functionalities. If the adversary can arbitrarily change the algorithm implemented by the honest party, he can cause the algorithm to output all the secret information. Thus, both read-proofness and tamper-proofness are necessary assumptions. This raises the following natural questions:

Is it necessary to have a component which is both read-proof and tamper-proof? Can we decouple these assumptions and achieve security when the adversary has arbitrary tampering powers for any secret information, and complete knowledge of any unchangeable information? What are the minimal physical assumptions necessary for the existence of provably secure implementations of major cryptographic primitives?

Clearly, if the secret data is only secured via a read-proof hardware then the adversary can destroy the information by overwriting it. Our goal, however, is to prevent the adversary from compromising the security of the card with respect to the original secret data (e.g., by forging a valid digital signature).

In addition to being a natural next step in a line of research aiming to achieve security against ever stronger adversaries, these questions also have direct significance to reducing the gap between cryptographic proofs and practical implementations. The motivation for decoupling is further driven by the current state in secure hardware design. There are two fronts which support the need for decoupling: attacks on, and manufacturing of, the devices.

Known attacks show that it is hard to preserve the security of the cards. Works such as [KJJ99,AARR03] show that a wide variety of "side channels" exist that enable an adversary to read off secret keys. On the other hand, many physical tampering attacks have proved successful, see for example [AK96,SA03]. Boneh, DeMillo, and Lipton [BDL01] show how to use a small number of random faults to break specific, public-key based schemes. Biham and Shamir [BS97] show how to break even unknown secret-key schemes, using a specific kind of random faults. They give these attacks the name *differential fault analysis*.

These types of attacks are of particular concern in light of the way cryptography is used today. For one, many cryptographic applications are carried out by small devices outside the security of a protected environment (e.g., smartcards and PDAs). Such gadgets may fall into the wrong hands with great ease, giving an adversary ample opportunity to apply a battery of *physical attacks*. Moreover, today's widespread use of cryptography, by virtue of its ubiquity, opens the door to increased vulnerabilities, such as opportunities for *insider attacks* by naive or malicious users. Thus it is important to reduce as much as possible the assumptions on the adversary's limitations.

On the manufacturing front, if we wish to store data which is both hardwired and secret this would need to be done at manufacturing time. This implies that the user's secret key should, at some level, be known to the device manufacturer, and this is clearly not desirable. Moreover, producing one-of-a-kind hardware for each of many users, which would be required if a unique key is hardwired in each device, may be totally impractical.

This body of evidence argues that to assume hardware that is both read-proof and tamper-proof is a big leap of faith. From this perspective, granted that both tamper-proof and read-proof security are assumptions, we wish to understand their relative strength. We are asking whether, for a fixed cryptographic algorithm, and a secret key which is stored in a read-proof hardware, the read-proof hardware can be bootstrapped via an algorithm to provide tamper-proofness? We introduce the notion of *Algorithmic Tamper-Proof (ATP) Security* which addresses security in the decoupled environment.

Our Model. We will model devices with two separate components, one being tamper-proof yet readable, and the other being read-proof yet tamperable. These components may be thought of as corresponding to the traditional notions of a hardware (circuitry) and software (memory) components of a given device. We allow only data that is common to all devices (and considered universally known) to be hardwired beyond the tampering reach of the adversary.

We define a very strong *tampering adversary* and the notion of security in our new model. The adversary considers the device's memory, M, as an n-tuple of individual bits, $x_1, \ldots, x_n$, and knows the functionality of each bit-position (e.g., where a given secret key begins and ends). We allow the adversary to specify any polynomial-time computable function $f : \{0,1\}^n \to \{0,1\}^n$ and transform M to $f(M)$. More precisely, we envisage that the adversary may adaptively interact with the device by repeating the following a polynomial number of times:

1. choose a polynomial-time computable function f and replace the current memory content, M, with the new content $f(M)$; and
2. interact with the device with memory content $f(M)$ (e.g., input a message to be signed with the current secret key, enter a PIN. etc.)

We define the notion of *algorithmic tamper-proof security* and require that whatever such an attacker can achieve, could also be achieved by a black-box attack on the system. This definition may be formulated either as a simulation-based definition, or by a direct definition of security for the cryptographic primitive (signature or encryption) with a tampering adversary.

We believe this to make a clear and attractive model for studying our problem. The model unifies and provides a theoretical framework for practical attacks such as differential fault analysis, while at the same time maintaining a more general view of security. The model also provides the next natural step in security against strong adversaries (e.g., for encryption, this is the next step after CCA2 attacks). Further applications may be possible.

Our Answers. We first show that in the model as described above ATP security cannot be achieved. That is having secret data stored in read-proof only hardware does not even preserve the secrecy of the data, let alone provide security for the cryptographic function.

Thus, we consider modifications to the model which still preserve the decoupling property in order to achieve ATP security. The modifications are done in two directions, one to enhance the physical design and the second to limit the tampering capabilities of the adversary.

Enhancing the Physical Design. We show that ATP security in the above model can be achieved iff the device is enhanced with: (1) a self-destructing capability, and (2) some hardwired data (public parameter) which is produced by a separate server that cannot be tampered with.

Specifically, we show that without (1), any cryptographic algorithm can be completely broken by a memory tampering attack, and that without (2), there are signature and encryption schemes that cannot be implemented securely in the face of a tampering attack.

Then we proceed to show that the two enhancements are *sufficient*. We achieve algorithmic tamper-proof security with respect to *arbitrary, feasible functions* f, for fundamental public-key applications such as signing and decryption; but our techniques also apply in the secret-key setting.

One way to interpret these results, is that to achieve general ATP for cryptographic schemes (e.g., signature or decryption), we *do* need a component which is both read-proof and tamper-proof (the memory of the server used for condition 2). However, this component *need not* be part of every device instantiating the scheme, as assumed in traditional models (where the secret key is stored in that component). Rather, it is sufficient to have *one* such component, used *only at setup time*, in order to provide algorithmic tamper-proof security for all instantiations of the scheme on different devices.

Restricting the Power of Tampering. We then initiate a study of tampering attacks under a restricted class of functions. We show that the situation is not hopeless even with a more basic device, that is not enhanced with self-destruct and an external public key. In particular, we show how to achieve ATP security when the adversary is limited to choosing functions f from some restricted, yet useful, classes of functions. The results presented have some practical significance as they address precisely such classes of functions that were successfully used before to attack existing systems [BS97,BDL01]. These include random hardware faults (differential fault analysis), and flipping (or "zapping") specified bits.

PIN-Protected Hardware. The main direct application of our results is in the protection of portable devices such as smartcards or PDAs. Indeed tampering attacks are most likely to be feasible when the device storing the secret key is completely in the hands of the adversary (though one can envision other scenarios). Portable devices are commonly protected by PIN numbers or passwords, to prevent unauthorized access by an adversary. We show how to incorporate PIN numbers in our model and how to make sure that the tampering powers of the adversary are not used to circumvent this extra layer of protection.

Related Work. In addition to the related work mentioned above, there are several works that address the *physical* (as opposed to algorithmic) aspects of tamper-proofing a specific device (typically a smartcard), such as [QS01]. There are many approaches that address security when the *read-proof* (as opposed to tamper-proof) assumption is relaxed in some way. Most relevant in our context, are the recent works of [ISW03], who consider security when the adversary may read part of the inputs going through the circuitry of the device, and of [MR03], who consider a general new model for security against an adversary that can observe arbitrary physical characteristics of a computation ("side channels"). The work of [CGGM00] on resettable zero knowledge can be viewed as a special case of algorithmic tamper-proof security, where the adversary's tampering powers are limited to resetting the randomness.

2 The New Model

2.1 The Device and Adversarial Capabilities

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 A which uses the secret content (we may think of A as the circuitry component).

We say that the system implements a certain function F, if for any input a, $A(sc, a) = F(a)$. We say that A implements a keyed cryptographic function $F(\cdot, \cdot)$, if for every key sk (from the appropriate domain) there exists a setting sc_{sk} of the secret data, such that the system (A, sc_{sk}) implements the function $F(sk, \cdot)$. 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 A, possibly with some additional state information to be passed to the second algorithm. The second is a software setup algorithm: given input sk and A, 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.

Consider A which implements some $F(\cdot, \cdot)$ (e.g., a signature algorithm). We define a *tampering adversary* who can request three commands to be carried out: $\mathsf{Run}(\cdot)$ and $\mathsf{Apply}(\cdot)$, and Setup.

– The command $\mathsf{Run}(a)$, invokes the cryptographic computation A using the software content sc on input a. The output is the output of such computation, i.e., $A(sc, a)$. 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.[1]
– The command $\mathsf{Setup}(sk)$ invokes the software setup algorithm, outputting sc such that the device (A, sc) implements the function $F(sk, \cdot)$.

The device may also have a *self-destruct* capability, called by the algorithm A. 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”.

Incorporating PIN Numbers. Consider the application of the model to smartcards. One goal is to prevent a tampering adversary from learning information about the contents of the card, so that he cannot duplicate and distribute devices with the same functionality (e.g., decryption cards for pay-TV applications). However, it is also often desirable to prevent the adversary from using the functionality of the device *himself*.

Towards this goal, we propose that the card be augmented with a short memorizable PIN, to be entered by the user before any application. That is, a Run query, where it previously took one input, now should take two: the PIN and the input (such as a message m to be signed). The card will only function if the PIN is correct, and, moreover, it will permanently stop functioning (or self-destruct) after a certain (not too big) number of wrong attempts. This requires a counter mechanism.

It is not hard to show that if the adversary cannot tamper with the counter, all our results carry through by considering the PIN as part of the secret key. In Section 5 we show how to achieve ATP security in the setting with PIN, by showing a cryptographic implementation of a counter which is ATP secure, based on one-way permutations or on forward-secure signature schemes. (We thus will not directly deal with the PIN setting in the other parts of the paper.)

[1] It is clear that if this command was allowed any output, then it could just output the secret key. Moreover, we cannot even allow f to produce outputs by making calls to Run, or security would be unachievable. Consider the following attack. The adversary chooses two inputs x_0 and x_1. Given that the secret key on the card is $s_1 s_2 ... s_l$, the function f is “for $i = 1$ to l $\mathsf{Run}(x_{s_i})$”. Clearly, by executing this function, we extract the whole secret key.

2.2 The Notion of Security

Intuitively, we would like that the extra power given to the adversary to be useless. We present definitions of security for signature and encryption schemes, and discuss the simulation technique that we use to achieve these goals.

Signature Cards. The classical definition of security for signature schemes is *security against adaptive chosen-message attack* introduced by [GMR88]. In our terminology, this corresponds to an adversary who is given a public key *pk* and the opportunity to issue as many Run commands as he wants on input messages $m_1, \ldots, m_n$, chosen adaptively, and get the corresponding signatures. Then we say that the scheme is *unforgeable* if the adversary is not able to produce a new message $m \neq m_j$ and a valid signature on it.

In our model we augment the power of the adversary by allowing him to also issue Apply commands. That may change the key pair corresponding to the card; namely, instead of the original key pair (pk, sk), the card may now be working relative to various different key pairs (pk', sk'). Yet, we will consider as *relevant* only Run queries for which the adversary gets a valid signature relative to the original public key *pk*. After interacting with the card, the adversary should not be able to produce a new message m and its valid signature under the public key *pk*. We count as a forgery a pair (m, s) even if m was asked before, but the card outputs an invalid signature because it had an incorrect secret key stored inside as a consequence of some Apply command.

Formally, let $\mathcal{S} = \langle \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver} \rangle$ be a signature scheme, where Gen is the key-generation algorithm, Sig is the signature algorithm, and Ver is the verification algorithm. We say that $\mathcal{S}$ is *algorithmically tamper-proof unforgeable* if for every probabilistic polynomial-time adversary $\mathcal{A}$, there exists a negligible function negl() such that

$$\Pr\left[\begin{array}{l}
(pk, sk) \leftarrow \mathsf{Gen}(1^k); \\
H \leftarrow \{\};\ \ I \leftarrow \{\};\ \ \mathsf{State} \leftarrow \epsilon \\
\text{for } i = 1 \ldots n \\
\quad (\mathsf{State}, \mathsf{Cmd}) \leftarrow \mathcal{A}(\mathsf{State}, pk, H); \\
\quad \textbf{if } \mathsf{Cmd} = \mathsf{Run}(m_i) \textbf{ then } s_i \leftarrow \mathsf{Sig}(sk, m_i); \\
\qquad \textbf{if } \mathsf{Ver}(pk, m_i, s_i) = yes \textbf{ then } I \leftarrow I \cup \{m_i\}; \\
\quad \textbf{if } \mathsf{Cmd} = \mathsf{Setup}(sk_i) \textbf{ then } s_i \leftarrow \mathsf{Setup}(sk_i); \\
\quad \textbf{if } \mathsf{Cmd} = \mathsf{Apply}(f_i) \textbf{ then } sk \leftarrow f_i(sk); \\
\quad H \leftarrow H \cup \{(\mathsf{Cmd}, s_i)\}; \\
(m, s) \leftarrow \mathcal{A}(pk, H); \\
m \notin I \quad \text{and} \quad \mathsf{Ver}(pk, m, s) = yes
\end{array}\right] = \mathsf{negl}(k)$$

Decryption Cards. In the full version of this paper, we give the definition of security for decryption cards. Here, we give an informal sketch of this definition.

Recall that security for encryption schemes comes in at least three different levels: semantic security (security against passive adversary) [GM84], security against lunchtime attacks (security against an adversary who can interact with the decryption oracle during a training stage, before receiving a challenge ciphertext) and security against adaptive chosen ciphertext attacks (CCA-2, where an adversary has a training stage before he receives a challenge ciphertext; once he receives the challenge ciphertext, he can ask the decryption oracle additional queries which must be *distinct* from his challenge ciphertext).

We say that a scheme is *secure against adaptive chosen-ciphertext attack with lunchtime tampering* (or *tamper-proof CCA-2 secure*) if we allow the adversary to issue both Run and Apply commands during the training stage. Then the adversary outputs two messages m_0, m_1 and is given the target ciphertext c, which is the encryption of either m_0 or m_1, chosen at random. Then the adversary can perform *only* Run queries on any ciphertexts other than c. We say that the scheme is secure, if the adversary cannot guess (with probability better than 1/2) the correct decryption of c.

Note that we do not allow the adversary to modify the secret key after the target ciphertext is released. This is because, for a challenge ciphertext c, and Apply query may be of the form "If c decrypts to 0, self-destruct," and therefore it leaks information about the plaintext.

Proofs by Simulation. The above security goal would follow if we were able to prove that this powerful adversary does not learn any more information about the secret key than an adversary who is simply limited by an input/output relationship with the card (because then, if we start from a card secure in the old model, it is also ATP secure).

We can use the concept of simulation to capture the above idea. Our theorems will be proven according to the following approach. We will construct simulators which have only $\mathsf{Run}(\cdot)$ access to the card and $\mathsf{Setup}(\cdot)$ access to the issuer, and make them interact with the tampering adversary. The card is resistant to the tampering powers of the adversary (namely Apply commands) if the adversary is not able to distinguish between the case that he interacts with the real card, and the case that he interacts with the simulator.

3 Enhancing the Physical Design

As stated in the Introduction we augment our model with two additions: public parameters and self destruct, and show that these additions are both necessary and sufficient to achieve ATP security.

These results are shown by exhibiting attacks when the enhancements are not available. First, we show an attack that extracts the entire secret key from any cryptographic algorithm, as long as the card never self-destructs. Then, we show that there is a signature scheme for which there is an attack that can extract the entire secret key, for any implementation without public parameters (even with self-destruct). This can be viewed as a very powerful and simple generalization of previous specific attacks such as [BDL01,BS97].

3.1 Self-Destruct Is Necessary

Testing for Malfunctioning. Intuitively, for any meaningful cryptographic functionality, we should be able to determine, perhaps with some degree of error, whether a given device functions properly, i.e., whether the secret content sc stored on the device gives rise to the right functionality.

If no one can tell that a given device is doing its job correctly, then this device can be replaced with another one, based on a secret content sc' that was generated by a separate invocation of the Setup algorithm, and no one will notice the difference. Hence sc is useless, since sc' works just as well, and there is no need to protect it!

For example, suppose that we have a signature device. Provided that we have the corresponding public key, we can test whether the device functions as prescribed by querying the device for a signature on some message, and then checking the validity of the signature. Similarly, for a decryption device in the public-key setting, whether or not it maintains its functionality can be determined by encrypting many messages and checking whether the device decrypted all of them correctly.

Such test may not be perfect. It is possible that, even though the device does not have the correct secret content sc, but some sc' that is close to the correct content, the device will still pass our test with non-negligible probability. It is easy to come up with schemes that still work, even if their secret keys have been altered slightly, but provide the correct output with decreased probability.

Let us assume that for the functionality at hand, we have a testing procedure Test-Dev such that (1) Test-Dev will always accept when given a device whose sc is correct; (2) if Test-Dev accepts a device with secret content sc' with non-negligible probability ϵ, then discovering sc' constitutes a successful attack on the functionality of the device.

The tests described above for signature and decryption functionalities satisfy these two conditions: discovering sc' that allows to generate correct signatures only an ϵ fraction of the time is still an attack on the signature functionality: now the adversary can create an existential forgery. Similarly, being able to decrypt with an ϵ advantage over random guessing constitutes an attack on a cryptosystem. We show the following claim (informally stated):

Claim. No cryptographic device that can be efficiently tested for malfunctioning, can be made tamper-proof without the self-destruct capability.

Sketch of Proof: The Key-Extraction Procedure. Suppose that we are given a procedure Test-Dev as described above. Suppose that the secret content sc of the device consists of n bits. Finally, suppose that the only operation the attacker is allowed to carry out on the secret component of the device is the $Set(i, b)$ operation that sets the i'th bit of sc to b.

Consider the following procedure, that outputs a candidate value $C = C_1 ... C_n$ for the secret content sc: Initialize $i = 1$. While $i \leq n$: (1) $Set(i, b)$, $b \in \{0, 1\}$ and run Test-Dev. Let b^* be the value such that, when $sc_i = b^*$, Test-Dev accepted more often than when $sc_i = \bar{b}^*$. (2) $Set(i, b^*)$, $C_i = b^*$. (3) Increment i. Upon completing the while-loop, output C.

The value C outputted at the end is identical to the value sc' stored on the device at the end of the procedure. Note that on each iteration of the loop, this procedure maintains the invariant that, with probability $1 - \nu(n)$ (where $\nu(n)$ is a negligible function), the value currently stored in the secret component sc' of the device is accepted by Test-Dev with non-negligible probability. This can be seen by induction: when $i = 1$, the secret component has not been altered yet, and so we are given that Test-Dev accepts. Suppose that $i > 1$. We know that the current sc is accepted by Test-Dev with non-negligible probability. Let b be the current bit sc_i. Suppose that setting $sc_i = \bar{b}$ results in Test-Dev accepting with only negligible probability ν. Then the probability that $b^* = \bar{b}$ is also negligible. Therefore, the device accepts with non-negligible probability when its secret content is C, thus discovering C constitutes a successful attack.

The above attack relies on the adaptiveness of the adversary, who decides which Apply command to issue next, depending on the result of the previous Run command. In the full version of this paper we show that even a non-adaptive adversary can extract the secret key using a *fixed* list of Run and Apply commands. The functions applied simply exchange pairs of specified bits in the string. □

3.2 Hardwiring an External Public Key Is Necessary

Let us start with some intuition. For simplicity, consider a card implementing a signature algorithm $F(\cdot, \cdot)$ (the same techniques will work for decryption cards). Having no public parameters means that there is a software setup function g, such that for any sk', $g(sk')$ outputs a corresponding sc' for a card implementing $F(sk', \cdot)$.[2] In particular, for a given card whose software sc corresponds to some sk, the adversary may be able to replace sc by sc' corresponding to another, "adversarial" sk'. Such an sk' might have the property that when the adversary now issues a Run command, the output will include the original sk, which will allow the adversary to completely break the signature scheme. Indeed, we will show below a signature scheme for which this is exactly the case, and thus there is no ATP method which works for this scheme. It follows that for any general ATP method, the software content cannot be computed solely from the information held by the device. Instead, it must make use of some hardwired cryptographic public key Π, such that the corresponding secret key is needed in the setup of sc.[3] Concretely, we prove that for any general algorithmic tamper-proofing method we can view the hardwired content of the device, A, as a public key for a weak signature scheme, secure against universal forgery (i.e., not all messages

[2] It may seem that this does not grant the adversary any special powers, since he can always compute this by issuing a Setup(sk') command. However, such a Setup command requires that the adversary knows sk'.

[3] It will be convenient to identify Π as the public key of the card manufacturer, though in reality the corresponding secret key may be held by a third party, or distributed among several parties who participate in the setup stage.

can be forged), in the face of a single known-message attack. We refer the reader to [GMR88] for definitions and discussion of these and other security levels for signature schemes.[4]

Towards making the above intuition formal, for any signature scheme F that has a tamper-proof implementation, consider the following weak signature scheme W_F. The key generation algorithm is the device's setup algorithm for a tamper-proof secure card implementing F. The public key Π is set to the entire content of the card's hardware (the algorithm A), and the secret key is the randomness used to generate the public key. The signing algorithm, upon receiving a message m, checks if m is of the form (pk, sk) which are valid public and secret key pairs for F. If so, output sc as appropriate for a tamper-proof-secure card for a user holding (pk, sk). To verify a signature sc on (pk, sk), the verifier checks if a card containing the hardware Π and the software sc would perform correctly as a signature card for (pk, sk) (this can be done by trying to sign). Accept if the check succeeds.

Claim. There exists a secure signature scheme F such that, if its tamper-proof implementation exists, then W_F (described above) is a weak signature scheme secure against universal forgery in the face of a single known-message attack.

Sketch of Proof. It suffices to show a secure signature scheme F and two messages a and b such that given a valid signature of W_F on a, it is computationally infeasible to compute a valid signature on b.
Consider any secure signature scheme comprised of Gen, Sig, Ver and a security parameter k. We define $F = \mathsf{Gen}', \mathsf{Sig}', \mathsf{Ver}'$ as follows.

- Gen' runs Gen to obtain the key pair pk, sk. Let R be a random string of length k. Let $sk' = sk \circ R$ and $pk' = pk$.
- $\mathsf{Sig}'(sk', m)$: for $sk' = sk \circ R$, if $R \neq 0^k$, obtain $\sigma \leftarrow \mathsf{Sig}(sk, m)$. Otherwise, output sk.
- $\mathsf{Ver}'(pk, m, \sigma)$ just runs the algorithm Ver.

The resulting signature scheme F is secure as long as the original one was secure (the probability that $R = 0^k$ happens to be chosen is negligible).
We now turn to W_F, and let $a = (pk, sk \circ R)$ and $b = (pk, sk \circ 0^k)$ for some (pk, sk) generated by Gen and for $R \neq 0^k$. Assume towards contradiction that given a signature $sc = W_F(a)$ one could forge a signature $\hat{sc} = W_F(b)$ by applying some feasible function f. It follows that a card for F containing sc, can be tampered with to produce a forgery. Indeed, the adversary can apply f to the content of the card, thus resulting with $\hat{sc}$ which is valid for the key-pair $(pk, sk \circ 0^k)$. Now the adversary can issue a Run command on any message. The card extracts

[4] We note that security against universal forgery with known-message attacks is not a strong enough notion of security for signature schemes (the standard one, which is our default definition for signature security, is security against existential forgery in the face of adaptive chosen-message attacks [GMR88]). Nevertheless, this weak signature scheme already implies that there is *some* cryptographic key Π which *must* be hardwired into the card, and thus in some sense "certifies" sc.

$\hat{sk} = sk \circ 0^k$, runs $F_{\hat{sk}}$ on the selected message, resulting in the output sk. Now the adversary can forge signatures with respect to F (with respect to the original pk). This contradicts the tamper-proof security of the card. (Note that even if the card contains self-destruct capability, it is not useful since there is no way the card can detect any problem, as $\hat{sc}$ encodes a valid $\hat{sk}$). □

3.3 ATP for Signature and Decryption Schemes

In this section we show how to realize ATP for signature and decryption schemes. Our results meet the definitions of Section 2 in the model enhanced with public parameters and self-destruct (as shown necessary above).

Consider a scheme $\mathcal{F}$ which is either a signature scheme or a public-key encryption scheme, and let sk be a secret signing or decryption key. We would like to store sk in the secret storage of the card, so that an adversary cannot tamper with it in any useful way. A very natural approach is for the card issuer to digitally sign sk and store the signature together with sk in sc, and have the card verify the signature against the issuer's public key before using sk for signing or verifying.

This is indeed the approach that we use, with a few details that need to be taken care of. First, as we already discussed, in order for this to work we must ensure that the card contains the public key of the issuer hardwired into its circuitry, and that the card self-destructs if the check does not succeed. However, it turns out that this is not enough: even if the card issuer uses a signature scheme secure against chosen message attack in the standard sense of [GMR88], we will see an attack that completely recovers sk.

Instead, we will assume that the signature scheme used by the card issuer satisfies a stronger property: not only is it hard to forge a signature on a *new* message, it is hard to forge a *new* signature on an old message. Although this is stronger than the traditional definition, most signature schemes known (c.f., [GMR88,FS87,GQ88,Sch91,CS99,GHR99]) already satisfy it. We call this notion *strong* security against chosen message attack (the formal definition is straight forward and omitted here). The scheme is described in Figure 1.

Theorem 1. *If strong unforgeable signature schemes exist, then there exist ATP unforgeable signature schemes. Specifically, if $\mathcal{I}$ is a strong signature scheme, and $\mathcal{F}$ is a standard signature scheme (unforgeable against adaptive chosen message attack), then the implementation in Figure 1 is an ATP unforgeable signature scheme.*

The proof is given in the full version. Very briefly, the proof proceeds by constructing a simulator that, for any adversary, launches an adaptive chosen message attack on the underlying signature scheme $\mathcal{F}$. The simulator guesses which query of the adversary changes sc, and guesses that this query in fact replaced sc with some pair $(sk', \sigma_{\Pi}(sk'))$ which is one of the queries the adversary issued to the card issuer's signing oracle. For these guesses, the simulator can now an-

Let $\mathcal{I} = (G, \sigma, V)$ be a *strong* signature scheme (used by the card issuer). Let $\mathcal{F}$ be either a signature scheme of the form $\mathcal{F} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ or an encryption scheme of the form $\mathcal{F} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, and let F be the algorithm Sig or Dec, respectively. Let $(\Sigma, \Pi) \leftarrow G(1^k)$ be the secret and public signing keys of the card issuer, and let $(sk, pk) \leftarrow \mathsf{Gen}(1^k)$ be secret and public (signing or encryption) keys for $\mathcal{F}$.

During card setup, Π is hardwired into the card's circuitry (as part of the algorithm below), and the pair $(sk, \sigma_\Pi(sk))$ is stored in the protected memory sc (where $\sigma_\Pi(\cdot) = \sigma(\Sigma, \cdot)$ is the issuer's signing algorithm).

Upon receiving a $\mathsf{Run}(a)$ query, the card performs the following algorithm:
(1) Checks that the storage is of the form $(sk, \sigma_\Pi(sk))$ (using the verification algorithm V).
(2) If so, run $F(sk, a)$ (either signature or decryption) and output the result. Otherwise: self-destruct.

Fig. 1. Tamper-Proofing a Signature or Decryption Scheme

swer all of the adversary's queries, as it knows the content of sc.[5] We then prove that either: (1) the simulator succeeds in producing a forgery with probability polynomially related to that of the adversary (thus breaking the underlying signature scheme), or (2) another simulator can be constructed which produces a forgery to the card issuer's signature scheme.

Theorem 2. *If CCA2 (resp., CCA1) secure encryption schemes and strong unforgeable signature schemes exist, then there exist cryptosystems that are ATP CCA2-secure with lunchtime tampering (resp., ATP CCA1-secure). Specifically, if $\mathcal{I}$ is a strong signature scheme, and $\mathcal{F}$ is a CCA2 (resp., CCA1) secure encryption schemes, then the implementation in Figure 1 is secure against CCA2 with lunch time tampering (resp., tamper-proof CCA1 secure).*

The proof of this theorem is slightly more complicated than the proof for the signature scheme, but it follows the same general idea. It also appears in the full version of this paper.

A strong signature scheme is necessary for this construction. The following attack works in the case where the issuer's signature scheme is unforgeable according to the traditional definition. In other words, assume that it is possible, given a valid message/signature pair (m, σ), to construct a new valid pair (m, σ') with $\sigma' \neq \sigma$.

[5] Intuitively, the only useful change to sc that the adversary can make is by replacing it with a signed pair. This is where the proof requires that the signature scheme for the issuer is strong: this property guarantees that the only signatures the adversary can get are exactly those directly queried, thus allowing the simulator to answer Apply queries from the adversary.

Assume that the issuer's signature scheme has the following property: a signature σ consists of two parts, σ_1 and σ_2. The second component is ignored by the verification algorithm, which really uses only σ_1 to authenticate messages. Thus, the pair (sk, σ) is stored on the card, where $\sigma(\sigma_1, \sigma_2)$ is the manufacturer's signature on sk.

The adversary does the following: first he obtains, from the issuer via a Setup query, a signature $\sigma' = (\sigma_1', \sigma_2')$ on a secret key sk'. Then he replaces the value σ_2 in the card, with the values (sk, σ_1). Note that this will not have any effect on the card's functionality and will not cause the card to self-destruct. Then for each bit sk_i of the secret key he will do the following: if $sk_i = 0$ then do nothing, otherwise replace sk with sk', σ_1 with σ_1', but do not touch the modified σ_2 (this way a record of the old secret key remains). Now by simply querying the card on a given message m, the adversary will be able to see if sk or sk' is being used, and thus if $sk_i = 0$ or not. The adversary then restores sk and σ_1 and repeats the above process for $i + 1$.

On Private Card Setup. In the above solutions, we need to have the issuer's signature on the secret key sk. It is important to note that this does not imply that the card's issuer must know sk. Indeed, one solution is running generic secure two-party protocols [Yao82,GMW87,Gol98], as a result of which the user obtains $\sigma_\Pi(sk)$, and the issuer obtains nothing. The proof of security can be extended to this case as well, by having the simulator *extract* the value sk from the adversary (who no longer sends sk in the clear to the signing oracle). The drawback of this general solution described above is that it may be expensive. Another existing solution is blind signatures. Although providing the desired property that the issuer learns nothing about sk, they are an overkill because neither does the issuer learn σ! A more efficient solution follows.

TIGHT COMMITMENT SCHEME. Recall that a non-interactive commitment scheme Com is (informally) a function such that for all x, for a random r, it is hard to infer any information about x from the value $\mathsf{Com}(x, r)$, and it is infeasible to find (x, r) and (x', r') such that $\mathsf{Com}(x, r) = \mathsf{Com}(x', r')$, and $x \neq x'$. Let Com be a secure non-interactive commitment scheme with a special security property that is similar to the special security scheme of the signature scheme that we use for the device issuer. Namely, not only is it infeasible to open a commitment in two different ways, but it is infeasible to even find a value x and values $r \neq r'$ such that $\mathsf{Com}(x, r) = \mathsf{Com}(x, r')$. Let us call a commitment scheme with this property a *tight* commitment scheme. Pedersen commitment [Ped92] is an example of a tight commitment scheme.

Suppose that we are given a tight commitment scheme with an efficient zero-knowledge proof of knowledge of how to open a commitment. For example, the Pedersen commitment has such a protocol. Then the issuing protocol can be implemented as follows: the user forms a commitment $c = \mathsf{Com}(sk, r)$. He then proves knowledge of the commitment opening. Finally, the issuer sends to the user the value $\sigma = \sigma_\Pi(c)$. The secret component sc of the device will consist of (sk, r, σ).

The proof that a tight commitment scheme is necessary for the security of this construction is similar to the proof that a strong signature is necessary, and is omitted here.

4 ATP via Restricted Classes of Functions

In this section, we consider an adversary that is limited to issuing Apply commands from some restricted, yet useful, class of functions. It turns out that in this case, ATP results are possible even without self-destruct and public parameters.

The results presented below have some practical significance, not only because the model they require is more realistic (e.g., without the self-destruct requirement), but also since we address precisely such classes of functions that were successfully used before to attack existing systems [BS97,BDL01]. These include random hardware faults termed *differential fault analysis*, and flipping (or *zapping*) specified bits. Using our solutions, attacks like the above ones can be protected against.

Since our definition of security requires the functionality of the card to remain secure even when the adversary knows the PIN, we concentrate below on protecting the functionality of the card. Adding PIN protection can be done in a similar manner to our above general solutions.

Differential Fault Analysis. The following results holds for cards with any cryptographic functionality, with neither self-destruct nor a hardwired external key.

Suppose the adversary is limited to the following attack: He specifies two values $p_0, p_1 \in [0,1]$. $f_{p_0,p_1}(x)$ transforms *each* bit x_i of x as follows: if $x_i = b$, leave it that way with probability p_b otherwise flip it. Note that this transformation is exactly the same for each bit. (In information-transmission terms, this transformation can be viewed as sending x through an adversarial memoryless channel.)

Although seemingly benign compared to other attacks we have considered, this is in fact a very powerful attack, invented by Biham and Shamir [BS97] (following Boneh, DeMillo, and Lipton[BDL01]), and known as the *differential fault analysis*. Biham and Shamir use it to recover the entire secret key of a decryption card, such as DES.[6]

Securing a smart-card against such an attack does not require any enhancement to the minimal model. Rather, we can just encode the secret s using an error-detecting code whose distance d is such that $1/2^d$ is negligible. Before running its intended application, the card sees if there are any errors in its storage. If so, it does nothing, otherwise, it works as intended.

It is easy to see that this is sufficient, because if the card's storage changes, it is very unlikely that it will change into a valid encoding; therefore, a simulator that just computes the probability that the card is unusable after a given Apply query and acts accordingly is sufficient for the proof.

[6] Their attack uses asymmetric memory, where $p_0 = 1$, and p_1 is relatively large, but less than 1. That is, a bit which is 1 has a small non-negligible probability to flip.

We note that using error-detecting codes follows the approach alluded to by Boneh, DeMillo, and Lipton [BDL01], who suggest that a cryptographic computation needs to be checked before the output is given.

The Flip Function in the Model Without Self-Destruct. Suppose the external public key of the issuer is wired in, but there is no self-destruct.

Consider the function $\mathsf{Flip}(x, i) = x'$ where x' is equal to x in all the bits except the i^{th} one which is complemented. (This generalizes differential fault analysis by giving the adversary control over which bits to flip, and the certainty that the bit has been flipped). If the adversary is limited to issuing commands of the form $\mathsf{Apply}(\mathsf{Flip}(x, i))$, then the self-destruction property is not required.

Suppose sk is the secret that needs to be stored on the card. Each bit sk_i of sk is encoded using two random bits, $e_{i,1}$ and $e_{i,2}$ such that $e_{i,1} \oplus e_{i,2} = sk_i$. The resulting encoding, $e(sk)$, is then signed by the card manufacturer. The values $(e, \sigma(e))$ are stored on the card.

For each Run command, the card first checks that in its storage, (e, σ), σ is a valid signature on e. If so, the card reconstructs the secret key sk from $e(sk)$ and performs whatever operation is required. Otherwise, the card does nothing.

A sketch of the proof that the latter solution provides ATP security against an adversary limited to flipping bits can be found in the full version of this paper.

5 ATP of Devices Using PIN Numbers

We saw that security of portable devices, such as smart-cards, provides strong motivation for considering ATP security. Indeed, one goal is to prevent an adversary capable of tampering with the device from learning information about its contents, so that such an adversary cannot duplicate and distribute devices with the same functionality (e.g., decryption cards for pay-tv applications). However, it is also often desirable to prevent the adversary from using the functionality of the device *himself*.

To address this problem, we propose that the device be augmented with a short memorizable PIN to be entered by the user before any application. That is, a Run query, where it previously took one input, now should take two: the PIN and the input (such as a message m to be signed). The device will only function if the PIN entered is correct, and, moreover, it will permanently stop functioning (or self-destruct) after a certain (not too big) number of wrong attempts. This can be implemented by a counter which is incremented with every failed attempt. We may consider a model where the device self-destructs once the counter has reached a certain number. A better model, but harder to achieve, is one where the number of *consecutive* wrong attempts is also limited (this limit could then be very small, such as 3).

As a starting point, assume that the adversary cannot tamper with the counter implementation. In this case, all the results we saw so far can be extended to the PIN setting, by considering the PIN as part of the secret key. In particular, in the model with public parameters the signature of the card issuer will be on the secret key *together with the PIN*.

We now turn to addressing the implementation of the counter. Clearly, if the counter is kept in regular tamperable memory, the adversary can recover the PIN by trying all possible PIN combinations, zeroing the counter after each failure. In order to avoid this attack, we suggest two types of counter implementations.

Hardware Implementation. In some situations it may be reasonable to assume that the counter is implemented in hardware, in such a way that the adversary cannot tamper with it. Note that this assumption is more reasonable than assuming all of the secret key is in non-tamperable hardware. Indeed, the counter mechanism is the same for all cards, and is not secret, making it easier to mass produce on hardware. However, the counter (unlike our other public parameters) cannot be implemented by a write-once memory, since it needs to be incremented with every failed attempt. This can be addressed by using an asymmetric type of memory, where incrementing (e.g. by zeroing one bit) is easy, while undoing it is very hard. For example, an operation akin to cutting a wire would be very appropriate. We note that [BS97] also use, in a different way, an asymmetric type of memory where flipping a bit from 1 to 0 is much easier than flipping it from 0 to 1.

Counter Implementation in Tamperable Memory. Consider now the case that the counter can only be implemented in regular (tamperable) memory. Below we provide a solution which is tamper-proof secure, based on any one-way permutation. In the full version we generalize the idea to construct a solution based on any forward-secure digital signature scheme. This generalization provides more flexibility in obtaining good trade-offs among the time and space parameters according to the constraints of the given application, and can allow for better performance overall. All our solutions rely on the mechanisms of self-destruct and public parameters, as described in previous sections. We start by assuming that the model requires a limit M on the total number of failed attempts.

Intuitively, our goal is to construct a counter such that even a tampering adversary can only increment it (or destroy it), but not decrease it. Consequently, such an adversary will not be able to try more than the specified number of guesses for the PIN before the device self-destructs. Our solution will use the existence of one-way permutations, namely, informally, permutations which are easy to compute but hard to invert (for formal definitions see, e.g., [Gol01]).

It works as follows: Let f be a one-way permutation, and let M be the total number of failed attempts we are willing to tolerate. Let R_0 be a random string from the domain of f, generated by the Setup algorithm. For $j = 1, \ldots, M$ we define $R_j = f(R_{j-1})$, namely $R_j = f^j(R_0)$. The setup algorithm will output counter value $(R_0, 0)$ as part of the secret component sc, and the value R_M to be stored and signed together with sk and the PIN. Every failed attempt to enter the PIN will result in replacing the current counter value (R_i, i) with $(f(R_i), i + 1)$.

Every time the device is invoked, it checks the validity of the current value in the counter, and in particular whether $f^{M-j}(R_j) = R_M$. This can generally be done by applying f $M - j$ times. Depending on f, this computation may be done much more efficiently. For instance, assume f is the Rabin function, namely squaring modulo a product of two Blum primes, or the RSA function with a small exponent (both are standard candidates for a one-way permutations). In this case, raising a number to a small power T times can be done efficiently, requiring $O(\log T)$ multiplications.

A more detailed description and proof of security are given in the full version, where we also give a more general implementation based on forward-secure signatures.

Limiting the Number of Consecutive Failed Attempts. Limiting the number of *consecutive* failed attempts to some small number m can be done whenever the adversary is restricted to a certain class of functions, which does not include functions allowing to update the counter (e.g., in our solution above, the one-way permutation f or any power of it). In this case, we can change the device algorithm as follows: Before the validity check, check whether the counter value mod$m = 0$ and if so self-destruct. Also, after the PIN check step, if the PIN is correct, update the counter to the next value which equals 1 *mod m*.

It is not hard to prove that this implementation is ATP secure against a restricted adversary which cannot apply the update function. We leave it as an open problem to construct general tamper-proof counters which limit number of consecutive failed attempts (or conversely to prove that this is not possible in this strong model).

Acknowledgments. We acknowledge Eric Lehman for suggesting the general solution for the non-adaptive key extraction procedure described in the full version of this paper. We also thank Yevgeniy Dodis for valuable discussions. Finally, we thank Charanjit Jutla, Hugo Krawczyk, and Adi Shamir for participating in the lunch-time conversation where the initial seeds for this work were planted.

References

[AARR03] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2003.

[AK96] Ross Anderson and Markus Kuhn. Tamper Resistance - a Cautionary Note. In *Proceedings of the Second Usenix Workshop on Electronic Commerce*, pages 1–11, November 1996.

[BDL01] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14(2):101–119, 2001.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burt Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer Verlag, 1997.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 235–244, 2000.

[CS99] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. In *Proc. 6th ACM Conference on Computer and Communications Security*, pages 46–52. ACM press, nov 1999.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solution to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer Verlag, 1987.

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139. Springer Verlag, 1999.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[Gol98] Oded Goldreich. Secure multi-party computation. Manuscript. Available from `http:www.wisdom.weizmann.ac.il/~oded/pp.html`, 1998.

[Gol01] Oded Goldreich. *Foundations of Cryptography.* Cambridge University Press, 2001.

[GQ88] Louis C. Guillou and Jean-Jacques Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 216–231. Springer Verlag, 1988.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology — CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*. Springer Verlag, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology — CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer Verlag, 1999.

[MR03] Silvio Micali and Leonid Reyzin. Physically observable cryptography. `http://eprint.iacr.org/2003/120`, 2003.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Verlag, 1992.

[QS01] J. J. Quisquater and D. Samyde. Electro magnetic analysis (EMA): Measures and countermeasures for smart cards. In *International Conference on Research in Smart Cards – Esmart*, volume 435 of *Lecture Notes in Computer Science*, pages 200–210, Cannes, France, 2001. Springer Verlag.

[SA03] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2003.

[Sch91] Claus P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.