The Physically Observable Security of Signature Schemes

Alexander W. Dent[1] and John Malone-Lee[2]

[1] Information Security Group
Royal Holloway, University of London
Egham, Surrey, TW20 0EX, UK
a.dent@rhul.ac.uk
http://www.isg.rhul.ac.uk/~alex
[2] Department of Computer Science
University of Bristol
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1UB, UK
malone@cs.bris.ac.uk
http://www.cs.bris.ac.uk/~malone

Abstract. In recent years much research has been devoted to producing formal models of security for cryptographic primitives and to designing schemes that can be proved secure in such models. This line of research typically assumes that an adversary is given black-box access to a cryptographic mechanism that uses some secret key. One then proves that this black-box access does not help the adversary to achieve its task. An increasingly popular environment for cryptographic implementation is the smart-card. In such an environment a definition of security that provides an adversary with only black-box access to the cryptography under attack may be unrealistic. This is illustrated by attacks such as the power-analysis methods proposed by Kocher and others. In this paper we attempt to formally define a set of necessary conditions on an implementation of a cryptosystem so that security against an adversary with black-box access is preserved in a more hostile environment such as the smart-card. Unlike the previous work in this area we concentrate on high-level primitives. The particular example that we take is the digital signature scheme.

1 Introduction

The idea of formally modelling cryptographic security originates in *Probabilistic Encryption* [11], the seminal work of Goldwasser and Micali. Since the publication of that paper, a huge body of research has been devoted to designing schemes that provably meet some definition of security. The basic tenet of all this work is as follows. Start with an assumption about some atomic primitive (for example, the assumption that a particular function is one-way); design a

scheme in such a way that an adversary cannot break the scheme without violating the assumption about the atomic primitive. This property is demonstrated using a complexity-theoretic reduction: one shows that, if an adversary of the scheme exists, then this adversary could be used as a subroutine in an algorithm to violate the assumption about the atomic primitive. Following this procedure one concludes that, if the assumption about the atomic primitive is correct, the scheme satisfies the chosen security definition.

Until recently, the idea of showing that the security of a cryptosystem relies only on the properties of a set of critical atomic primitives has been applied in a "black-box" manner. That is to say, an adversary is given black-box access to an instance of the cryptosystem with a randomly generated secret key. It may have complete control over all inputs, and see all outputs, but it has no knowledge of the internal state of the black-box implementing the cryptosystem. This approach has been used in both the symmetric-key [2] and asymmetric-key [3] settings.

Such models may be appropriate for applications in which all potential adversaries are remote from the legitimate user; however, increasingly cryptography is used in, for example, smart card applications where this is not the case. An attack in this setting was discovered by Kocher *et al.* [14]. They showed how, by monitoring the power consumption of a smart card running DES [9], the secret key could be recovered. These attacks have since become known as *side-channel attacks*; the side channel is the information leaked by the physical implementation of an algorithm – the power consumption in the example above.

There is no avoiding the fact that a physical device performing sensitive operations may leak information. Countermeasures preventing specific attacks can often be designed but they may be expensive. It is therefore imperative that security models are developed that are capable of explicitly isolating the security-critical operations in the implementation of a cryptosystem. This will allow appropriate countermeasures to be focused exactly where they are necessary.

Recently, the first steps have been taken towards formally defining security models for environments where an adversary is able to mount side-channel attacks. One such approach, proposed by Micali and Reyzin [15], is known as *physically observable cryptography*. In this model, every cryptographic operation gives off physical *observables*. For example, these observables could be related to the power consumption [14] or the electro-magnetic radiation emitted by a device during computation [1]. The model does not deal with attacks such as those proposed in [5, 6] in which an adversary actively attempts to alter the operation of a cryptosystem.

The original paper of Micali and Reyzin [15], very properly, concentrates on physically observable cryptography on a "micro" scale: it examines the effect that physical observables have on proofs of security for fundamental primitives such as one-way functions and permutations. They left open the question of how their model could be applied on a more "macro" scale, to primitives such as encryption and signatures – for which there already exist schemes with black-box

security proofs. In this paper we start to address this question. More specifically, we will describe how a set of necessary conditions can be established on the implementations of components of certain cryptosystems so that a black-box security proof holds in the setting where an adversary is given access to the implementation of the cryptosystem.

2 Physically Observable Cryptography

In this section we begin by reviewing the model proposed by Micali and Reyzin [15]. Once we have done so we describe how to extend the model to deal with higher-level primitives than those considered to-date.

2.1 Informal Axioms

The model of Micali and Reyzin [15] requires several "informal axioms". These axioms are assumed to apply to any computational device used to implement the primitives under consideration. We state these axioms briefly below and elaborate on them where necessary. Further details may be found in [15].

Axiom 1 Computation, and only computation, leaks information. Hence, unaccessed memory is totally secure.

Axiom 2 The same computation leaks different information on different devices.

Axiom 3 Information leakage depends upon the chosen measurement.

Axiom 4 Information measurement is local: the information leaked by a component function is independent of the computations made by any of the other component functions.

Axiom 5 All leaked information is efficiently computable from the device's internal configuration. In particular, this means that the leakage is efficiently simulatable if you know all the inputs to a component function.

We note that these axioms cannot be applied indiscriminately to all devices implementing cryptography. For example the cache-based cryptanalysis proposed by Page [16] and developed by Tsunoo *et al.* [18] exploits certain implementations of DES where Axiom 4 above fails to hold. In particular these attacks exploit the fact that, when implemented on a piece of hardware with cache memory, the time taken to access S-box data may vary according to previous S-box access. This time-based side-channel can be exploited to recover the key.

2.2 Computational Model

In the traditional Turing machine (TM) model the tape of the machine is accessed sequentially. This is not consistent with Axiom 1 in Section 2.1: to move from one cell to another the machine may need to scan many intermediate cells thereby leaking information about data that is not involved directly in computation. To overcome this problem Micali and Reyzin augment the model with

random access memory so that each bit can be addressed and accessed independently of all other bits.

As we noted at the end of Section 2.1, Axiom 4 requires us to work in a model where the leakage of a given device is independent of all the computation that precedes it. Micali and Reyzin point out that this means we cannot work in a single TM model [15]. To provide modularity for physically observable cryptography, the model of computation consists of multiple machines. These machines may call one another as subroutines. A second requirement of the model in order to preserve independence of leakage is that each machine has its own memory space that only it can see. To implement this requirement the model is augmented with a *virtual memory manager*.

We now proceed to formalise these concepts following the ideas of Micali and Reyzin [15]. We comment on the differences between our version and the original as and when they occur.

Abstract Virtual-Memory Computers An *abstract virtual-memory computer* (abstract VMC or simply VMC for short) consists of a collection of special Turing machines. We call these machines *abstract virtual-memory Turing machines* (abstract VTMs or simply VTMs for short). We write $A = (T_1, \ldots, T_n)$ to denote the fact that abstract virtual-memory computer A consists of abstract VTMs $T_1, \ldots, T_n$, where T_1 is distinguished: it is invoked first and its inputs and outputs coincide with those of A.

The specialisation of this model that we will use will have the following features. We will assume that T_1 calls each of $T_2, \ldots, T_n$ in turn; that none of these is called more than once; and that T_i does not call T_j if $i \neq 1$. We will demonstrate these properties with a concrete example in Section 3.2.

Virtual-Memory Management In addition to the standard input, output, work and random tapes of a probabilistic TM, a VTM has random access to its own *virtual address space* (VAS). There is in fact a single *physical address space* (PAS) for the entire VMC. A *virtual-memory manager* takes care of mapping the PAS to individual VASs. Complete details of how the virtual-memory manager works may be found in [15]. We only mention here the properties that we require.

Each VTM has a special VAS-access tape. To obtain a value from its VAS it simply writes the location of the data that it wishes to read on its VAS-access tape. The required data then appears on the VTMs VAS-access tape. The mechanics of writing to VAS are equally simple: a VTM simply writes the data and the location that it wishes it to be stored at on its VAS-access tape.

The only special requirement that we have in this context is that the virtual-memory manager should only remap memory addresses, but never access the data.

Input and Output The implementations of the cryptosystems that we will consider in this paper will have the following form. The only VTM that will take

any external input or produce any external output is T_1. So, if $A = (T_1, \ldots, T_n)$, the external input and external output of A are exactly those of T_1.

At the start of the computation the input is on the start of T_1's VAS. At the end of the computation the output occupies a portion of T_1's VAS. Further details of this may be found in [15].

Calling VTMs as Subroutines As we mentioned above, for the implementations of the cryptosystems that fit our version of the model, the only VTM that will call any other VTM is T_1. This VTM has a special *subroutine-call tape*. To call a subroutine T_i, T_1 specifies where the input for T_i is located on its – T_1s – VAS. It also specifies where it wants the output. The virtual memory manager takes care of mapping address locations.

2.3 Physical Security Model

The physical implementation of a cryptosystem, itself modelled as a virtual-memory computer $T = (T_1, \ldots T_n)$, will be modelled as a *physical virtual-memory computer* (physical VMC). This is a collection of *physical virtual Turing machines* (physical VTMs) $\mathcal{P} = (P_1, P_2, \ldots, P_n)$. Each physical VTM P_i consists of a pair $P_i = (L_i, T_i)$ where T_i is a VTM and L_i is a *leakage function* associated with the physical implementation of T_i.

A leakage function is used to model the information leaked to an adversary by a particular implementation. As defined by Micali and Reyzin [15], it has three inputs: (1) the current configuration of the physical VTM under consideration; (2) the setting of the measuring apparatus used by the adversary; and (3) a random string to model the randomness of the measurement. Further details may be found in [15].

We say that an adversary *observes* a physical VTM if it has access to the output of the leakage function for the VTM and can decide upon the second input: the measuring apparatus to use. As in [15], we denote the event that an adversary A outputs y_A after being run on input x_A and observing a physical VTM P_i being executed on an input $x_{\mathcal{P}}$ and producing an output $y_{\mathcal{P}}$, by

$$y_{\mathcal{P}} \leftarrow \mathcal{P}(x_{\mathcal{P}}) \rightsquigarrow A(x_A) \rightarrow y_A.$$

3 A Definition of Security for Physical Virtual-Memory Computers

Recall from Section 2.2 that we are interested in implementations of cryptosystems with the following form. The cryptosystem must be susceptible to being modelled as a VMC $T = (T_1, \ldots, T_n)$ such that the input and output of T correspond exactly to the input and output of T_1, and T_1 calls each T_i once and once only. We will also require in our model that T_1 is not responsible for any computation itself. It simply maps addresses in its VAS to the VAS-access tapes

of the VTMs $T_2, \ldots, T_n$. It therefore follows from Axiom 1 in Section 2.1 that T_1 does not leak any side-channel information to an adversary.

The final point that we should make about T_1 is that it has the secret key of the cryptosystem concerned hard-coded into its VAS before any adversary is given access to the implementation. We will also assume that the secret key for the cryptosystem is of the form $sk = (sk_2, \ldots, sk_n)$ where the sk_i is the secret key material used by T_i. At present our model can only deal with the case where the sk_i are distinct and generated independently of one another by the key generation algorithm for the scheme (modulo some common parameter such as a group or the bit-length of an RSA modulus). A good example of a cryptosystem with such a property is the CS1a scheme of Cramer and Shoup [8]. We will comment on why this property is necessary at the appropriate point in our proof of security.

At this point we will provide an example to illustrate the concepts that we are introducing. The example that we will use is a version of the PSS variant [4] of the RSA signature scheme [17]. We also introduce this example because we will prove our result specifically for the case of digital signature schemes.

Before going into details of the specific scheme we remind ourselves of the definition of a signature scheme and the definition of security for signature schemes.

3.1 Signature Schemes and their Security

A signature scheme *SIG* consists of three algorithms *KeyGen*,*Sig* and *Ver*. These have the following properties.

- The *key generation algorithm KeyGen* is a probabilistic algorithm that takes as input a security parameter 1^k and produces a public/secret key pair (pk, sk).

- The *signing algorithm Sig* takes as input the secret key sk and a message m; it outputs a signature s. The signing algorithm may be probabilistic or deterministic.

- The *verification algorithm Ver* takes as input a message m, the public key pk and a purported signature s; it outputs 1 if s is a valid signature on m under pk, otherwise it outputs 0.

Let us also recall the standard definition of (black-box) security for a signature scheme: *existential unforgeability under adaptive chosen message attack* [12]. This notion is described using the experiment below involving a signature scheme $SIG = (KeyGen, Sig, Ver)$, an adversary $\mathcal{A}$ and a security parameter k.

$\mathbf{Exp}^{\mathrm{euacma}}_{SIG,\mathcal{A}}(k)$

Stage 1 The key generation algorithm *KeyGen* for the signature scheme in question is run on input of a security parameter 1^k. The resulting public key is given to adversary $\mathcal{A}$.

Stage 2 Adversary $\mathcal{A}$ makes a polynomial number (in the security parameter) of queries to a signing oracle. This oracle produces signatures for $\mathcal{A}$ on messages of its choice. The oracle produces these signatures using the secret key generated in Stage 1 and algorithm *Sig*.

Stage 3 Adversary $\mathcal{A}$ attempts to output a message and a valid signature such that the message was never a query to the signing oracle in Stage 2. If it succeeds in doing this we say that $\mathcal{A}$ *wins* and we output 1, otherwise we output 0.

The adversary $\mathcal{A}$'s advantage is the probability that it wins in the above. We say

$$\mathbf{Adv}^{\text{euacma}}_{SIG,\mathcal{A}}(k) = \Pr[\,\mathbf{Exp}^{\text{euacma}}_{SIG,\mathcal{A}}(k) = 1\,] \tag{1}$$

If, for all probabilistic polynomial time $\mathcal{A}$, (1) is a negligible function k then *SIG* is said to be *existentially unforgeable under adaptive chosen message attack*.

Having defined the black-box version of the definition of security for a signature scheme, it is a straightforward manner to define the physically observable analogue. To do this we simply replace the black-box queries that $\mathcal{A}$ has in Stage 2 of the above definition with *physically observable queries* as defined in Section 2.3. In other words, we give $\mathcal{A}$ access to an oracle for the leakage function of the system. The adversary supplies the measurement information that is input to the leakage function, and the oracle uses the machines current state and randomness as the other inputs. Note that this model reduces to the standard chosen message attack model if the measurement information consists of a message, and the leakage function returns a randomly generated signature on that message. We denote the experiment where $\mathcal{A}$ has access to a leakage oracle $\mathbf{Exp}^{\text{eupoacma}}_{T(SIG),\mathcal{A}}(k)$ and define the advantage of an adversary $\mathcal{A}$ in this game by

$$\mathbf{Adv}^{\text{eupoacma}}_{T(SIG),\mathcal{A}}(k) = \Pr[\,\mathbf{Exp}^{\text{eupoacma}}_{T(SIG),\mathcal{A}}(k) = 1\,]. \tag{2}$$

In the above $T(SIG)$ denotes the fact that we are concerned with the actual implementation T of the scheme *SIG* rather than *SIG* itself.

3.2 The RSA-PSS Signature Scheme

In order to make the concepts we are discussing more concrete, we give an example: the RSA-PSS signature scheme [4, 17].

The signing algorithm for RSA-PSS involves formatting the message and then performing modular exponentiation using a secret exponent. Here we describe how this process can be decomposed into its various subroutines in our model.

Suppose that signing of n-bit messages is performed using a k-bit RSA modulus N and a secret exponent d. This requires two hash functions

$$H : \{0,1\}^{n+k_0} \to \{0,1\}^{k_1} \text{ and } G : \{0,1\}^{k_1} \to \{0,1\}^{n+k_0} \tag{3}$$

where $k = n + k_0 + k_1 + 1$.

The signing procedure will be modelled as a VMC $T = (T_1, \ldots, T_5)$. We describe the roles of the various VTMs below. The message to be signed is m.

- T_2 requires no input. It simply generates a k_0 bit random number r and writes r to the appropriate location in its VAS.

- T_3 requires m and r as input. The addresses of these are provided by T_1 and the virtual memory manager does the appropriate address mapping. Having recovered m and r from its VAS, T_3 computes $u = H(m||r)$ and writes u to the appropriate location in its VAS.

- T_4 requires m, r and u as input. The addresses of these are provided by T_1 and the virtual memory manager does the appropriate address mapping. Having recovered m, r and u from its VAS, T_4 computes $v = G(u) \oplus (m||r)$ and writes v to the appropriate location in its VAS.

- T_5 requires u, v, d and N as input. The addresses of these are provided by T_1 and the virtual memory manager does the appropriate address mapping. Having recovered u, v, d and N from its VAS, T_5 converts the bit-string $0||u||v$ into an integer x and computes $s = x^d \bmod N$. It then writes s to the appropriate location in its VAS.

- T_1 takes external input m and has d and N hard-coded into its VAS. Its role is simply to write the appropriate addresses from its VAS to its subroutine-call tape and invoke $T_2, \ldots, T_5$ in turn (the appropriate addresses are implicit in the descriptions of $T_2, \ldots, T_5$ above). The final job of T_1 is to output the data from the portion of its VAS where s is located after T_5 has been called.

Note that in the description of T_5 we include the 0 in the string $0||u||v$ to insure that, once the string is converted into an integer, that integer is less than N.

3.3 Definition of Security for Implementations

In this paper we are starting with the assumption that a cryptosystem satisfying the constraints that we outlined above is secure in a black-box setting. For a signature scheme such as RSA-PSS this means existential unforgeability under adaptive chosen message attack. Our aim is to provide sufficient conditions on the various components of the implementation such that security in the black-box setting translates into security of the physical implementation.

Let us consider an implementation $T = (T_1, \ldots, T_n)$ of some cryptosystem with public key pk and secret key $sk = (sk_2, \ldots, sk_n)$. For $i = 2, \ldots, n$ we let

$$\underline{x}_i \leftarrow T|^i(m, pk, sk_i)$$

denote the action of executing T and halting after T_i has been run. We denote by $\underline{x}_i$ the vector of outputs from $T_i, \ldots, T_2$.

Also, for $i = 3, \ldots, n$ we let

$$s \leftarrow T|_i(m, \underline{x}_{i-1}, pk, sk_i)$$

denote the action of executing T from the point of T_i onwards where $\underline{x}_{i-1}$ denotes the vector of outputs produced by $T_{i-1}, \ldots, T_2$. Note that in a complete execution of T, T_1 would know the locations of $\underline{x}_{i-1}$ VAS. Using these it would be able to provide the necessary input for $T_i, \ldots, T_n$.

We say that T_i is secure if there exists a polynomial-time simulator S_i such that no adversary $\mathcal{A}$ can win the following game ($\mathbf{Exp}^{\mathrm{lor}}_{\mathcal{A},T_i}(k)$) with probability significantly greater than 1/2. Note that, since S_i produces no output that is used by any later process, it can be thought of as either a VTM with a leakage function or a function that simulates the leakage function of the VTM T_i. In the description below, the symbol q represents an upper bound on the number of *queries* that $\mathcal{A}$ is able to make to the implementation that it is attacking. In the description of $\mathbf{Exp}^{\mathrm{lor}}_{\mathcal{A},T_i}(k)$ below lor is an acronym for left-or-right; either $\mathcal{A}$ ends up being run in the experiment on the left or in the experiment on the right. This idea has been used extensively in the literature, see [2] for example.

Experiment $\mathbf{Exp}^{\mathrm{lor}}_{\mathcal{A},T_i}(k)$
Run the key generation algorithm for the scheme
  on input 1^k to produce a key-pair (pk, sk)
Prepare implementation of T_i using sk_i (where $sk = (sk_2, \ldots, sk_n)$)
Choose b at random from $\{0, 1\}$
If $b = 1$ run $\mathbf{Exp}^{\mathrm{real}}_{\mathcal{A},T_i}(k)$
If $b = 0$ run $\mathbf{Exp}^{\mathrm{sim}}_{\mathcal{A},T_i}(k)$
If $b' = b$ return 1, otherwise return 0

Experiment $\mathbf{Exp}^{\mathrm{real}}_{\mathcal{A},T_i}(k)$
$state \leftarrow (pk, \{sk_l\}_{l \neq i})$
for$(j = 0,\ j < q,\ j = j + 1)$
  {
    $m_j \leftarrow \mathcal{A}(state)$
    $\underline{x}_{i-1} \leftarrow T|^{i-1}(m_j, pk, sk_i)$
    $x_i \leftarrow T_i(\underline{x}_{i-1}, pk, sk_i)$
      $\rightsquigarrow \mathcal{A}(state) \rightarrow state$
    $\underline{x}_i = (x_i, \underline{x}_{i-1})$
    $s_j \leftarrow T|_{i+1}(m_j, \underline{x}_i, pk, sk_i)$
    $state \leftarrow \mathcal{A}(state, s_j)$
  }
$b' \leftarrow \mathcal{A}(state)$

Experiment $\mathbf{Exp}^{\mathrm{sim}}_{\mathcal{A},T_i}(k)$
$state \leftarrow (pk, \{sk_l\}_{l \neq i})$
for$(j = 0,\ j < q,\ j = j + 1)$
  {
    $m_j \leftarrow \mathcal{A}(state)$
    $\underline{x}_i \leftarrow T|^{i}(m_j, pk, sk_i)$
    $Null \leftarrow S_i(m_j, pk)$
      $\rightsquigarrow \mathcal{A}(state) \rightarrow state$
    $s_j \leftarrow T|_{i+1}(m_j, \underline{x}_i, pk, sk_i)$
    $state \leftarrow \mathcal{A}(state, s_j)$
  }
$b' \leftarrow \mathcal{A}(state)$

Note that we assume that $\mathcal{A}$ is given all the secret-key material for all the T_l with $l \neq i$. This is crucial for our security proof.

We define

$$\mathbf{Adv}^{\mathrm{lor}}_{T_i,\mathcal{A}}(k) = |2 \cdot \Pr[\,\mathbf{Exp}^{\mathrm{lor}}_{T_i,\mathcal{A}}(k) = 1\,] - 1|.$$

We say that it is possible to implement T_i securely if there exists and S_i such that, for all probabilistic polynomial-time $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{lor}}_{T_i,\mathcal{A}}(k)$ is a negligible function of k. Henceforth we refer to S_i as the *physical simulator* for T_i.

3.4 Result

In this section we present our result for an implementation $T = (T_1, \ldots, T_n)$ of a signature scheme *SIG*. Our result holds in the model described in Section 2. We state it formally below.

Theorem 1 Suppose that $\mathcal{A}$ is an adversary that succeeds in forging a *SIG* signature by using a physical adaptive chosen message attack on the implementation $T = (T_1, \ldots, T_n)$. We show that there are adversaries $\mathcal{A}_2, \ldots, \mathcal{A}_n, \mathcal{A}'$ such that

$$\mathbf{Adv}^{\text{eupoacma}}_{T(SIG),\mathcal{A}}(k) \leq \mathbf{Adv}^{\text{lor}}_{T_2,\mathcal{A}_2}(k) + \ldots + \mathbf{Adv}^{\text{lor}}_{T_n,\mathcal{A}_n}(k) + \mathbf{Adv}^{\text{euacma}}_{SIG,\mathcal{A}'}(k). \quad (4)$$

The execution times of $\mathcal{A}_2, \ldots, \mathcal{A}_n$ and $\mathcal{A}'$ are all essentially the same as that of $\mathcal{A}$ and the number of oracle calls made by each one is the same as the number make by $\mathcal{A}$.

Proof. To prove our result we define a sequence $\mathbf{G}_0, \ldots, \mathbf{G}_{n-1}$ of modified attack games. The only difference between games is how the environment responds to $\mathcal{A}$'s oracle queries. For any $0 \leq i \leq n-1$, we let W_i be the event that $\mathcal{A}$ succeeds in producing a valid forged signature in game $\mathbf{G}_i$. This probability is taken over the random choices of $\mathcal{A}$ and those of $\mathcal{A}$'s oracles.

The first game $\mathbf{G}_0$ is the real attack game in which $\mathcal{A}$ my physically observe the execution of $T = (T_1, \ldots, T_n)$ on chosen input m. It may do this q times and may choose its inputs adaptively based on information gleaned from previous queries. From the definition of $\mathbf{Adv}^{\text{eupoacma}}_{T(SIG),\mathcal{A}}(k)$ it follows that

$$\Pr[\, W_0 \,] = \mathbf{Adv}^{\text{eupoacma}}_{T(SIG),\mathcal{A}}(k). \quad (5)$$

In the second game $\mathbf{G}_1$ we replace the implementation of T_n with which $\mathcal{A}$ interacts with S_i - the physical simulator for T_n. We claim that there exists a polynomial time adversary $\mathcal{A}_n$, whose execution time is essentially the same as that of $\mathcal{A}$, such that

$$|\Pr[\, W_0 \,] - \Pr[\, W_1 \,]| \leq \mathbf{Adv}^{\text{lor}}_{T_n,\mathcal{A}_n}(k). \quad (6)$$

It is easy to construct such an adversary $\mathcal{A}_n$. According to the definition of $\mathbf{Exp}^{\text{lor}}_{T_i,\mathcal{A}_n}(k)$, $\mathcal{A}_n$ is given as input $pk, (sk_2, \ldots, sk_{n-1})$. Now, to construct $\mathcal{A}_n$ we simply prepare implementations of $T_2, \ldots, T_{n-1}$ which we can use to simulate $\mathcal{A}$'s view in its attack on T. Now, in the case where the bit hidden form $\mathcal{A}_n$ is 1, $\mathcal{A}$ is run by $\mathcal{A}_n$ in exactly the same way that the former would be run in game $\mathbf{G}_0$. Also, in the case where the bit hidden from $\mathcal{A}_n$ is 0, $\mathcal{A}$ is run by $\mathcal{A}_n$ in exactly the same way that the former would be run in game $\mathbf{G}_1$. It follows that any perceptible difference in $\mathcal{A}$'s performance in the transition from game $\mathbf{G}_0$ to game $\mathbf{G}_1$ would provide us with an adversary $\mathcal{A}_n$ of the implementation of T_i.

We repeat this procedure, replacing T_{n-1} with S_{n-1} and so on, until we have replaced T_2 with S_2. For $j = 1, \ldots, n-2$ this gives us

$$|\Pr[\, W_j \,] - \Pr[\, W_{j+1} \,]| \leq \mathbf{Adv}^{\text{lor}}_{T_{n-j},\mathcal{A}_{n-j}}(k), \quad (7)$$

where $\mathcal{A}_{n-j}$ is an adversary of the implementation of T_{n-j} whose execution time is essentially the same as that of $\mathcal{A}$.

Finally, once this process has been completed, $\mathcal{A}$ does not have access to any genuine physically observable components of T. We may therefore consider that in game $\mathbf{G}_{n-1}$, $\mathcal{A}$ is an adversary of the scheme in the black-box setting. We conclude that there exists some adversary $\mathcal{A}'$ such that

$$\Pr[\,W_{n-1}\,] \leq \mathbf{Adv}^{\mathrm{euacma}}_{SIG,\mathcal{A}'}(k). \tag{8}$$

The result now follows from (5), (6), (7) and (8).

4 Conclusion

We have provided a set of sufficient conditions for the implementation of a cryptosystem to be no less secure than the abstract cryptosystem itself. The sufficient conditions come in two parts. Firstly we assume that the implementation of the cryptosystem fits a computational model based on that proposed by Micali and Reyzin [15]. This is the model that we described in Section 2. Secondly, in Section 3.3, we gave an indistinguishability-based definition of security that should be satisfied by the subroutines used by the implementation of the cryptosystem. In Theorem 1 we proved that, in this model, if the subroutines satisfy our definition then the implementation is no less secure than the cryptosystem itself.

The model that we have considered here is designed to cope with attacks that are in some sense passive; the adversary is assumed not to actually tamper with the internal workings of the implementation. This means that the model does not say anything about attacks such as the fault attacks of Biham *et al.* [5]. However, there has been some preliminary research into the possibility of a theoretical model to treat such cases [10].

Although we believe that a formal security treatment for side-channel environments may provide valuable insight, we recognise that it may be very difficult to prove indistinguishability results about implementations in a complexity-theoretic sense. An orthogonal line of research is to develop concrete tests that can be applied to implementations in order to assess and compare their security. For example, Coron *et al.* have proposed a set of statistical tests for the detection of leaked secret information [7]. A second interesting approach is the hidden-markov model technique proposed by Karlof and Wagner [13]. The aim of this technique is to infer information about a secret key based on side-channel information.

While our theoretical approach provides us with sufficient conditions for secure implementations, the statistical techniques above provides us with necessary conditions. By working on the problem from both these ends (so to speak), one hopes to over time converge on a realistic, well-defined set of conditions that a secure implementation of a cryptosystem should satisfy.

References

1. 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, *Chryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45, Redwood Shores, CA, USA, August 13–15 2003. Springer-Verlag, Berlin, Germany.
2. Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science*, pages 394–403, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.
3. Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45, Santa Barbara, CA, USA, August 23–27, 1998. Springer-Verlag, Berlin, Germany.
4. Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and rabin. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, Saragossa, Spain, May 12–16, 1996. Springer-Verlag, Berlin, Germany.
5. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 513–525, Konstanz, Germany, May 11–15, 1997. Springer-Verlag, Berlin, Germany.
6. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51, Konstanz, Germany, May 11–15, 1997. Springer-Verlag, Berlin, Germany.
7. Jean-Sébastien Coron, David Naccache, and Paul Kocher. Statistics and secret leakage. *ACM SIGOPS Operating Systems Review*, 3(3):492–508, August 2004.
8. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.
9. Federal Information Processing Standards Publication 46-3 (FIPS PUB 46-3): Data Encryption Standard, October 1999.
10. Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 258–277, Cambridge, MA, USA, February 19–21, 2004. Springer-Verlag, Berlin, Germany.
11. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
12. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.
13. Chris Karlof and David Wagner. Hidden markov model cryptanalysis. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Chryptographic Hardware and*

Embedded Systems – CHES 2003, 5th International Workshop, volume 2779 of *Lecture Notes in Computer Science*, page 2, Cologne, Germany, September 8–10 2003. Springer-Verlag, Berlin, Germany.

14. Paul 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, Santa Barbara, CA, USA, 1999.
15. Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296, Cambridge, MA, USA, February 19–21, 2004. Springer-Verlag, Berlin, Germany. Full version available at `http://eprint.iacr.org2003/120`.
16. Dan Page. Theoretical use of cache memory as a cryptanalytic side-channel. Technical Report CSTR-02-003, University of Bristol Department of Computer Science, June 2002.
17. Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.
18. Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, Maki Shigeri, and Hiroshi Miyauchi. Cryptanalysis of DES implemented on computers with cache. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Chryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop*, volume 2523 of *Lecture Notes in Computer Science*, pages 62–ŋ76, Redwood Shores, CA, USA, August 13–15 2003. Springer-Verlag, Berlin, Germany.