Information Leakage Attacks Against Smart Card Implementations of Cryptographic Algorithms and Countermeasures

A Survey

Erwin Hess[1], Norbert Janssen[2], Bernd Meyer[1], and Torsten Schütze[1]

[1] Siemens AG, Corporate Technology, 81730 München, Germany, Email:
{erwin.hess|bernd.meyer|torsten.schuetze}@mchp.siemens.de

[2] Infineon Technologies AG, CC PD 1, 81617 München, Germany, Email:
norbert.janssen@infineon.com

Abstract. Every practical implementation of a cryptographic algorithm represents a physical device possessing potential side channels not covered by the security models of theoretical cryptography. Hence, even provable secure cryptographic algorithms may be attacked due to leakage of information. Smart cards and security ICs are often used as tamper-proof security devices. To prevent an attacker from exploiting easily accessible information like power consumption, running time, input-output behavior under malfunctions caused, i. e., by irregular clocking, radiation, power peaks, special precautions have to be taken. Commonly used countermeasures against information leakage are the reduction of the signal-to-noise ratio using special implementation techniques for hardware and software and the decorrelation of secret internal data from the channels observable by an attacker.
In this contribution we survey the basic concepts of known attacks based on information leakage, i. e., timing attack, differential fault analysis, SPA, and DPA, and the countermeasures proposed in the literature. These methods comprise hardware design techniques and the design and implementation of modifications of cryptographic algorithms.

1 Introduction

In traditional cryptography, cipher systems have been analyzed by modeling cryptographic algorithms as ideal mathematical objects. If any attacker is able to find any sophisticated strategy, this would lead to a low-complexity algorithm for the solution of a problem commonly believed to be intractable. This can be shown by complexity-theoretical reductions. Trust gained in this way is independent of the concrete implementation of a cryptographic algorithm.

On the other hand, any practical implementation represents necessarily a physical device with additional properties not covered by the mathematical model. This allows an adversary to break even provably secure algorithms by manipulating the device in an unintended way. In general, the methods for these invasive attacks rely on special knowledge and are done with the help of costly equipment [2, 1, 26]. This was considered to be a serious danger only for classified or military applications. Defenses against attacks like probing [19], re-engineering, and memory readout techniques [26] etc., comprise protecting shields and chip coatings, top layer metal grids, sensors to defend glitch attacks against clock and voltage supply, and are a common part of todays smart cards.

During the last four years, a new class of attacks became public [5, 3, 22, 23]. These attacks exploit easily accessible information like power consumption, running time, and input-output behavior under malfunctions, and can be mounted by anyone using low-cost equipment. The adversary targets specific implementation details which may reveal information about secret data or the internal state of a device. These so-called *side-channel attacks* amplify and evaluate the leaked information with the help of statistical methods and are often much more powerful than classical

cryptanalysis. Examples show that a very small amount of side-channel information is enough to completely break a cryptosystem [21].

In this paper, we give an overview of the existing side-channel attacks presented in the literature. In Section 2, we survey the basic principles of *simple power analysis*, *differential failure analysis*, *timing attacks*, and *differential power analysis*. We give examples for the vulnerabilities of specific algorithms and implementations. In Section 3, we discuss possible countermeasures proposed in the literature.

2 Principles of attacks

2.1 Simple power analysis—SPA

Simple Power Analysis (SPA) is the first developed—but also the least powerful—of the information-leakage attacks described in this paper. It involves the direct interpretation of power consumption measurements collected during cryptographic operations.

Messerges et. al. [28], for example, present actual results from monitoring power signals. They describe two different types of information which can be derived from the activity on the data bus: Hamming weight information (leakage occurs, for example, in a precharged bus design) and transition count information (number of switching gates that are driven by the data bus).

The SPA attack can be performed with the following steps: first, a *single* encryption is run. The power consumption data of the device are recorded and, finally, the key bits can either directly be seen or easily be computed. If an attacker can determine where certain instructions are being executed, it can be relatively easy to extract useful information. SPA attacks can, for example, be used to break simple RSA implementations by revealing differences between multiplying and squaring, see the square-and-multiply algorithm in Section 2.3. The processing of 0 and 1 bits of the secret exponent can directly be recognized in the power trace. Messerges et. al. [28] give a detailed account on how to break DES using the Hamming weight data of the key bytes.

In SPA attacks, the aim is essentially to guess from the power trace which particular instruction is being executed at a certain time and what values the input and output have. Therefore, the adversary needs an exact knowledge of the implementation to mount such an attack.

2.2 Differential fault analysis—DFA

Differential fault analysis exploits weaknesses of the robustness of cryptographic algorithms under malfunctioning. Many cryptographic systems leak information which can be used to reconstruct secret keys hidden in a tamper-proof device if computational errors affect the computation. The adversary can (possibly repeatedly) compute correct and perturbed results, compare them, and try to deduce the keys.

Mainly two different fault models for DFA attacks are discussed in the literature: in the *transient fault model* or *random register fault model* it is assumed, that an attacker can flip a single bit (or a small number of bits) of a value contained in a register. The moment at which the bit flips and the position of the memory cell of the changed bit are randomly distributed among the overall computation time and memory locations. Transient faults can, for example, be caused by irregular clocking, radiation, or peaks in the power supply voltage. Transient faults change only data stored in volatile memory, but do not change or destroy the device or data stored in non-volatile memory. In the *persistent fault model* the adversary is allowed to change a single bit (or a small number of bits) in non-volatile memory (i. e., erasing EEPROM cells using ultraviolet light or X-rays, setting or clearing EEPROM cells using micro-probing, destroying ROM cells or introducing stuck-at faults using laser cutters) or to change the device (i. e., cutting wires).

In [5] the authors describe several attacks against public-key cryptosystems in the transient fault model: implementations of RSA which use chinese remaindering (CRT) to speed up signing may be completely broken if an error occurs in one of the exponentiation steps for each prime factor of the modulus. In this case, the difference between the correct result E and the faulty

result $\tilde{E}$ is divisible by a prime factor of the modulus N with high probability, hence, computing $\gcd(E - \tilde{E}, N)$ factors the modulus. Even in implementations of RSA without CRT, the secret exponent can be deduced from transient faults. If the error affects one of the last iterations of the exponentiation, see Section 2.3, it is possible to find the exact position of the flipped bit and part of the secret key by exhaustive search. The Fiat-Shamir and Schnorr-identification schemes can be attacked with register faults while the prover is waiting for a challenge.

Biham and Shamir describe in [3] how the DES encryption algorithm can be broken using transient faults. In addition, they introduce a persistent fault model which assumes that an attacker can erase an arbitrary bit of the key. This model is justified by the asymmetric behavior of memory cells: if the fault is induced by external radiation, then the charges are more likely to leak out of the gate than to be forced into the gate. Biham and Shamir propose to compute a sequence of encryptions with keys having successively erased bits. This sequence allows one to find the positions and the values of the erased bits and, thus, to reconstruct the secret key in reverse. The attack is applicable, too, if the cryptosystem is completely unknown to the attacker. Biham and Shamir show several variations of these attacks under the assumption that the attacker is able to cut wires or to destroy gates.

The transient and persistent fault models have been criticized for being purely theoretical. In [2] it is argued that a random one-bit error would be more likely to crash the processor of the tamper-proof device or yield an uninformative error than to produce a faulty ciphertext. Instead, *glitch attacks* which have already been used in the pay-TV hacking community, are presented in [2, 1, 26] as a more practical approach to non-invasive (differential) fault analysis. The attacker applies a rapid transient in the clock or the power supply to the chip. Due to different delays in various signal paths, this affects only some signals and by varying the parameters of the attack, the CPU can be made to execute a number of wrong instructions. By carefully choosing the timing of the glitch, the attacker can read out secret data, by-pass security checks, or weaken cryptographic algorithms by DFA attacks.

2.3 Timing attacks

The execution time of cryptographic algorithms often shows slight differences dependent on the input of the algorithm. This data dependent variation is due to performance optimization, conditional statements, handling of special cases, cache misses, and a variety of other causes. Since an adversary can easily measure the execution time of a tamper-proof device like a smart card with high accuracy, the dependence between public inputs, secret data hidden in the device, and changes in execution time can be used to derive valuable information about the secret data [11, 22, 20, 25, 17, 18]. In the following, we describe the principles of timing attacks according to [11] and [22].

In order to mount a timing attack, it must be possible to start repeatedly the device with different inputs (randomly chosen or carefully computed by the attacker) and measure the execution time. During the experiments, the measured timings must be reproducible (at least up to a small error bound). For most attacks described in the literature, the adversary needs to have detailed knowledge about the cryptographic algorithm, its implementation, and the reasons for the variations of the execution time. To be more specific, consider the following square-and-multiply algorithm for modular exponentiation which is the basis of many public-key cryptosystems (like RSA, ElGamal, Diffie-Hellman):

Input: basis b, exponent $e = (e_1, \ldots, e_\ell)_2$, and modulus m
Output: $c = b^e \mod m$

1) $c \leftarrow 1$;
2) for $i \leftarrow 1$ to ℓ do
3) $\quad c \leftarrow c^2 \mod m$;
4) $\quad$ if $e_i = 1$ then $c \leftarrow c \cdot b \mod m$; fi
5) endfor

We denote by $t(y, b_1, \ldots, b_k)$ the execution time of the first k iterations of the for-loop for a given input y and exponent bits $b_1, \ldots, b_k$. Assume that an attacker has already reconstructed the first k bits $e_1, \ldots, e_k$ of the secret exponent e. Using his knowledge about the implementation of the algorithm, he can calculate the execution time $t(y, e_1, \ldots, e_k)$ of the first k iterations. If we subtract this value from the total execution time $t(y, e_1, \ldots, e_\ell)$ of the exponentiation measured by the attacker, we get the execution time $T(y)$ of the last $\ell - k$ iterations of the loop

$$T(y) = t(y, e_1, \ldots, e_\ell) - t(y, e_1, \ldots, e_k).$$

Under the assumption that $e_{k+1} = 1$, the attacker partitions the set of inputs into a set Y_1 of inputs which result in high execution time for the multiplication in step 4 in iteration $k + 1$, and a set Y_0 of inputs which result in low (or normal) execution time for this multiplication. The distinction between high and low (or normal) execution time depends obviously on the weaknesses of the implementation under attack (see the description of example attacks below). If inputs are randomly sampled from the sets Y_0 and Y_1 we get two random variables $T(Y_0)$ and $T(Y_1)$ for the execution time of the last $\ell - k$ iterations. If we assume further that the probabilities that a randomly chosen input results in high execution time are not significantly correlated for subsequent multiplications of the exponentiation (including squarings from step 3), an attacker can make the following observation: if $e_{k+1} = 0$ then it should make no difference for the average running time of the last $\ell - k$ iterations whether the input was chosen from Y_0 or Y_1. If $e_{k+1} = 1$ then for most samples from Y_1 the running time should be greater than the time for samples from Y_0. This means, if $e_{k+1} = 0$, the statistical properties of the random variables $T(Y_0)$ and $T(Y_1)$ are expected to be very similar. For the case $e_{k+1} = 1$ there should exist a difference in the distributions of the random variables $T(Y_0)$ and $T(Y_1)$ which can be noticed by an adversary doing several measurements using random samples from Y_0 and Y_1 and applying a statistical test.

In [11] the authors give a detailed description of timing attacks against exponentiation using Montgomery's representation of the operands. In this attack the variation of the running time is due to a last subtraction of the modulus which has to be done if the result of the modular multiplication is out of range. Essentially the same reduction step in an optimized finite field multiplication is the basis for an attack against an insecure implementation of AES candidate Rijndael in [25].

A more general type of timing attack which does not need a partition of the inputs according to a particular delay of the execution time is described in [22]: using the already reconstructed k bits $e_1, \ldots, e_k$ of the secret exponent e the attacker can compute two random variables

$$T_b(Y) = t(Y, e_1, \ldots, e_\ell) - t(Y, e_1, \ldots, e_k, b), \text{ where } b \in \{0, 1\},$$

dependent on the next unknown bit b of the exponent. Under the assumption that the iterations of the exponentiation algorithm are sufficiently stochastically independent, the variance of $T_b(Y)$ is expected to decrease for a correct guess of bit b and to increase otherwise. With randomly chosen inputs $y \in Y$ it is possible to approximate the variance of $T_b(Y)$ and, thus to compute the next unknown bit of the exponent. Note that it is not necessary for the attacker to know the reasons for the variations of the execution time.

A completely different method for a timing attack against the block cipher RC5 is described in [17]: the security of RC5 relies on the usage of data dependent cyclic rotations in its round function. Under the assumption that the execution time of a cyclic shift is proportional to the number of rotated bit positions and that rotations of different rounds are sufficiently independent, it is shown in [17] that the total amount of rotations during an encryption is correlated with the number of rotated bit positions of the last round. This can be used to reconstruct the round key of the last round from measurements of the total execution time and to break the cipher by iterating this process.

Plaintext- and ciphertext-only timing attacks against IDEA are described in [21]. IDEA uses in its round function multiplications of non-zero elements of the finite field $\mathbb{Z}/(2^{16}+1)\mathbb{Z}$, where the value 2^{16} is represented by the string 0^{16} to obtain a bijective mapping over the strings $\{0, 1\}^{16}$. In some applications, multiplication by 2^{16} (i. e. the string 0^{16}) is specially coded in the program and results in much faster execution than multiplication by other values.

2.4 Differential power analysis—DPA

Differential Power Analysis (DPA), introduced by Kocher et. al. [23, 24], is an attack that allows one to obtain information about the secret key by performing a statistical analysis of the power consumption measured for a large number of computations with the same key. In contrast to SPA attacks, it is not required for the attacker to have a detailed knowledge of the implementation of the algorithm. In the following, we describe the basic principles of DPA according to [24] and [15].

To be more specific we consider the case of the DES algorithm. In each of its 16 rounds the DES algorithm performs table lookup operations in 8 S-boxes. Each S-box uses 6 key bits and 6 data bits as inputs and produces 4 output bits. The DPA attack proceeds as follows:

1. Run the encryption algorithm for N random values of plaintext input Y_i, $i = 1, \ldots, N$. For each of the N plaintext inputs, a discrete time power consumption curve C_i of the relevant part of the algorithm is recorded, i. e., $C_{ij} = \{\text{power consumption for input } Y_i \text{ at time } t = t_j\}$. To amplify the signals contained in the power traces C_i we also compute the average power trace $\bar{C}$ ($\bar{C}_j = \frac{1}{N}\sum_{i=1}^{N} C_{ij}$).
2. Next, the attacker chooses one target bit, for example, the first output bit of the first S-box in the first round of the DES algorithm. Let b be the value of that bit. Obviously, b depends only on 6 bits of the secret key.
3. The attacker makes a hypothesis H_0 in a statistical sense on the involved 6 key bits, i. e., he guesses the value of these bits. He can now compute the theoretical value of b for the known plaintext Y_i under hypothesis H_0. Depending on the theoretical value of b, he separates the inputs and the corresponding power traces into two sets: those giving $b = 0$ and those giving $b = 1$.

$$C^0 := \{C_{ij} | b = 0\}, \qquad C^1 := \{C_{ij} | b = 1\}$$

4. The attacker computes the average trace $\bar{C}^0$ for all inputs Y_i leading to $b = 0$. If the guess for the 6 key bits was wrong, then the power traces $\bar{C}^0$ and $\bar{C}$ will not show any statistically significant differences. (If the guess for the key is incorrect, then the computed value of b will differ from the actual target bit for about half of the plaintexts Y_i, thus, the power traces are uncorrelated.) If, on the other hand, the guess was correct and the target bit b is involved in the further computation, we will see a significant difference between $\bar{C}^0$ and $\bar{C}$, i. e., we can observe peaks in the power trace. In other words, if the power traces $\bar{C}^0$ and $\bar{C}$ are statistically indistinguishable with a certain error probability, then we must reject hypothesis H_0 and repeat the last steps with a different H_0. (Note, that there are only 2^6 possible values of H_0.) Otherwise, the correct 6 key bits have been found.
5. Repeating steps 1–4 for the other 7 S-boxes the attacker learns 48 key bits. The remaining 8 key bits can be found by exhaustive search.

We would like to point out that DPA heavily relies on the following fundamental hypothesis from [15]:

> *There exists an intermediate variable [target bit b=first output bit of first S-box in the first round], that appears during the computation of the algorithm, such that knowing a few key bits [6 key bits for H_0] (in practice less than 32 bits) allows to decide whether two inputs (respectively two outputs) give or not the same value for this variable.*

Since step 3 of the original DPA attack requires the computation of the expected value of b under hypothesis H_0, DPA attacks concentrate on the beginning of an algorithm using known plaintext or on the end of an algorithm using known ciphertext.

Remark 1.

(i) The number N of necessary plaintext inputs depends very strongly on the signal-to-noise ratio of the smart card being investigated.

(ii) Before further computations are performed in step 1, the power traces have to be aligned in time, so that one point $t = t_j$ corresponds to the same operation for all inputs Y_i. However, such timing variations can be filtered out by statistical means.

(iii) The DPA attack can be further enhanced by considering d target bits instead of one target bit b (multiple bit DPA). In addition to the sets $C^0 := \{C_{ij} | b = 0^d\}$, $C^1 := \{C_{ij} | b = 1^d\}$ one forms a set $C^2 := \{C_{ij} | C_{ij} \notin C^0, C^1\}$. All signals falling into C^2 do not give enough power differences and are therefore discarded. This increases the signal-to-noise ratio considerably, see [28] for specific values of signal levels.

While the first DPA attacks were aimed at symmetric algorithms like DES [24, 15, 21], algorithms for public-key cryptography, specifically methods for modular exponentiation, are also vulnerable. The basis of DPA attacks against the square-and-multiply method for modular exponentiation, see Section 2.3, is the fact that after the i-th iteration the ciphertext depends only on the first i key bits. The authors of [29] describe three types of concrete attacks against smart card implementations of square-and-multiply which differ in the assumptions on the capabilities of the adversary or the smart card. Coron [8] presents a practical attack against an SPA-resistant version of double-and-add, the equivalent to square-and-multiply in elliptic curve cryptography, and extends the attack to other scalar multiplication algorithms (addition-subtraction methods, window-method).

Kocher's original attack has the disadvantage that the adversary must know all the cleartexts (attack the subkey of the first round) or all the ciphertexts (attack the subkey of the last round). Recently, attacks have been proposed which try to circumvent these problematic issues: in [13] the authors introduce the so-called *inferential power analysis* (IPA). They separate the power attack into two stages: a profiling stage and the actual key extraction stage. During the profiling stage, a large number of power traces is used to learn about the key scheduling (location and identification of the target key bits). In the key extraction stage, only a few power traces are necessary to find the subkey bits and then the master key itself. Since the profiling data depend only on the implementation and *not* on the key that was used, it is possible to perform the profiling stage only once (on a specific smart card) and then to use this knowledge for other specimen of this card type. Further advantages of IPA attacks are the ability to look at the middle of an algorithm and that they do not require either known plaintext or known ciphertext.

In [4] Biham and Shamir described a power attack which falls into the same category: they use the power traces of several different cards to identify when key bits are handled. It is shown that DES-like key scheduling algorithms are especially vulnerable to such attacks, while Skipjack-like key scheduling structures seem to be more difficult to attack. Finally, the key scheduling of the AES candidates is investigated with respect to these attacks.

In [6] a further attack against particular operations used to generate subkeys (key whitening process) is reported and demonstrated on the basis of a reference implementation of Twofish on a certain smart card. However, this attack is not limited to Twofish or this specific card. A general evaluation of the resistance of AES candidates against timing and differential power attacks can be found in [10]. See [30] for a comparative summary.

At this time, only a few *concrete* attacks against implementations of the AES candidates are reported in the literature: instead there are rather general accounts of their vulnerability against DPA. Techniques to secure the AES finalists are proposed in [27].

3 Countermeasures against side-channel attacks

3.1 Countermeasures against SPA

SPA attacks are relatively easy to circumvent. Of course, one tries to reduce the leaked power signal and prevent the attacker from learning the implementation. Since the attacker cannot easily amplify the secret information contained in the power trace, it suffices to decrease the signal-to-noise ratio of the side-channel information considerably. By avoiding secret keys for conditional branching, one can mask many SPA characteristics. For example, one can always compute $c \cdot b \mod m$ in step 4 of the modular exponentiation algorithm in Section 2.3 and only store the result if

$e_i = 1$. Introducing random timing shifts also helps to make SPA attacks more difficult. However, these countermeasures do not prevent more powerful attacks.

3.2 Countermeasures against DFA

Differential failure attacks which do not rely on altering the circuitry of the tamper-proof device can be defeated by plausibility checks for the result of the cryptographic computation, for the internal state of a protocol, and for the keys. The simplest method of protection for cipher algorithms consists of running the algorithm twice and comparing the results. Unfortunately in most cases, this results in intolerable execution times. Better performance can be reached by special algorithms that check whether a randomized relation holds during the computation. In [32, 33] Shamir describes such an algorithm for RSA with CRT which avoids the costly double computation of the modular exponentiation. Identification schemes like Fiat-Shamir or the Schnorr-identification scheme can be secured if the internal state of the prover is saved in a redundant way. This allows the prover to check the consistency of the messages before answering a challenge, see [5]. Similar countermeasures are presented in [31] against the key erasing attacks of [3]. If the key is stored with some redundancy, the device can detect flipped bits.

Glitch attacks and (differential) failure attacks by cutting wires or destroying gates are more difficult to control. In these cases, the device cannot reliably perform plausibility checks and special hardware is necessary to take precautions, see [2, 1, 26].

3.3 Countermeasures against timing attacks

Since timing attacks rely on data dependent variations of the execution time the most obvious countermeasure is to avoid such changes in the time pattern of a cryptographic algorithm. This makes the implementation of cryptographic algorithms very complicated and costly in general. The programmer must have a detailed knowledge of the processor, the hardware platform, the algorithm, and sometimes the complete application. On the extreme, the execution of the program has to be linear and cannot take advantage of performance optimizations, i. e., the running time is always worst-case and the implementation is likely to be inacceptably slow. A carefully designed implementation which is immune against timing attacks on one platform may become insecure on others. Sometimes, timing variations are due to events which cannot be controlled by the application programmer, like interference with other processes, memory mapping, hardware exceptions, cache misses, and variable execution of machine instructions. Counting clock cycles to ensure a constant execution time may be circumvented by observing other leakage channels (for example power consumption).

To protect the exponentiation algorithm from Section 2.3 against timing attacks it is sufficient for most applications to implement multiplication and squaring in such a way that they take always the same amount of time. Although it was observed in [22] that in this case the total execution time of the exponentiation still corresponds to the Hamming weight of the key, the information computable by the attacker is negligible. If keys of length n bits are randomly chosen the information leaked out by the Hamming weight can be upper bounded using Shannon's entropy formula by

$$2^{-n} \sum_{i=0}^{n} \binom{n}{i} \log_2 \left(2^n / \binom{n}{i} \right).$$

This adds up to ≈ 6.047 bits of key information for a random 1024 bit key, see [20] for a discussion of Hamming weight of DES keys and timing attacks.

Another possibility to prevent timing attacks is based on blinding the input. To apply this method, the cryptographic algorithm has to be transformed in a new scheme with an additional parameter that influences all intermediate values but not the result. The value of this parameter is randomly chosen for each application of the algorithm. It is necessary that the random value is generated inside the tamper-proof device and not known to the attacker. Blinding decorrelates

inputs, intermediate values of the computation, and the execution time. Therefore, the time measurements by the attacker are no longer reproducible. In [22, 32, 33] the authors describe several methods for blinding exponentiation algorithms.

3.4 Countermeasures against DPA

One of the first—and most natural—attempts to counteract DPA attacks which have been proposed in the literature and implemented in practical devices relies on the reduction of side channel information: by introducing random timing shifts to complicate the alignment of power traces, by replacing some of the critical instructions by "consumption-friendly" instructions, and, finally, by adding filters and performing random power consuming operations one tries to decrease the signal-to-noise ratio and, thereby, to increase the number of necessary power samples. The use of differential logic, i. e., representing 0 by "01" and 1 by "10" respectively, leads to a lower signal-to-noise ratio and a constant Hamming weight of the operands.

Another popular approach is to randomize the execution sequence, i. e., perform the DES-S-box look-up in random order or perform additional rounds that do not affect the mathematical result. However, unless this randomization is done extensively, which is often prohibited due to performance constraints, it can be undone by only slightly more samples and some statistics.

The blinding technique proposed as a countermeasure against timing attacks [22] can be applied in the DPA context, too. Since one has to protect against known plaintext and known ciphertext attacks one should use both exponent and message blinding, see [29]. Instead of computing $c = b^e \mod m$ one performs:

0) Compute $v_f = (v_i^{-1})^e \mod m$ for a random value v_i
1) Blind the message b: $\hat{b} = (v_i b) \mod m$
2) Blind the exponent e: $\hat{e} = e + r\varphi(m)$, r random value, φ Euler phi-function
3) Exponentiate: $\hat{c} = (\hat{b})^{\hat{e}} \mod m$
4) Unblind the result: $c = (v_f \hat{b}) \mod m$
5) Update the pair (v_i, v_f)

Efficient ways to calculate and update the blinding pair (v_i, v_f) are discussed in [22].

For elliptic curve systems, mathematically equivalent countermeasures to those proposed for modular exponentiation have been investigated by Coron [8]. A further countermeasure, specific to elliptic curve cryptosystems, is the randomization of the projective coordinate representation of a point.

Although no single defense makes a system impervious to DPA, adding a variety of these countermeasures will likely increase the difficulty of attacks. A new quality of countermeasures has been achieved with the definition of leakage immunity and the formal analysis of side leakage information. Coron et. al. [9] develop statistical tests capable of detecting information leakage in unknown cryptosystems. However, these tests are rather generic and passing these tests should not give too much trust in a leakage-resistant cryptosystem. Chari et. al. [7] also give a definition of leakage immunity, but in contrast to [9] they describe a provable secure instance under their specific power consumption model. Utilizing the method of secret sharing, they propose to randomly split each bit of the original computation into k shares and prove a lower bound on the number of samples required to distinguish the distributions in step 4 of the attack in Section 2.4. A similar approach is taken in [15] where a splitting of the S-boxes for the DES algorithm and a simple splitting scheme for RSA is proposed. In summary, it can be stated, that first steps into the formal analysis of computing with side-channel information have been taken. However, substantial effort is still required to refine the power consumption models and to improve their theoretical analysis.

Finally, it should be noted that some cryptographic algorithms like ECDSA are by their nature rather immune against statistical attacks like DPA and timing attacks, since in this case scalar multiplication is performed with a random exponent instead of a fixed exponent.

4 Conclusion

In this paper, we reviewed the current status of side-channel attacks against smart cards. We did not describe new attacks, but instead tried to outline the basic principles and to work out the main differences, ignoring the technical details of specific threats.

The importance of information leakage attacks has been acknowledged by the industry and mechanisms to protect against it are now a major part of todays security design. Shortly after the publication of these new types of attacks, only ad-hoc measures, like randomization and noise generation, were incorporated into smart cards. Today, more sophisticated countermeasures are implemented.

A combination of the countermeasures mentioned in this article should give enough security for next generation smart cards. However, the development of a theoretical, unified power consumption model, its formal analysis, and the derivation of quantitative security measures is still in its infancy. First steps in this direction have been taken in recent research papers.

It is expected that by combining different sources of side-channel information, e. g. power consumption and data from fault attacks, using powerful statistical methods and observing multiple cryptographic operations, new, much more powerful attacks will be developed in the near future. Although none of these techniques are known to exist at present, their potential threats and possible countermeasures have to be taken into account by system designers and researchers.

References

1. R. J. Anderson and M. G. Kuhn, *Tamper resistance—a cautionary note*, Proceedings of Second USENIX Workshop on Electronic Commerce (Oakland, California), 1996, pp. 1–11.
2. ———, *Low cost attacks on tamper resistant devices*, Proceedings of International Workshop on Security Protocols 1997 (Paris, France) (M. Lomas et.al., ed.), Lecture Notes in Computer Science, vol. 1361, Springer-Verlag, 1997, pp. 125–136.
3. E. Biham and A. Shamir, *Differential fault analysis of secret key cryptosystems*, Proceedings of CRYPTO '97 (Burton S. Kaliski Jr., ed.), Lecture Notes in Computer Science, vol. 1294, Springer-Verlag, 1997, pp. 513–525.
4. ———, *Power analysis of the key scheduling of the AES candidates*, The Second AES Conference, March 22-23, 1999, 1999, pp. 115–121.
5. D. Boneh, R. A. DeMillo, and R. J. Lipton, *On the importance of checking cryptographic protocols for faults*, Proceedings of EUROCRYPT '97 (W. Fumy, ed.), Lecture Notes in Computer Science, vol. 1233, Springer-Verlag, 1997, pp. 37–51.
6. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, *A cautionary note regarding evaluation of AES candidates on smart-cards*, The Second AES Conference, March 22-23, 1999, 1999, pp. 133–147.
7. ———, *Towards sound approaches to counteract power-analysis attacks*, Proceedings of CRYPTO '99 (M. J. Wiener, ed.), Lecture Notes in Computer Science, vol. 1666, Springer-Verlag, 1999, pp. 398–412.
8. J.-S. Coron, *Resistance against differential power analysis for elliptic curve cryptosystems*, Proceedings of CHES '99 (C. K. Koç and Chr. Paar, eds.), Lecture Notes in Computer Science, vol. 1717, Springer-Verlag, 1999, pp. 292–302.
9. J.-S. Coron, P. Kocher, and D. Naccache, *Statistics and secret leakage*, 2000, 17 pages.
10. J. Daemen and V. Rijmen, *Resistance against implementation attacks—a comparative study of the AES proposals*, The Second AES Conference, March 22-23, 1999, 1999, pp. 122–132.
11. J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems, *A practical implementation of the timing attack*, Technical Report CG-1998/1, Université catholique de Louvain, June 1998.
12. ———, *A practical implementation of the timing attack*, Proceedings of CARDIS '98, Lecture Notes in Computer Science, Springer-Verlag, 1998.
13. P. N. Fahn and P. K. Pearson, *IPA: A new class of power attacks*, Proceedings of CHES '99 (C. K. Koç and Chr. Paar, eds.), Lecture Notes in Computer Science, vol. 1717, Springer-Verlag, 1999, pp. 173–186.
14. O. Goldreich and R. Ostrovsky, *Software protection and simulation on oblivious RAMs*, Journal of the ACM **43** (1996), no. 3, 431–471.

15. L. Goubin and J. Patarin, *DES and differential power analysis*, Proceedings of CHES '99 (C. K. Koç and Chr. Paar, eds.), Lecture Notes in Computer Science, vol. 1717, Springer-Verlag, 1999, pp. 158–172.
16. G. Hachez, F. Koeune, and J. J. Quisquater, *Timing attack: What can be achieved by a powerful adversary*, Proceedings of the 20th symposium on Inform. Theory in the Benelux (A. Barb/'e, E. C. van der Meulen, and P. Vanroose, eds.), Werkgemeenschap Informatie- en Communicatietheorie, Enschede (NL), 1999, pp. 63–70.
17. H. Handschuh, *A timing attack on RC5*, Technical Report SC02-1998, Gemplus' Corporate Product R&D Division, 1998.
18. H. Handschuh and H. M. Heys, *A timing attack on RC5*, Proceedings of SAC '98, Lecture Notes in Computer Science, vol. 1556, Springer-Verlag, 1999, pp. 306–320.
19. H. Handschuh, P. Pallier, and J. Stern, *Probing attacks on tamper-resistant devices*, Proceedings of CHES '99 (C. K. Koç and Chr. Paar, eds.), Lecture Notes in Computer Science, vol. 1717, Springer-Verlag, 1999, pp. 303–315.
20. A. Hevia and M. Kiwi, *Strength of two data encryption standard implementations under timing attacks*, ACM Transactions on Information and System Security **2** (1999), no. 4, 416–437.
21. J. Kelsey, B. Schneier, D. Wagner, and C. Hall, *Side channel cryptanalysis of product ciphers*, Proceedings of ESORICS '98, Lecture Notes in Computer Science, vol. 1485, Springer-Verlag, 1998, pp. 97–110.
22. P. Kocher, *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*, Proceedings of CRYPTO '96 (N. Koblitz, ed.), Lecture Notes in Computer Science, vol. 1109, Springer-Verlag, 1996, pp. 104–113.
23. P. Kocher, J. Jaffe, and B. Jun, *Introduction to differential power analysis and related attacks*, `http://www.cryptography.com/dpa/technical/`, 1998.
24. ———, *Differential power analysis*, Proceedings of CRYPTO '99 (M. J. Wiener, ed.), Lecture Notes in Computer Science, vol. 1666, Springer-Verlag, 1999, pp. 388–397.
25. F. Koeune and J.-J. Quisquater, *A timing attack against Rijndael*, Technical Report CG-1999/1, Université catholique de Louvain, June 1999.
26. O. Kömmerling and M. G. Kuhn, *Design principles for tamper-resistant smartcard processors*, Proceedings of USENIX Workshop on Smartcard Technology, 1999, pp. 9–20.
27. T. S. Messerges, *Securing the AES finalists against power analysis attacks*, Proceedings of Fast Software Encryption Workshop 2000, Lecture Notes in Computer Science, Springer-Verlag, 2000, to appear.
28. T. S. Messerges, E. A. Dabbish, and R. H. Sloan, *Investigations of power analysis attacks on smartcards*, Proceedings of USENIX Workshop on Smartcard Technology, 1999, pp. 151–161.
29. ———, *Power analysis attacks of modular exponentiation in smartcards*, Proceedings of CHES '99 (C. K. Koç and Chr. Paar, eds.), Lecture Notes in Computer Science, vol. 1717, Springer-Verlag, 1999, pp. 144–157.
30. James Nechvatal, Elaine Barker, Donna Dodson, Morris Dworkin, James Foti, and Edward Roback, *Status report on the first round of the development of the advanced encryption standard*, Tech. report, National Institute of Standards and Technology (NIST), August 1999, `http://csrc.nist.gov/encryption/aes/round1/r1report-addenda.pdf`.
31. P. Paillier, *Evaluating differential fault analysis of unknown cryptosystems*, Technical Report AP05-1999, Gemplus' Corporate Product R&D Division, 1999.
32. A. Shamir, *How to check modular exponentiation*, Rump Session of EUROCRYPT '97, 1997.
33. ———, *Method and apparatus for protecting public key schemes from timing and fault attacks*, United States Patent 5991415, November 23, 1999.