Fault based cryptanalysis of the Advanced Encryption Standard (AES)

Johannes Blömer[1] and Jean-Pierre Seifert[2]

[1] University of Paderborn
D-33095 Paderborn, Germany
bloemer@upb.de
[2] Infineon Technologies, Security & ChipCard ICs, CC TI CI
D-81609 Munich, Germany
Jean-Pierre.Seifert@infineon.com

Abstract. In this paper we describe several fault attacks on the Advanced Encryption Standard (AES). First, using optical fault induction attacks as recently publicly presented by Skorobogatov and Anderson [SA], we present an implementation independent fault attack on AES. This attack is able to determine the complete 128-bit secret key of a sealed tamper-proof smartcard by generating 128 faulty cipher texts. Second, we present several implementation-dependent fault attacks on AES. These attacks rely on the observation that due to the AES's known timing analysis vulnerability (as pointed out by Koeune and Quisquater [KQ]), any implementation of the AES must ensure a data independent timing behavior for the so called AES's `xtime` operation. We present fault attacks on AES based on various timing analysis resistant implementations of the `xtime`-operation. Our strongest attack in this direction uses a very liberal fault model and requires only 256 faulty encryptions to determine a 128-bit key.

Keywords: AES, Cryptanalysis, Fault attacks, Side-channel attacks, Smartcards.

1 Introduction

Recently, Rijndael has been chosen as the winner of the Advanced Encryption Standard (AES) contest. Thus Rijndael has become the AES. Within the near future AES will replace DES as the worldwide standard in symmetric key encryption. Not surprisingly, a lot of research so far focused on the mathematical security of AES. Likewise, the security of AES against side-channel attacks like timing analysis and power analysis, has also been examined. On the other hand, no fault-based cryptanalysis of AES has been reported so far. This is surprising as the frauds with smartcards by inducing faults are real, cf. [A,AK1,AK2], whereas no frauds via Timing or Power Analysis attacks have been reported so far.

In this paper we describe several methods for a fault based cryptanalysis of AES. We present an implementation independent attack as well as attacks on several implementations of AES aimed at making AES timing analysis secure.

Several different fault models are used in our attacks. The first attack is implementation independent but uses the most restrictive fault model. It is aimed at the first transformation during an encryption with AES, the so called `AddRoundKey`. In this attack we use a rather strong but seemingly realistic fault model. We assume, that an attacker can set a specific memory bit to a fixed value, e.g., to the value 0. Moreover, we assume that the attacker can do this at a precise time of his choice. The practicality of this model has recently been demonstrated in [SA]. Using this model we show that an attacker can determine the complete 128-bit secret key by computing 128 encryptions inducing a single fault each time. We also show that the fault model can be relaxed in two ways. First, an attacker need not be able to set a fixed memory bit with probability 1 to the value 0, say. Instead, it is sufficient that the value of a certain a bit will be changed with slightly higher probability from 1 to 0 than from 0 to 1. Second, we show that the attacker need not be able to change the bit at a precise point in time. Hence, to fend off the attack it is not sufficient to equip a cryptographic device with a randomized timing behavior. We note that our implementation independent attack can also be mounted against other symmetric encryption ciphers like IDEA, SAFER, and Blowfish. In fact, like AES, these ciphers have an initial key addition. This alone renders them immediately susceptible to our implementation independent attack. Moreover, in a forthcoming paper [BS02] we are going to describe how optical attacks can be used to break symmetric or asymmetric ciphers in a trivial way, when implemented on an unprotected platform.

Our second class of attacks are implementation dependent. More precisely, we consider several different implementations for the so called `xtime` operation that is performed during the transformation `MixColumn`. As it is well-known, AES is susceptible to timing attacks if `xtime` is not implemented appropriately. We consider various timing attack resistant implementations and show that all of them lead to a simple fault based cryptanalysis of AES. The various implementations we consider vary from software solutions to hardware based implementations. Definitely, we do not consider all implementations of `xtime` currently in use. However, we are confident that most if not all implementations are susceptible to the attacks described in this paper. The fault models we use in the attacks range from very liberal models (arbitrary fault at a specific location) to more restricted models, like the one we use in the implementation independent attack. Our strongest result shows that the most obvious timing attack resistant implementation of `xtime` opens the door for a simple fault based attack on AES. With this attack the complete secret key can be determined encrypting roughly 256 plaintexts. In each encryption an attacker must induce an *arbitrary* fault at a specific memory byte. Using the methods we describe for the implementation independent attack, one can argue, that it is not necessary for the attacker to induce the fault at a specific time. We like to point out that our implementation dependent attacks use a technique introduced by [KQ] and were inspired by ideas in [YJ].

In all our attacks we assume, that from the behavior of a cryptographic device we can deduce whether a fault leading to a wrong ciphertext has actually occurred during an encryption. If a fault occurs, the device may simply output a wrong ciphertext. However, an attacker can compute the correct ciphertext by rerunning the device without inducing a fault, thereby detecting whether a fault occured during the original encryption. The device may also check its own result, or may even check intermediate results and answer with an error warning in case a fault has been detected (see [KWMK]). This clearly tells an attacker that a fault has occurred. Thirdly, the card may check whether a fault has occurred, and if so, recompute the encryption. However, in this case by measuring the time before the device outputs the ciphertext, an attacker can tell whether a fault occurred. To simplify the presentation, we assume that a cryptographic device will answer with the output `reset`, whenever a fault caused a wrong ciphertext.

From the results in this paper, it follows that it is absolutely necessary to incorporate both hardware and software means into AES realizations to guard against fault attacks. Of course, some modern high-end crypto smartcards are protected by various sophisticated hardware mechanisms to detect any intrusion attempt to their system behavior. However, as techniques to induce faults into the computations of smartcards become more sophisticated, it is mandatory to use also various software mechanisms to fend off fault attacks. We like to close with an advice due to Kaliski and Robshaw [KR] from RSA Laboratories that *good engineering practices in the design of secure hardware are essential*, we only would like to add, that the same applies to secure software.

The paper is organized as follows. We first describe the AES algorithm. In the next section we briefly turn to the physics of realizing faults during computations. In particular, we sketch and characterize the physical attacks used by us in later sections. The following section then describes our general independent fault attack on AES. In this section we also describethe two aforementioned relaxations concerning a probabilistic fault model and a loose time control on the induced error. The next section starts with the explanation of the basic idea for all of our implementation dependent fault attacks. Then, we will continue to describe incrementally more and more sophisticated fault attacks. Eventually, we give some hints on conceivable countermeasures to defeat our fault attacks on the AES.

2 Preliminaries

2.1 Description of the Advanced Encryption Standard

In this section we briefly describe the Advanced Encryption Standard (AES). For a more detailed description we refer to [DR2].

AES encrypts plaintexts consisting of $\mathtt{lb}$ bytes, where $\mathtt{lb} = 16, 24$, or 32. The plaintext is organized as a $(4 \times \mathtt{Nb})$ array (a_{ij}), $0 \leq i < 4, 0 \leq j < \mathtt{Nb} - 1$, where $\mathtt{Nb} = 4, 6, 8$, depending on the value of $\mathtt{lb}$. The n-th byte of the plaintext is stored in byte $a_{i,j}$ with $i = n \mod 4$, $j = \lfloor \frac{n}{4} \rfloor$.

AES uses a secret key, called *cipher key*, consisting of $\mathtt{lk}$ bytes, where $\mathtt{lk} = 16, 24$, or 32. Any combination of values $\mathtt{lb}$ and $\mathtt{lk}$ is allowed. The cipher key is organized in a $4 \times \mathtt{Nk}$ array (k_{ij}), $0 \leq i < 4, 0 \leq j \leq \mathtt{Nk} - 1$, where $\mathtt{Nk} = 4, 6, 8$, depending on the value of $\mathtt{lk}$. The n-th key byte is stored in byte k_{ij} with $i = n \mod 4$, $j = \lfloor \frac{n}{4} \rfloor$.

The AES encryption process is composed of *rounds*. Except for the last round, each round consists of four transformations called `ByteSub`, `ShiftRow`, `MixColumn`, and `AddRoundKey`. In the last round the transformation `MixColumn` is omitted. The four transformations operate on intermediate results, called *states*. A state is a $4 \times \mathtt{Nb}$ array (a_{ij}) of bytes. Initially, the state is given by the plaintext to be encrypted. The number of rounds $\mathtt{Nr}$ is 10, 12, or 14, depending on $\max\{\mathtt{Nb}, \mathtt{Nk}\}$. In addition to the transformations performed in the $\mathtt{Nr}$ rounds there is an `AddRoundKey` applied to the plaintext prior to the first round. We call this the *initial* `AddRoundKey`.

Next, we are going to describe the transformations used in the AES encryption process. We begin with `AddRoundKey`.

The transformation `AddRoundKey` The input to the transformation `AddRoundKey` is a state (a_{ij}), $0 \leq i < 4, 0 \leq j < \mathtt{Nb}$, and a *round key*, which is an array of bytes $(\mathtt{rk}_{ij})$, $0 \leq i < 4, 0 \leq j < \mathtt{Nb}$. The output of `AddRoundKey` is the state $(b_{ij}), 0 \leq i < 4, 0 \leq j < \mathtt{Nb}$, where

$$b_{ij} = a_{ij} \oplus \mathtt{rk}_{ij}.$$

The round keys are obtained from the cipher key by expanding the cipher key array (k_{ij}) into an array (k_{ij}), $0 \leq i < 4, 0 \leq j \leq \mathtt{Nr} \cdot \mathtt{Nb}$, called the *expanded key*. The exact procedure by which the expanded key is obtained from the cipher key is of no importance for the attacks described in this paper. The round key for the initial application of `AddRoundKey` is given by the first $\mathtt{Nb}$ columns of the expanded key. The round key for the application of `AddRoundKey` in the m-th round of AES is given by columns $m\mathtt{Nb}, \ldots, (m+1)\mathtt{Nb} - 1$ of the expanded key, $1 \leq m \leq \mathtt{Nr}$.

The transformation `ByteSub` Given a state (a_{ij}), $0 \leq i < 4, 0 \leq j < \mathtt{Nb}$, the transformation `ByteSub` applies an invertible function $S : \{0,1\}^8 \to \{0,1\}^8$ to each state byte a_{ij} separately. The exact nature of S is of no relevance for the attacks described later. We just mention that S is non-linear, and in fact, it is the only non-linear part of the AES encryption process. In practice, S is often realized by a substitution table or *S-box*.

The transformation `ShiftRow` The transformation `ShiftRow` cyclically shifts each row of a state (a_{ij}) separately to the left. Row 0 is not shifted. Rows 1, 2, 3 are shifted by C_1, C_2, C_3 bytes, respectively, where the values of the C_i depend on $\mathtt{Nb}$.

The transformation `MixColumn` The transformation `MixColumn` is crucial to some of our attacks. The transformation `MixColumn` operates on the columns of a state separately. To each column a fixed linear transformation is applied. To do so, bytes are interpreted as elements in the field $\mathbb{F}_{2^8}$. As is usually done, we will denote elements in this field in hexadecimal notation. Hence 01, 02 and 03 correspond to the bytes 00000001, 00000010, and 00000011, respectively. Now `MixColumn` applies to each row of a state the linear transformation defined by the following matrix

$$\begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix}. \qquad (1)$$

One complete round of the AES encryption procedure is schematically shown in figure 1.

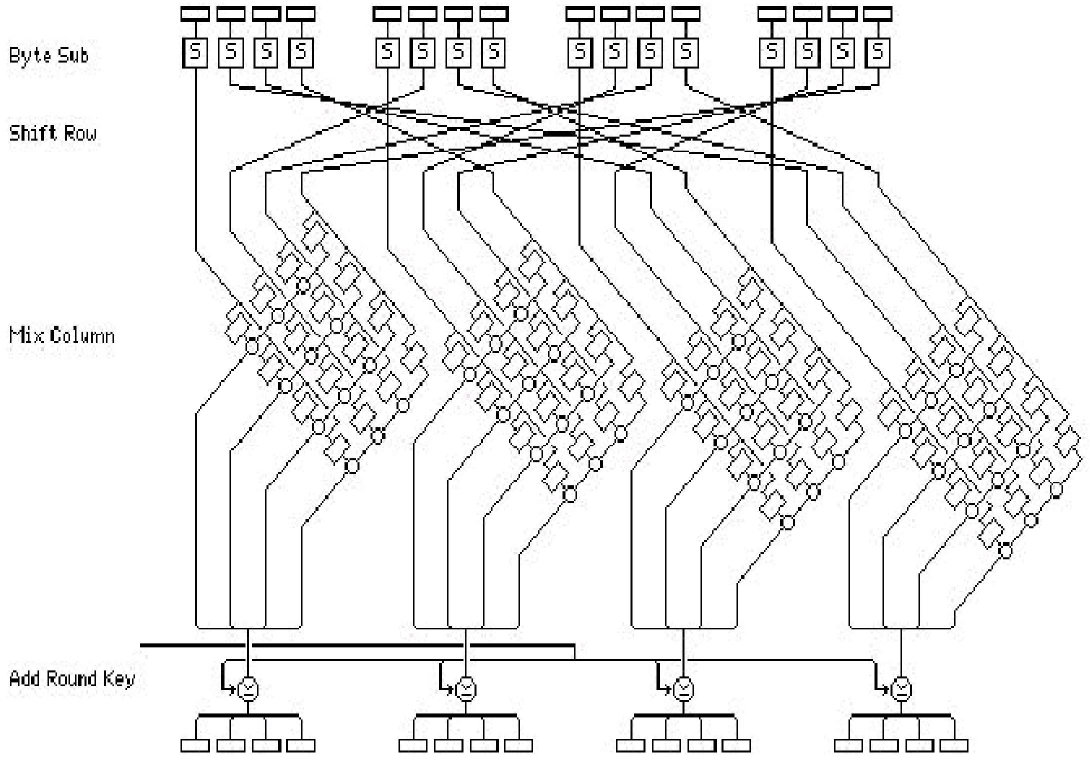

Fig. 1. Schematic for one AES round.

The operation `xtime` The multiplications in $\mathbb{F}_{2^8}$ necessary to compute the transformation `MixColumn` are of great importance to some of our attacks. Therefore we are going to describe them in more detail. First we need to say a few words about the representation of the field $\mathbb{F}_{2^8}$. In AES the field $\mathbb{F}_{2^8}$ is represented as

$$\mathbb{F}_{2^8} = \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1). \qquad (2)$$

That is, elements of $\mathbb{F}_{2^8}$ are polynomials over $\mathbb{F}_2$ of degree at most 7. The addition and multiplication of two polynomials is done modulo the polynomial $x^8 + x^4 + x^3 + x + 1$. Since this is an irreducible polynomial over $\mathbb{F}_2$, (2) defines a field. In this representation of $\mathbb{F}_{2^8}$ the byte $\mathbf{a} = (a_7, \ldots, a_1, a_0)$ corresponds to the polynomial $a_7x^7 + \cdots a_1x + a_0$. The multiplication of an element $\mathbf{a} = (a_7, \ldots, a_1, a_0)$ in $\mathbb{F}_{2^8}$ by $01, 02$, and 03 is realized by multiplying the polynomial $a_7x^7 + \cdots a_1x + a_0$ with the polynomials $1, x, x + 1$, respectively, and reducing the result modulo $x^8 + x^4 + x^3 + x + 1$. Hence

$$\begin{aligned} 01 \cdot \mathbf{a} &= \mathbf{a} \\ 03 \cdot \mathbf{a} &= 02 \cdot \mathbf{a} + \mathbf{a}. \end{aligned}$$

We see that the only non-trivial multiplication needed to multiply a column of a state by the matrix in (1) is the multiplication by 02. Following the notation in [DR2] we denote the multiplication of byte $\mathbf{a}$ by 02 by $\texttt{xtime}(\mathbf{a})$. The crucial observation is that $\texttt{xtime}(\mathbf{a})$ is simply a shift of byte $\mathbf{a}$, followed in some cases by an xor of two bytes. More precisely, for $\mathbf{a} = (a_7, \ldots, a_0)$

$$\texttt{xtime}(\mathbf{a}) = \begin{cases} (a_6, \ldots, a_0, 0) & \text{if } a_7 = 0 \\ (a_6, \ldots, a_0, 0) \oplus (0,0,0,1,1,0,1,1) & \text{if } a_7 = 1 \end{cases} \tag{3}$$

This finishes our brief description of the AES encryption procedure.

3 Physical faults attacks

Although there are lots of possibilities to introduce an error during the cryptographic operation of an unprotected smartcard IC, we will only briefly explain so called spike attacks, glitch attacks and the recently developed optical attacks. We selected these attacks, as they are so called non invasive attacks or at least only semi-invasive, meaning that these methods require no physical opening, no chemical preparation nor an electrical contact to the chip's metal surface. Therefore, these attacks are the most obvious methods for attacking smartcard ICs by fault attacks. For a thorough treatment of tamper-resistance and especially on methods how to enforce erroneous computations of microcontroller chips we refer to [A,AK1,AK2,Gu1,Gu2,Koca,SA,KK,Ma].

Moreover, although the effects of applying physical stress via the above mentioned methods to a smartcard IC can be completely explained by numerous physical and technical considerations, these explanations are clearly beyond the scope of this paper, meaning that our physical explanations will be pretty rough.

3.1 Spike attacks

As required by [ISO], a smartcard IC must be able to tolerate on the contact V_{CC} a supply voltage between $4,5$V and $5,5$V, where the standard voltage is specified at 5V. Within this range the smartcard must be able to work properly. However, a deviation of the external power supply, called *spike*, of much more than the specified 10% tolerance might cause problems for a proper functionality of the smartcard IC. Indeed, it will most probably lead to a wrong computation result, provided that the smartcard IC is still able to finish its computation completely.

Although a spike seems from the above explanation very simple, a specific type of a power spike is determined by altogether nine different parameters. These nine parameters are determined by a combination of time, voltage values, and the shape of the voltage transition. This indicates the range of different parameters which an attacker can try to modify in order to induce a fault.

As spike attacks are non-invasive attacks, they are the most obvious method for inducing computational faults on smartcard ICs. In particular, they require no physical opening, no chemical preparation of the smartcard IC and do not require making an electrical contact to the IC's metal surface.

3.2 Glitch attacks

Similar to above, [ISO] prescribes that a smartcard IC must be able to tolerate on the clock contact CLK a voltage for V_{IH} of $0,7V_{CC},\ldots,V_{CC}$ and for V_{IL} of $0V_{CC},\ldots,0,5V_{CC}$. Moreover, it must be also able to tolerate clock rise and clock fall times of 9% from the period cycle. Within this range the smartcard must be able to work properly. However, a deviation of the external CLK, most often called *glitch*, of much more than the specified tolerances clearly will cause problems for a proper functionality of the smartcard IC. Again, similar to voltage spikes there is huge range of different parameters determining a glitch, which can be altered by an attacker to induce a faulty computation on a device performing a encryption.

Interestingly, a finely tuned clock glitch is able to completely change a CPU's execution behavior including the omitting of instructions during the executions of programs. For physical explanations of this nice effect we refer the interested reader to [AK1,AK2,KK]. According to [KK], around 1999 clock glitch attacks have been the simplest and most practical attacks.

3.3 Optical attacks

Without doubt, light attacks now belong to the classical non invasive methods to realize faulty computations on smartcards by manipulating their non-volatile memory, i.e., their EEPROM, cf. [A,AK1,AK2,BS99,KK,Koca,NR,Ma,Pe]. Unfortunately, such light attacks could be used so far only in an unsystematic way, for e.g. flipping randomly selected bits from 1 to 0 with some non zero probability.

However, recently it was demonstrated by [SA] that by using focussed camera flash-light on the right places of a microcontroller it is indeed possible to set or reset any individual bit of its memory at a time specified by the attacker. Such an attack enables a very timing-precise controlled transient fault on a given individual bit. Moreover, it is claimed by [SA] to be practical and seems to require only very cheap and simple public available photo equipment. Therefore, [SA] called it optical attack instead of light attack, as it mainly relies on using optical equipment.

3.4 Differentiating the physical attacks

In order to sort the zoo of possible effects by physically stressing a chip, we will now characterize the above attacks concerning:

- Control on fault location.
- Precision of timing.
- Fault model.
- Number of faulty bits induced.

Control on fault location For the issue of the control on the resulting fault location, one can clearly define three natural classes: no control, loose control and complete control. The definition of them is self contained.

Precision of timing As with the former issue it is obvious to define three natural classes whose definition speaks for themselves: no control on the fault occurence time, loose control on the fault occurence time and very precise control on the fault occurence time.

Fault model Although there are usually only three fault models specified, cf. [BDL,BDHJNT,BS97,YJ,YKLM1,YKLM2,ZM], we will introduce a new fault model. This model reflects the power of the optical induced fault models as developed in [SA]. In addition to the classical fault models, which are given by the stuck at faults model (saf), bit flip model (bf) or random fault model (rf)), we will define the *bit set or reset model* (bsr). In this model the attacker is able to set or reset the value of any target bit at any time specified by himself, regardless of its previous contents, cf. [SA].

Number of faulty bits induced Clearly, the number of induced faulty bits due to physical stress is very important for a fault based cryptanalysis. Here one usually makes the distinction between single faulty bit, few faulty bits and random number of faulty bits.

We present the characterization of the different physical attacks presented previously by a table.

	Control on fault loc.	Prec. of timing	Fault model	#(faulty bits induc.)
Spike	no	precise	random	random
Glitch	depending	precise	depending	depending
Optical	complete	precise	bsr and bf	as required

4 A general fault attack on AES

First we will show how to compute the complete cipher key under the assumption that $\texttt{Nk} \leq \texttt{Nb}$, that is, the block length is greater or equal than the key size. In this case the complete cipher key is used in the initial `AddRoundKey`. We will show how to compute the cipher key bit by bit.

As described in Section 2.1 the cipher key is stored in a $4 \times \texttt{Nr}$ array of bytes (k_{ij}). We denote the l-th bit in byte k_{ij} by $k_{ij}^l, 0 \leq l < 7$. Similarly, the l-th bit in state byte a_{ij} will be denoted by a_{ij}^l. We will show how an attacker can determine k_{ij}^l by inducing a single fault during one encryption. Consider the plaintext $\mathbf{0} = 0^{8 \cdot \texttt{lb}}$, i.e., each bit of $\mathbf{0}$ has value 0. The attacker encrypts $\mathbf{0}$. During the initial transformation `AddRoundKey` the operation

$$a_{ij} := 0^8 \oplus k_{ij}$$

will be performed. Of course, after this operation has been performed, we have $a_{ij} = k_{ij}$. Before the next transformation is performed, the attacker tries to set a_{ij}^l to 0. After this fault has been induced, the encryption proceeds without further faults being induced.

The main observation is that if $k_{ij}^l = 0$, then setting the value of a_{ij}^l to 0, did not change the value of this bit and we still get a correct encryption of $\mathbf{0} = 0^{8 \cdot \texttt{lb}}$. However, if $k_{ij}^l = 1$, then setting the value of a_{ij}^l to 0 results in an incorrect ciphertext and the cryptographic device will answer with `reset`. Hence, from the behavior of the device the attacker can deduce the value of k_{ij}^l.

Altogether, we see that in case $\texttt{Nk} \leq \texttt{Nb}$ we can determine the complete cipher key by encrypting `8lk` times the message $\mathbf{0}$, each time inducing a single fault.

Next we consider the case $\texttt{Nk} > \texttt{Nb}$. In case $\texttt{Nk} > \texttt{Nb}$, an attacker can determine the first `Nb` bytes of the cipher key using the attack described above. Once the attacker knows the first `Nb` blocks of the cipher key, he can simulate the AES encryption process up to the `AddRoundKey` transformation in round 1. More importantly, the attacker can compute a plaintext P, such that during the encryption of P the state (a_{ij}) prior to the application of `AddRoundKey` in round 1 consists of zero bytes only, i.e., $a_{ij} = 0^8$ for all i, j.

To determine a_{ij}^l for $j \geq \texttt{Nk}$, the attacker encrypts the plaintext P. After the transformation `AddRoundKey` has been performed in the first round, the attacker resets a_{ij}^l to 0. After that, the encryption procedure proceeds without further faults. As before and by choice of P, if $k_{ij}^l = 0$ resetting the value of a_{ij}^l has no effect and the correct encryption of P is computed. However, if $k_{ij}^l = 1$, the ciphertext will not be correct. As before, the attacker can determine the value of k_{ij}^l.

Recall that we always have $\texttt{Nk} \leq \texttt{2Nb}$ (see Section 2.1). Hence in the way just described the attacker can determine the complete cipher key. Overall, to compute the cipher key, an attacker needs to compute the encryption of `8lk` plaintexts, each time inducing a single fault.

4.1 A probabilistic fault model

Inspired by technological considerations we will now consider a slightly relaxed fault model. Namely, given the rapid shrink processes of semiconductor technologies, it is plausible to assume that optical attacks using a cheap and unprofessional lithography equipment will not achieve a sufficient resolution to exactly attack single bits in the desired way. In the best case an optical attack will approximately hit the correct physical location to set or reset a specific bit. We model this in the following way. If the value of the bit an attacker tries to reset is 1, then we assume that with probability $p_1 > \frac{1}{2}$ the attacker successfully resets the value of that bit to 0. If the value of the bit the attacker tries to reset is 0, then we assume that the value will change to 1 with probability $p_0 < \frac{1}{2}$. We can assume that the probabilities p_0, p_1 are known to the attacker.

We only describe how to determine the value of k_{00}^0 with this fault model. The generalization to arbitrary bits of the cipher key is straightforward. To compute k_{00}^0 the attacker encrypts the message $\mathbf{0}$ m-times, where m is a parameter to be specified later. In each encryption of $\mathbf{0}$, after the operation

$$a_{00} = 0^8 \oplus k_{00}$$

has been performed in the initial `AddRoundKey`, the attacker tries to reset the value of a_{00}^0. For each encryption the attacker can deduce from the behavior of the cryptographic device whether it yielded a correct or an incorrect ciphertext. If the number of incorrect ciphertexts is at least

$$m\left(\frac{p_1 + p_0}{2}\right)$$

the attacker guesses $k_{00}^0 = 1$. Otherwise, the attacker guesses $k_{00}^0 = 1$.

Let us analyze the probability that the attacker guesses k_{00}^0 correctly. If $k_{00}^0 = 1$, then in each encryption of $\mathbf{0}$ the value of a_{00}^0 will be set to 0 with probability p_1. Hence in the case $k_{00}^0 = 1$, each encryption of $\mathbf{0}$ results in an incorrect ciphertext with probability p_1. Therefore, we expect $p_1 m$ incorrect ciphertexts. By Bernstein's inequality (see for example [GS]), the probability that in this case fewer than $m\left(\frac{p_1+p_0}{2}\right)$ ciphertexts are incorrect is bounded from above by $\exp(-m(p_1 - p_0)^2/16)$.

Similarly, one can show that in case $k_{00}^0 = 0$, the probability that the number of incorrect ciphertexts is larger than $m\frac{p_1+p_0}{2}$ is also bounded from above by $\exp(-m(p_1 - p_0)^2/16)$. Altogether we obtain that the attacker guesses bit k_{00}^0 correctly with probability at least

$$1 - \exp\left(-m\frac{(p_1 - p_0)^2}{16}\right).$$

One checks that with the choice

$$m = \frac{176}{(p_1 - p_0)^2}$$

the attacker correctly guesses k_{00}^0 with probability $1 - 2^{-15}$. Analogously, the other cipher key bits can be determined. If for every cipher key bit we choose $m = 176/(p_1 - p_0)^2$ plaintexts, the probability that every cipher key bit key is guessed correctly is $1 - 2^{-8}$.

For example, if $p_1 = \frac{3}{4}$, $p_0 = \frac{1}{4}$ and `lk` $= 16$, then with the choice $m = 90112$ the attacker guesses the complete cipher key correctly with probability $1 - 2^{-8}$.

We note that the bounds on the probability that the attacker guesses a bit incorrectly can be reduced somewhat by using Chernoff bounds instead of Bernstein's bound (see [GS]).

4.2 Relaxing the timing constraint

In the basic fault model as well as in the probabilistic fault model, we always assumed that the attacker is able to exactly determine the time when he resets the value of a memory bit b. Unfortunately, as described in [CCD], some modern secure microcontrollers are equipped with a randomized timing behavior to counteract statistical side-channel attacks. However, in this section we will show that under some very plausible assumptions this hardware mechanism doesn't yield sufficient protection. More specifically, we only want to assume that the attacker has a certain window of time in which he can reset the value of a particular memory bit b. We assume, that within the time window there are only c operations performed by the AES encryption procedure that involve bit b. The attacker knows c. Furthermore we assume that the time at which bit b is reset is randomly distributed. That is, for each $t = 0, \ldots, c$ the probability that b is reset after the execution of the t-th operation involving b, but before the following operation involving b, is $\frac{1}{c}$. We describe the modifications that have to applied to the basic attack to take care of this weaker assumption. Similar modifications can be carried out for the attack in the probabilistic fault model.

We only describe how to determine the first cipher key bit k_{00}^0. Instead of encrypting the plaintext $\mathbf{0}$, the attacker encrypts several plaintexts $P_1, \ldots, P_m$. In all these plaintexts P_i the leftmost bit has value 0. The remaining bits of the plaintexts are chosen uniformly at random. As before the attacker tries to reset the value of a_{00}^0 after the initial `AddRoundKey` has been performed. Instead he only manages to reset a_{00}^0 within a time window that besides the initial operation $a_{00} := a_{00} \oplus k_{00}$ includes $c-1$ other operations involving a_{00}^0. However, the initial `AddRoundKey` is the first transformation involving a_{00}^0. The next operation involving a_{00}^0 is the `ByteSub` of round 1. We now make the following heuristic assumption:

Assume that bit a_{00}^0 has a fixed value b, while the remaining bits of a_{00} are chosen uniformly at random. Then the leftmost bit of `ByteSub`(a_{00}) is distributed uniformly at random.

From this it follows that unless the attacker manages to reset a_{00}^0 immediately after the initial `AddRoundKey` has been performed, the attacker tries to reset a bit whose value is distributed uniformly at random.

Next we compute the probability that the fault induced by the attacker during the encryption of plaintext P_i leads to an incorrect cipher text. With probability $\frac{1}{c}$ the attacker resets a_{00}^0 immediately after the initial `AddRoundKey`. In this case, the ciphertext will not be correct if and only if $k_{00}^0 = 1$. With probability $1 - \frac{1}{c} = \frac{c-1}{c}$ the attacker resets bit a_{00}^0 following the `ByteSub` of round 1 or later. From the assumption stated above, it follows that in this case the attacker resets the value of a bit whose value is distributed uniformly at random. Therefore the ensuing ciphertext will not be correct with probability $\frac{1}{2}$.

We conclude that for all plaintexts P_i, if $k_{00}^0 = 0$ the encryption of P_i will be incorrect with probability

$$\frac{c-1}{2c}.$$

On the other hand, if $k_{00}^0 = 1$ then the ciphertext for P_i will not be correct with probability

$$\frac{c-1}{2c} + \frac{1}{c}.$$

As in the probabilistic fault model, this difference in probability can be exploited to guess the value of k_{00}^0 correctly with high probability. The attacker simply chooses m large enough and depending on the number of incorrect ciphertexts he sets k_{00}^0 to 0 or 1. The details are exactly as in the previous section, so we omit them.

5 Implementation specific fault attacks

Since a fault-based cryptanalysis (actually an engineering type of attack) might also exploit some peculiar implementation properties, we now take a closer look at some particular implementation details to understand the fault-based vulnerability of AES. We will first present the idea for our implementation specific fault attacks. Then we we will apply this idea to several conceivable implementations of the `xtime` operation. Nevertheless, it should be clear that also other implementations of `xtime` might be suspectible to the kind of attack described below. Moreover, we would like to stress that the ideas developed within Sections 4.1 and 4.2 also apply to the following implementation specific fault attack scenarios.

5.1 Description of the underlying idea

Combining an idea of [YJ] together with ideas from the Timing Analysis of the AES, we will turn the attack of [KQ] into a fault based cryptanalysis of the AES. Depending on the actual realization of the `xtime` operation, different attack scenarios will follow.

Table based fault attacks of the AES First we describe how to determine the first byte of the cipher key, when given the information that a specific `xtime` operation reduces its result by xoring it with the byte $(0, 0, 0, 1, 1, 0, 1, 1)$ (see (3)). But, in contrast to [KQ] we will get this information by inducing a computational error during that specific `xtime` operation. The information retrieval process itself is the implementation despendent part of the attack. For some conceivable implementations it will be described later.

First a $256 \times N$ table T is set up. Here $N \leq 256$ is some parameter to be specified later. Every row in this table corresponds to a possible value for the first cipher key byte k_{00}, while the columns correspond to possible values for the first plaintext byte a_{00}. The table entries $T[k, m], 0 \leq k < 256, 0 \leq m < N$ are defined as follows

$$T[k,m] = \begin{cases} 1 \text{ if the leftmost bit of } \texttt{ByteSub}(k \oplus m) \text{ is } 1 \\ \\ 0 \text{ if the leftmost bit of } \texttt{ByteSub}(k \oplus m) \text{ is } 0 \end{cases} \qquad (4)$$

Next the attacker constructs N plaintexts $m_0, \ldots, m_{N-1}$. The first byte of the plaintext m_i is i. The remaining text bytes are chosen uniformly at random. Now an attacker encrypts the messages m_i and for each i he enforces an error during the encryption of the plaintext. As already said, the actual time and kind of error will be specified below depending on the specific implementation of the `xtime` operation. At this point it suffices to note that during the encryption of plaintext m_i the transformation `MixColumn` in round 1 multiplies the byte $\texttt{ByteSub}(k_{00} \oplus i)$ by 02, or equivalently `MixColumn` applies $\texttt{xtime}(\texttt{ByteSub}(k_{00} \oplus i))$. This operation will be shown to be highly vulnerable to a fault attack revealing whether the leftmost bit of $\texttt{ByteSub}(k_{00} \oplus i)$ is 1 or 0. Hence, the attacker is then able to predict the leftmost bit of $\texttt{ByteSub}(k_{00} \oplus i)$.

Then the attacker compares his predictions for the leftmost bits of $\texttt{ByteSub}(k_{00} \oplus i), i = 0, \ldots, N-1$, with the entries in table T. If the attacker chooses $N = 256$ and his predecitions are correct, this comparison reveals the first cipher key byte. However, we expect that fewer than 256 plaintexts P_i will suffice to determine the first cipher key byte. In fact, Koeune and Quisquater observed that in their timing attack $N = 20$ already suffices. Since a fault attack will yield much more reliable predictions for the leftmost bits of $\texttt{ByteSub}(k_{00} \oplus i)$ we expect that in our case $N \approx 16$ will suffice.

Note that the main diagonal of the matrix in (1) consists of 02 only. Hence during `MixColumn` every state byte gets multiplied by 02 once. From this, one concludes that the method to compute the cipher key byte k^0_{00} can be used to compute the other cipher key bytes as well. Due to the AES's key schedule as described in section 2.1, the cipher is broken once the attacker has determined `Nk` consecutive bytes of the round key. Thus, for $\texttt{Nk} \leq \texttt{Nb}$ the above methods breaks AES. For smaller block sizes, similiar ideas as presented in section 4 apply.

Based on this table approach we now present in figure 2 the resulting common skeleton for all of our following implementation specific fault attacks.

5.2 The simplest attack against an unskilled textbook-secured implementation

The simplest and also most obvious way for a Timing Analysis (and SPA) resistant implementation of the `xtime` operation seems to be given by the following. Save the most significant bit of the input byte, shift the byte by one bit to the left, perform *two* xor's with the shifted byte. Finally, according to the most significant bit of the input byte, return one of the two previously computed results as shown in the figure 3.

```
build table $T[k, m]$
build plaintexts $m_0, \ldots, m_N$

**for** $i := 0$ **to** $N$ **do**
    encrypt the message $m_i$ and
    disturb the operation xtime(ByteSub($k_{00} \oplus i$)) within MixColumn
    of round 1 by an appropriate physical attack as described later
    **if** output refused or incorrect **then**
        $\tilde{T}[i]$ is set to 0 or 1 depending on the xtime realization
**od**

by comparing the tables $T$ and $\tilde{T}$ determine the first key byte $k_{00}$

**output:** $k_{00}$
```

Fig. 2. Common fault attack skeleton, revealing the first key byte.

```
**input:** $a = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$

$f := a_7$
$a := (a_6, a_5, a_4, a_3, a_2, a_1, 0)$
xtime[0] $:= a \oplus (0, 0, 0, 0, 0, 0, 0, 0)$
xtime[1] $:= a \oplus (0, 0, 0, 1, 1, 0, 1, 1)$
**return**(xtime[$f$])

**output:** xtime($a$)
```

Fig. 3. Timing Analysis secured `xtime(a)` realization.

Let us now analyze what happens, if an attacker uses the fault attack skeleton outlined above. That is, the attacker encrypts N plaintexts m_i, $i = 1, \ldots, N$. Moreover, in each encryption the attacker disturbs the computation of `xtime`[0]. Here we can apply a very liberal fault model. We assume that the attacker is able to enforce an arbitrary wrong value

$$\texttt{xtime}[0]' \neq \texttt{xtime}[0].$$

Then, depending on bit a_7 of the original byte a, the following will result. If $a_7 = 1$, due to the fact that `xtime`[1] will be returned, the wrong result `xtime`[0]$'$ is of no further interest and will be therefore discarded. Therefore the encryption yields a correct ciphertext. However, if $a_7 = 0$, the wrong result `xtime`[0]$'$ will lead to a wrong ciphertext and the cryptographic device will answer with `reset`. Hence, from the behavior of the device, the attacker can determine the bit a_7. Now the attacker can proceed as described in Section 5.1 to completely break the cipher.

Note, that we did not make any assumption on the kind of introduced error during the computation of `xtime`[0]. We simply need a wrong result. Thus it is indeed easy to realize by any one of the methods presented in 3, indicating that this attack is actually devestating.

5.3 An attack against a possible real implementation

After we have seen an unskilled textbook implementation of the `xtime` operation, we will now consider a slow but real implementation written in an extended 8051 assembler language. We selected an extended 8051 microcontroller as it is the most commonly used controller in todays' smartcards. Consider the following `xtime` realization, which is obviously secure against a Timing Analysis.

```
; xtime(a)

; parameters: "a" in accu

; return value: "xtime(a)" in accu

xtime:

  MOV B, A                          ; B:=a
  SLL B                             ; B:=(a_6,a_5,...,a_1,0)
  DIV A, #10000000b                 ; A:=(0,0,...,0,a_7)
  MUL A, #00011011b                 ; A:=a_7*(0,0,0,1,1,0,1,1)
  XRL A, B                          ; A:=A⊕(a_6,a_5,...,a_1,0)
  RET
```

Fig. 4. Timing Analysis secured `xtime`(**a**) realization.

As above, let us analyze what happens, if an attacker applies the fault attack skeleton, i.e., he encrypts N plaintexts m_i, $i = 1, \ldots, N$. However, this time the attacker will apply a glitch attack, cf. Section 3.2, to omit the execution of the `MUL A, #00011011b` instruction.

Then, depending on bit a_7 of the input byte a, the following will result. If $a_7 = 0$, the register `A` will be zero after the instruction `DIV A, #10000000b`. Thus, omitting via a glitch the following `MUL A, #00011011b` instruction does not matter, as it would write back to register `A` a zero, still guaranteeing a correct encryption of m_i. Hence, in case $a_7 = 0$, the encryption processes ends with a correct ciphertext. However, in the case of $a_7 = 1$, omitting the `MUL A, #00011011b` instruction, will result in a value 1 in register `A`, whereas $(0, 0, 0, 1, 1, 0, 1, 1)$ would be the correct value. Thus, the cryptographic device will detect a computational error and will answer with `reset`, resulting in an answer which indicates that a wrong computation happened which will actually be observed by the attacker. Therefore, from the behavior of the cryptographic device, the attacker can deduce the value of bit a_7. As before, applying in this fashion the fault attack skeleton to every one of the 32 key bytes will completely break the cipher.

5.4 An attack against a suggested implementation

Now, that we have seen some vulnerable `xtime` realizations we will consider the implementation actually suggested by the inventors of the AES, as proposed in [DR2]. Inspired by the knowledge about the timing analysis vulnerability of the AES, they proposed to implement `xtime` as an array T consisting of 256 bytes, where

$$T[\mathbf{b}] := \texttt{xtime}(\mathbf{b}).$$

As above, an attacker applies the fault attack skeleton as described in Section 5.1. This time the attacker will apply an optical attack, cf. Section 3.3. Clearly it is most conceivable that the table T will be stored in ROM and therefore is of no use to an attacker. However, the whole *current* state of the AES encryption clearly must be stored in RAM. Therefore, this time the attacker will reset via an optical attack on the RAM the bit a_7 of the state byte a.

Although the analysis is very similiar to the former two cases, for completeness sake we will include it. Depending on the bit a_7 of the input byte a, the optical attack will have the following effect. If $a_7 = 0$, trying to reset it via optical attacks has no effect. Therefore, the table T is consulted for the correct value and the cryptographic device will answer with a correct ciphertext. However, in case $a_7 = 1$, resetting this bit to 0 must result in a wrong encryption, since the table T is consulted for a wrong value. Hence, from the behavior of the cryptographic device an attacker can determine the value of bit a_7. As before described in Section 5.1 this can be used to completely break the cipher.

We note that recently the table based approach for `xtime` was also shown to be suspectible to a Differential Power Analysis by [YJ]. Given all this, one can definitely say that the suggestion of [DR2] is completely insecure against physical side-channel attacks.

5.5 An attack against a hardware implementation

So far we have only considered software realizations of `xtime`, eventually we will briefly strive over a possible hardware realization. However, as a complete hardware description even if only for the `MixColumn` transformation is clearly out of the present papers scope, we will concentrate again on the `xtime` circuit. In this vein of building a dedicated AES circuit, it is widely anticipated, cf. [Wo], that `xtime` should be realized by the following very simple circuit, being beside clearly secure against a Timing Analysis.

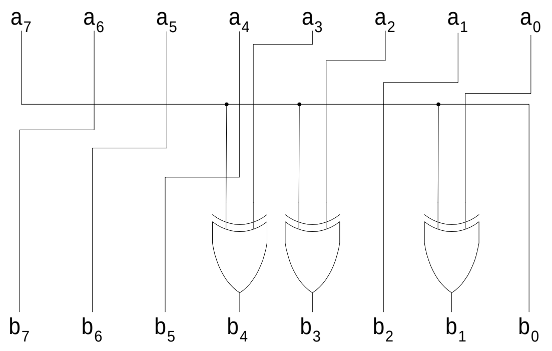

Fig. 5. Circuit for $\mathbf{b} :=$ `xtime`$(\mathbf{a})$.

We will now elaborate this situation a bit more in depth. Forced by the tough area requirements for a chipcard IC, most AES hardware architecture proposals are silicon size optimized. This means in particular, that such AES hardware modules really compute the AES operations `ByteSub`, `MixColumn` and of course `xtime`, instead of using large ROM based lookup tables to compute the corresponding transformations, cf. [SMTM,Wo,WOL]. However, due to the logical depth of the corresponding computations, those architectures have to take into account that every single transformation needs at least one clock cycle, and most often it will require more cycles. Thus, between different transformations the whole AES encryption state is stored within a register bank, which is actually realized by a large number of flip-flops, cf. [WE]. In particular, the whole encryption state prior to the execution of `MixColumn` must be stored within flip-flops, cf. [WE]. Indeed, the inputs $(a_7, \ldots, a_0)$ to a corresponding `xtime` circuit are actually flip-flops, storing the results from the previous `ShiftRow` transformation. But this in turn means that again we can use an optical attack, cf. Section 3.3, against these flip-flops, exactly as described in [SA]. Indeed, this time the attacker will reset (during the time period when the results from the previous `ShiftRow` are stored within the flip-flops) the bit a_7 of the state byte a via an optical attack on the aforesaid flip-flop. The rest of the analysis is completely analog to the table based `xtime` realization.

6 Countermeasures

We would like to point out that some modern high-end crypto smartcards are protected by various and numerous means of sophisticated hardware mechanisms to detect any intrusion attempt to their system behavior, cf. [Ma,MACMT,MAK,NR]. Various but not all hardware manufacturers of cryptographic devices such as smartcard ICs have been aware of the importance of protecting their chips against intrusions by, e.g., external voltage variations, external clock variations, light attacks, etc. To do so they use carefully developed logic families, sensors, filters, regulators, etc.

And indeed, only this special hardware countermeasures might give rise to a trustworthy functionality of the chip. The reason is that only those proprietary and secretly kept hardware mechanisms are indeed designed to counteract the source of a physical attack and not to counteract their effect on computations. Counteracting effects rather than sources is usually done by naive software countermeasures or some proposed hardware architectures [KWMK]. Namely, the latter ones simply check the computed ciphertext for correctness and will cause the chip to react with some kind of alarm if an erroneous ciphertext has been detected. Unfortunately, as shown in the present paper, this is obviously detectable and exploitable to mount our fault attacks.

Another possibility could be a complete and good randomization of the AES's computation, i.e., randomizing the plaintext and the cipherkey. A first step in this direction was recently done by [AG] and might be helpful to develop appropriate software countermeasures.

7 Acknowledgments

We would like to thank Alexander May for careful reading of our paper.

References

[A] R. Anderson, *Security Engineering*, John Wiley & Sons, New York, 2001.

[AG] M. L. Akkar, C. Giraud, "An implementation of DES and AES, secure against some attacks", *Proc. of CHES '01*, Springer LNCS vol. 2162, pp. 315-324, 2001.

[AK1] R. Anderson, M. Kuhn, "Tamper Resistance – a cautionary note", *Proc. of 2nd USENIX Workshop on Electronic Commerce*, pp. 1-11, 1996.

[AK2] R. Anderson, M. Kuhn, "Low cost attacks attacks on tamper resistant devices", *Proc. of 1997 Security Protocols Workshop*, Springer LNCS vol. 1361, pp. 125-136, 1997.

[BDL] D. Boneh, R. A. DeMillo, R. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations" *Journal of Cryptology* **14**(2):101-120, 2001.

[BDHJNT] F. Bao, R. H. Deng, Y. Han, A. Jeng, A. D. Narasimbalu, T. Ngair, "Breaking public key cryptosystems on tamper resistant dives in the presence of transient faults", *Proc. of 1997 Security Protocols Workshop*, Springer LNCS vol. 1361, pp. 115-124, 1997.

[BS97] E. Biham, A. Shamir, "Differential fault analysis of secret key cryptosystems", *Proc. of CRYPTO '97*, Springer LNCS vol. 1294, pp. 513-525, 1997.

[BS99] E. Biham, A. Shamir, "Power analysis of the key scheduling of the AES candidates", *Proc. of the second AES conference*, pp. 115-121, 1999.

[BS02] J. Blömer, J.-P. Seifert, "On the power of optical attacks to induce faults on cryptosystems".

[BMM] I. Biehl, B. Meyer, V. Müller, "Differential fault attacks on elliptic curve cryptosystems", *Proc. of CRYPTO '00*, Springer LNCS vol. 1880, pp. 131-146, 2000.

[CCD] C. Clavier, J.-S. Coron, N. Dabbous, "Differential Power Analysis in the presence of Hardware Countermeasures", *Proc. of CHES '00*, Springer LNCS vol. 1965, pp. 252-263, 2000.

[CJRR] S. Chari, C. Jutla, J. R. Rao, P. J. Rohatgi, "A cautionary note regarding evaluation of AES candidates on smartcards", *Proc. of the second AES conference*, pp. 135-150, 1999.

[CKN] J.-S. Coron, P. Kocher D. Naccache, "Statistics and Secret Leakage", *Proc. of Financial Cryptography*, Springer LNCS, 2000.

[DR1] J. Daemen, V. Rijmen, "Resistance against implementation attacks: a comparative study", *Proc. of the second AES conference*, pp. 122-132, 1999.

[DR2] J. Daemen, V. Rijmen, *The Design of Rijndael*, Springer-Verlag, Berlin, 2002.

[DPV] J. Daemen, M. Peeters, G. Van Assche, "Bitslice ciphers and implementation attacks", *Proc. of Fast Software Encryption 2000*, Springer LNCS vol. 1978, pp. 134-149, 2001.

[GS] G. R. Grimmett, D. R. Stirzaker, *Probability and random processes*, Oxford Science Publications, Oxford, 1992.

[Gu1] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory", *Proc. of 6th USENIX Security Symposium*, pp. 77-89, 1997.

[Gu2] P. Gutmann, "Data Remanence in Semiconductor Devices", *Proc. of 7th USENIX Security Symposium*, pp. ?-?, 1998.

[ISO] International Organization for Standardization, "ISO/IEC 7816-3: Electronic signals and transmission protocols", `http://www.iso.ch`, 2002.

[JLQ] M. Joye, A. K. Lenstra, J.-J. Quisquater, "Chinese remaindering based cryptosystem in the presence of faults", *Journal of Cryptology* **12**(4):241-245, 1999.

[JPY] M. Joye, P. Pailler, S.-M. Yen, "Secure Evaluation of Modular Functions", *Proc. of 2001 International Workshop on Cryptology and Network Security*, pp. 227-229, 2001.

[JQBD] M. Joye, J.-J. Quisquater, F. Bao, R. H. Deng, "RSA-type signatures in the presence of transient faults", *Cryptography and Coding*, Springer LNCS vol. 1335, pp. 155-160, 1997.

[JQYY] M. Joye, J.-J. Quisquater, S. M. Yen, M. Yung, "Observability analysis — detecting when improved cryptosystems fail", *Proc. of CT-RSA Conference 2002*, Springer LNCS vol. 2271, pp. 17-29, 2002.

[KR] B. Kaliski, M. J. B. Robshaw, "Comments on some new attacks on cryptographic devices", *RSA Laboratories Bulletin* **5**, July 1997.

[Kn] D. E. Knuth, *The Art of Computer Programming, Vol.2: Seminumerical Algorithms*, 3rd ed., Addison-Wesley, Reading MA, 1999.

[KK] O. Kömmerling, M. Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors", *Proc. of the USENIX Workshop on Smartcard Technologies*, pp. 9-20, 1999.

[KQ] F. Koeune, J.-J. Quisquater, "A timing attack against Rijndael", *Université catholique de Louvain*, TR CG-1999/1, 6 pages , 1999.

[Koca] O. Kocar, "Hardwaresicherheit von Mikrochips in Chipkarten", *Datenschutz und Datensicherheit* **20**(7):421-424, 1996.

[Koch] P. Kocher, "Timing attacks on implementations of Diffie-Hellmann, RSA, DSS and other systems", *Proc. of CYRPTO '97*, Springer LNCS vol. 1109, pp. 104-113, 1997.

[KJJ] P. Kocher, J. Jaffe, J. Jun, "Differential Power Analysis", *Proc. of CYRPTO '99*, Springer LNCS vol. 1666, pp. 388-397, 1999.

[KWMK] R. Karri, K. Wu, P. Mishra, Y. Kim, "Concurrent error detection of fault-based side-channel cryptanalysis of 128-bit symmetric block ciphers", *Proc. of IEEE Design Automation Conference*, pp. 579-585, 2001.

[Ma] D. P. Maher, "Fault induction attacks, tamper resistance, and hostile reverse engineering in perspective", *Proc. of Financial Cryptography*, Springer LNCS vol. 1318, pp. 109-121, 1997.

[MvOV] A. J. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, 1997.

[Me] T. Messerges, "Securing the AES finalists against power analysis attacks", *Proc. of Fast Software Encryption 2000*, Springer LNCS vol. 1978, pp. 150-164, 2001.

[MAK] S. W. Moore, R. J. Anderson, M. G. Kuhn, "Improving Smartcard Security using Self-Timed Circuit Technology", *Fourth AciD-WG Workshop*, Grenoble, ISBN 2-913329-44-6, 2000.

[MACMT] S. W. Moore, R. J. Anderson, P. Cunningham, R. Mullins, G. Taylor, "Improving Smartcard Security using Self-Timed Circuit Technology", *Proc. of Asynch 2002*, IEEE Computer Society Press, pp. ?-?, 2002.

[NR] D. Naccache, D. M'Raihi, "Cryptographic smart cards", *IEEE Micro*, pp. 14-24, 1996.

[Pai] P. Pailler, "Evaluating differential fault analysis of unknown cryptosystems", *Gemplus Corporate Product R&D Division*, TR AP05-1998, 8 pages, 1999.

[Pe] I. Petersen, "Chinks in digital armor — Exploiting faults to break smartcard cryptosystems", *Science News* **151**(5):78-79, 1997.

[RSA] R. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Comm. of the ACM* **21**:120-126, 1978.

[SQ] D. Samyde, J.-J. Quisquater, "ElectroMagnetic Analysis (EMA): Measures and Countermeasures for Smart Cards", *Proc. of Int. Conf. on Research in Smart Cards, E-Smart 2001*, Springer LNCS vol. 2140, pp. 200-210, 2001.

[SMTM] A. Satoh, S. Morioka, K. Takano, S. Munetoh, "A compact Rijndael hardware architecture with S-Box optimization", *Proc. of ASIACRYPT '01*, Springer LNCS, pp. 241-256, 2001.

[SA] S. Skorobogatov, R. Anderson, "Optical Fault Induction Attacks", *Proc. of 2002 IEEE Symposium on Security and Privacy*, 2002.

[Sh] A. Shamir, "Method and Apparatus for protecting public key schemes from timing and fault attacks", U.S. Patent Number 5,991,415, November 1999; also presented at the rump session of EUROCRYPT '97.

[WE] N. H. E. Weste, K. Eshraghian, *Principles of CMOS VLSI Design*, 2nd ed., Addison-Wesley, Reading MA, 1994.

[Wo] J. Wolkerstorfer, "An ASIC implementation of the AES MixColumn-operation", Graz University of Technology, Institute for Applied Information Processing and Communications, Manuscript, 4 pages, 2001.

[WOL] J. Wolkerstorfer, E. Oswald, M. Lamberger, "An ASIC implementation of the AES S-Boxes", *Proc. of CT-RSA Conference 2002*, Springer LNCS vol. 2271, 2002.

[YJ] S.-M. Yen, M. Joye, "Checking before output may not be enough against fault-based cryptanalysis", *IEEE Trans. on Computers* **49**:967-970, 2000.

[YKLM1] S.-M. Yen, S.-J. Kim, S.-G. Lim, S.-J. Moon, "RSA Speedup with Residue Number System immune from Hardware fault cryptanalysis", *Proc. of the ICISC 2001*, Springer LNCS, 2001.

[YKLM2] S.-M. Yen, S.-J. Kim, S.-G. Lim, S.-J. Moon, "A countermeasure against one physical cryptanalysis may benefit another attack", *Proc. of the ICISC 2001*, Springer LNCS, 2001.

[YT] S.-M. Yen, S. Y. Tseng, "Differential power cryptanalysis of a Rijndael implementation", LCIS Technical Report TR-2K1-9, Dept. of Computer Science and Information Engineering, National Central University, Taiwan, 2001.

[ZM] Y. Zheng, T. Matsumoto, "Breaking real-world implementations of cryptosystems by manipulating their random number generation", *Proc. of the 1997 Symposium on Cryptography and Information Security*, Springer LNCS, 1997.