Passive and Active Combined Attacks on AES
–Combining Fault Attacks and Side Channel Analysis–

Christophe Clavier
INSTITUT D’INGÉnierIE INFORMATIQUE DE LIMogES (3iL)
UNIVERSITÉ DE LIMogES - XLIM
F-87000 Limoges, France
christophe.clavier@{3il.fr, unilim.fr}

Benoît Feix, Georges Gagnerot and Mylène Roussellet
INSIDE CONTACTLESS
41 Parc Club du Golf
13856 Aix-en-Provence, Cedex 3, France
firstnamefirst-letterfamilyname@insidefr.com

Abstract

Tamper resistance of hardware products is currently a very popular subject for researchers in the security domain. Since the first Kocher side-channel (passive) attack, the Bellcore researchers and Biham and Shamir fault (active) attacks, many other side-channel and fault attacks have been published. The design of efficient countermeasures still remains a difficult task for IC designers and manufacturers as they must also consider the attacks which combine active and passive threats. It has been shown previously that combined attacks can defeat RSA implementations if side-channel countermeasures and fault protections are developed separately instead of being designed together.

This paper demonstrates that combined attacks are also effective on symmetric cryptosystems and shows how they may jeopardize a supposedly state of the art secure AES implementation.

Keywords: Advanced Encryption Standard, Collision Fault Analysis, Correlation Side Channel Analysis.

1 Introduction

Protecting smart secure devices such as smartcards has become a challenging task. Many different attacks can threaten these products. From the invasive attacks which require expensive material such as a probing station or a Focused Ion Beam to the most recent non invasive attacks such as side-channel and fault attacks a developer has to consider all these threats and implement the appropriate countermeasures. These latest attacks do not require as expensive material as the invasive attacks do and are thus more accessible to hackers.

In the first half of the 90’s smartcards were considered as tamper resistant products until the original Simple side-channel analysis (SSCA). Introduced by Kocher et al. [15, 16], these attacks include Timing Attacks (1996), SPA(1998) and SEMA, and Differential side-channel analysis (DSCA) including DPA (1998) and DEMA. The smartcard industry responded rapidly to these new threats and implemented appropriate countermeasures to protect products. At the same time the first fault (active) attacks, named Differential Fault Analysis DFA, were published by Boneh, DeMillo and Lipton [6] and Biham and Shamir [4]. The first technique, also known as the Bellcore attack, threatens implementations of the RSA cryptosystem [22] while the second one targets the DES algorithm [9].

Since these publications many new passive and active attack techniques have been discovered and many countermeasures have been designed.

Countermeasures against active and passive attacks are generally studied and proposed separately, and protecting a product from both techniques usually consists in superposing both kinds of countermeasures. In [3] the authors showed that simply superposing countermeasures is not sufficient as they succeeded in break-
ing an RSA implementation that used state of the art countermeasures against side-channel attacks and fault attacks. Their Passive and Active Combined Attack, PACA, on an RSA implementation demonstrates that naively adding countermeasures together is not sufficient and that implementing these protections must be done carefully be means e.g. of the infective methodology.

In this paper we present another passive and active combined attack on a state of the art SCA protected AES [10]. We combine a particular fault attack technique named Collision Fault Analysis (CFA) that was introduced by Brier et al. [7] in 2004, with the classic Correlation side-channel analysis (CSCA) introduced by Blömer and Seifert [5] in 2003, with the complex Correlation side-channel analysis (CSCA) introduced by Brier et al. [7] in 2004.

This paper is organized as follows. Section 2 gives an overview of active and passive attacks with a focus on the collision fault analysis and the correlation side-channel analysis. We present in Section 3 the AES state of the art implementation chosen for this study and explain why this implementation is resistant to the previously published CFA. In Section 4 we introduce our combined attack and explain how, with the same fault model as the CFA, it can recover the secret key on our AES implementation. We discuss the countermeasures in Section 5, describe a safe-error variant of our attack which defeats these countermeasures in Section 6, and conclude the paper in Section 7.

2 Side Channel and Fault Analysis Background

Passive attacks consist in observing side-channel information, such as the power consumed by the chip while performing sensitive operations during a cryptographic computation. Active attacks consist in perturbing the device when it is processing sensitive data or calculations. Both techniques may result in the recovery of the secrets.

2.1 Side Channel Analysis

Power and electromagnetic analysis rely on the following physical property: a microprocessor is physically made of thousands of logical gates which switch differently depending on the operations being executed and the data being manipulated. The power consumption and the electromagnetic radiation, which depend on these gate switching may leak information on the executed instructions and the manipulated data. Consequently, by monitoring the power consumption or radiation of a device performing cryptographic operations, an observer can infer information on the implementation of the program executed and on the secret data involved. Basic SSCA consists in analyzing the secret key manipulations on a single power curve, for instance during the key scheduling operations in the DES or AES rounds. DPA is a more powerful method which consists in validating an hypothesis on some key bits, by a statistical treatment on many execution power curves of the targeted embedded implementation. The complexity is then reduced to very few calculations compared to a classical cryptanalysis or to a brute force attack on the key. For instance a classical DPA on an unprotected software implementation of AES on a standard microcontroller typically requires between 500 to few thousands power curves. Some improvements of DPA have been published in the previous years, among which the Correlation Power Analysis CPA technique requires far fewer curves for recovering the key than the original DPA. Recovering the key by CPA on unprotected AES implementations may require between only 50 and few hundred curves. Other recent methods have been published that also improve the side-channel attacks [11, 24, 21]. Most common countermeasures against power analysis, and particularly DPA and CPA, consist in using random values for masking the operations. In this case even if an attacker makes guesses on some secret key bits, he can not predict any intermediate value as another unknown variable, the random mask, is part of any intermediate data during the computation. However in this case a more complex but realistic attack, named High Order Differential Power Analysis (HODPA), presented by Messerges [18], is still applicable if the mask values are identical on different bytes, and/or if some different instants on a same power curve can be used to eliminate the random mask effect.

We now briefly present the CPA technique which is the passive component of our combined attack.

2.1.1 Correlation Power Analysis

The power consumption of the device is supposed to vary linearly with $H(D \oplus R)$, the Hamming distance between the data manipulated $D$ and a reference state $R$. The power consumption model $W$ is then defined as $W = \mu \cdot H(D \oplus R) + \nu$, where $\nu$ captures both the experimental noise and the non modeled part of the power consumption. The linear correlation factor: $\rho_{C,H} = \frac{\text{cov}(C,H)}{\sigma_C \sigma_H}$ is then used to correlate the power curves $C$ with this value $H(D \oplus R)$. Knowing that the maximum correlation factor is obtained for the correct guess of secret key bits, an attacker can try all possible secret bits values and select the value corresponding to the highest correlation.
2.2 Fault Analysis

Fault effects and perturbations on electronic devices were first observed in the 1970’s in the aerospace industry. Later the Differential Fault Analysis, DFA for attacking embedded symmetric cryptosystems was introduced by Biham and Shamir [4] in 1997. In this paper the authors explain how to recover the secret key by using between 50 and 200 ciphertexts. For years this threat was considered as only theoretical until the first practical results of light attacks were presented (on an RSA implementation) by Anderson and Skorobogatov [23]. The DFA has subsequently been studied and applied on DES in [12] where Giraud and Thiebaud recover the key by means of only 2 faulty ciphertexts. In the case of the AES many attacks have been proposed [20, 8, 13] that allow the secret key to be recovered by using as few as 2 faulty ciphertexts.

We now present the Collision Fault Analysis technique which is the active component used in our combined attack.

2.2.1 Collision Fault Analysis

In [5] Blömer and Seifert first published a CFA on the first XOR of the AES. They assume a fault model where the attacker has the ability to force to zero any chosen bit of the result of this XOR operation. Then they compare a correct and a faulted AES execution for the same message. If both ciphertexts are equal the original value of the result bit is 0, otherwise it is 1. Knowing the message and scanning the different key bytes, the whole 128-bit AES key is retrieved with 128 faulty executions. An interesting property is that the classical countermeasures which consist in checking the computation, for instance by executing the AES twice and comparing the results, do not prevent this attack. Indeed whether the card detects the fault or not will provide the attacker with the same information as whether or not the fault corrupted the ciphertext.

Later Hemme [14] presented the first CFA on the DES. His attack consists in introducing one bit errors in the first rounds of the algorithm. Then by computing chosen message encryption with the card (without injecting faults) the attacker obtains collisions that he can exploit to recover information on the secret key. With enough collisions he can recover the whole secret key. In this case, verifying the whole DES computation is an efficient countermeasure.

Another CFA analysis on the AES first XOR computation can be done when the fault effect resets a whole byte (or many bytes) instead of a bit. In this attack an induced fault resets the result of a XOR between one message byte $M_j$ and one key byte $K_j$ – with the other key addition byte results not being affected. The attacker stores the faulty ciphertext $C'$ and asks the card to encrypt the 256 messages $M$ with $M_j$ taking all possible byte values. One of these 256 ciphertexts will be equal to $C'$. This collision is produced for $M_j$ verifying $M_j \oplus K_j = 0$, which indicates that $K_j = M_j$.

Amiel et al. [2] adapted this CFA to an AES protected from first order DPA by random masking. In this implementation the same random byte $r_1$ is used for masking all 16 message bytes and the same random byte $r_2$ is applied on all 16 key bytes. In that case a single random byte $r = r_1 \oplus r_2$ is applied on all the bytes of intermediate values throughout the computation. The authors succeeded in faulting 2 to 16 bytes of the result of the first XOR. Then by searching collisions they obtained relations between known input bytes and key bytes masked. Exploiting these relations allows them to recover the secret key. This attack needs the pre-computation with the card of $2^{23}$ non faulty ciphertexts and in practice 112 faulty ciphertexts were used. Note that this attack is applicable only if the same random mask $r$ is applied on all of the 16 bytes of the intermediate values. The targeted implementation was not protected against high order differential analysis and in particular against a second order analysis. State of the art implementations are thus not vulnerable to this CFA.

3 Targeted AES Implementation

We present here the implementation targeted by our attack. We have chosen a state of the art side channel resistant AES implementation. To prevent DPA and CPA attacks, a 16-byte random mask is used to mask the input message (and another one to mask the key). This random mask is composed of 16 different random bytes that can change at each round. This targeted implementation is designed to resist to the HODPA attack presented in [1] and [17].

To realize such an implementation it is not possible to use a 256-byte substitution table as randomizing this substitution table for each random byte $r_0, \ldots, r_{15}$ would necessitate precomputing and storing $16 \times 256$-byte substitution tables, one for each byte $r_i$. Moreover these tables would need to be recomputed at each round for changing the mask between each round. The chosen implementation is the one presented by Oswald et al. in [19], the inversion is here computed masked in $GF(2^4)$. In this case all the 16 bytes of the message and the intermediate calculations are masked with different random bytes. This implementation is described in Fig. 3 in Appendix.
Table 1. Performance (cycles) and memory costs (bytes) for AESDPA and AESHODPA implementations

<table>
<thead>
<tr>
<th></th>
<th>Cycles</th>
<th>ROM</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESDPA</td>
<td>20000</td>
<td>4000</td>
<td>256 + 65</td>
</tr>
<tr>
<td>AESHODPA</td>
<td>51000</td>
<td>5500</td>
<td>75</td>
</tr>
</tbody>
</table>

Note that, as previously stated, the CFA presented by Amiel et al. is not applicable on our targeted implementation.

We have carried out two implementations of a secure AES on an 8-bit microprocessor with different security levels. The first one is resistant to DPA attacks and takes 20000 cycles (2 ms at 10 MHz). Data are masked by the same byte which requires precomputing only one substitution table for one AES execution. This implementation is not resistant to HODPA attacks and is also vulnerable to Amiel et al. CFA. We also carried out the implementation described above which uses inversion in $GF(2^4)$. All data are masked by different bytes which change between each round. This implementation is resistant to HODPA attacks and takes 51000 cycles (5.1 ms at 10 MHz). We will refer to both these implementations as AESDPA and AESHODPA respectively. The performance and memory footprint figures for both implementations are presented in Table 1. We introduce the AESDPA implementation here only for comparison purposes to illustrate the cost implied for protecting an AES from HODPA. Only the AESHODPA implementation will be referred to in our attacks.

4 Passive and Active Combined Attack on Masked AES

In this section we present an analysis that can be carried out on our AESHODPA implementation with the same fault model as in previous publications. Our proposed attack targets the first round calculations of the AES and necessitates choosing the input message and obtaining the ciphertext computed by the card. Compared to previous CFA on masked AES, it does not require a large number of messages to be encrypted by the card.

Notation: We denote by $M = (M_0, \ldots, M_{15})$ the input message and by $K = (K_0, \ldots, K_{15})$ the key used by the card for encrypting $M$. Given a message $M$, we also denote by $(M | \text{condition})$ the message $M$ modified so that the condition holds. For instance, messages $(M | M_j = 0)$ and $(M | M_j = 1)$ are identical except on the $j^{th}$ byte which is 0 in first case and 1 in the other. We will also refer to a faulty computation or a result thereof by means of the superscript symbol $^\dagger$. For instance $C^\dagger$ will refer to a faulty ciphertext.

4.1 Fault Model

We consider the following fault model: the result of an operation XOR can be set to zero – or to a not necessarily known constant value – by the attacker. This model has previously been assumed in several fault analysis papers and can be considered as realistic since practical results were also presented. In practice such kind of fault effect can be induced in a card by the following events:

- An operation can be bypassed. For instance the instruction to be executed is replaced by a NOP. As explained in [25] the opcode fetched by the microprocessor may be replaced by 0x00 that, in some products, corresponds to the NOP operation. In this case the expected computation is not done, and the result register keeps its previous value that may be either 0 or another constant value. If the value is not constant the attack will be not possible.
- A loop counter can be modified, for instance in [2] the AddRoundKey operation is bypassed on some bytes by modifying (reducing) a counter value.
- The processing in the ALU can be perturbed and an XOR result can thus be modified to zero or a constant value.

Figure 1 gives an example of a typical code on an 8-bit microprocessor which can be attacked by what has been presented above. It represents the XOR operation between the masked message and the masked key carried out at the beginning of a secure AES. We can observe in this code that our fault model can be used to model different effects, if a NOP operation is executed, if a XOR result is forced to 0; if the counter value R2 is modified or if the JNZ operation is perturbed, etc...

4.2 Attack on the First Key Addition

As in [2] we assume that the key addition before the first round can be perturbed and one or many bytes resulting from this addition can be set to zero. We describe our analysis for the first bytes $M_0$ and $K_0$ of
the message and the key. The analysis will be identical for the other byte indices.

We denote by \( rm = (r_{m0}, \ldots, r_{m15}) \) and by \( rk = (r_{k0}, \ldots, r_{k15}) \) the two 16-byte random masks on the message and the key respectively. The resulting mask of the XOR between the message and the key is denoted by \( r = rm \oplus rk \).

For a normal execution the first byte of the XOR result is:

\[
B_0 = (M_0 \oplus r_{m0}) \oplus (K_0 \oplus r_{k0}) = M_0 \oplus K_0 \oplus r_0
\]

For a faulty execution we have:

\[
B^\delta_0 = 0
\]

The key observation is that the effect of the fault is to introduce a differential \( \delta \) on the byte value just before the first round S-Box computation.

\[
B^\delta_0 = B_0 \oplus \delta, \quad \text{with} \quad \delta = M_0 \oplus K_0 \oplus r_0
\]

The same effect on the execution would have been obtained, without a fault, when using an input message which differs from the initial one by this differential.

Considering without loss of generality an initial message \( M = (0, \ldots, 0) \), the differential then reduces to \( \delta = K_0 \oplus r_0 \), and we have a collision opportunity corresponding to the following equation:

\[
C^i = AES^i(M) = AES(M|M_0 = \delta)
\]

Note that a normal execution with \((M|M_0 = \delta)\) will produce a collision with \( C^i \) whatever the random mask values for this execution.

For any ciphertext \( C^i \) obtained by injecting a fault we have the following properties:

1. \( C^i \) is characteristic of the mask value \( r_0 \) in the faulty execution.

2. We can recover the value \( K_0 \oplus r_0 \) of this execution by identifying which input message leads to a collision with \( C^i \) without fault.

By computing \( AES(M|M_0 = u) \) for all \( u = 0, \ldots, 255 \), we can identify the \( u \) value which verifies the relation:

\[
u = \delta = K_0 \oplus r_0
\]

At this point we are able, for any faulty execution, to recover the value of \( \delta = K_0 \oplus r_0 \) involved in that execution. It is then possible to reproduce this analysis many times to obtain \( k (k \leq 256) \) such relations for \( k \) different values \( \delta_i, i = 1, \ldots, k \), and store the power consumption curve \( W_i \) of the faulty execution corresponding to each \( \delta_i \).

Now, observe the following property: for any possible guess \( g \) about \( K_0 \), \( g = 0, \ldots, 255 \), we obtain a unique set \( S_g = \{r_{0,1}, \ldots, r_{0,k}\} \) of \( k \) random mask values. In the HODPA resistant implementation these random values are generated and manipulated in the card at different moments during the inversion in \( GF(2^4) \) and during the MixColumn computation applied to the mask in the first round. It is then possible to correlate these random values with the power curves \( W_i \). By computing the linear correlation factor between the set of curves \( \{W_1, \ldots, W_k\} \) and the set of random masks \( \{r_{0,1}, \ldots, r_{0,k}\} \), the most important correlation peak over the different guesses identifies the correct set of random values manipulated in the card and thus indicates that the corresponding guess \( g \) is equal to the secret key byte \( K_0 \).

The analysis can be repeated on the next bytes of the key addition operation to recover the other key bytes \( K_1 \) to \( K_{15} \).

We summarize the different steps of the attack in Figure 2. Note that in phase 2 of the attack, the expected number of faulty executions needed to obtain a new informative \( \delta_i \) grows constantly with \( i \). The expected number of faults \( N_k \) required to gather \( k \) relations is equal to

\[
N_k = \sum_{i=1}^{k} \frac{256}{256 - (i - 1)}
\]

so that an average of 126 faults generates 100 relations, while the complete set of 256 relations requires \( N_{256} = 1568 \) faults.
Phase 1: dictionary precomputation
\( M = (M_0, \ldots, M_{15}) \leftarrow (0, \ldots, 0) \)
for \( u = 0 \) to 255 do
\( C_u \leftarrow \text{AES}(M|M_n = u) \)

Phase 2: collision search
\( \Gamma = \emptyset \)
i ← 1
while \( (i < k) \) do
\( C^i = \text{AES}^i(M) \)
if \( C^i \notin \Gamma \) do
\( \delta_i \leftarrow u \text{ s.t. } C^i = C_u \text{ with } u \in \{0, \ldots, 255\} \)
\( W_i \leftarrow \text{power curve of the faulted execution} \)
\( \Gamma \leftarrow \Gamma \cup \{C^i\} \)
i ← i + 1

Phase 3: correlation
for \( g = 0 \) to 255 do
for \( i = 1 \) to \( k \) do
\( r_{n,i} \leftarrow \delta_i \oplus g \)
\( \rho_g \leftarrow \text{correlation trace between } \{r_{n,1}, \ldots, r_{n,k}\} \) and \( \{W_1, \ldots, W_k\}\)
\( K_n \leftarrow g \text{ which gives the highest correlation peak} \)

Figure 2. The attack algorithm on key byte \( K_n \)

Remark: Our attack is also applicable when the fault effect does not result in a 0 value but in an unknown constant \( c \). Instead of recovering the 16-byte key \( K \) we recover \( K \oplus (c, \ldots, c) \). Then we just have to exhaust all 256 keys until a key matching some correct plaintext/ciphertext pair is found.

5 Countermeasures

While data randomization with a full mask (i.e. 16 different bytes) is enough to protect the AES algorithm against the Amiel et al. attack as well as high order differential analysis, it is not sufficient against our combined attack. Below we mention some possible countermeasures.

5.1 Inverse computation

A classical and efficient countermeasure used to protect cryptographic algorithms against fault attacks in smartcards is the verification of the computation done. Before returning the ciphertext the card performs the inverse computation on the result. If the value obtained corresponds to the input message then the computation is considered valid and the card can return the result. However if the comparison is not successful, it means that a fault was generated during one of the two cryptographic computations. In this case, nothing is returned by the card. As our attack requires faulty ciphertexts, it is not applicable when the inverse computation countermeasure is implemented.

5.2 Duplicated Rounds

An alternative to the previous countermeasure consists in duplicating the execution of the rounds exposed to the attacks, for instance the first and the last rounds. The rounds which are duplicated must be performed with two different masks. Their executions are carried out together, and bytes are processed in a random order regardless of their masks. At the end of the round the following property must hold: the addition of the two results must be equal to the addition of the masks. If this property does not hold then a fault is detected. In order to protect against DFA, it is recommended to duplicate the first three and the last three rounds.

5.3 Data error

Another way to protect an algorithm may be the deliberate introduction of data errors appearing under some kinds of sequence flow disruptions. This notion of infective countermeasure can be applied to loop counters, round counters,... Some infectious data are supposed to be equal to zero or to a fixed value in a normal execution, but when a fault modifies a loop counter (for instance during the AddRoundKey operation in AES) these values become erroneous when they are XORed with this counter.

5.4 Checksums

Data used at the beginning of a cryptographic algorithm (message, key, mask, ...) may be associated with a checksum. After executing sensitive operations a checksum on the obtained data is computed and compared to these values stored in memory. A comparison error implies that a fault occurred during this part of the algorithm. These checksums can be carried out using hardware mechanisms.

We have implemented the first two countermeasures on the HODPA secure algorithm described in Section 3. Their performances and memory costs are presented in the Table 2.
6 Passive and Active Combined Attack on Masked AES with Safe Errors

Here we present a variant of our combined attack which is not precisely based on collisions but rather on safe errors, also known as ineffective faults.

In this variant, the way to obtain the knowledge of $K_n \oplus r_n$ for some faulty execution differs from the attack described in Section 4.2. Instead of comparing a faulty ciphertext $C^i = AES^i(M|M_n = 0)$ with a pool of normal ciphertexts $\{C_u = AES(M|M_n = u)\}$, the attacker repeatedly compares some normal ciphertext $C_u = AES(M|M_n = u)$ with a faulty one $C^i_u = AES^i(M|M_n = u)$ obtained with the same input, until both ciphertexts collide. Once this collision occurs the attacker knows that $K_n \oplus r_n = u$.

At first sight this variant may seem irrelevant since the distributions of the two random variables $C^i$ and $C^0$ are the same. Also this variant requires 256 faulty executions on average to obtain one single collision, while the previously described attack requires only one.

The great advantage of the safe errors variant becomes clear when the HODPA resistant implementation is also protected against CFA, either by means of the inverse computation countermeasure or by means of the duplicated rounds one. In both cases the attacker identifies the collision event each time a result is returned. Indeed the result is returned whenever the fault has not been detected, that is whenever it was safe and had no local effect on the XOR result. Note that it is not possible to distinguish a safe error event from a no fault at all event. Consequently the safe errors variant may be difficult to perform in practice if the fault injection tool is not highly reliable.

An interesting and unexpected property of the safe errors variant is that it is easier to perform when the computation checking countermeasures are implemented than when they are not. Indeed when either of these countermeasures is present the attack consists in a known message attack, otherwise it is a chosen message attack.

### 6.1 Countermeasures

The countermeasures presented in Section 5 no longer work here as no data has been modified. Then the question is how can we prevent attacks which do not modify data processed by the card? It seems to be an open problem and we do not have any good response. However we can mention several mechanisms which can complicate the attacker’s task.

We have not yet addressed hardware mechanisms. As stated previously by Blümmer and Seifert [5] we must insist on the fact that efficient hardware mechanisms to detect or resist light injections help software implementations and help to prevent many fault attacks.

The randomization (time or order) during execution is also a good means to destabilize the attacker as he does not exactly know where the targeted data is manipulated.

Due to the somewhat large number of fault injections required to perform the safe errors variant, an efficient means to defeat the attacker could be to limit the number of possible faulty executions. If more than a specified number of faults is detected the card can kill itself or at least refuse to answer except under privileged conditions. Note however that this principle may be difficult to implement in practice depending on the card operating environment and requirements.

The best solution will be to mask the key with a value which is never manipulated during the processing. It is then not possible to correlate power curves with the mask and the only data that an attacker can find would is the masked key.

### 7 Conclusion

Sound countermeasures are known for protecting embedded cryptographic implementations from either high order side-channel analysis or differential and collision fault analysis.

In this paper we have shown that simply putting together different kinds of countermeasures may not be sufficient. We have presented a Combined Active – CFA – and Passive – CPA – Attack (PACA) which breaks a proposed state of the art side-channel resistant AES implementation with a limited number of faults. We have not yet addressed hardware mechanisms.

As stated previously by Blümmer and Seifert [5] we must insist on the fact that efficient hardware mechanisms to detect or resist light injections help software implementations and help to prevent many fault attacks.

The randomization (time or order) during execution is also a good means to destabilize the attacker as he does not exactly know where the targeted data is manipulated.

Due to the somewhat large number of fault injections required to perform the safe errors variant, an efficient means to defeat the attacker could be to limit the number of possible faulty executions. If more than a specified number of faults is detected the card can kill itself or at least refuse to answer except under privileged conditions. Note however that this principle may be difficult to implement in practice depending on the card operating environment and requirements.

The best solution will be to mask the key with a value which is never manipulated during the processing. It is then not possible to correlate power curves with the mask and the only data that an attacker can find would is the masked key.

### Table 2. Performance and memory costs (bytes) for two FA resistant implementations

<table>
<thead>
<tr>
<th></th>
<th>Cycles</th>
<th>ROM</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESHODPA with Inverse Computation</td>
<td>102 000</td>
<td>5 500</td>
<td>75</td>
</tr>
<tr>
<td>AESHODPA with 6 Duplicated Rounds</td>
<td>81 000</td>
<td>5 900</td>
<td>91</td>
</tr>
</tbody>
</table>
highly reliable fault injection tool. Although we have given some hints about how our last attack may be rendered more difficult, it seems to be an open problem how to protect implementations from ineffective faults, which are informative even though they do not alter the computations.

Acknowledgments

The authors would like to thank Sean Commercial for his valuable comments and advice on this manuscript.

References


A Targeted AES Implementation

Figure 3. Secure HODPA Implementation