# A SURVEY ON FAULT ATTACKS

Christophe Giraud
*Oberthur Card Systems*
*25, rue Auguste Blanche*
*92 800 Puteaux, France.*
c.giraud@oberthurcs.com

Hugues Thiebeauld*
*Thales Microelectronics*
*CNES LAB. BPI 1414*
*18, Avenue Edouard Belin*
*31 401 Toulouse Cedex 9, France.*
hugues.thiebeauld@cnes.fr

**Abstract** Fault attacks described in cryptographic papers mostly apply to cryptographic algorithms, yet such attacks may have an impact on the whole system in a smart card. In this paper, we describe what can be achieved nowadays by using fault attacks in a smart card environment. After studying several ways of inducing faults, we describe attacks on the most popular cryptosystems and we discuss the problem of induced perturbations in the smart card environment. Finally we discuss how to find appropriate software countermeasures.

**Keywords:** Fault Attack, Differential Fault Attack, Side-Channel Attack, Tamper Resistance, Smart Card.

## 1. Introduction

In the smart card industry, fault attacks have been increasingly studied since the publication of Boneh, DeMillo and Lipton's paper [10] in 1996. At that time, the attack described in the Bellcore paper was merely theoretical, but since then improvements in technology have allowed attackers to put fault attacks into practice. In 2001, Skorobogatov and Anderson presented a practical attack [27] requiring very cheap equip-

* Research done while at Oberthur Card Systems.

ment: by using a photoflash lamp, they succeeded in flipping a bit in a memory cell. Despite the fact that the component was an old unprotected chip, their paper introduced a means of putting into practice theoretical fault attacks on cryptographic algorithms and they also showed that every software application dealing with security can be threatened by fault attacks. Nowadays, faults attacks are very powerful and may allow an attacker to break an unprotected system faster than any other kind of side-channel attacks, such as SPA, DPA or EMA.

In this paper, we discuss what is now possible concerning fault attacks from both a theoretical and a practical point of view. We deal with fault attacks applied to cryptographic algorithms but also with fault attacks applied to the whole system used in a smart card environment.

Firstly, we describe different ways to set up a fault attack. After studying the different kinds of induced faults and their implications during the execution of a software, we describe how to use these faults to break the most well-known cryptographic algorithms such as DES, AES, RSA, DSA and ECDSA. Finally, we discuss the security of software applications including cryptographic algorithms.

## 2. How to perform a fault attack ?

There are many ways of performing a fault attack. We briefly describe the three which are most often published in cryptographic papers. These techniques are all moderately difficult to carry out and require relatively simple equipment. Moreover, they are all non-invasive or semi-invasive, which means that they do not require any physical tampering with the chip, although the packaging may be destroyed in some cases. Finally, we discuss the importance of synchronization and localization to successfully perform such attacks.

### 2.1 Glitch attacks

Historically the first way to induce a faulty behaviour in a smart card was to induce a glitch of power on one of the contacts. Good results were obtained with a power glitch applied to Vcc (power supply), GND (ground) or to the clock. A very powerful and brief addition of power can perturb the component, and hence generates a fault. However, the additional signal has to be short enough to avoid its detection or the destruction of the card. In order to achieve a successful attack, one can only set the timing of the glitch, its amplitude and its duration.

The obvious advantage is that the attack is very easy to set up, because it is truly non invasive and the equipment needed is easily available and cheap. The main drawback is that it is impossible to perturb a specific part of the chip, since the perturbation has an impact on the whole smart card.

Moreover, nowadays smart cards resist this kind of attack : thanks to filters, DC converters or power sensors, the card can either detect or cancel an out of specifications peak of power that might induce faulty behaviour. The hardware reacts accordingly by, for example, forcing a reset.

## 2.2 Light attacks

Light attacks were introduced in 2000 by Skorobogatov and Anderson [27].

Nowadays this is the most common way to induce faults on smart cards. The idea behind this attack is to use the energy of a light emission to perturb the silicon of a chip. For this kind of attack the preparation of the chip is more difficult than for a glitch attack because it is semi-invasive : the plastic layer on the component has to be removed in order to reach the silicon with the light beam. Such a preparation is described in [1] and [5].

**Physical Effects.** Why is the component perturbed ? With a penetration depth dependant on the light wavelength, the energy carried by the light beam is absorbed by the silicon's electrons, which allows them to jump from the valence band to the conductance band. Each new electron in the conductance band creates an electron-hole pair and the set of all of these pairs forms a local photoelectric current. The logical gates or the memory cells of the component, depending on their technology, are sensitive to such a current and behave accordingly. This can therefore be exploited for an attack

**Parameters.** To succeed in generating a fault, we have to take into account the following parameters : the energy of the light, its wavelength, the duration of the emission, the location and the extent of the impact zone. Let us detail each of them :

- The energy of the light beam must be considered as an energy per surface on the chip. The amount of energy per square inch must be enough to allow the electrons to jump to the conductance band, but low enough not to destroy the chip.

- The penetration depth of the energy depends on the wavelength of the light. For CMOS technology, if we succeed in creating electron-hole pairs close to the N-channel or the P-channel, the memory cells become very sensitive ; the wavelength has to be chosen accordingly.

- The longer the light emission, the larger the electron-hole pairs that are created. If the duration of the emission can be adjusted, the duration of the local current generated can be controlled.
- One of the advantages of light attacks compared to glitch attacks is that one can choose the location of the attack on the chip. The thinner the source of light, the better the precision of the impact on the chip.

**Material.** In view of the previous parameters, the source of light has to be well chosen. The characteristics of the two most common sources for such an application are described here.

From [27], a simple camera flash installed on a microscope allows the execution of a light attack. The spectrum of a flash is composed of all visible wavelengths (white light) and a great part of infra red. The main advantage of this equipment is that it is very cheap and easy to install. Many attacks can be performed, but the attacks remain very limited. For example, with current smart cards, it is very difficult to change the value of a memory cell or to perturb the execution of the code. With a camera flash the different parameters described above (duration, power and localization of the emission) are difficult to control.

By using laser equipment one can significantly enhance the accuracy of the attack. The main characteristics of a laser are a discrete wavelength emission, a very thin beam and an emission which is easy to control in terms of duration and power. As previously explained, the depth penetration of the energy is in direct relation with the light wavelength, so the spectrum of the laser has to be chosen accordingly. The maximal power emission also has to be taken into account. The single drawback of using a laser is the price of the material, clearly much more expensive than a camera flash.

Chips have been equipped for some time with hardware countermeasures that render simple flash attacks quite inoperative. However, an attacker, with experience in the field and more enhanced equipment, can still find ways to perform light attacks.

## 2.3 Magnetic attacks

Introduced in [26], another way to induce transient faults is to emit a powerful magnetic pulse near the silicon. The magnetic field creates local currents on the surface of the component, which can generate a fault. From a practical point of view, the attack is performed with very cheap material: a wire is wound around a very sharp needle, a current flowing through the wire creates an oriented magnetic field. For better results, the attack has to be semi-invasive : the needle must be as close as possible to the silicon, so the plastic layer has to be removed.

Depending on the equipment used and its characteristics (size of the needle, number of loops around the needle) the following parameters have to be taken into account : the power of the magnetic field, its duration and localization on the chip. This technique seems difficult to set up practically, because the generation of a high magnetic field requires a high current, which would destroy the wire. Moreover, even if the needle concentrates the magnetic field, the laser still focuses the perturbation on a very small part of the chip much more accurately than the needle.

## 2.4 Synchronization issues

In order to set up a successful attack, the attacker must understand which errors have to be introduced from a theoretical point of view. He must then try to put these errors into practice. According to the theoretical analysis, induced faults can be more or less coarse (change any or several bits versus changing one given bit), which explains why the synchronization has to be more or less precise. For example, a DFA attack on RSA only needs a faulty computation on one of the CRT components; as the computations are quite long, a precise synchronization is not necessary. On the other hand, a DFA attack on a hardware DES (cf. section 4.1) requires the uttermost precision in synchronization due to the very fast execution of the computation we want to perturb. As the attack needs to be accurate in time, a good synchronization is critical.

There are many ways of synchronizing an attack. It is possible to use side channel information, such as monitoring the power consumption or the magnetic radiations resulting from the activity of the chip. Events such as writing in EEPROM can thus be recognized allowing us to determine when the attack must be performed. Signal processing can enhance the quality of the signal and its interpretation. All these techniques can be combined to obtain complementary information to help target the attack more accurately.

## 2.5 Localization issues

As we have already pointed out, a glitch attack perturbs every part of the chip. On the other hand, light or magnetic attacks allow a more accurate localization. It is easy to understand that to perturb only a specific part of the chip, for example a DES module, a light or magnetic emission has to be focused on this specific part. Sometimes, in order to induce desired faults, the targeted area has to be accurately circumscribed and exposed. An optical circuit is thus necessary in order to perform a light attack with the required precision.

## 3. Kind of faults and consequences

As soon as the attacker is able to perform a perturbation on the chip, many errors are observed. Two kinds of faults, permanent or transient, can be generated.

**Permanent faults.** Permanent faults occur when the value of a cell is definitively changed. It can concern either data (contained in EEPROM or in RAM) or code (contained in EEPROM). Modifying data definitively could be very powerful, particularly when the data is related to sensitive objects of the smart card, like a PIN or a key. Nevertheless, perturbing many successive logical cells or choosing the value of the perturbed cell is very difficult when the data is ciphered or the memory is scrambled.

**Transient faults.** Transient faults are the most common ; they occur when a code execution or a computation is perturbed. An error in a code execution can be caused when the logical parts, including CPU and registers (data and control) are affected, or when reading code or data is perturbed. On the other hand, a computation error occurs when CPU or peripherals (such as cryptographic modules) are affected.

**Consequences on the execution.** A single modified byte can severely perturb the execution flow of a code. At the lowest level of the micro-controller, each operation is encoded over several bytes : some for the instruction and others for the parameters. So assuming the attacker modifies one byte of code during its execution, the fault thus generated can affect either an instruction or a parameter.

The following cases can then occur:

- An instruction byte is modified : a different operation is executed. First consequence, the expected instruction is not executed : a call to a subroutine can be skipped, a test can be avoided, etc. Second consequence, one or several different instructions are executed, which can result in strange behaviour.
- A parameter byte is modified : a different address or value is considered. In the former case the target address is not modified but another is, or a wrong value is fetched from the memory, or

the program counter is indirectly modified in a case of a JUMP instruction. In the latter case the operation is performed with a different operand, with the consequence that the result is wrong.

Given dedicated tools, techniques and some knowledge of the topology of the chip, most parts of a micro-controller can be attacked to cause transient faults.

## 4. Algorithms attacks

Fault attacks on cryptographic algorithms have been studied since 1996 and since then, nearly all the cryptographic algorithms have been broken by using such attacks. In this section we describe fault attacks on the best-known cryptosystems. We begin with symmetric algorithms such as DES and AES and then we focus on asymmetric algorithms such as RSA and DSA.

### 4.1 Symmetric algorithms

In this section we focus on the two best-known block-ciphers: the DES and the AES. Firstly we describe DFA attacks on these two block-ciphers and then we describe an efficient countermeasure against these attacks.

**DES.** The main paper about DFA on DES was written in 1996 by Biham and Shamir [7]. In this paper, they explain how to obtain the secret key by using between 50 and 200 faulty ciphertexts. Let us describe how it works.

We suppose that a single-bit fault is induced on the right part of the temporary result at the beginning of the last round. We denote respectively by $C$ and $C*$ the correct and the faulty ciphertexts of a message $M$. We also denote respectively by $T_{0-31}$ and $T_{32-63}$ the left and the right parts of a 64-bit value $T$.

If we compare the left parts of the correct and the faulty temporary results at the end of the 16th round (i.e. $IP(C)_{32-63}$ and $IP(C^*)_{32-63}$), there is only one bit which differs. This bit corresponds to the induced fault so we obtain the position of the faulty bit and we deduce which S-boxes have been affected. By computing $EP(IP(C)_{32-63}) \oplus EP(IP(C^*)_{32-63})$ where $EP$ is the DES expansive permutation, we obtain thedifference $\Delta_{inputs}$ between the correct and faulty inputs of the S-Box in the last round.

Then we look at thedifference $\Delta_{outputs}$ between the correct and faulty outputs of the S-Box in the last round, which is equal to $P^{-1}(IP(C)_{0-31})$

$\oplus P^{-1}(IP(C^*)_{0-31})$ where $P$ is the permutation used at the end of each round.

$\Delta_{inputs}$ (respectively $\Delta_{outputs}$) contains non-zero bits only in input bits (respectively outputs bits) of the one (or two) S-box(es) which has been affected by the fault.

In the following part, we suppose that the faulty bit affected only one S-box, let us say $Sbox_k$. In order to find the 6-bit input of this S-box, we sort the possible values out by using the following procedure:

1 List all the 6-byte couples $(B_i,\ B_j)$ such as $B_i \oplus B_j = \Delta_{inputs}$

2 For each couple in this list, if $Sbox_k(B_i) \oplus Sbox_k(B_j) \neq \Delta_{outputs}$ then eliminate the couple $(B_i,\ B_j)$.

By using several faulty ciphertexts, we recover the 6-bit input of the $k^{th}$ S-box.

By iterating this attack we know the whole 48-bit input $Input\text{-}SB_{16}$ of the S-Box in the last round. Then we can compute the $16^{th}$ subkey which is equal to $EP(IP(C)_{32-63}) \oplus Input\text{-}SB_{16}$. From this subkey we find the whole DES key by a very fast exhaustive search on the last unknown 8 bits.

We can extend this attack with a fault induced at the beginning of the $11^{th}$, $12^{th}$, $13^{th}$, $14^{th}$ or $15^{th}$ round. But in these cases we use a counting method: instead of eliminating the couple $(B_i, B_j)$ which does not verify the relation $Sbox_k(B_i) \oplus Sbox_k(B_j) = \Delta_{outputs}$, we increase the counter by one of any pair which verifies the previous relation, the right value is expected to be counted more frequently than any wrong value.

This attack can also be extended if we induce a fault on a whole (or even several) byte(s) or if we disturb the key scheduling.

Such an attack has been successfully achieved on an unprotected software DES. By using a camera flash, as described in section 2.2, we succeeded in recovering the secret key by using only 2 ciphertexts. This could be achieved thanks to the fact that in practice we disturb one or several bytes compared to the single-bit fault model used by Biham and Shamir. Thus one faulty ciphertext provides information on several bytes of the $16^{th}$ subkey.

**AES.** On the $2^{nd}$ October 2000, the AES was chosen to be the successor of the DES. Since then, many papers have been published about DFA attacks on the AES [9, 12, 16, 17, 25]. Here we focus on the most powerful attack which was published recently [25].

For the sake of simplicity, we suppose the attack is done on an AES-128. This attack is based on the following observation: the MixColumns function operates on its input 4 bytes by 4 bytes, so if we induce a fault on one byte of one of this 4-byte block, the number of possible differences at the input of the MixColumns transformation is 255*4. Due to the linearity of the MixColumns function, we have 255*4 possible differences at its output.

If we suppose that a fault on one byte has been induced on the input of the last MixColumns and if we denote by $M$ the plaintext, $C$ and $C^*$ the corresponding correct and faulty ciphertexts, the attack can be mounted as described hereafter:

1 Compute the 255*4 possible differences at the output of the Mix-Columns function and store them in a list $\mathcal{D}$

2 $C$ and $C^*$ differ only on 4 bytes, let us say bytes 0, 13, 10 and 7 (it corresponds to a fault on one of the first four bytes of the input of the last MixColumns). Take a guess on the 4 bytes in the same position of the last round key (i.e. $K^{10}_{0,13,10,7}$).

3 Compute $invSB(invSR(C_{0,13,10,7} \oplus K^{10}_{0,13,10,7})) \oplus invSB(invSR(C^*_{0,13,10,7} \oplus K^{10}_{0,13,10,7}))$ and check if this value is in $\mathcal{D}$. If so, add the round key to the list $\mathcal{L}$ of possible candidates.

4 Go back to step 2 by using another correct/faulty ciphertexts pair (D, D*) (which could be obtained from another plaintext) which differs on the same bytes as $C$ and $C^*$, and by choosing $K^{10}_{0,13,10,7}$ from the list $\mathcal{L}$. Repeat until there remains only one candidate in $\mathcal{L}$.

After this attack, 4 bytes of the last round key are known and we re-iterate three times this attack with pairs *(C, C*)* which differ respectively on bytes (1, 14, 11, 4), (2, 15, 8, 5) and (3, 12, 9, 6).

This attack implies a guess on 4 bytes which is not very practical. In [25], an ingenious implementation of this attack is described by guessing only 2 bytes of the last round key at each iteration.

Moreover Piret and Quisquater remark that if we induce a fault on one byte between the two calls to the MixColumns function of the 7th and the 8th round, we obtain a fault on one byte for each 4-byte block of the input of the MixColumns function of the 9th round. So we obtain information on 16 bytes of the last round key instead of information on only 4 bytes. By using this attack the key could be retrieved by using only 2 faulty ciphertexts.

**Countermeasures.** If we suppose that an attacker cannot induce the same fault twice, one of the best countermeasures to protect the DES and the AES is to compute the last rounds twice (including the key scheduling). For the DES, doubling the last 8 rounds is efficient and for the AES, doubling the last 4 rounds prevents nearly all known DFA attacks[1]. Of course this kind of countermeasure must be adapted for each symmetric cryptosystem, but doubling the whole or a part of symmetric algorithms is generally an effective countermeasure against fault attacks.

## 4.2 Public Key Algorithms

In this section we firstly describe fault attacks applied to the RSA, before looking at fault attacks on some signature schemes such as DSA and EC-DSA. Finally, we present fault attacks on the scalar multiplication used in Elliptic Curve Cryptography (ECC).

**RSA.** The first published fault attack was applied to the RSA [10] and was improved shortly after by Lenstra [22]. This attack on the RSA-CRT is very simple and efficient in practice.

Firstly, we describe how to sign a message by using the RSA-CRT and how to find the secret key by using the DFA attack described in [10]. We then describe how Lenstra improved this attack.

Let $N = pq$ be a product of two large prime integers, $e$ chosen such as $gcd(e, (p-1)(q-1)) = 1$ and $d = e^{-1} \bmod (p-1)(q-1)$, $(d, p, k)$ is the secret key and $(e, N)$ is the public key.

To sign a message $m < N$, the signer computes $s = m^d \bmod N$. To improve the time calculation, this signature is performed by using the Chinese Reminder Theorem:

1 Compute $d_p = d \bmod p - 1$ and $d_q = d \bmod q - 1$

2 Compute $s_p = m^{d_p} \bmod p$ and $s_q = m^{d_q} \bmod q$

3 Compute $s = ((((s_q - s_p) \bmod q) * (p^{-1} \bmod q)) * p + s_p) \bmod N$

We can find two integers $a$ and $b$ such as $s = as_p + bs_q$ where:

$$\begin{cases} a & \equiv & 1 \bmod p \\ a & \equiv & 0 \bmod q \end{cases} \text{and} \begin{cases} b & \equiv & 0 \bmod p \\ b & \equiv & 1 \bmod q \end{cases} \qquad (1)$$

because $s_p = s \bmod p$ and $s_q = s \bmod q$.

If the signer signs the same message twice, a fault attack can then be performed: during the second signature, a fault is induced during the computation of $s_p$ (this attack works in the same way if a fault is induced during the computation of $s_q$). So we obtain a faulty signature $\tilde{s}$:

$$\begin{cases} s &= as_p + bs_q \bmod N \\ \tilde{s} &= a\tilde{s}_p + bs_q \bmod N \end{cases} \tag{2}$$

We remark that:

$$s - \tilde{s} \equiv a(s_p - \tilde{s}_p) \bmod N \tag{3}$$

Moreover $a \equiv 0 \bmod q$, so if $p$ does not divide $s_p - \tilde{s}_p$ then:

$$\gcd(s - \tilde{s},\ N) = \gcd(a(s_p - \tilde{s}_p),\ N) = q \tag{4}$$

Then we can easily find the other part $p$ of the secret key by dividing the modulus $N$ with $q$.

An improvement on this attack was found by Lenstra [22]: by using the same fault as above (i.e. fault induced during the computation of $s_p$), we have $s \equiv \tilde{s} \bmod q$ and $s \not\equiv \tilde{s} \bmod p$ so:

$$gcd(m - \tilde{s}^e, N) = q \tag{5}$$

where $e$ is the public exponent used to verify the signature. By using this attack, we only need one faulty signature of a known message to find the secret key.

An efficient countermeasure against this attack is to verify the signature by using the public key. As the public key is often very short ($e = 2^{16} + 1$ for example), the verification by using the RSA-SFM is very fast. A faster countermeasure was described by Blömer *et al.* in [8]: instead of verifying the signature, they rewrote the CRT recombination in such a way that if a fault is induced on a CRT component ($s_p$ or $s_q$), the error also affects the other CRT component (respectively $s_q$ or $s_p$).

**DSA.** Let us briefly describe the signature of the DSA: firstly the signer chooses a 160-bit prime number $q$ and a 1024-bit prime $p$ such as $q$ divides $p - 1$. Then he chooses a positive integer $g$ less than $p - 1$ of order $q$ modulo $p$. Finally he chooses a positive integer $a$ less than $q - 1$ and computes $A = g^a \bmod p$. His public key is $(p,\ q,\ g,\ A)$ and his secret key is $a$.
To sign a message $m$, the signer chooses a non-null random $k$ less than $q - 1$ and computes

$$\begin{cases} r &= (g^k \bmod p) \bmod q \\ s &= k^{-1}(h + a.r) \bmod q \end{cases} \tag{6}$$

where $h$ is the hash of $m$ obtained by using the SHA-1 algorithm.
The signature of the message $m$ is the couple $(r,\ s)$.

The only existing DFA attack on the DSA was published in [4]. Thanks to this attack we can find a bit of the secret key each time we succeed in flipping a bit of the secret key during the computation of the second part of the signature.

Let us describe this attack: if an attacker succeeds in inducing a fault on only one bit of the secret key $a$, he obtains a faulty signature $(r,\ \tilde{s})$ where $\tilde{s} = k^{-1}(h + \tilde{a}.r) \bmod q$. The attacker can compute

$$\begin{aligned} T &= g^{\tilde{u}.h \bmod q}.A^{\tilde{u}.r \bmod q} \bmod p \bmod q \\ &= g^{\tilde{u}(h+a.r) \bmod q} \bmod p \bmod q \end{aligned} \tag{7}$$

where $\tilde{u} = \tilde{s}^{-1} \bmod q$. Let

$$R_i = g^{\tilde{u}.r.2^i \bmod q} \bmod p \bmod q \text{ for } i = 0,\ 1, \ldots,\ 159 \tag{8}$$

then he obtains from (7) and (8):

$$TR_i = g^{\tilde{u}(h+r(a+2^i)) \bmod q} \bmod p \bmod q \tag{9}$$

and

$$T/R_i = g^{\tilde{u}(h+r(a-2^i)) \bmod q} \bmod p \bmod q \tag{10}$$

So he obtains

$$\begin{cases} TR_i &= r \bmod p \bmod q \text{ if the } i^{\text{th}} \text{ bit of } a \text{ is } 0, \\ T/R_i &= r \bmod p \bmod q \text{ if the } i^{\text{th}} \text{ bit of } a \text{ is } 1. \end{cases} \tag{11}$$

By iterating $i$ from 0 to 159 and by comparing $r$ with $TR_i$ and $T/R_i$, the attacker can discover the value of a bit of the secret key. The complete secret key can be recovered by performing 161 signatures of the same (unknown) message: one correct and 160 faulty.

**ECDSA.** With the same approach described previously, we can attack the ECDSA. This attack was firstly described in [14].

We denote by $(q,\ a,\ b,\ r,\ G,\ W)$ the signer's public key and by $s$ his private key where $q$ is a 160-bit prime, $a$ and $b \in GF(q)$ the coefficients defining the elliptic curve, $G$ a curve point generating the subgroup of order $r$ and $W = sG$.

To sign a message $m$ the signer generates a random $u$ and computes the point $V = uG = (x_V,\ y_V)$. He then converts $x_V$ into an integer $k$. Finally he computes the two parts of the signature:

$$\begin{cases} c &= k \bmod r, \\ d &= u^{-1}(f + s.c) \bmod r. \end{cases} \tag{12}$$

where $f = SHA - 1(m)$.

If an attacker succeeds in inducing a fault on only one bit of the secret key $s$ during the computation of $d$, he obtains a faulty signature $(c, \tilde{d})$ where $\tilde{d} = u^{-1}(f + \tilde{s}c) \bmod r$. The attacker can compute

$$\begin{cases} h_{1,i,+} & = \tilde{d}^{-1}(f + 2^i c) \bmod r \\ h_2 & = c\tilde{d}^{-1} \bmod r. \end{cases} \tag{13}$$

He then computes the point

$$P_{i,+} = h_{1,i,+}G + h_2 W = \left(u\frac{f + (2^i + s)c}{f + \tilde{s}c}\right)G = (x_{P_{i,+}}, y_{P_{i,+}}) \tag{14}$$

He then converts $x_{P_{i,+}}$ into an integer $k_{P_{i,+}}$.
Now, we have two cases:

1 If the fault has flipped the $i^{\text{th}}$ bit of $s$ from 1 to 0, then $\tilde{s} = 2^i + s$ so from (14) $P_{i,+} = uG$. By iterating the value of $i$ from 0 to 159 and by testing if $k_{P_{i,+}} \bmod r = c$, the attacker can discover the value of a bit of the secret key.

2 But if the fault has flipped the $i^{\text{th}}$ bit of $s$ from 0 to 1 then $\tilde{s} = s - 2^i$. The attacker can also discover the value of this bit by applying the same method as described above with $P_{i,-}$ instead of $P_{i,+}$ where $P_{i,-} = h_{1,i,-}G + h_2 W$ and $h_{1,i,-} = \tilde{d}^{-1}(f - 2^i c) \bmod r$.

So by using several faulty signatures (at least 160), an attacker can recover the whole value of the secret key $a$.

**Remark.** The fault attacks on the DSA and the ECDSA described in this paper use the fact that the value of the secret key is disturbed during the computation of the second part of the signature. To protect such signature schemes against this kind of attack, we can implement one of the following countermeasures:

- we can check the integrity of the secret key at the end of the signature computation, for example by adding a CRC to the secret key: we check if the CRC of the secret value used during the signature computation is the same as the CRC of the secret key stored in non-volatile memory,
- we can also verify the signature by using the public key before sending the signature out. But this countermeasure is very costly in terms of execution time because the verification takes nearly twice as long as the signature computation.

**Summary table.**

| | Fault model | Fault location | Minimum number of required faulty results |
|---|---|---|---|
| DES | Byte *(could be more)* | Anywhere among the last 6 rounds | 2 |
| AES | Byte | Anywhere between the MixColumns of the 7th and 8th round | 2 |
| RSA-CRT | Size of the modulus | Anywhere during the computation of one of the CRT components | 1 |
| DSA | Bit | Anywhere among 20 bytes | 160 |
| ECDSA | Bit | Anywhere among 20 bytes | 160 |

**ECC.** Only two papers dealing with elliptic curve scalar multiplication in the presence of faults have been published. The first fault attacks on scalar multiplication were published at Crypto 2000 by Biehl, Meyer and Müller [6]. In 2003, Ciet and Joye relaxed assumptions on these attacks [13].

In the sequel, we suppose that a scalar multiplication is performed with a scalar $d$ and a point *P* which lays on the elliptic curve $E$: $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ where the $a_i$'s belong to a finite field $\mathcal{K}$.

Let us describe the fault attacks of Biehl *et al.* [6]: by observing that the elliptic curve parameter $a_6$ is not used in the addition and the doubling formulas on elliptic curves, they remark that addition and doubling operations with a random point $P'$ lying on a curve $E'$: $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6'$ are the same as addition and doubling operations with a point *P* lying on the elliptic curve *E*. By supposing the Elliptic Curve Discrete Logarithm Problem (ECDLP) could be solved on $E'$, if the smart card executes the scalar multiplication with $P'$ and sends out the result $d.P'$, we could then obtain the secret scalar $d$.

As a result of this observation, they developed two attacks. For the first one, they induce a bit-fault on one of the coordinates of the point $P \in E$. They obtain a point $P'$ which lies on an elliptic curve $E'$ of the form $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6'$. After the scalar multiplication, they obtain a point $d.P'$ which lies on the same curve as $P'$. From $dP'$ they can compute $a_6'$ and so define the elliptic curve $E'$. By trying all the different candidates $P'$ which only differ from the known point *P* by one bit, they check if this candidate is a point on $E'$ and if so, they try to solve the ECDLP on $E'$.

For the second attack, a fault on one bit is induced on a temporary point $Q^{(i)}$ during the scalar multiplication. We make a supposition on which $Q^{(i)}$ the fault is induced and we guess the latest $n-i$ bits of the scalar. By using these $n-i$ bits, we compute $\tilde{Q}^{(i)}$ from $\tilde{Q}^{(n)}$. By flipping one bit of $\tilde{Q}^{(i)}$, we obtain $\hat{Q}^{(i)}$ and we then compute $\hat{Q}^{(n)}$. If $\hat{Q}^{(n)} = Q^{(n)}$, the right value for the latest $n-i$ bits of the secret scalar has been guessed. Otherwise they continue the attack by changing another bit of $\tilde{Q}^{(i)}$. If the attack is not successful after trying each possibility they make another supposition about which $Q^{(i)}$ was perturbed.

By exploiting random faults induced in either coordinates of $P$, in the elliptic curve parameters or in the field representation, Ciet and Joye showed in [13] that it is possible to recover a part of the secret scalar.

All of these attacks can be avoided by checking that $dP = (x, y)$ satisfies the relation $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$.

## 5. About security

### 5.1 The importance of securing hardware and software

As attacks are developed and improved, hardware evolves and gets more secure. However, protecting all the silicon surface against each and every type of attack is a difficult and very costly process. On the other hand, securing an unprotected micro-controller by only adding software countermeasures is also extremely costly in terms of execution time and memory space.

It appears that following a careful analysis of the attacks described above, a combination of hardware and software countermeasures yields a very good security/cost ratio.

Moreover, it is extremely important to consider the security of an embedded platform as a whole, no level being excluded from the analysis, whether it is the cryptographic functionalities, the low level operating system functions or the virtual machine and the high level applications.

For example, having a very secure PIN check is pointless if it is easy to bypass the high level call to this function and continue as if it had been satisfactorily executed.

### 5.2 How can we determine appropriate countermeasures?

Firstly one has to fix what an attacker is able to do : for example is he able to induce the same fault twice in the same code execution ? Is he able to choose a modified value ? ...

All these assumptions allow a developer to have a framework, which helps him to choose the most efficient countermeasures.

Next one must decide what has to be secured : which objects and what kind of processing are sensitive. From there on, one can go up in the software architecture and decide up to which layers the security level has to be enforced. Finally one must add the security features which are needed : redundancy on the objects, ensuring that operations have been well executed (for example by doubling them), securing the cryptographic algorithms, ...

One very helpful way to make it difficult to perform a fault attack efficiently is to desynchronize as much as possible the execution of the sensitive parts, by using for example random waits, dummy instructions, jitter on clocks, etc.

As it appears clearly in this paper, making an embedded platform secure demands a thorough understanding of the attacks, which comes only through practical experiments, through a good knowledge of making codes secure and through fully exploiting all the hardware features.

## 6. Conclusion

Although the equipment to set up fault attacks appears to be quite common, we have seen that putting such attacks into practice requires technical experience. Moreover, it is increasingly difficult to perturb the latest micro-controllers, which are designed to resist fault attacks. Nonetheless, the danger exists and the risk has to be seriously considered. As we have seen, protecting algorithms and software implemented on smart cards or any tamper-proof device is best achieved by combining both software and hardware countermeasures. Efficient countermeasures are well-known, but they have to be implemented with care and sparingly in view of the cost of security in terms of time and memory space, especially in the restrictive smart card environment.

## Notes

1. Blömer and Seifert presented in [9] a bit-fault attack on the AES with a very restrictive fault model: they suppose it is possible to set to zero one chosen bit. With such an attack, doubling the last 4 rounds is pointless, but in any case their fault model is too restrictive to be put into practice. Indeed, a lot of improvements in perturbation techniques would have to made for their attack to be effective from a practical point of view.

## References

[1] R. Anderson and M. Kuhn. Tamper Resistance - a Cautionary Note. In *Proceedings of the* 2nd *USENIX Workshop on Electronic Commerce,* pages 1–11, 1996.

[2] R. Anderson and M. Kuhn. Low cost attacks on tamper resistant devices. In B. Christianson, B. Crispo, T. Mark, A. Lomas, and M. Roe, editors, $5^{th}$ *Security Protocols Workshop,* volume 1361 of *LNCS,* pages 125–136. Springer, 1997.

[3] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In B. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002,* volume 2523 of *LNCS,* pages 260–275. Springer, 2002.

[4] F. Bao, R. Deng, Y. Han, A. Jeng, A. D. Narasimhalu, and T.-H. Ngair. Breaking Public Key Cryptosystems an Tamper Resistance Devices in the Presence of Transient Fault. In $5^{th}$ *Security Protocols Workshop,* volume 1361 of *LNCS,* pages 115–124. Springer-Verlag, 1997.

[5] F. Beck. *Integrated Circuit Failure Analysis – A Guide to Preparation Techniques.* Wiley, 1998.

[6] I. Biehl, B. Meyer, and V. Müller. Differential Fault Analysis on Elliptic Curve Cryptosystems. In M. Bellare, editor, *Advances in Cryptology – CRYPTO 2000,* volume 1880 of *LNCS,* pages 131–146. Springer-Verlag, 2000.

[7] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystem. In B.S. Kalisky Jr., editor, *Advances in Cryptology – CRYPTO '97,* volume 1294 of *LNCS,* pages 513–525. Springer-Verlag, 1997.

[8] J. Blömer, M. Otto, and J.-P. Seifert. A New RSA-CRT Algorithm Secure Against Bellcore Attacks. In *ACM-CCS'03.* ACM Press, 2003.

[9] J. Blömer and J.-P. Seifert. Fault based cryptanalysis of the Advanced Encryption Standard. In R.N. Wright, editor, *Financial Cryptography – FC 2003,* volume 2742 of *LNCS.* Springer-Verlag, 2003.

[10] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT '97,* volume 1233 of *LNCS,* pages 37–51. Springer-Verlag, 1997.

[11] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the Importance of Eliminating Errors in Cryptographic Computations. *Journal of Cryptology,* 14(2):101–119, 2001. An earlier version was published at EUROCRYPT'97 [10].

[12] C.-N. Chen and S.-M. Yen. Differential Fault Analysis on AES Key Schedule and Some Countermeasures. In R. Safavi-Naini and J. Seberry, editors, *Information Security and Privacy - 8th Australasian Conference – ACISP 2003,* volume 2727 of *LNCS,* pages 118–129. Springer-Verlag, 2003.

[13] M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. In *Designs, Codes and Cryptography,* 2004. To appear.

[14] E. Dottax. Fault Attacks on NESSIE Signature and Identification Schemes. Technical report, NESSIE, Available from `https://www.cosic.esat.kuleuven.ac.be/nessie/reports/phase2/SideChan\_1.pdf`, October 2002.

[15] E. Dottax. Fault and chosen modulus attacks on some NESSIE asymetrique Primitives. Technical report, NESSIE, Available from `https://www.cosic.esat.kuleuven.ac.be/nessie/reports/phase2/ChosenModAtt2.pdf`, February 2003.

[16] P. Dusart, G. Letourneux, and O. Vivolo. Differential Fault Analysis on A.E.S. Cryptology ePrint Archive, Report 2003/010, 2003. `http://eprint.iacr.org/`.

[17] C. Giraud. DFA on AES. Cryptology ePtint Archive, Report 2003/008, 2003. http://eprint.iacr.org/.

[18] M. Joye, A.K. Lenstra, and J.-J. Quisquater. Chinese Remaindering Based Cryptosystems in the Presence of Faults. *Journal of Cryptology,* 12(4):241–246, 1999.

[19] M. Joye, J.-J. Quisquater, F. Bao, and R.H. Deng. RSA-type Signatures in the Presence of Transient Faults. In M. Darnell, editor, *Cryptography and Coding,* volume 1355 of *LNCS,* pages 155–160. Springer-Verlag, 1997.

[20] M. Joye, J.-J. Quisquater, S.-M. Yen, and M. Yung. Observability Analysis - Detecting When Improved Cryptosystems Fail -. In B. Preneel, editor, *Topics in Cryptology – CT-RSA 2002,* volume 2271 of *LNCS,* pages 17–29. Springer-Verlag, 2002.

[21] V. Klíma and T. Rosa. Further Results and Considerations on Side Channel Attacks on RSA. In B. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002,* volume 2523 of *LNCS,* pages 244–259. Springer-Verlag, 2002.

[22] A.K. Lenstra. Memo on RSA Signature Generation in the Presence of Faults. Manuscript, 1996. Available from the author at arjen.lenstra@citicorp.com.

[23] F. Koeune M. Joye and J.-J. Quisquater. Further results on Chinese remaindering. Technical Report CG-1997/1, UCL, 1997. Available from http://www.dice.ucl.ac.be/crypto/techreports.html.

[24] D.P. Maher. Fault Induction Attacks, Tamper Resistance, and Hostile Reverse Engineering in Perspective. In R. Hirschfeld, editor, *Financial Cryptography – FC '97,* volume 1318 of *LNCS,* pages 109–121. Springer-Verlag, 1997.

[25] G. Piret and J.-J. Quisquater. A Differential Fault Attack Technique Against SPN Structures, with Application to the AES and KHAZAD. In C.D. Walter, Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003,* volume 2779 of *LNCS,* pages 77–88. Springer-Verlag, 2003.

[26] D. Samyde, S. Skorobogatov, R. Anderson, and J.-J. Quisquater. On a New Way to Read Data from Memory. In *First International IEEE Security in Storage Workshop,* pages 65–69. IEEE Computer Society, 2002.

[27] S. Skorobogatov and R. Anderson. Optical Fault Induction Attack. In B. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002,* volume 2523 of *LNCS,* pages 2–12. Springer, 2002.

[28] S.-M. Yen and J.Z. Chen. A DFA on Rijndael. In A.H. Chan and V. Gligor, editors, *Information Security – ISC 2002,* volume 2433 of *LNCS.* Springer, 2002.

[29] S.-M. Yen and M. Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. on Computers,* 49(9):967–970, 2000.

[30] S.-M. Yen, S.-J. Kim, S.-G. Lim, and S.-J. Moon. A Countermeasure against one Physical Cryptanalysis May Benefit Another Attack. In K. Kim, editor, *Information Security and Cryptology – ICISC 2001,* volume 2288 of *LNCS,* pages 414–427. Springer-Verlag, 2001.

[31] S.-M. Yen, S.J. Moon, and J.-C. Ha. Permanent Fault Attack on RSA with CRT. In R. Safavi-Naini and J. Seberry, editors, *Information Security and Privacy - 8th Australasian Conference – ACISP 2003,* volume 2727 of *LNCS,* pages 285–296. Springer-Verlag, 2003.