Static Fault Attack on Hardware DES Registers

Philippe Loubet-Moundi, Francis Olivier, and David Vigilant

Gemalto, Security Labs, France
{philippe.loubet-moundi,david.vigilant,francis.olivier}@gemalto.com
http://www.gemalto.com

Abstract. *In the late nineties, Eli Biham and Adi Shamir published the first paper on Differential Fault Analysis on symmetric key algorithms. More specifically they introduced a fault model where a key bit located in non-volatile memory is forced to* 0/1 *with a fault injection. In their scenario the fault was permanent, and could lead the attacker to full key recovery with low complexity.*
In this paper, another fault model is considered: forcing a key bit to 0/1 *in the register of a hardware block implementing Data Encryption Standard. Due to the specific location of the fault, the key modification is not permanent in the life of the embedded device, and this leads to apply a powerful safe-error like attack. This paper reports a practical validation of the fault model on two actual circuits, and discusses limitations and efficient countermeasures against this threat.*

Keywords: Hardware DES, fault attacks, safe-error, register attacks

1 Introduction

Fault attacks against embedded systems exploit the disturbed execution of a given program to infer sensitive data, or to execute unauthorized parts of this program. Usually a distinction is made between *permanent faults* and *transient faults*. A *permanent fault* alters the behavior of the device from the moment it is injected, and even if the device is powered off [8]. On the other hand, a *transient fault* has an effect limited in time, only a few cycles of the process are affected. This paper deals with *static faults* which lie in between. A *static fault* modifies a value loaded in a volatile storage, until the next power off or sooner if the effect is erased and repaired by the device itself, or if the value is electrically re-programmed.

In 1996, Boneh et al. published the first paper on Differential Fault Analysis (DFA) against cryptographic implementations [5]. Targeting the RSA-CRT, they demonstrated that exploiting a faulted cryptographic result could lead to a recovery of the private key. This sort of attack has been widely applied to other cryptographic algorithms during the last two decades. Naturally, a block cipher such as the famous Data Encryption Standard (DES), standardized in 1977 [13], quickly became a privileged target for DFA.

Indeed a few weeks later, Eli Biham and Adi Shamir published a very complete and interesting paper [3] presenting a large coverage of DFA applied to DES. More particularly, they imagined a strong fault model applicable to any secret key algorithm. Forcing key bits in non-volatile memory (NVM) to 0 one by one by fault injections, they were able to recover the key. Their fault model has been identified as very realistic, because of the NVM memory structure.

In this paper, a fault model slightly different is considered and validated by the practice on two hardware DES engines. This fault model allows mounting a "Safe-Error"(SE)-like attack. The "Safe-Error" naming has been introduced by Sung-Ming Yen and Marc Joye and then mainly applied to public key algorithms (RSA, DH, ECC). This attack technique was already used but it has been formally identified and well described in their paper [20]. Indeed their attack mainly targets the "square-and-multiply always" algorithm [6]. By disturbing the i^{th} multiplication in the exponentiation, the attacker is able to recover the i^{th} bit of the secret exponent. If the result is good, it means that the multiplication is fake (bit = 0) otherwise it is a true one (bit = 1). Each exponent bit can be disclosed by repeating this process. Even if there is a specific reaction from the device when the faulty result is detected, the attacker has the information. They pointed out in their paper that this approach can also be applied to symmetric key algorithms. Indeed, whatever the algorithm, if an attacker is able to guess a bit value from an error-free and a faulty result, or even a fault reaction, this attack remains very powerful. However they admitted that the theoretical work on the extension and exact cryptanalytic process for specific systems are still under construction. Only a few concrete scenarios of SE attacks on symmetric key algorithm have already been published so far [4].

Considering a new fault model, this paper shows another powerful SE attack scenario on a TDES implementation, even with actual countermeasures.

Organization of the paper: This paper first briefly reminds the context of hardware DES, related fault attacks and existing countermeasures. In the next part, after having defined the fault model and the static fault attack on key registers, experimental results validating our fault model are reported. These experiments are described from the set-up phase, through the fault injection and the results observed. Finally in the last section, presenting the required success conditions in an objective manner, the limitations of the attack in the real world, as well as efficient countermeasures are discussed.

2 Hardware DES, DFA and Existing Countermeasures

2.1 Hardware DES

After its standardization in 1977, DES has been widely deployed. It has been revoked and replaced by the Advanced Encryption Standard (AES) in 2001 [15].

However, under its Triple DES (TDES) version using 3 56-bit keys [14], the Data Encryption Standard is still approved by NIST until 2030 [16]. As the main block cipher algorithm in the late nineties, it has been the objects of many hardware implementations. Today, most smart cards host a hardware DES engine by default because it is intensively used in many cryptographic applications. Today's hardware DES engines usually implement the whole algorithm and an interface between software and hardware which consists of:

- A key register dedicated to the key value. This register is write-only
- A data register dedicated to the input data block. This register is write-only
- A configuration register dedicated to configure the performed calculation (encryption, decryption, 3DES key length, DES start engine, DES completion notification)

Implementing TDES using a dedicated hardware engine is user-friendly, and it significantly improves timing performances.

2.2 DFA on DES and Existing Countermeasures

Many vulnerabilities against fault attacks have been identified in DES, as for example the DFA in the 14^{th} round [3], or in other rounds [17,11]. As a consequence, protections have also been developed to counter DFA. Several of the most popular ones (software or hardware) are described below:

DES hiding: One may hide the DES signal on the power trace. If the attacker faces difficulty to recognize with side-channels means where the DES is performed, it should be more difficult to inject a fault.

Random timing jitter: One may add random timing jitter during DES phases to make synchronization (hence fault injection) more complex.

Rounds redundancy: One also may perform several times some of or all the rounds, and compare the intermediate results. This may impact performances.

Reverse computation: The ciphertext block may be decrypted just after an encryption, and the result compared with the original plain text.

Code tracing: Automatic increment of flag bits in the code can be useful to trace the DES execution flow and guarantee that all critical steps of the code have been executed.

Logic gates modification: Multiple rails logic (custom cells) with forbidden states may be used to detect perturbations. Even if the resistance needs to be improved [19], the main drawbacks of these technologies is the effective cost of hardwired macro blocks compared to a standard implementation of cells in advanced semiconductor technologies.

3 The Attack: Description and Experiments

3.1 The Attack Description

Fault Model A strong fault model is considered for our attack. We assume that a fault injection is able to force the key bit value stored in the DES hardware key register described in 2.1, until the next update, the next reset, or the next start up of the device.

These registers are the basic elements of a microcontroller's circuit. They are used to store volatile data that are manipulated by the device close to the processing unit. At start up, the register can be fixed to a default value, or can remain uninitialized. Each bit structure is made of a basic combination of a few gates to build a storage element. Most frequently, latches or flip-flops are used to store the bit value. This value is changed on a master signal depending on the input value. The latch is sensitive to a level applied on the master signal, the flip-flop reacts on the edge of the master signal. The number of gates used in a latch or a flip-flop can vary from 4 to 8 in simple designs.

A huge work has been undertaken for years to evaluate the resistance of latches and flip-flops against perturbations [12]. Single event upset or single event transient can be prevented with hardware redundancy techniques. But, for cost reasons, it is a real constraint to duplicate or triplicate all the registers of a secure chip used in commercial devices (e.g. smart card).

Attack Scenario The fault is induced after the key loading and before the DES/3DES computation. Let us consider that the i^{th} key bit can be forced to a specific value 0 (resp. 1) in the DES hardware key register. The attacker runs an encryption of an unknown block, without injecting any perturbation, and gets the result. Then, the encryption of the same block is rerun, but by forcing the i^{th} key bit value to 0 (resp. 1). Comparing the two results, the attacker is able to retrieve the value of the i^{th} key bit. Indeed, if the original value is 0 (resp. 1), then both results must be equal. Otherwise, the two results should be different. Note that the attacker must be able to run the encryption of the same block several times (once without error, n times with an error on a different bit position every time, to recover n bits of the key). Otherwise, running twice the same block, one with an error and the other without, this being repeated n times, could also be sufficient to recover n bits.

3.2 The Fault Model in Practice

Setting up the Experiment Most of hardware DES modules currently available on the market have been studied with the following set-up. However, for demonstration purpose and simplification, only tests performed on two $0.13\mu m$ or $0.18\mu m$ integrated circuits are presented in this paper. Open samples are used

so that full control of the executed code is enabled. Besides all possible configurations of the chip can be set-up. During the experiments, all hardware and software protections against fault injection are disabled to greatly simplify the exploitation of the results. Due to the important number of metal layers on the top of the chip - 5 metal layers with dense metal lines and large power routing grid - the samples are mechanically prepared for laser enlightment [18] from the backside.

The light source is a NIR YAG ($1064\mu m$) [2] laser cutter generating powerful $6ns$ pulses. The laser beam is focused through an optical microscope and the spot can be tuned thanks to several objectives from x2 up to x100 and a XY laser shutter that makes rectangle illumination beam. The laser characteristics and set up are of higher importance to get significant experimental test results. Several trials are necessary before choosing an efficient laser spot size which is very close to the standard cells size of the considered circuit.

Assuming that the DES execution is a straightforward non-secure implementation, the experimental test code performs the following sequence of operations:

```
1. load key in HW DES key register
2. load data in HW DES data register
3. laser shot
4. execute the DES encryption
5. get DES result
```

Key and data registers are write only, therefore they cannot be read back after the shot. This limitation is present on most of hardware DES engines. The *get DES result* is used to recover the encrypted data. The laser shot is applied between the *load data* and the *execute DES* in order to modify the key or the data register. Disturbing internal working registers of the HW DES is not in the test scope.

Methodology for Register Location Locating the registers is time consuming, even if relatively the HW DES macrocell is a large part of the logic area. The entire logic can be first roughly scanned with a $100\mu m$ step. This first mapping gives sufficient data to localize an area where faulty outputs can be observed. In a second phase, deeper investigations through this area are conducted with a fine step ($10\mu m$) to find a more accurate matrix of register bit locations.

Experimental Results DES registers being not readable, ciphertexts must be analyzed. When the output differs from the normal result, an explanation is searched for by exhausting "separately" all internal variables of the algorithm. If all internal variables are considered as coded within a single byte it is possible, under this limitation, to build up a data base of candidate faulty outputs. Then experimental faulty outputs are searched within this list of candidates. This approach is not exhaustive since it does not consider combinations of several

altered byte, but it covers single bit modifications of the input data (plain text and key). In case of hardware coprocessor the notion of machine word or byte does not make sense since the internal state is hard coded with bit field registers which size fits the algorithm requirements (32, 48, 56, 64 bits depending on the DES steps). Besides, some faulty results remain mysterious. But many of them can find an explanation which is sufficient to feed further discussions.

1^{st} **experiment:** The test vector used in the 1^{st} experiment is the following:

```
Input    :  0000000000000000
Key      :  00000000FFFFFFFF
Output   :  7EC4125DD3118B2D
```

Experiment observations are reported in figure 1. Faulty ciphertexts are listed in the first column, and the associated fault on key or/and Permutation Choice 1 (PC1) bytes are detailed in the second and third columns. $I[i], K[i], PC1[i], IP[i] = xx$ means that the byte i of the Input or the Key or the value of the PC1 or the Initial bit Permutation have been modified with the value xx.

```
A6B2BA441A3B3C79  I[0]   =  C0
6B821715F6EAC91E  Key[0] =  04   PC1[5]  =  1F
4C2912268670B997  Key[0] =  18   PC1[6]  =  11
056BB9D9C0F01F4A  Key[0] =  80   PC1[0]  =  F1
738E7D3A3CE3CC7F  Key[0] =  E0
148F55FE1AE8C6B0  Key[1] =  02   PC1[4]  =  2F
D22523C61D4C23E8  Key[1] =  80   PC1[0]  =  F2
3CBD46B51B136AF8  Key[2] =  08   PC1[6]  =  40
6DB62FF095E5C24E  Key[3] =  08   PC1[6]  =  80
510F4013B51D9EA1  Key[3] =  E0
5B5E126E6F95CBC2  Key[4] =  EE   PC1[3]  =  EF
6976903C543159B4                 PC1[5]  =  2D
```

Fig. 1. Faulty results for the 1^{st} experiment

These results motivate the following observations:

- The alteration can concern the input message (I) as well as the key.
- Key alterations can also occur due to PC1 alterations. However the relationship is not bijective in the sense that some key bytes modifications cannot also be explained by a single byte alteration in PC1. Conversely, the last example shows that a PC1 modification may occur without single byte alteration of the key. This may be due to combinations of several bytes.
- The fault can set or clear bits as well.

– The fault can affect several bits.

When proceeding with such experimentation, an important point regards the possibility of reproducing it. This has been done by keeping every set-up parameters unchanged (key, scanning, laser), except the value of the input message.

2^{nd} **experiment:** Here input has been replaced by a more balanced one. Below are the new data used:

```
Input    :  A5A5A5A55A5A5A5A
Key      :  00000000FFFFFFFF
Output   :  BD2C9DA36BD38AD1
```

```
1977E5A462570475  I[0]   =  A7   IP[7]   = F1
FEDB3FD2B2C86811  I[0]   =  AD   IP[6]   = F1
138AB840885CFC67  I[1]   =  A7   IP[7]   = F2
324ADEACA89C9EA8  I[1]   =  F5
056B38442A84A25E  I[2]   =  E5   IP[0]   = F4
91DD763823C6EE94  I[3]   =  B5   IP[1]   = F8
2D4A7F714481F2A8  I[7]   =  1A   IP[0]   = 70
BE974DF14DC25F99  I[7]   =  58   IP[7]   = 70
CBAFD304C88A8D74  Key[0] =  C0
DE3C3CAC1844CA35  Key[1] =  80   PC1[0]  = F2
7FDB38A839E5057A  Key[2] =  08   PC1[6]  = 40
289113B0972DDF2B  Key[2] =  80   PC1[0]  = F4
EF03D42414B841C0  Key[3] =  02   PC1[4]  = 8F
F0C85F7E03A77799  Key[3] =  04   PC1[5]  = 8F
C778FCEAC6A8C333  Key[3] =  08   PC1[6]  = 80
66CDBF7C71A8FE81  Key[3] =  A0
5467B5C532D3B009  Key[6] =  BE   PC1[1]  = B0
039684C9DC6C62FE  Key[6] =  FA   PC1[4]  = 0B
95F5354B84F59BF6  Key[7] =  5E
0D1E71BCDD933108  Key[7] =  F6   PC1[5]  = 07
```

Fig. 2. Faulty results for the 2^{nd} experiment

Erroneous outputs are listed in figure 2. Results lead to the following remarks:

– Scanning exhaustively the chip did not affect all the key bits.
– When considering the first half of the key (Key[0-3] initially set with zeros) there are some points where single bit setting is observable.
– On the second half (Key[4-7] initially set with ones) some single bit clearing are observable too.
– These remarks lead to think that there are some points where single bit alteration is possible.

– There are also some other points where the resulting key byte does not seem to depend on the initial value, as if fully reprogrammed by the perturbation.

One may incriminate the experimental set-up instability as a reason for the variety of results from one experiment to the other. But the reproducibility is attested by the sensitivity mapping resulting from the chip scanning. It appears that the sensitive points are not so many and possibly spread over several unconnected regions. Experimentally they are easy to retrieve. From what is known of the chip design, these points do not necessarily belong to the DES engine area. The perturbation may have propagated from the enlightened point through the location of the registers.

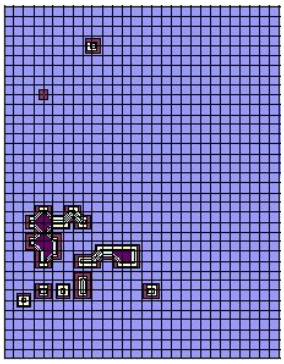

Fig. 3. fault mapping example

Sensitive regions are identified on this picture:

– Effects observed in the rightmost and upper areas remain unexplained.
– More interesting is the scattered lower left region where modifications occur on the key (upper points) and data bytes (lower points).

3^{rd} **experiment:** In order to validate that previous observations may be transposed on circuits with a different technology, a chip in $0.18\mu m$ (instead of $0.13\mu m$) is used in the following experiments. Registers modification are targeted and the attack must be reproducible, so the attack is repeated 10 times for the same physical location. The following data are used:

```
Input    :  AA55AA55AA55AA55
Key      :  FFFFFFFFFFFFFFFF
Output   :  C0BE513D923789CC
```

Resulting faulty ciphertexts, input bytes modified and the direction of the bit modification can be respectively seen in the two latest columns of the figure 4.

This third result slightly differs from those obtained on previous chip, indeed:

```
E25BA12ACEC346F2  I[4]  = AB   0>1
B5181C6E3C982121  I[3]  = 15   1>0
1836DFD7DAF34AB9  I[4]  = EA   0>1
0F6686100071DA1D  I[5]  = 5D   0>1
48688788ED15C8BE  I[5]  = 45   I[6]  = BE  0>1(x2) 1>0(x1)
997E5F9E1AAFAC3D  I[7]  = 5D   0>1
D99BBDF6521A78CA  I[5]  = 7D   0>1(x3)
FB4D1E8A48C0C3A1  I[5]  = D5   0>1
394F9FAFC535671E  I[4]  = E8   I[5]  = D5  0>1(x2) 1>0(x1)
BA5038A5193CDB43  I[4]  = 9E   I[5]  = 57  0>1(x2) 1>0(x2)
01DDEA4DF50A4A64  I[4]  = BE   0>1(x2)
DB024C5C31B5C0E6  I[3]  = 54   1>0
CC1F3E2DE62FD64C  I[3]  = 5D   0>1
36457C9572A9AA0B  I[6]  = AB   0>1
7876B26C4B5C91C1  I[7]  = 57   0>1
```

Fig. 4. Faulty results for the 3^{rd} experiment

- Only bits located in the data register are modified.
- Some bits of the data register are not modified.
- Several locations are identified where the attack has a good yield - close to 100%. So a strong relationship can be established between the laser scanning fail map and the bit modified within the register.

Several additional scans are performed to find out the remaining bits of the data. After a significant number of iterations and small optimizations of the laser set up, only a few bits of the data register are retrieved, and some of them were still missing. The key bits were not disturbed at all. Thus only a limited amount of data bits can be mapped but with a very high repetition rate. The effect of the perturbation on the bit value has also been analyzed. A quick statistic of the faulty results shows an asymmetric behavior of the flipping bits: $0,75\%$ of the bits had switch from 0 to 1 and $0,25\%$ had switch from 1 to 0. The difficulty to flip one key bit cannot be explained entirely by the initial FF value .

Physical Analysis of the Circuit The previous results show a strong relationship between the registers modification and the laser shot location. So it seems interesting to go further and to analyze the circuit used for the third experiment at the silicon -gate- level. Usually in smart cards or secure devices, the physical layout of the circuit is not public. A basic chemical sample preparation with hydrofluorhydric acid (HF) is then useful to reach the silicon surface where transistors implants are visible (bulk level). Performing a deep reverse engineering of each gate is very costly and demands a careful sample preparation: this is beyond the scope of the present study. But it is possible to distinguish the shape of the gates in their bulk implantation without any indication of their function. An optical microscope with high magnification (x500) provides with pictures of the die logic.

A focused is put on the limited area where the fail map presents several bit errors. For crosschecking the bit fail map (on the picture below, the red squares represent faulty register bits and black squares the non exploitable faults) is superposed and aligned with specific patterns on the picture. For that purpose, each standard cell is manually highlighted with different colors and overlaid with the mapping resulting from the laser scan. According to the figure 5, this method shows that the latches or flip-flops do not match easily with the map of the bits errors on the analyzed chip. It is impossible to confirm a direct correspondence between the fault injection location and the physical implantation of the registers.

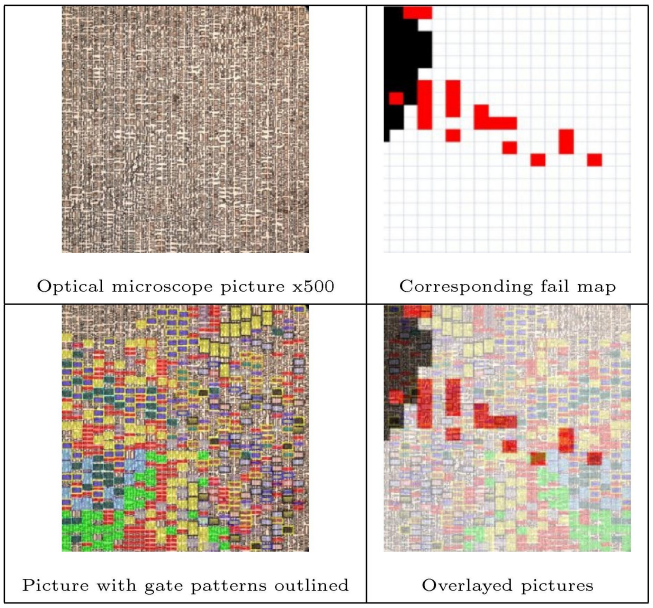

Fig. 5. Microscope picture, fail map, gate patterns and their superimposing

Trial Summary Practical results given by this couple of experiments allows discussing the basic assumptions made by E. Biham and A. Shamir in their original paper, but applied to hardware DES registers. Trying to implement the methodology they suggested, experimental conditions reveal themselves as less clear than what an attacker might expect. Fine calibration is mandatory. But it remains a difficult and risky task. The effects are not easy to control for the following reasons:

– The obtained values look sometimes random.

- The impacted bits can be sometimes cleared and sometimes set.
- The key bits can be impacted sometimes individually and sometimes by groups.
- The results are seldom complete since a full scanning does not necessarily affect all the key bits, one by one.
- The relation between the physical location of the register and the fail map is not systematic.
- The effects are different between two chips from different semiconductor technology nodes
- The design register used by the chip maker could have an impact on the results
- It may be argued that the experiment may be improved with smaller scanning steps and beam width.

As a consequence it might be possible to infer not all, but a piece of key information using Biham and Shamir's ideas, especially bit to bit inference. On the one hand the experimental set-up can be improved and thus the cryptanalysis made more effective; but on the other hand additional cryptanalysis like complementary brute force seems still necessary. However these experiments show that specific protections against this type of safe-error attacks shall be implemented.

4 The Attack in the Real World

4.1 Limitations

Limitations of the set up: The feasibility of this attack on a real product without dedicated countermeasures depends strongly on the reliability of the test set up. Compared to transient fault injections -i.e. during a HW DES computation - the timing synchronization of the laser shot is not such critical. Nevertheless, others items must be taken into consideration to evaluate the capabilities or limitations of a laser bench to implement such an attack. Indeed the laser beam positioning is of major importance. A reliable and accurate set-up is mandatory not only to perform fine scans but to come back to the desired position as well. Without such an equipment, static faults on registers are impossible. Moreover, due to dense metal layer routing in the top surface of the chip, the attack must be conducted from the backside, so that sensitive layers are directly reached. The same position between the device used for calibration and the real target must be found. This can be achieved by:

- performing a new scan with the risk to be caught by countermeasures on the real product
- using a dedicated infrared camera with high magnification

Finally, the laser itself should be able to reach only the expected bit. The small spot size used in the reported experiments for $0.13\mu m$ or $0.18\mu m$ technologies should be not suitable for advanced semiconductors technologies.

Laser positioning and effect propagation: This first result gives some indications about the mechanism of fault injection at silicon level. Moreover it shows that effects of the laser on a real circuit are not easily predictable. Firing with the laser just above the flip-flop or the latch is not the only way to induce faults in a register. Several hypotheses can explain this phenomenon. The real surface exposed by the laser beam after propagation within the silicon substrate may be different from the spot observed upon the die surface. The dense metal routing on the chip reflects the beam that can hit sensitive areas in the neighborhood of the initial spot. As the real electrical effect is unknown, the flip of the bit value in the register can also be generated by a glitch that appears on the control signal of the flip-flop or the latch.

Another hypothesis is a cell reset caused by a trigger of the reset signal, if such a signal is implemented on that register. A better understanding of the real effect produced by the laser source would require deeper investigations on specific test circuits, with dedicated test structures. Such results would help finding the optimum laser position to change one single bit.

Operating system and chip security features: According to the described experiments, injecting a fault in a HW DES register seems feasible and realistic on chip samples. However, attacking a real smart card product becomes more tricky. The calibration phase needs open samples allowing test code loading for key and message setting. Performing a precise characterization of the behavior and the location of the HW DES registers must be done before any final product attack. Otherwise, the probability to be detected by software or hardware countermeasures during the laser scanning is huge and the attacker has no chance to succeed. And even if open samples are managed carefully by chip manufacturers, criticality of such materials should be of high interest. The supply of such samples must remain restricted, as well as their traceability must be perfectly controlled.

Highlights on products: Real smart cards show very stringent limitations that make a full calibration practically impossible. The attack with a very accurate calibration (chip layout, fail map) phase may even remain unfruitful. Indeed, beyond the experimental difficulties and related risks of killing the product, the presence of anti replay schemes in most of designed cryptographic protocols in the field can represent a major obstacle. Below are famous examples of anti replay seeds:

- The application transaction counter in standard EMV banking applications [7]: the so-called ATC is upgraded in non volatile memory at each new transaction and contaminates every generated cryptogram either through session key derivation or data to be signed.
- Similarly, the sequence number used in 3GPP [1] authentication and key agreement (AKA).

– The internally generated random challenge, which is involved in Global Platform [10] secure channel protocols (SCP01/02) for mutual authentication.

In such a context, detecting a key register modification through the changing of an expected cryptogram does not make sense anymore since every cryptogram is updated at each iteration. The card is responsible of the anti replay seed management, hence its implementation must be done securely. Obviously this protocol feature is efficient only if malicious perturbations that may intend to freeze it are thwarted.

4.2 Efficient Countermeasures

Attack versus existing countermeasures None of the countermeasures cited in the previous section are inherently efficient. Indeed regarding the DES hiding for instance, a hardware engine uses a significant number of additional gates which increase the power consumption and should be more visible in power traces. Moreover the random delays in the execution make synchronization more difficult but not impossible because the delay between the key loading and the beginning of the DES computation takes a significant numbers of clock cycles. If the key is not reloaded, even before the inverse operation, the comparison will not reveal the fault injection. If the key is reloaded or if the computation is replayed, it is not much more difficult to reinject the same fault at each time.

Efficient HW countermeasures: Hardware countermeasures against single bit modification are already known in aerospace or aeronautics industries. Duplication and triplication are used in high reliable systems in space products. In the smart card world, the cost is prohibitive. Chip manufacturers need to implement cheaper solutions to prevent or detect potential safe-error attacks on HW DES registers modifications.

The detection of faults injection can be done by adding partial redundancy like additional bit for parity check or multiple bits with error correction code. Depending on the size of the register - 8 bits or 32 bits - the efficiency versus cost of the ECC is not the same.

A different approach can be to tackle the safe-error attack itself. The attacker should made hypothesis about the value of the bit regarding the output or the behavior of the device. By randomly changing the value of the physical storage of the bit the possibility to perform the attack disappear. A simple XOR with a random mask changed at each reset of the HW DES engine might be sufficient.

A similar and complementary approach would consist of changing the scrambling - position of the bit within the register - for each loading of the key or data in the HW DES module.

Efficient SW countermeasures: Although the actual hardware DES on smart cards offer user-friendly interface for programmers, low-cost implementations can benefit software protections against fault attacks. But detecting the error is not sufficient. Indeed if a key register fault detection stops the process and branches it into an infinite loop, the attacker has retrieved the information: the key bit value is different from the forced value. The safe-error is still feasible, and paradoxically less costly, since the error-free computation is not needed anymore for the attack. Thus existing countermeasures need to be improved. But finding generic countermeasures against bit forcing on key register is not trivial. Below are two examples of improved software countermeasures defeating the safe-error attack on hardware register.

One possible idea is the "Predictable Pattern Technique". It consists in hiding the true DES amongst fake ones, where fake calculations are made with fixed patterns, that is totally known parameters and predictable results. All DES are performed in a random order, and it must be impossible for the attacker to distinguish the true DES, amongst fake predictable DES. Finally, all results of predictable DES are verified. When the attacker sees that the fault injected has been detected (or not), it gives no information about the secret key. Indeed, the attacker just knows that the i^{th} bit of one of key disturbed has (or does not have) value b, the forced value. But the attacker does not know whether it concerns the true key, so the bit value of real key cannot be guessed.

Input:

Keys: $K_i = \begin{cases} true\ key\ K, \text{ if } i = 0 \\ fake\ keys, \text{ if } 0 < i < 2n \\ \overline{K_{i-2n}}, \text{ if } 2n \geq i < 4n \end{cases}$

Blocs: $M_i = \begin{cases} M, \text{ if } 0 \geq i < 2n \\ \overline{M}, \text{ if } 2n \geq i < 4n \end{cases}$

Output: Cipher Bloc R

for $i = 0$ to $4n - 1$ In Random Order **do**
  $R_i = DES(M_i, K_i)$
end for
if $R_0 \neq \oplus_{i=1}^{i<4n} R_i$ **then**
  "FAULT DETECTED"
else
  return R_0
end if

Fig. 6. Example of DES implementation defeating safe-error

Another idea is the "Complementation Property Test". It is the combination of the well known DES hiding techniques, with verification of the complementation property of the DES: $DES(M,K) = \overline{DES(\overline{M},\overline{K})}$, M and K representing respectively the input block and the key, and where $\bar{M}$ represents the bit complementation of M. Let us suppose that several fake keys are stored in non-volatile memory or randomly generated before the DES computation. The software counter-measure consists in performing all the DES (fake and true) and their complement, all in a random order. Obviously, the attacker must not be able either to distinguish a DES from a complement DES, or the true one from fake ones. Thanks to the complementation property of the DES, there are many ways to verify that all results are consistent.

This method [9] defeats entirely the attack. The figure 6 represents an example of implementation of this counter-measure for the DES cipher. This protection detects fault injections on fake and complemented keys. Here if a fault is detected, the attacker is unable to deduce something from a fault reaction of the program. Indeed, it is impossible to guess whether the injected fault modified either the true DES key, either a complemented one, or a fake one. Obviously, the final comparison must be implemented such that it is not possible for an attacker to disturb the branch instruction. Thus, the safe-error described in previous chapters is defeated.

5 Conclusion

This paper has presented a strong fault model, consisting in forcing key bits in the register of DES hardware engine. This fault model leads to a powerful safe-error attack scenario that we have described. Our fault model has been practically validated and results on two actual circuits have been given here.

This paper also shows that the attack is not so easy in practice, by discussing the requirements, the limitations of the set-up, the limitations imposed by cryptographic protocols (anti-replay). However, these experiments show that only dedicated software countermeasures may bring a sufficient confidence regarding the resistance against this attack. Some of these countermeasures have been proposed in this paper.

Acknowledgments. The authors would like to thank Pascal Paillier and all the others reviewers for their *smart* remarks.

References

1. 3GPP: 3G security; Security architecture. TS 33.102, 3rd Generation Partnership Project (3GPP) (Jun 2008), http://www.3gpp.org/ftp/Specs/html-info/33102.htm
2. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerers apprentice guide to fault attacks. Cryptology ePrint Archive, Report 2004/100 (2004), http://eprint.iacr.org/
3. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology. pp. 513–525. Springer-Verlag, London, UK (1997), http://portal.acm.org/citation.cfm?id=646762.706179
4. Blömer, J., Seifert, J.P.: Fault based cryptanalysis of the advanced encryption standard (AES), vol. 2742, pp. 162–181. SPRINGER-VERLAG BERLIN (2003), http://www.springerlink.com/index/v3epxbdwvgb5wvhn.pdf
5. Boneh, D., R.DeMillo, R.Lipton: New threat model breaks crypto codes. Bellcore Press Release (September 1996)
6. Coron, J.S.: Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. In: Ç.K. Koç, Paar, C. (eds.) Cryptographic Hardware and Embedded Systems — CHES 1999. Lecture Notes in Computer Science, vol. 1717, pp. 292–302 (1999)
7. EMV: Book 2 Security and Key Management. EMV Integrated Circuit Card Specifications for Payment Systems (June 2008)
8. Fournier, J., Loubet-Moundi, P.: Memory address scrambling reveals using fault attacks. In: Fault Diagnostic and Tolerance in Cryptography. pp. 30–36. FDTC '10 (2010)
9. Gemalto, S.A.: Cryptographic algorithm fault protections - pct patent 2010/046251 (2008), http://www.wipo.int/patentscope/search/en/WO2010046251
10. GlobalPlatformInc: Global platform card specification version 2.2 (March 2006), http://www.globalplatform.org
11. Hemme, L.: A differential fault attack against early rounds of (triple-)des. In: Joye, M., Quisquater, J.J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004, Lecture Notes in Computer Science, vol. 3156, pp. 170–217. Springer Berlin / Heidelberg (2004)
12. NASA: Nasa/gsfc radiation effects & analysis home page (2009), http://radhome.gsfc.nasa.gov/
13. NIST: Data Encryption Standard. Federal Information Processing Standards Publications (May 1977), http://www.itl.nist.gov/fipspubs/fip46.htm
14. NIST: Data Encryption Standard. Federal Information Processing Standards Publications (1999), http://www.itl.nist.gov/fipspubs/fip46-3.htm
15. NIST: Advanced Encryption Standard. Federal Information Processing Standards Publications (November 2001), http://www.itl.nist.gov/fipspubs/fip197.htm
16. NIST: NIST Special Publication 800-67 Version 1.1. Special Publication 800-67 Version 1.1 (May 2008), http://csrc.nist.gov/publications/nistpubs/800-67
17. Rivain, M.: Differential fault analysis on des middle rounds. In: Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems. pp. 457–469. CHES '09, Springer-Verlag, Berlin, Heidelberg (2009)
18. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: CHES proceedings. pp. 2–12. Springer-Verlag (2002)

19. Waddle, J., Wagner, D.: Fault attacks on dual-rail encoded systems. In: Proceedings of the 21st Annual Computer Security Applications Conference. pp. 483–494. IEEE Computer Society, Washington, DC, USA (2005), http://dl.acm.org/citation.cfm?id=1106778.1106851
20. Yen, S.M., Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Transactions on Computers 49, 967–970 (2000)