The Sorcerer's Apprentice Guide to Fault Attacks

Hagai Bar-El[1] Hamid Choukri[2,3] David Naccache[3] Michael Tunstall[3,4] Claire Whelan[5]

[1] Discretix Technologies Ltd. *hagai.bar-el@discretix.com*
[2] University Bordeaux 1 *hamid.choukri@gemplus.com*
[3] Gemplus Card International *david.naccache@gemplus.com*
[4] Royal Holloway, University of London *michael.tunstall@gemplus.com*
[5] Dublin City University *claire.whelan@computing.dcu.ie*

Abstract

The effect of faults on electronic systems has been studied since the 1970s when it was noticed that radioactive particles caused errors in chips. This led to further research on the effect of charged particles on silicon, motivated by the aerospace industry who was becoming concerned about the effect of faults in airborn electronic systems. Since then various mechanisms for fault creation and propagation have been discovered and researched. This paper covers the various methods that can be used to induce faults in semiconductors and exploit such errors maliciously. Several examples of attacks stemming from the exploiting of faults are explained. Finally a series of countermeasures to thwart these attacks are described.

1. Introduction

One of the first examples of faults being injected into a chip was accidental. It was noticed that radioactive particles produced by elements naturally present in packaging material [23] caused faults in chips. Specifically, Uranium-235, Uranium-238 and Thorium-230 residues present in the packaging decay to Lead-206 while releasing α particles. These particles create a charge in sensitive chip areas causing bits to flip. Whilst these elements were only present in two or three parts per million, this concentration was sufficient to affect chip behavior. Subsequent research included studying and simulating the effects of cosmic rays on semiconductors [32]. Cosmic rays are very weak at ground level due to the earth's atmosphere, but their effect becomes more pronounced in the upper atmosphere and outer space. This problem is further compounded by the fact that the more RAM a computer has the higher the chance of a fault occurring. This has provoked a great deal of research by organizations such as NASA and Boeing. Most of the work on fault resistance was motivated by this vulnerability to charged particles. Considerable engineering endeavors were devoted to the 'hardening' of electronic devices designed to operate in harsh environments. This has mainly been done using simulators to model circuits and study the effect of randomly induced faults. Various fault induction methods have since been discovered but all have in common similar effects on chips. One such example is the use of a laser to imitate the effect of charged particles [16]. The different faults that can be produced have been characterized to enable the design of suitable protections. The first attack that used a fault to derive secret information [8] targeted the RSA public-key cryptosystem. Basically, a fault was introduced to reveal the two secret prime numbers that compromised the RSA system. This led to similar attacks on other cryptographic algorithms. The countermeasures that can be used to thwart fault attacks had already been largely defined and successfully deployed.

This survey is organized as follows: In section 2 the various methods of fault injection and their effects are described. We then turn to theoretical (section 3) and practical (section 4) attacks. Finally, countermeasures are described in section 5.

2 Methods of Fault Injection

The most common fault injection techniques are:

1. *Variations in Supply Voltage* during execution may cause a processor to misinterpret or skip instructions. This method is widely researched and practiced behind closed doors by the smart-card industry but does not often appear in the open literature.

2. *Variations in the External Clock* may cause data misread (the circuit tries to read a value from the data

bus before the memory had time to latch out the asked value) or an instruction miss (the circuit starts executing instruction $n+1$ before the microprocessor finished executing instruction n).

3. *Temperature*: circuit manufacturers define upper and lower temperature thresholds within which their circuits will function correctly. The goal here is to vary temperature using an alcoholic cooler until the chip exceeds the threshold's bounds. When conducting temperature attacks on smart-cards (never documented in the open literature to the authors' knowledge) two effects can be obtained: the random modification of RAM cells due to heating and the exploitation of the fact that read and write temperature thresholds do not coincide in most non-volatile memories (NVMs). By tuning the chip's temperature to a value where write operations work but reads don't or the other way around a number of attacks can be mounted (components are classified into three temperature vulnerability classes which description is beyond the scope of this survey).

4. *White Light*: All electric circuits are sensitive to light due to photoelectric effects. The current induced by photons can be used to induce faults if a circuit is exposed to intense light for a brief time period. This can be used as an inexpensive means of fault induction [3].

5. *Laser* can reproduce a wide variety of faults and can be used to simulate [16] faults induced by particle accelerators [11, 29]. The effect produced is similar to white light but the advantage of a laser over white light is directionality that allows to precisely target a small circuit area. Gemplus' laser fault injection laboratory is shown in figures 1 and 2.

6. *X-rays and ion beams* can also used as fault sources (although less common). These have the advantage of allowing the implementation of fault attacks without necessarily de-packaging the chip.

2.1 The Different Types of Faults

Electronic circuits can be subject to two classes of faults: provisional (transient) and destructive (permanent) faults. In a provisional fault, silicon is locally ionized so as to induce a current that, when strong enough, is falsely interpreted by the circuit as an internal signal. As ionization ceases so does the induced current (and the resulting faulty signal) and the chip recovers its normal behavior. By opposition, destructive faults, created by purposely inflicted defects to the chip's structure, have a permanent effect. Once inflicted, such destructions will affect the chip's behavior permanently.

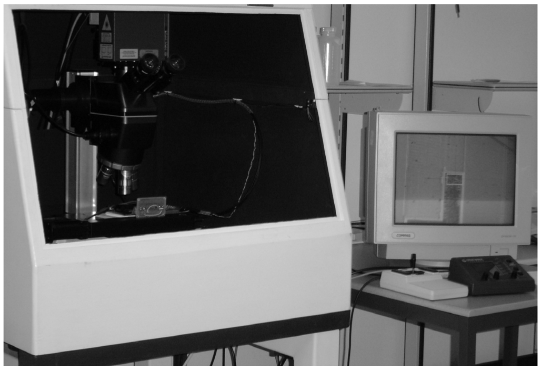

Figure 1. Laser Fault Injection Equipment

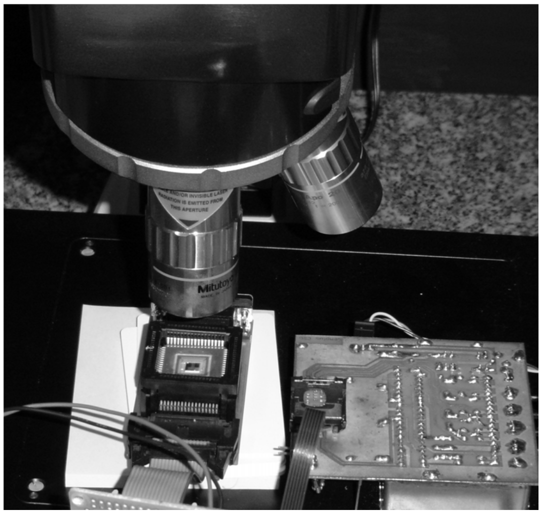

Figure 2. Laser Fault Injection Equipment (Inner View)

2.1.1 Provisional Faults (Taxonomy)

Provisional faults have reversible effects and the circuit will recover its original behavior after the system is reset or when the fault's stimulus ceases.

- *Single Event Upsets* (SEUs) are flips in a cell's logical state to a complementary state. The transition can be temporary, if the fault is produced in a dynamic system, or permanent if it appears in a static system. SEU was first noticed during a space mission in 1975 [13, 27] and stimulated research into the mechanisms by which faults could be created in chips. SEUs can also manifest themselves as a variation in an analogue signal such as the supply voltage or the clock signal.

- *Multiple Event Upsets* (MEUs) are a generalization of SEUs. The fault consists of several SEUs occurring simultaneously. A high integration density is a risk factor that can provide conditions favorable to the genesis of MEUs.

- *Dose Rate Faults* [18] are due to several particles whose individual effect is negligible but whose cumulative effect generates a sufficient disturbance for a fault to appear.

2.1.2 Destructive Faults (Taxonomy)

- *Single Event Burnout faults* (SEBs) are due a parasitic thyristor being formed in the MOS power transistors [20, 31]. This can cause thermal runaway in the circuit causing its destruction.

- *Single Event Snap Back faults* (SESs) [17] are due to the self-sustained current by the parasitic bipolar transistor in MOS transistor channel N. This type of fault is not likely to occur in devices with a low supply voltage.

- *Single Event Latch-up faults* (SELs) [1, 11] are propagated in an electronic circuit by the creation of a self-sustained current with the releasing of PNPN parasitic bipolar transistors in CMOS technology. This can potentially destroy the circuit.

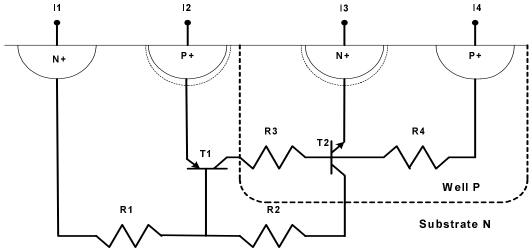

Figure 3. Single Event Latch-up - Parasitic Transistors T1 and T2.

- *Total Dose Rate faults* [9] are due to a progressive degradation of the electronic circuit subsequent to exposure to an environment that can cause defects in the circuit [30].

When using fault injection as an attack strategy provisional faults are the method of choice. These allow for faults under numerous experimental conditions to be attempted until the desired effect is achieved. As a side-bonus the system remains functional after the attack's completion. By opposition, a destructive fault would (usually) render the target unusable and will necessitate the manufacturing of a clone.

3 Fault Attacks in Theory

The first academic fault attack paper [8], proposed a number of methods for attacking public key algorithms. One attack focused on an implementation of RSA using the Chinese Remainder Theorem (CRT). The attack is very simple as it only requires one fault to be inserted in order to factor the RSA modulus. Basically the attack works as follows:

3.1 Fault Attack on RSA Signature

Let $N = p \times q$, where p and q are two large prime numbers. Let $m \in \mathbb{Z}_N^*$ be the message to be signed, d the private key and s the RSA signature. We denote by a and b the pre-computed values required for use in the CRT, such that:

$$\left\{\begin{array}{ll} a \equiv 1 & \pmod p \\ a \equiv 0 & \pmod q \end{array}\right. \quad \text{and} \quad \left\{\begin{array}{ll} b \equiv 0 & \pmod p \\ b \equiv 1 & \pmod q \end{array}\right.$$

and define:

$$\begin{array}{ll} d_p = d & \pmod{p-1} \\ d_q = d & \pmod{q-1} \end{array}$$

Using repeated squaring calculate:

$$s_p = m^{d_p} \pmod p$$

$$s_q = m^{d_q} \pmod q$$

The RSA signature s is then obtained by the linear combination $s = a \times s_p + b \times s_q \pmod N$

The attack is based on being able to obtain two signatures of the same message, where one signature is correct and the other faulty. By "faulty" we mean that a fault injected during the computation corrupted either the computation of s_p or s_q.

Let $\hat{s} = a \times s_p + b \times \hat{s}_q \pmod N$ be the faulty signature (we arbitrarily assume that the error occurred during the computation of s_q but the attack works just as well when s_p is corrupted). Subtraction yields:

$$\Delta = s - \hat{s} = (a \times s_p + b \times s_q) - (a \times s_p + b \times \hat{s}_q) = a(s_q - \hat{s}_q)$$

Hence, (given that $a \equiv 0 \pmod q$) one notes that $\Delta = a(s_q - \hat{s}_q)$ is a multiple of q. A simple GDC calculation will thus factor N:

$$\text{GCD}(\Delta, N) = q$$

In summary all that is required to break RSA is one correct signature and one faulty one. This attack will be successful regardless of the type or number of faults injected during the process provided that all faults affect the computation of s_p or (mutually exclusive or!) s_q.

Although initially theoretical, this attack (implemented in [5]) stimulated the genesis of a variety of fault attacks against a wide gamut of cryptographic algorithms. The following subsections describe some more of these attacks.

3.2 Fault Attack on RSA Decryption

Suppose that one bit in the binary representation of d flips from from 1 to 0 or vice versa, and that this faulty bit position is randomly located. An attacker arbitrarily chooses a plaintext m and computes the ciphertext c. He then injects a fault during c's decryption and gets a faulty plaintext $\hat{m}$. Assuming that bit $d[i]$ flips to $\overline{d[i]}$, then division of the faulty plaintext by the correct one will yield:

$$\frac{\hat{m}}{m} = \frac{c^{2^i \overline{d[i]}}}{c^{2^i d[i]}} \pmod N$$

Obviously, if

$$\frac{\hat{m}}{m} = \frac{1}{c^{2^i}} \pmod N \Rightarrow d[i] = 1$$

and if

$$\frac{\hat{m}}{m} = c^{2^i} \pmod N \Rightarrow d[i] = 0$$

This process is repeated until enough information is obtained on d. The attack works if and only if one bit is changed. If for example two bits (i and j) are changed then the result will resemble the changing of one bit (k), where the the sign depends on how the bit is changed:

$$\pm 2^i \pm 2^j = \pm 2^k$$

It should be noted that the attack also works for multiple bit errors. The more bits that are changed the more pronounced the effect becomes. Details and variants can be found in [6]. This attack can also apply to discrete logarithm based public key cryptosystems such as DSA.

3.3 Fault Attacks on Key Transfer or NVM

In this scenario [7] a fault is injected during the transfer of secret data from one memory component to another. Although the attack is applicable to any algorithm let us assume that a DES key is being transferred from EEPROM to RAM in a smart card. If we change the value of parts of the key to some fixed value (for example one byte at a time), it becomes possible to derive the secret key.

We DES-encrypt a message M to obtain a faultless ciphertext C_0. Then, during the key transfer from EEPROM to RAM, one key byte is changed to a fixed known value (00 in our example). The resulting C_1 is recorded and the process is repeated by forcing two bytes to a fixed value, then three bytes, and so on. This continues until the whole key but one byte has been set, byte by byte, to the fixed value.

This procedure shown in table 1, where C_i represents the ciphertext of an unknown key with i bytes set to a fixed value . Once this data has been collected it can be used to derive the DES key.

Table 1. The Biham-Shamir Attack

Input	DES Key	Output
$M \rightarrow$	K_0 =`XX XX XX XX XX XX XX XX`	$\rightarrow C_0$
$M \rightarrow$	K_1 =`XX XX XX XX XX XX XX 00`	$\rightarrow C_1$
$M \rightarrow$	K_2 =`XX XX XX XX XX XX 00 00`	$\rightarrow C_2$
$M \rightarrow$	K_3 =`XX XX XX XX XX 00 00 00`	$\rightarrow C_3$
$M \rightarrow$	K_4 =`XX XX XX XX 00 00 00 00`	$\rightarrow C_4$
$M \rightarrow$	K_5 =`XX XX XX 00 00 00 00 00`	$\rightarrow C_5$
$M \rightarrow$	K_6 =`XX XX 00 00 00 00 00 00`	$\rightarrow C_6$
$M \rightarrow$	K_7 =`XX 00 00 00 00 00 00 00`	$\rightarrow C_7$

Let K_n represent the original DES key with n bytes replaced with known values. To find K_7 the 128 different possible values for the first byte of the DES key are tried until one produces the ciphertext C_7[1]. After this K_6 can be found by searching through the 128 different possible values for the second byte, as the first byte will be known. Finding the entire key will require a search through a key space of 1024 different keys. This attack can also be used when unknown data is manipulated by an known algorithm.

Historical note: An attack similar to [7] was discovered and documented (but never published) during a code audit in Gemplus back in 1994. The code was that of a smart-card operating system where a special file contained DES keys saved in records. This OS featured two commands: `erase` i, a command that erases the i-th key record and `encrypt` i, M a command that outputs the ciphertext of the message M using the key contained in the i-th record. While invisible for the user, the OS was using the convention that all-zero keys are free records (an `encrypt` command on a zero (erased) record would return an error). The attack here was exploiting the fact that EEPROM could only be erased by 32-block units. In other words, upon an `erase`, the OS would erase twice four bytes. The attack consisted of encrypting a message with an unknown key and then instructing the OS to erase this key but cutting power just after the first 32-bit block's deletion. The card will then contain a 56-bit key which rightmost half is zeroed (which is not

[1] Although a byte is changed only 128 different values are possible as the least significant bit is a parity bit.

interpreted by the OS as an empty key record!). An encryption with this key followed by two 2^{28} exhaustive search campaigns would have eventually revealed the key.

Since that date, OSs associate a security bit σ to each key. When a user instructs to delete a key, the σ bit is erased first, thereby recording the information that the key cannot be used anymore for cryptographic operations. Only then will the OS undertake the task of erasing the key's actual bits. Upon reset, the OS ascertains that all $\sigma = 0$ keys contain zero bytes if any nonzero $\sigma = 0$ keys are found, the OS simply resumes the deletion of their bits.

3.4 Fault Attacks on DES

DES is a 16-round secret key algorithm based on a Feistel structure. This attack targets DES' fifteenth round. We use a simplified description of the last round (figure 4) to explain what happens when the fifteenth round does not execute properly[2].

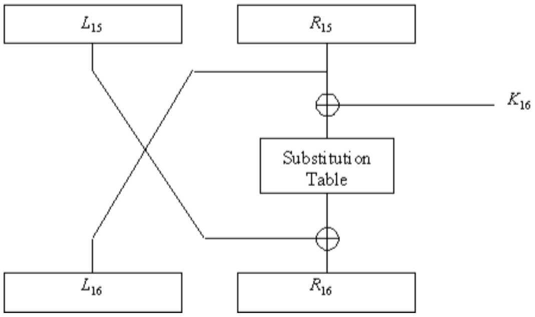

Figure 4. Simplified DES Last Round Model.

The output of the last round can be expressed as:

$$R_{16} = S(R_{15} \oplus K_{16}) \oplus L_{15}$$

$$= S(L_{16} \oplus K_{16}) \oplus L_{15}$$

If a fault occurs during the execution of the fifteenth round, *i.e.* R_{15} is changed into a faulty $\hat{R}_{15}$, then:

$$\hat{R}_{16} = S(\hat{R}_{15} \oplus K_{16}) \oplus L_{15}$$

$$= S(\hat{L}_{16} \oplus K_{16}) \oplus L_{15}$$

If we xor R_{16} and $\hat{R}_{16}$ we get:

$$R_{16} \oplus \hat{R}_{16} = S(L_{16} \oplus K_{16}) \oplus L_{15} \oplus S(\hat{L}_{16} \oplus K_{16}) \oplus L_{15}$$

$$= S(L_{16} \oplus K_{16}) \oplus S(\hat{L}_{16} \oplus K_{16})$$

This gives a relationship where only the value of the sixteenth subkey (K_{16}) is unknown; all the other variables being given directly as an output of the DES. For each substitution table used in the last DES round this relationship will be true. An exhaustive search of the 64 possible values that validate this equation can be conducted for each of the six bits corresponding to the input of each substitution table. This will give approximately 2^{18} different hypotheses for the last subkey leading to a final exhaustive search through 2^{26} DES keys to find the whole key. In practice, it is simplest to conduct the attack several times either at different positions in the fifteenth round or with a varying message. When the lists of possible hypotheses are generated the actual subkey will show up in the intersection of all the sets of hypotheses. If the difference between the two output values for a given substitution table (R_{16} and $\hat{R}_{16}$) is zero then all the possible values of K_{15} for that substitution table will be valid. This means that it is advantageous to induce a fault as early as possible in the fifteenth round so that the effect of the fault spreads over as many different substitution tables in the sixteenth round as possible.

3.5 Fault Attacks on Other Algorithm - Further Reading

While the bibliography on the matter would be too voluminous to overview exhaustively, the authors attract the reader's attention to a more powerful attack [28] applicable to all secret key algorithms. Several authors *e.g.* [12, 10] present fault attacks on AES or RC5 [2]. The details of these are beyond the scope of this article and are presented as further reading.

4 Some Experimental Fault Attacks

In a glitch attack, the attacker deliberately generates a malfunction that causes one or more flip-flops to transition into a wrong state. The aim is usually to replace a single critical machine instruction with an almost arbitrary one. Glitches can also aim to corrupt data values as information is transferred between registers and memory [19]. There are three main techniques for creating fairly reliable malfunctions that affect only a very small number of machine cycles in smart-card processors. These are clock signal transients, power supply transients, and external electrical field transients. All three were successfully experimentally implemented by Gemplus. Particularly interesting instructions, that an attacker might want to target with glitches, are conditional jumps or the test instructions preceding them. They create a window of vulnerability in the processing stages of many security applications that often allow the attacker to bypass sophisticated cryptographic barriers by simply preventing the execution of the code that detects that an au-

[2] In figure 4 bit permutations were removed as these do not fundamentally change theory although they somewhat complicate explanation.

thentication attempt was unsuccessful. Instruction glitches can also be used to extend the runtime of loops, for instance in serial port output routines, to see more of the memory after output buffer, or reduce the runtime of loops, thereby transforming an iterated block-cipher into an easy to break single-round variant [19]. Clock-signal glitches are currently the simplest and most practical ones. They temporarily increase the clock frequency for one or more half cycles, such that some flip-flops sample their input before the new state has reached them. Power analysis was used by this survey's authors to monitor how far a program has progressed and launch a fault as the power profile of a specific instruction was recognized. This in turn can be used to determine when, for example, a branch instruction is about to be taken. A more rapid clock cycle at this point (a clock glitch) may provide insufficient time for the processor to write the jump address to the program counter, thereby annulling the branch operation [24]. A similar clock-glitch attack is also presented in [2]. Because of the different number of gate delays in various signal paths and the varying parameters of the circuits on the chip, this affects only some signals, and by varying the precise timing and duration of the glitch, the CPU can be fooled to execute a number of completely different, wrong instructions. These will vary from one instance of the chip to another, but can be found by a systematic search using specialized hardware.

The following figures illustrate different effects that glitches can have. In this experiment power was dropped from V_{cc} to 0V during a few nanoseconds. By carefully playing with the glitch's parameters (duration, falling edge, amplitude *etc.*) two types of behavior were obtained:

- Under a first set of conditions (figure 5), the processor just skipped a number of instructions and resumed normal execution several microseconds after the glitch. This fault allows the selective execution of instructions in a program.

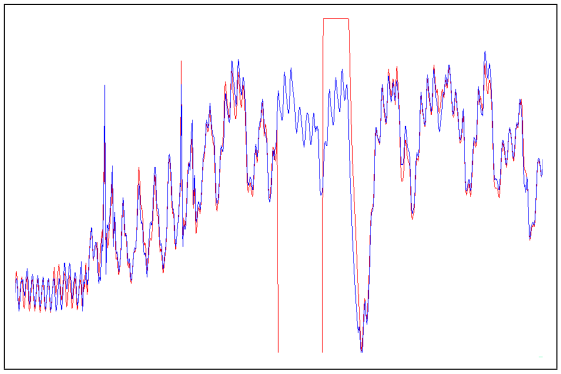

Figure 5. Instruction Only Glitch Attack

- Under a second set of conditions, not only does the processor skip instructions - but the value of data manipulated by the processor is also modified in a precise manner. This is visually reflected in the power curves of figure 6.

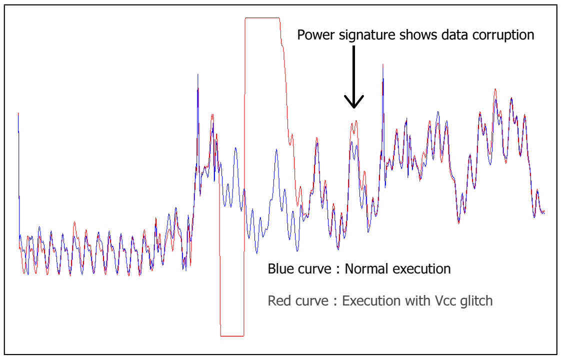

Figure 6. Instruction and Data Glitch Attack

It should be noted that a third set of conditions was tested in this experiment. Although the results are not shown here, the outcome was that the value of data could be corrupted while the interpretation of instructions was left unchanged.

The following two images show glitch injection electronics used in mounting these attacks. The data aquisition board shown in figure 7 was initially developed for performing differential power analysis. It was then extended to incorprate glitch attacks. The board accepts a signal from a CLIO reader instructing the aquisition board to apply a lower voltage to the V_{cc} for the duration of that signal. The levels of voltage that are applied during the glitch are controlled via potentiometers configured with a screwdriver.

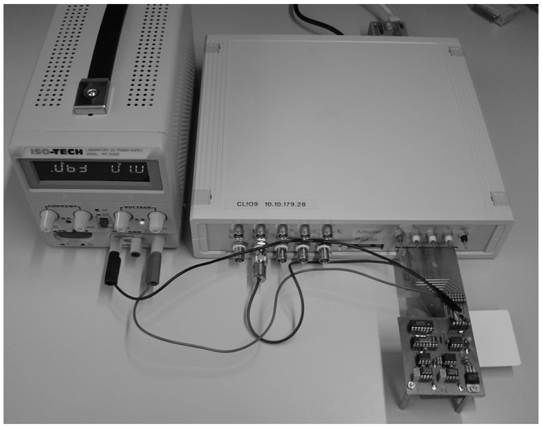

Figure 7. Data Aquisition Board with CLIO Reader

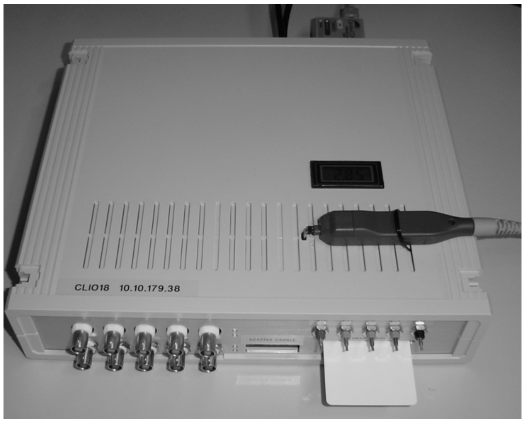

Figure 8. A Modified CLIO Reader

Figure 8 shows a modified clio reader that can be used to inject a glitch at a specific point during a command. This setup can be configured via the network to allow for a large number of glitch configurations to be tested when searching for vulnerablities in new chips.

Glitch attacks have been reported against a number of cryptographic systems. We will describe here a few such attacks in further detail.

4.1 Glitch Attack on RSA

The GCD attack presented in section 3 was implemented by [5] and others. We also refer the reader to [6] and [15] who report clock-glitch attacks against RSA and DES.

4.2 Glitch Attack on DES

When we can cause an instruction of our choice to fail, then there are several fairly straightforward ways to attack DES. We can remove one of the 8-bit xor operations that are used to combine the round keys with the inputs to the S-boxes from the last two rounds of the algorithm, and repeat this for each of these key bytes in turn. The erroneous ciphertext outputs that we receive as a result of this will each differ from the genuine ciphertext in the output of usually two, and sometimes three, S-boxes. Using the techniques of differential cryptanalysis, we obtain about five bits of information about the eight key bits that were not xor'ed as a result of the induced fault. So, for example, six ciphertexts with faulty last rounds should leak about thirty key bits, leaving an easy brute-force search [2]. An even faster attack brutally reduces the number of DES rounds to one or two by corrupting the appropriate loop variable or conditional jump. As a conclusion, unprotected DES can be compromised in a variety of ways with somewhere between one and ten faulty ciphertexts. Analogous attacks on AES were successfully mounted in Gemplus' laser laboratory.

4.3 Glitch Attack on EEPROM

EPROM stores information as charges in the gate insulator of a MOSFET; charge is stored on the floating gate of a MOS transistor and the control gate is used to program the transistor as shown in figure 9. EEPROM transfers electrons by Fowler-Nordheim tunnelling and program/erase operations are carried out by electrons tunnelling through the thin oxide. Control gate voltage is high for programming while for erasure the control gate is grounded and the drain voltage is raised. To read information from a cell, the cell's static voltage is compared to a reference detection voltage V_{det} (usually $V_{det} = V_{cc}/2$). Consequently, if programming is done under the lowest tolerable voltage a lesser amount of particles will be forced into the cell. Then, if during reading V_{cc} is increased to the highest value tolerated by the circuit V_{det} is artificially boosted and hence data will be read as zero regardless it's actual value. To attack an n byte key one can simply subject the circuit to $n - 1$ power glitches to obtain the encryption of a known plaintext under a vulnerable key of the form:

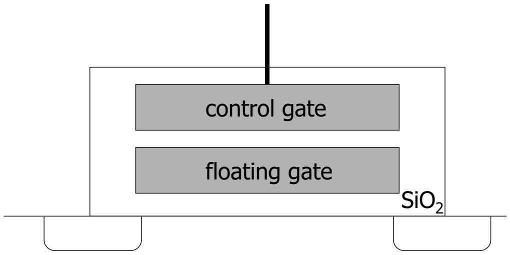

Figure 9. EPROM

```
00 00 ... 00 00 XX 00 00 ... 00 00
```

The attacker will then move the glitch's position to successively scan the entire key. This attack was implemented by Gemplus in the late 1990s.

4.4 Analogous Laser Attack on a Data Bus

In a specific smart card chip, a laser impact on the data bus during information transfer has the effect of reading the value 255 (0xFF) regardless the transferred information's actual value. The attack described in the previous subsection could hence be directly re-adapted in Gemplus' laser laboratory.

4.5 The Java Sandbox

The Java sandbox is an environment in which applets are run without direct access to the computer's resources. The idea being that an applet need not be trusted as it is incapable of running malicious code. The most common example of Java programs being used is on the Internet, where an applet is downloaded and executed on a PC to achieve a given effect on the webpage being observed. A relatively recent paper [14] describes a fault attack on a PC forcing the Java Virtual Machine to execute arbitrary code. This was done by using a spotlight to heat up the PC's RAM to the point where a fault (in this case a bit flip) occurs. In this case a special applet was loaded into the computer's memory and the RAM heated up to the point where some bits would change their value. The expected fault was that the address of a function a called by the applet would have one bit changed, so that the address called was $a \pm 2^i$, where $0 \leq i \leq 31$ (the computer's word size). The programmer arranges to have a function present at that address that will return a variable of a type that is not expected by the calling function, for example an integer to a pointer. This can then be used to read/write to arbitrary addresses in the computers memory. One of the possible uses of such a fault would be to change fields in the Java runtime system's security manager to grant the applet illegal rights.

5 Countermeasures

Since the identification of faults as a problem in electronic systems several hardening methods were deployed. These solutions help circuits to avoid, detect and/or correct faults. Hardware and software countermeasures will be overviewed separately for the sake of clarity.

5.1 Hardware Countermeasures

Hardware protections are implemented by the chip manufacturer and can be further sub-divided into two categories: *active* and *passive* protections.

5.1.1 Active Protections:

- *Light detectors* detect changes in the gradient of light.
- *Supply voltage detectors* react to abrupt variations in the applied potential and continuously ascertain that voltage is within the circuit's tolerance thresholds.
- *Frequency detectors* impose an interval of operation outside which the electronic circuit will reset itself.
- *Active shields* are metal meshes that cover the entire chip and has data passing continuously in them. If there is a disconnection or modification of this mesh the chip will not operate anymore. This is primarily a countermeasure against probing, although it helps protecting against fault injection as it makes the location of specific blocks in a circuit harder.
- Hardware redundancy:

1. *Simple Duplication with Comparison (SDC)* is a the duplication of hardware blocks followed by a test by a comparator. When the two blocks' results don't match, an alert signal is transmitted to a decision block. Two types of reaction can be implemented: a hardware reset or the activation of an interruption that triggers dedicated countermeasures. SDC protects against single focused errors and only permits their detection. A feedback signal is usually triggered to stop all outgoing data flows.

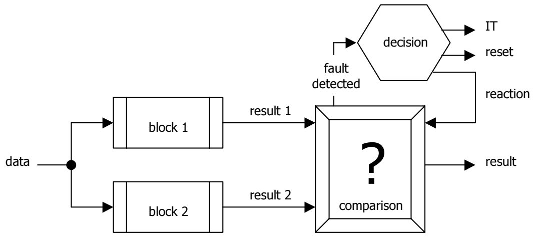

Figure 10. Simple Dupilcation with Comparison.

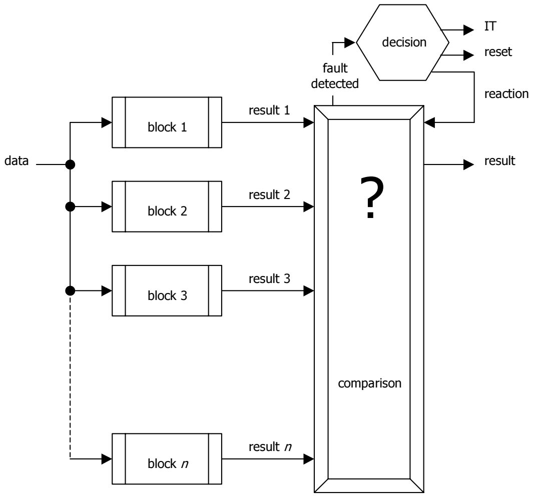

Figure 11. Multiple Duplication with Comparison.

2. *Multiple Duplication with Comparison (MDC):* each hardware block is duplicated at least thrice. The comparator detects any mismatch between results and transmits the alert signal to the decision block. As previously, two types of reaction can be implemented, a hardware reset or the activation of an interruption. The difference with SDC being the possibility to correct the fault through a majority vote and correct the outgoing signal.

3. *Simple Duplication with Complementary Redundancy (SDCR)* is based on the same principles as SDC but the two blocks store complemented data. When the result of the two blocks match, the comparison block transmits an alert to the system that triggers a hardware reset or an interrupt. SDCR protects against multiple focused errors since it is difficult to inject two different errors with complementary effects, but (just as SDC) SDCR only permits error detection.

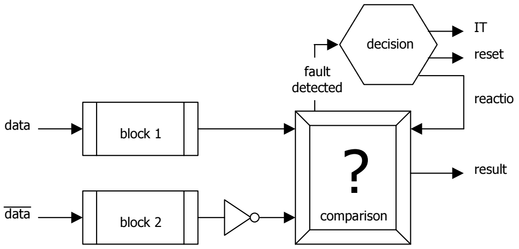

Figure 12. Simple Duplication with Complementary Redundancy.

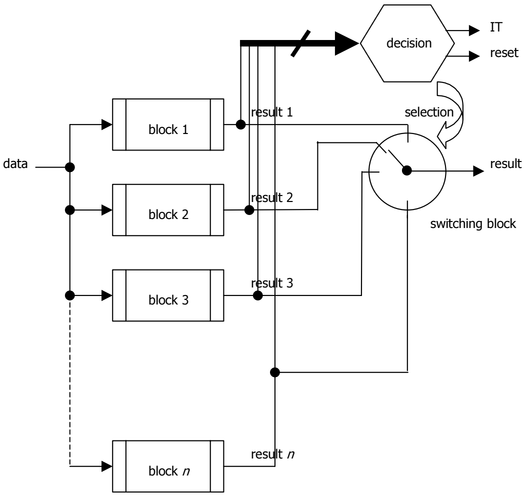

Figure 13. Dynamic Duplication

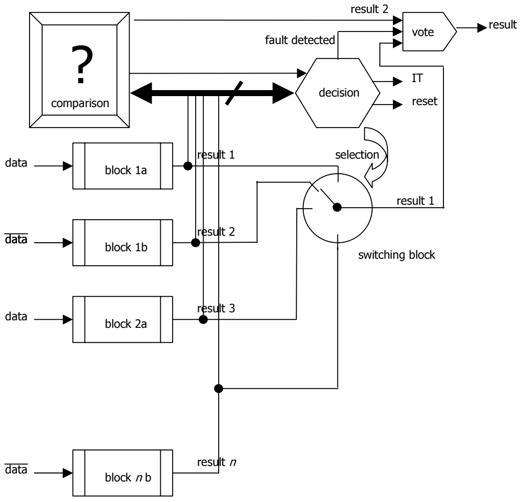

Figure 14. Hybrid Duplication.

4. *Dynamic Duplication* consists of multiple redundancies with a decision module, commanding a data switch upon fault detection. The vote block is a switch, which transmits the correct result as instructed by the comparator. Corrupted blocks are disabled and their results discarded. This type of implementation permits detection and subsequent reaction to the detected error [22].

5. *Hybrid Duplication* is a combination of multiple duplications with complementary redundancy and dynamic duplication. This protects against single and multiple focused faults, as it is very difficult to inject multiple faults with complementary effects.

- Protection using time redundancy:

1. *Simple Time Redundancy with Comparison (STRC)* consists of processing each operation twice and comparing results [4]. This protects against single and multiple time synchronized errors but is only capable of detecting faults. Reaction is limited to the discarding of the corrupted results.

2. *Multiple Time Redundancy with Comparison* is based on the principle used by STRC but the result is processed more than twice. This detects, reacts and possibly corrects single and multiple faults.

3. *Re-computing with Swapped Operands* consists of re-computing results with the operands' little endian and big endian bits swapped. The result is re-swapped and compared to detect potential faults. This type of protection has the advantage of de-synchronizing two different processes and makes fault attacks very difficult.

This countermeasure protects against single and multiple time synchronized errors.

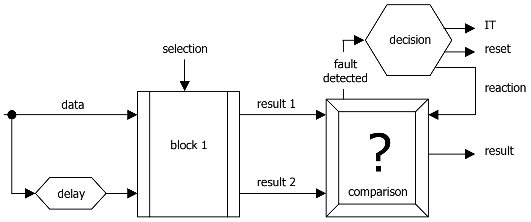

Figure 15. Simple Time Redundancy with Comparison

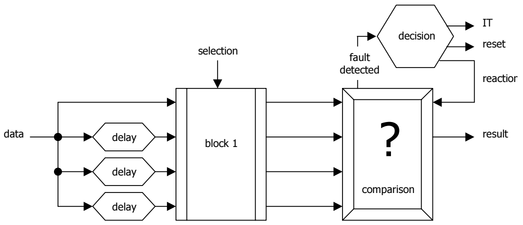

Figure 16. Multiple Time Redundancy with Comparison

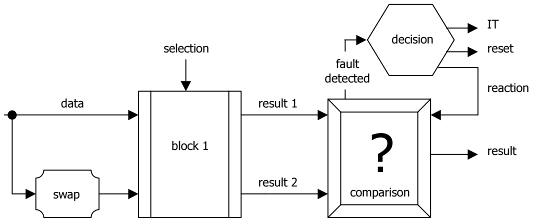

Figure 17. Re-computing with Swapped Operand

4. *Re-computing with Shifted Operands:* [25] operations are recomputed by shifting the operands by a given number of bits. The result is shifted backwards and compared to the original one.

5. *Re-computing with Duplication with Comparison* is a combination of time redundancy and hardware redundancy. This protects against single, multiple and time synchronized faults but the time penalty and the increase in block size limit this countermeasure's use.

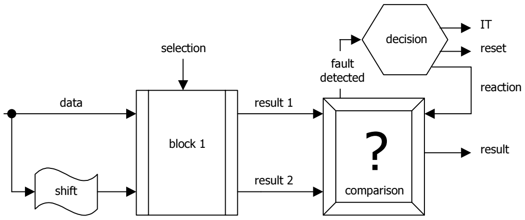

Figure 18. Re-computing with Shifted Operand

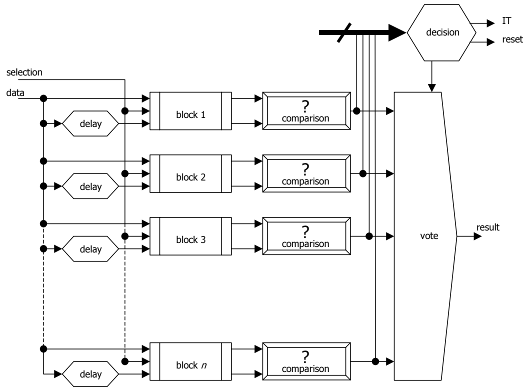

Figure 19. Re-computing with Duplication with Comparison

- *Protection by Redundancy Mechanisms* such as Hamming codes [21], hardwired checksums and error correction codes are also used to avoid or detect faults [26]. The typical example being checksums attached to each machine word in RAM or EEPROM to ensure integrity.

5.1.2 Passive Protections:

The second class of hardware protection mechanisms consists of *passive protections* that increase the difficulty of successfully attacking a device. These protections can be self-activated or managed by the device's programmer:

- Mechanisms that introduce *dummy random cycles* during code processing.

- *Bus and memory encryption.* Let h be a hardwired keyed permutation and f a simple hardwired block-cipher. Upon power-on, the chip generates an ephemeral key k. When the microprocessor wishes to write the value m at RAM address i, the system stores $v = f_k(m, i)$ at address $h_k(i)$. When the microprocessor requires the contents of address i, the system recomputes $h_k(i)$, fetches v from address $h_k(i)$, decrypts $m = f_k^{-1}(v, i)$ and hands m to the microprocessor. This makes laser or glitch targeting of a specific memory cell useless as successive computations with *identical* data use *different* memory cells.

- *Passive shield*: a full metal layer covers some sensitive chip parts, which makes light or electromagnetic beam attacks more difficult as the shield needs to be removed before the attack can proceed. This also allows to contain information leakage through electromagnetic radiations (*i.e.* thwart some side-channel attacks).

- *Unstable internal frequency generators* protect against attacks that need to be synchronized with a certain event, as events occur at different moments in different executions.

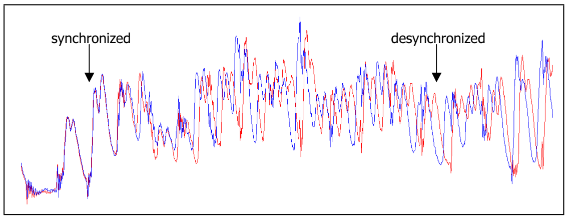

Figure 20. Unstable Internal Frequency Generation Reflected in Power Consumption.

5.2 Software Countermeasures

Software countermeasures are implemented when hardware countermeasures are insufficient or as cautious protection against future attack techniques that might defeat present-generation hardware countermeasures. The advantage of software countermeasures is that they do not increase the hardware block size, although they do impact the protected functions' execution time.

- *Checksums* can be implemented in software. This is often complementary to hardware checksums, as software CRCs can be applied to buffers of data (sometimes fragmented over various physical addresses) rather than machine words.

- *Execution Randomization*: If the order in which operations in an algorithm are executed is randomized it becomes difficult to predict what the machine is doing at any given cycle. For most fault attacks this countermeasure will only slow down a determined adversary, as eventually a fault will hit the desired instruction. This will however thwart attacks that require faults in specific places or in a specific order, such as the transferring of secret data attack described previously.

- *Variable redundancy* is nothing but SDC in software.

- *Execution redundancy* is the repeating of algorithms and comparing the results to verify that the correct result is generated. As SDCR, redundancy is more secure if the second calculation is different than the first (for example its inverse[3]) so that two identical faults cannot be used at different times.

- *Ratification counters and baits*: baits are small ($<$ 10 byte) code fragments that perform an operation and test it's result. A typical bait writes, reads and compares data, performs xors, additions, multiplications and other operations whose results can be easily checked. When a bait detects an error it increments an NVM counter and when this counter exceeds a tolerance limit (usually three) the card ceased to function.

In theory all data redundancy method used in hardware can be implemented in software. The problem then becomes execution time rather than block size. As some of the proposed hardware designs become extremely time consuming when imitated by software.

6 Conclusion

Various methods for creating faults were presented. Practical applications of these attacks were presented. These applications included attacks on keys and symmetric and asymmetric cryptosystems. Finally, hardware and software countermeasures were overviewed. Unfortunately, these countermeasures never come for free and impact the cost of the system being developed. Also, the resulting system will be slower and may feature an increased block size. There will always be a tradeoff between cost, efficiency and security, and it will be a judgement call by designers, developers and users to choose which of these requirements best suit their needs. There is still much work to be done in this area with the ultimate goal being an optimal balance between security, efficiency and cost.

References

[1] L. Adams, E. J. Daly, R. Harboe-Sorensen, et al. "A verified Proton Induced Latchup in Space", *In IEEE Transactions on Nuclear Science*, vol. 39, pp. 1804-1808, 1992.

[3] Encrypt-decrypt, sign-verify *etc.*

[2] R. Anderson and M. Kuhn. “Low Cost Attacks on Tamper Resistant Devices”, *IWSP: 5th International Workshop on Security Protocols*, LNCS 1361, Springer-Verlag, pp. 125-136, 1997.

[3] R. Anderson and S. Skoroboatov.“Optical Fault Induction Attacks”, *In Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002)*, LNCS 2523, Springer-Verlag, ISBN 3-540-00409-2, pp. 2-12, 2002.

[4] L. Anghel and M. Nicolaidis. “Cost Reduction and Evaluation of a Temporary Faults Detecting Technique”, in Proceedings of *Design, Automation and Test in Europe (DATE ’00)*, pp. 591-597, 2000.

[5] C. Aumüller, P. Bier, P. Hofreiter, W. Fischer and J.-P. Seifert. “Fault attacks on RSA with CRT: Concrete Results and Practical Countermeasures”, Cryptology ePrint Archive: Report 2002/073. http://www.iacr.org.

[6] F. Bao, R.H. Deng, Y. Han, A. Jeng, A.D. Narasimhalu and T. Ngair. “Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults”, the Proceedings of the *5th Workshop on Secure Protocols*, LNCS 1361, Springer-Verlag, pp. 115-124, Paris, April 7-9, 1997.

[7] E. Biham and A. Shamir. “Differential Fault Analysis of Secret Key Cryptosystems”, in Prceedings of *Advances in Cryptology - CRYPTO ’97*, LNCS 1294, pp. 513-525, 1997.

[8] D. Boneh, R. DeMillo and R. Lipton. “On the Importance of Checking Cryptographic Protocols for Faults”, *Journal of Cryptology*, Springer-Verlag, Vol. 14, No. 2, pp. 101-119, 2001.

[9] Ph. Cazenave, P. Fouillat, X. Montagner, H. Barnaby, R. D. Schrimpf, L. Bonora, J. P. David, A. Touboul, M. -C. Calvet and P. Calvel. “Total dose effects on gate controlled lateral PNP bipolar junction transistors”, *In IEEE Transactions on Nuclear Science*, vol. 45, pp. 2577-2583, 1998.

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

[11] P. Fouillat. “Contribution à l’étude de l’interaction entre un faisceau laser et un milieu semiconducteur”, Applications à l’étude du Latchup et à l’analyse d’états logiques dans les circuits intégrés en technologie CMOS, Thèse de doctorat de l’université Bordeaux I, 1990.

[12] Ch. Giraud, “DFA on AES”, Cryptology ePrint Archive: Report 2003/008. http://www.iacr.org.

[13] T. J. O’Gorman. “The effect of cosmic rays on soft error rate of a DRAM at ground level”, *In IEEE Transactions On Electronics Devices*, vol. 41, pp. 553-557, 1994.

[14] S. Govindavajhala and A. W. Appel. “Using Memory Errors to Attack a Virtual Machine”, in the *2003 IEEE Symposium on Security and Privacy*, pp. 154-165, 2003.

[15] O. Grabbe. “Smartcards and Private Currencies”, `http://www.aci/net/kalliste/smartcards.htm`

[16] D.H Habing. “The Use of Lasers to Simulate Radiation-Induced Transients in Semiconductor Devices and Circuits”, *In IEEE Transactions On Nuclear Science*, vol.39, pp. 1647-1653, 1992.

[17] R. Koga and W. A. Kolasinski. “Heavy ion induced snapback in CMOS devices”, *In IEEE Transactions on Nuclear Science*, vol. 36, pp. 2367-2374, 1989.

[18] R. Koga, M. D. Looper, S. D. Pinkerton, W. J. Stapor, and P. T. McDonald. “Low dose rate proton irradiation of quartz crystal resonators”, *In IEEE Transactions on Nuclear Science*, vol. 43, pp. 3174-3181, 1996.

[19] O. Kommerling and M. Kuhn. “Design Principles for Tamper Resistant Smartcard Processors”, *Proceedings of the USENIX Workshop on Smartcard Technology*, pp. 9-20, 1999.

[20] S. Kuboyama, S. Matsuda, T. Kanno and T. Ishii. “Mechanism for single-event burnout of power MOSFETs and its characterization technique”, *In IEEE Transactions On Nuclear Science*, vol. 39, pp. 1698-1703, 1992.

[21] F. Lima, E. Costa, L. Carro, M. Lubaszewski, R. Reis, S. Rezgui and R. Velazco. “Designing and Testing a Radiation hardened 8051-like Micro-Controller”, in the *3rd Military and Aerospace Applications of Programmable Devices and Technologies International Conference*, 2000.

[22] J. Losq. “Influence of fault detection and switching mechanisms on reliability of stand-by systems”, *In Digest 5th International Symp Fault-Tolerant Computing*, pp. 81-86, 1975.

[23] T. May and M. Woods. “A New Physical Mechanism for Soft Errors in Dynamic Memories”, in the Proceedings of the *16th International Reliability Physics Symposium*, April, 1978.

[24] S. Moore, R. Anderson and M. Kuhn. “Improving Smartcard Security using Self-Timed Circuit Technology”, *IEEE International Symposium on Ansychnronous Circuits and Systems*, pp. 120-126, 2002.

[25] J. H. Patel and L. Y. Fung. “Concurrent Error Detection in ALU’s by Recomputing with Shifted Operands”, *In IEEE Transactions On Computers*, vol. C-31, pp. 589-595, 1982.

[26] M. Pflanz, K. Walther, C. Galke and H. T. Vierhaus. “On-Line Detection and Correction in Storage Elements with Cross-Parity Check”, In Proceedings of the *8th IEEE International On-Line Testing Workshop (IOLTW’02)*, pp. 69-73, 2002.

[27] J. C. Pickel and J. T. Blandford, Jr. “Cosmic ray induced errors in MOS memory circuits”, *In IEEE Transactions On Nuclear Science*, vol. NS-25, pp. 1166-1171, 1978.

[28] G. Piret and J. J. Quisquater. “A Differential Fault Attack Technique Against SPN Structure, with Application to the AES and KHAZAD”, in *Cryptographic Hardware and Embedded Systems (CHES 2003)*, LNCS 2779, Springer-Verlag, pp. 77-88, 2003.

[29] V. Pouget. “Simulation expérimentale par impulsions laser ultra-courtes des effets des radiations ionisantes sur les circuits intégrés”, Thèse de doctorat de l’Université de Bordeaux I, 2000.

[30] B. G. Rax, C. I. Lee, A. H. Johnston and C. E. Barnes. “Total dose and proton damage in optocouplers”, *In IEEE Transactions on Nuclear Science*, vol. 43, pp. 3167-3173, 1996.

[31] E. G. Stassinopoulos, G.J. Brucker, P. Calvel, A. Baiget, C. Peyrotte and R. Gaillard. “Charge generation by heavy ions in power MOSFETs, burnout space predictions and dynamic SEB sensitivity”, *In IEEE Transactions On Nuclear Science*, vol. 39, pp. 1704-1711, 1992.

[32] J. Ziegler. “Effect of Cosmic Rays on Computer Memories”, *Science* , Vol. 206, pp. 776-788, 1979.