Fault Injection Attacks on Cryptographic Devices: Theory, Practice and Countermeasures

Alessandro Barenghi
Politecnico di Milano
Milan, Italy
barenghi@elet.polimi.it

Luca Breveglieri
Politecnico di Milano
Milan, Italy
luca.breveglieri@polimi.it

Israel Koren
University of Massachussets
Amherst, MA, USA
koren@ecs.umass.edu

David Naccache
Ecole Normale Supérieure
Paris, France
naccache@ens.fr

***Abstract*—Implementations of cryptographic algorithms continue to proliferate in consumer products due to the increasing demand for secure transmission of confidential information. Although the current standard cryptographic algorithms proved to withstand exhaustive attacks, their hardware and software implementations have exhibited vulnerabilities to side channel attacks, e.g., power analysis and fault injection attacks. This paper focuses on fault injection attacks that have been shown to require inexpensive equipment and a short amount of time. The paper provides a comprehensive description of these attacks on cryptographic devices and the countermeasures that have been developed against them.**

After a brief review of the widely used cryptographic algorithms, we classify the currently known fault injection attacks into low cost ones (which a single attacker with a modest budget can mount) and high cost ones (requiring highly skilled attackers with a large budget). We then list the attacks that have been developed for the important and commonly used ciphers and indicate which ones have been successfully used in practice. The known countermeasures against the previously described fault injection attacks are then presented, including intrusion detection and fault detection. We conclude the survey with a discussion on the interaction between fault injection attacks (and the corresponding countermeasures) and power analysis attacks.

I. Introduction

Cryptographic algorithms are being employed in an increasing number of consumer products, e.g., smart-cards, cellphones and set-top boxes, to meet their high security requirements. Many of these products require high-speed operation and include, therefore, dedicated hardware encryption and/or decryption circuits for the cryptographic algorithm. Unfortunately, these hardware circuits, unless carefully designed, may result in security vulnerabilities

The cryptographic algorithms (also called ciphers) that are being implemented, are designed so that they are difficult to break mathematically [1]. To obtain the secret key, which allows the decryption of encrypted information, an attacker must perform a brute force analysis that requires a prohibitively large number of experiments. For the most commonly used cryptographic algorithms, there is no known methodology to significantly reduce the secret key search space.

However, it has been shown that secret information (such as the key of the encryption algorithm) can leak through side channels. Examples of such side channels are the time needed to perform the encryption or the power consumed by the device implementing the encryption algorithm. Timing and power side channel attacks are based on the fact that the individual computation steps that are needed during the encryption are dependent on the bits of the secret key and thus, the time needed for these steps and the power consumed by them are directly correlated to the secret key bits. These attacks have proven to be effective and incur a relatively low cost. Furthermore, once a side-channel attack technique has been developed and made public, high technical skills and/or expensive equipment are not required to apply it in practice.

Another type of a side-channel attack is based on the electromagnetic radiation that emanates from the individual circuits executing the encryption/decryption. This attack is even more dangerous than power analysis since it can be performed at some distance from the circuit, and a direct contact with the circuit, that can be detected by suitable sensors, is not necessary.

Side-channel attacks have become a major industrial concern in the last fifteen years and resulted in an intensive research effort to develop suitable countermeasures that can defeat the attacks, or at least make them more difficult and time consuming to perform. Many different types of countermeasures have been developed, including: restructuring of the algorithm, shielding of the device, randomizing the computation, using power independent implementation, and others.

A different type of side-channel attack that proved to be very effective, is realized through the injection of deliberate (malicious) faults into a cryptographic device and the observation of the corresponding erroneous outputs [2], [3]. Using this type of attack and analyzing the outputs of the cryptographic device, called differential fault analysis (DFA) [4], the number of experiments needed to obtain the bits of the secret key can be drastically reduced. This kind of active side-channel attacks (in contrast to the previously described passive ones) has been in the last decade the subject of intense and expanding research, as it has been demonstrated to be highly effective [5]–[7].

Thus, incorporating countermeasures against fault injection attacks into cryptographic devices through some form of fault detection and possibly tolerance, is necessary for security purposes as well as for the more common objective of data integrity [8]–[10].

We start this survey paper with a brief overview of the two

important classes of ciphers, namely symmetric (or private) key and asymmetric (or public) key. We then explain the general approach to fault injection based attacks and describe the DFA technique. Next we present the state-of-the-art in fault injection attacks that can be mounted against symmetric and asymmetric key ciphers, and we illustrate them using two ciphers of each type. Finally, we present the currently known countermeasures against fault injection attacks including algorithmic changes, sensors and shields, and fault detection or correction techniques. A comprehensive list of references completes the survey and provides pointers to the main literature contributions to this rapidly evolving scientific and technological topic.

II. Description of Cryptographic Algorithms

Cryptographic algorithms use secret keys for encrypting the given data (known as *plaintext*) thus generating a *ciphertext*, and for decrypting the ciphertext to reconstruct the original plaintext. The keys used for the encryption and decryption steps can be either identical (or trivially related), leading to what are known as *symmetric key* ciphers, or they can be different, leading to what are known as *asymmetric key* (or *public key*) ciphers. Symmetric key ciphers have simpler, and therefore faster, encryption and decryption processes compared to asymmetric key ciphers. The main weakness of symmetric ciphers is the shared secret key which may be subject to discovery by an adversary, and therefore, must be changed periodically. The generation of new keys, commonly carried out using a pseudo random number generator, must be very carefully executed because, unless properly initialized, such generators may result in easy to discover keys. The new keys must then be distributed securely, preferably by using a more secure (but more computationally intensive) asymmetric cipher.

Symmetric key ciphers can be either *block ciphers* which encrypt a block consisting of a fixed number of plaintext bits at the same time, or *stream ciphers* which encrypt one bit at a time. Stream ciphers are not as frequently used as block ciphers, but still play a role in certain applications as we will see below.

Some well-known block ciphers include the Data Encryption Standard (DES) and the more recent Advanced Encryption Standard (AES). DES uses 64-bit plaintext blocks and a 56-bit key, while AES uses 128-bit blocks and keys of size between 128 and 256 bits. Longer secret keys are obviously more secure, but the size of the data block also plays a role in the cipher's security. For example, smaller blocks may allow frequency-based attacks, such as relying on the higher frequency of the letter "e" in an English text.

The encryption process for symmetric ciphers is designed with the goal of scrambling the plaintext as much as possible. This is done by repeating a computationally simple series of steps (called a *round*) several times to achieve the desired scrambling. This process must still be reversible so that the reverse process followed during decryption can generate the original plaintext using the same secret key.

In contrast, asymmetric key (public key) ciphers allow users to communicate securely without having access to a shared secret key. Here, the sender and recipient each have two cryptographic keys called the public key and the private key. The private key is kept secret, while the public key may be widely distributed. In a way, one of the two keys can be used to "lock" a safe; while the other key is needed to unlock it. If a sender encrypts a message using the recipient's public key, only the recipient can decrypt it using the corresponding private key.

Another noteworthy application of public key ciphers is sender authentication: the sender encrypts a message with her own private key. By managing to decrypt the message using the sender's public key, the recipient is assured that the sender (and no one else) has generated the message.

A. Simple symmetric key stream cipher: SNOW 3G

The simplest ciphers in use nowadays are stream ciphers that generate a pseudo-random stream of key bits that is bitwise XOR-ed with the plaintext to generate the ciphertext. The main advantage of stream ciphers is that they can be implemented using a small hardware circuit and can operate at a high speed, making them extremely suitable for power constrained devices such as mobile phones.

As an example of a lightweight stream cipher, we describe below SNOW 3G that has been chosen by the 3GPP committee (of the European Telecommunication Standards Institute) as one of the data confidentiality standards for phone calls [11].

SNOW 3G is the third instance of the SNOW cipher family proposed in [12]. It improves its predecessor - the SNOW 2.0 cipher, which was included in the ISO/IEC CD 18033-4 [13] standard, by enhancing robustness with respect to algebraic cryptanalysis. The cipher generates a sequence of 32-bit words from a 128-bit key and a 128-bit initialization variable following the scheme depicted in Figure 1. The circuit includes a shift register (composed of sixteen 32-bit wide elements, $s_{15} \dots s_0$) and a finite-state machine (composed of three 32-bit registers, R_1, R_2 and R_3) that is included to render the output highly non-linear. The $\boxplus$ symbol denotes addition modulo 2^{32}, while the $\oplus$ symbol denotes a 32-bit bitwise XOR. The two boxes marked S_1 and S_2 are lookup tables implementing nonlinear mappings of the input four bytes into different output four bytes, while a and a^{-1} denote multiplications over $\mathbb{Z}_{2^{32}}$ by fixed coefficients.

The cipher is initialized by filling the state and the three registers R_1, R_2 and R_3 with material from the key and the initialization vector. It is then updated for a number of cycles with no output produced. After this initialization phase, the circuit outputs 32 bits of keystream every cycle while updating the internal state as shown in the block diagram.

B. Symmetric key block ciphers: DES and AES

The more complex block ciphers like DES and AES can provide high security levels when large amounts of data must be encrypted and the available computing system is not overly constrained.

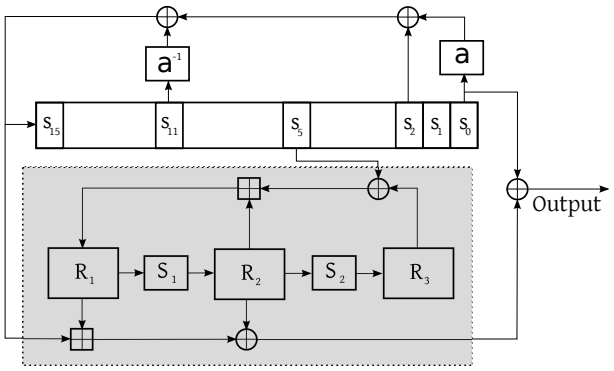

Figure 1. Snow 3G Block Diagram

Two crucial properties that every good block cipher must have are called *confusion* and *diffusion*. Confusion refers to establishing a complex relationship between the ciphertext and the key, while diffusion implies that any natural redundancy that exists in the plaintext (and can be exploited by an adversary) will dissipate in the ciphertext.

DES has been the first official standard cipher for commercial purposes [14]. In DES, most of the *confusion* is provided by the SBoxes: lookup tables representing nonlinear functions which are applied repeatedly to the input, while bitwise expansions and permutations provide the required *diffusion*.

In 1999, a specially designed circuit was successful in breaking DES in less than 24 hours [15], thus proving that the security provided by a 56-bit key is insufficient. Consequently, a newer standard (the Advanced Encryption Standard (AES)) has been established in 2002 [16], [17] with key size between 128 to 256 bits. The use of DES is however, still widespread either in its original form or, more frequently, in its more secure variation called Triple DES. Triple DES applies DES three times with different keys, and offers as a result a higher level of security. One variation uses 3 different keys for a total of 168 bits instead of 56, while another variation uses only 2 of them (112 bits in total).

The new standard block cipher, AES, is widely used and is now part of the IEEE P1619 standard for data encryption [18] and of the IEEE 802.11i standard for network communication [19]. AES is realized as a sequence of substitutions and permutations on a 128-bit plaintext, interleaved with the addition of the key through bitwise XOR. These operations are organized in so called *rounds* and the number of these rounds, denoted by N_r, depends on the length of the key, namely, $N_r = 10$, 12, or 14 for a 128, 192 or 256-bit AES key, respectively. The 128-bit data is usually represented as a 4×4 matrix of bytes called the *state* of the cipher, and is denoted by S with the byte elements $s_{i,j}$ ($0 \leq i, j \leq 3$). The state S is modified during each encryption round, until the final 128-bit ciphertext is produced.

Every round of the encryption process consists of the following four steps:

1) SubBytes - each state byte undergoes independently (of other bytes) a non-linear substitution of the form $T(s_{i,j}^{-1})$. Due to the complexity of this transformation, the 256 possible outcomes of this transformation are commonly pre-computed and stored in an 256×8 bits lookup table called S-Box. The non-linear function tabulated in the S-Box has been chosen in such a way that the distribution of the output bytes is robust against statistical attacks, i.e., small differences in the input will map to an arbitrary difference in the output. This property was explicitly required since the DES proved to be vulnerable to this type of attacks.
2) ShiftRows - the bytes of the first, second, third and fourth rows of the state matrix are rotated by 0, 1, 2 and 3 bytes, respectively. The state after this step is

$$S = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{bmatrix} \quad (1)$$

3) MixColumns - the four bytes in each column are used to generate four new bytes through linear transformations, as shown below ($j = 0, 1, 2, 3$)

$$\begin{aligned} s_{0,j} &= (\alpha \otimes s_{0,j}) \oplus (\beta \otimes s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s_{1,j} &= s_{0,j} \oplus (\alpha \otimes s_{1,j}) \oplus (\beta \otimes s_{2,j}) \oplus s_{3,j} \\ s_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (\alpha \otimes s_{2,j}) \oplus (\beta \otimes s_{3,j}) \\ s_{3,j} &= (\beta \otimes s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (\alpha \otimes s_{3,j}) \end{aligned} \quad (2)$$

where $\alpha = x$ (or 02 in hexadecimal notation), $\beta = x+1$ (or 03 in hexadecimal notation), $\otimes$ and $\oplus$ are the modulo 2 multiply and add operations, respectively, of the polynomial representations of the state bytes, and the α and β coefficients performed modulo the generator (irreducible) polynomial of AES which is $g(x) = x^8+x^4+x^3+x+1$. Polynomial presentations of binary numbers and operations modulo a given generator polynomial are described in [20].
4) AddRoundKey - the round sub-key is added through bit-wise XOR to the state. Separate round sub-keys are generated using a key schedule process.

All four steps are performed at each round except the last one, where the MixColumns step is omitted. In addition, prior to the first round, the first sub-key is added to the original plaintext.

The individual round sub-keys are generated using a key schedule/expansion procedure that computes the 128-bit round keys, k_j, given the input key k that consists of l 32-bit words where l is equal to 4, 6 or 8. Thus, the key schedule process generates a total of $4(N_r+1)$ 32-bit words organized as a linear array denoted by $W[0, \ldots, 4(N_r+1)-1]$. The first l words of W are loaded with the user supplied key. The remaining words are generated according to Algorithm II.1 where $RCON$ is an array of predetermined constants, $SubByte$ is the byte substitution of AES, and $<<< 8$ denotes a rotation of a word to the left by 8 bit positions.

Algorithm II.1: The AES key schedule.

Input: k: secret key, l: key length in words, N_r: number of rounds
Output: W: array containing round keys

```
begin
    for i = l to 4(N_r + 1) − 1 do
        if i ≡ 0 mod l then
            W[i] = W[i − l] ⊕ SubByte[W[i − 1] <<< 8]) ⊕ RCON[i/l]
        else if l = 8 and i ≡ 4 mod l then
            W[i] = W[i − l] ⊕ S[W[i − 1]]
        else
            W[i] = W[i − l] ⊕ W[i − 1]
    return W
end
```

C. Asymmetric key cipher: RSA

Although a number of asymmetric ciphers are in use, the most well-known and widely deployed public key cipher is the RSA algorithm named after its three inventors Rivest, Shamir and Adleman [21].

To employ the RSA cipher one must first generate a pair of keys, each one of which is able to decrypt what has been encrypted by the other one. One of these two keys (henceforth called the *public key*) will be publicly disclosed to everyone, thus allowing any person to encrypt a message and send it to the owner of the key pair. This owner is the only person able to decrypt the message, since the second key (which is referred as the *private key* and is never disclosed), is the only one that will enable decrypting the message encrypted with the public key.

The process required to generate the key pair consists of the following steps:

1) Select two large prime numbers p and q and calculate their product $n = pq$.
2) Select a small odd integer e that is relatively prime to
$$\varphi(n) = (p-1)(q-1)$$
Two numbers (not necessarily primes) are said to be relatively prime if their only common factor is 1. For example, 6 and 25 are relatively prime although none of them is a prime number.
3) Find the integer d that satisfies the relationship
$$de = 1 \mod \varphi(n)$$
d is thus the inverse of e if the calculations are done $\mod \varphi(n)$.

(e, n) constitutes the public key, while (d, n) will serve as the secret private key. The security provided by RSA depends on the difficulty of factoring the large integer n into its two prime factors. Since there is no known polynomial time algorithm to factor a composite number, it is sufficient to select two very large prime numbers p and q to construct the modulus n in order to make the key derivation extremely difficult. To make the factoring time prohibitively large, each of the prime numbers p and q must have at least hundreds of bits. The current practice, after massive computational efforts proved the feasibility of factoring a 768-bit number, is that the required length of the moduli, for civilian and military applications, are 1024 and 2048 bits, respectively. For confidential data that must be preserved for long time durations, 4096-bit moduli are recommended.

If a person wishes to send a message m to the owner of a key pair, he uses the public key to compute the ciphertext as $c = m^e \mod n$. Notice that this encryption scheme makes it necessary to restrict the message size of m so that is satisfies $0 < m < n$. Upon receiving the encrypted message c, the owner of the key pair will decrypt it using his private key by calculating $c^d \mod n = m^{de} \mod n = m \mod n$.

The most complex operation required when performing RSA encryption and decryption is exponentiation modulo n. A number of techniques are being used to reduce the complexity of this operation. First, since the only restriction on the choice of the exponent e is that it is relatively prime with $\phi(n)$, a small prime number is selected, e.g., 3, 17 or 65537, thus reducing the complexity of encryption. Another speedup in the exponentiation is achieved by employing a multiplication technique known as Montgomery multiplication [22] that greatly simplifies the required modular reductions. To speed up decryption, for which the private exponent d can not be chosen arbitrarily, the Chinese Remainder Theorem is used allowing to perform the computations modulo p and q separately (and in parallel) and then recombine the two results to obtain the final result modulo n. The separate computations are performed modulo smaller numbers (half the size of n), requiring less time and a smaller circuit.

D. Novel asymmetric algorithm: ECC

Elliptic curve ciphers are based on the use of points on a cubic curve P over a finite field $GF(p)$ where p is a prime number. The main idea is that it is possible to define an operation on the set points of an elliptic curve that behaves exactly like addition (i.e., it has a neutral element, is commutative and it is possible to find an inverse for every point of the curve). By using points on a curve P it is possible to obtain a trapdoor function akin to the one used in common discrete logarithm cryptosystems. This trapdoor function relies on the ease of adding a point of the curve k times to itself, a procedure known as point-scalar multiplication. Given a point $Q = kP$, it is computationally very difficult to find k provided that the curve has a sufficiently large number of points. A key feature of these ciphers is the choice the curve. Thus, it is necessary to select a curve that has a sufficiently large number of points and has no special properties which could reduce the difficulty of solving the so called Elliptic Curve Discrete Logarithm Problem.

The advantage of elliptic curve ciphers is the smaller size of the numbers involved in the computations. Typically, safe sizes for the prime p so that the curve will have a sufficient number of points range from 160 to 250 bits. This reduction in operand

size is especially important for computationally constrained devices such as smart cards [23], or when a large number of encryptions/decryptions is expected, such as in protecting Domain Name Systems (DNS) transactions [24].

III. Fault Injection Techniques

The fault injection techniques that have been developed in order to alter maliciously the correct functioning of a computing device currently include: variations in the power supply voltage level, injection of irregularities in the clock signal, radiation or EM disturbances, overheating the device or exposing it to intense light. Since the range of these techniques is wide and getting wider, we first classify the methodologies used for the attack according to their cost. We will also point out in this section the degree of technical skill and knowledge of the implementation required to perform the injection, and characterize the achievable faults with respect to their (temporal and spatial) precision and effectiveness. This classification would allow circuit designers to determine the possible threats to their secure implementation depending on the skill and budget of the perceived attackers.

A. Low cost fault injection techniques

We consider as *low cost* the injection methods requiring less than $3000 of equipment in order to set up the attack. This cost is well within the means of a single motivated attacker, and thus, these fault injection techniques should be considered as a serious threat to the implementations of secure chips that may be subjected to them.

The first fault injection technique we describe is the underpowering of the device. Through running the chip with a depleted power supply, the attacker is able to insert transient faults starting from single bit errors and becoming more invasive as the supply voltage gets lower. Since this technique does not require precise timing, the faults tend to occur uniformly throughout the computation, thus requiring the attacker to be able to discard results that are not fit to lead an attack. This methodology, reported to be effective on large integrated circuits such as the ARM9 processor [25], [26], as well as on small ASIC implementations of the ciphers [27], [28], results in delaying the correct set-up for the logic gates of the circuit. The voltage underfeeding, achieved by employing a precise power supply unit, requires the attacker to be able to tap into the power supply line of the device and connect his power supply unit. This requires only basic skills and can be easily achieved in practice without leaving evidence of tampering. Moreover, no knowledge of the implementation details of the device is needed.

One refinement of the aforementioned technique is the injection of well-timed power spikes or temporary brown-outs into the supply line of the circuit. Using this technique, it is possible to skip the execution of a single instruction in a software implementation of the cipher by reducing the feeding voltage for the duration of a single clock cycle. The authors of [29] report a successful application of this technique to an 8-bit microcontroller and over-voltage spikes have been successfully applied to de-packaged RFID tags in [30]. In order to inject a timed voltage lapse the attacker needs a custom circuit capable of dropping the feeding voltage below a certain threshold. This custom circuit should be supplied with the same clock that drives the microcontroller allowing it to correctly time the injection of the spike. The temporal precision of the fault injection is directly dependent on the accuracy of the voltage drop both in terms of duration and synchronization with the target device. The difficulties in applying this technique increase with the clock rate of the attacked circuit due to the mutual induction of the feeding line.

Another viable option for an attacker is to tamper with the clock signal. For example, it is possible to shorten the length of a single cycle through forcing a premature toggling of the clock signal. Such shortening, according to [31], causes multiple errors corrupting a stored byte or multiple bytes. These errors are transient and thus it is possible to induce such faults without leaving any tamper evidence. To alter the length of the clock cycle, the attacker needs to have direct control over the clock line, which is the typical case when smart cards are targeted [31]. It is not possible to attack chips that generate their own clock signal since disconnecting the clock line from the circuit is difficult. The attack mentioned in [31] involves a modified smart card reader that is capable of shortening the duration of a specific clock cycle through either forcing the raising edge to occur earlier or delaying the falling edge, depending on the kind of driven smart card. The modification to the card reader is not trivial but can still be performed without any special and expensive tools. Clock alteration techniques are hindered by the need to supply a regular clock within the working range of the device, while retaining the ability of altering a single clock edge. This implies that the equipment inducing the alteration must be working at a higher clock frequency than the attacked device, and this is intrinsically more difficult as the target device working frequency increases.

Another possibility for an attacker is to alter the environmental conditions, for instance, by causing the temperature to rise. A temperature rise has been reported to cause multiple multi-bit errors in DRAM memories [32]. The authors report a thermal fault injection attack against the DRAM chips of a common desktop computer. The reported number of flipped bits is around 10 per 32-bit word, when the working temperature of the DRAM is brought up to $100\,^{\circ}\mathrm{C}$. The number of faulty words is also reported to be in the range of tenths. The equipment used included a 50W light bulb and a thermometer. The level of heating was tuned through modifying the distance from the chip. The setup thus requires minimal technical knowledge, and the equipment is readily available. One drawback of this technique is that it tends to cause invasive faults in sensitive devices. Another downside is that the circuit may be destroyed through excessive heating.

A practical way to induce faults without having to tap into the device is to cause strong EM disturbances near it. The Eddy currents induced in the circuit by strong EM pulses

cause temporary alterations of the level of a signal, which may be recorded by a latch. Since the EM pulse is affecting uniformly the entire attacked device, it is necessary to shield the components which should not be subject to faults using a properly grounded metal plate or mesh. This technique has been shown to be effective against an 8-bit microcontroller [33] by employing, as a source of EM disturbances, a spark generator and placing it very close to the attacked chip. The authors have also demonstrated that a more efficient fault injection can be achieved by first removing the plastic package of the chip. Their spark generator consisted of a simple piezoelectric gas lighter that was held directly above the device. All the parts of the circuit which did not need to be disturbed were properly shielded through grounded aluminium plates. The above technique can not be applied to chips that have a grounded metal packaging (usually as a heat sink) that acts as an EM shield, unless the chip is decapsulated, adding a step that requires an uncommon technical skill. Still, decapsulation can be performed with low cost equipment (nitric acid and common glassware), thus not raising the cost of the attack considerably.

Assuming the attacker is able to successfully decapsulate a chip, he can perform fault injection attacks by illuminating the die with a high energy light source such as an UV lamp or a camera flash. The strong radiation directed at the silicon surface can cause the blanking of erasable EPROM and FLASH memory cells where constants needed for an algorithm execution are kept (e.g., the AES S-Boxes). Depending on the duration of the radiation process, the authors of [34] report a progressive blanking of all the memory cells as well as resetting the internal protection fuses of the microcontroller that was targeted. The authors also show that it is possible to selectively wipe out a part of the stored data in the memory by exposing only a part of the die to UV radiation. The required equipment consists only of an UV lamp that is placed closely to the exposed die. To shield the circuit parts which need not be exposed, they can be covered with a readily available UV-resistant dye. This technique is applicable only if the memory cells have not been covered by a metallic layer. For example, metal wires placed above the memory cells may provide a shield against radiation.

B. High cost fault injection techniques

A class of threats which cannot be ignored if the attackers have access to a larger budget (above the aforementioned $3000 and up to millions of dollars) includes fault injection techniques that rely on having a direct access to the silicon die and the ability to target individual circuits in a very precise manner. These techniques, albeit leaving evident traces of tampering, are very powerful and can considerably increase the probability of a successful attack.

A simple example of these techniques is based on the use of a strong and precisely focused light beam to induce alterations in the behavior of one or more logic gates of a circuit. A strong radiation of a transistor may form a temporary conductive channel in the dielectric, which, in turn, may cause the logic circuit to switch state in a precise and controlled manner (provided that the used etching technology is not too fine). For instance, it is possible, through targeting one of the transistors of an SRAM cell (in the memory of a microcontroller), to flip it up or down at will [35], [36]. In order to obtain a sufficiently focused light beam from a camera flash, a precision microscope must be used. The main limitation of this technique is the non-polarized nature of the white light emitted by the camera flash resulting in scattering of the light when focused through non-perfect lenses. Moreover, it is no longer possible to hit a single SRAM cell with the current etching technologies, since the width of the gate dielectric is now more than 10 times smaller than the shortest wavelength of visible light.

The most straightforward refinement of the previous technique is to employ a laser beam instead of a camera flash. The injected fault model is similar to that obtained when using a concentrated light beam [35], except for the fact that the laser beam is capable of always inducing faults. Near Infra-Red (NIR) lasers can also radiate the silicon die from the back allowing the attacker to hit circuits which are in the bottom layers of the chip although with a lower precision since the silicon substrate scatters the beam (a reduction in the scattering may be obtained by applying anti-reflective coatings). It is worth noting that the inability to hit only a single bit memory cell (due to the size of the concentrated beam) does not necessarily imply inability to inject a single bit fault. In [37], Agoyan *et al.* demonstrated how to inject single bit fault in a reproducible way, despite the optical precision of the equipment was not able to target the smallest features of the target chip.

Currently, commercially available fault injection workstations are composed of a laser emitter, focusing lens and a placement surface with stepper motors to achieve a very precise targeting of the beam. The main foreseen limitation of this fault injection technique is the fact that it is not possible to achieve sub-wavelength precision thus limiting the smallest number of gates hit by the radiation depending on the etching technology and the laser wavelength.

The most accurate and powerful fault injection technique uses Focused Ion Beam (FIB) that enables an attacker to arbitrarily modify the structure of a circuit, reconstruct missing buses, cut existing wires, mill through layers and rebuild them. Such FIB workstations are commonly used to debug and patch chip prototypes, or to reverse engineer unknown designs through adding probing wires to otherwise inaccessible parts of the circuit. For instance, [38] reports a successful reconstruction of an entire read bus of a memory containing a cryptographic key without damaging the contents of the memory. State of the art FIBs can operate at a precision of 2.5 nm, i.e., less than a tenth of the gate width of the smallest etchable transistor. FIB workstations require very expensive consumables and a strong technical background to fully exploit their capabilities. The only limit to the FIB technology is the diameter of the atoms whose ions are used as a scalpel. Currently, the most common choice is Gallium, which sets the

lower bound to roughly 0.135 nm.

Table I summarizes the important characteristics of the previously described fault injection techniques.

IV. Fault Injection Attacks

The variety of known attacks is already large and keeps on growing, as several new successful ones are demonstrated yearly. We first describe a few simple fault attacks on the SNOW 3G cipher to illustrate the differential fault analysis (DFA) methodology and show some practical implementations thereof. Then, we list and discuss several more complex fault attacks targeting the commonly used AES cipher, and several targeting RSA, exploiting different vulnerabilities. Next, the few available attacks targeting ECC are briefly presented, with some comparison to RSA.

A. Simple attacks on 3G-SNOW

A fault attack against SNOW 3G has been proposed by Debraize et al [39]. Their technique enhances the one proposed in [40] against SNOW 2.0 proving that the attack can be successfully extended despite the tweaks applied to the cipher to raise its security level. In this attack, the cryptanalysis is based on viewing the output of the cipher as a nonlinear function of the inner state, and the shift register is seen as a generator of a series of outputs which are dependent on the state of the three FSM registers. The proposed approach assumes that the attacker is able to introduce a fault into a specific 32-bit cell of the shift register, without a precise control on the timing of the fault. Since the target of the fault is a part of a shift register, the fault model may be expressed also through its dual, i.e., a fault injected with a clock accurate timing but without proper control on the location of the injected fault in the shift register. After injecting the fault, the attacker analyzes the faulty output differences with respect to a correct key stream and can deduce the position of the fault in the shift register based on the position of the 32-bit words which are different in the two key streams. Once the position of the fault has been determined, the attack continues by constructing an equation expressing the difference in the inner state as an unknown, while the difference between the fault and correct output word is the known term. After collecting a sufficient number of equations it is possible to remove the terms dependent on the registers and, in the best case, obtain a set of linear equations which can be solved through common Gaussian elimination. The authors have also proposed an alternative to Gaussian elimination (for the case where a complete linear set of equations can not be obtained using the above technique) that is based on Gröbner bases. Through decomposition in a Gröbner basis of a equation system it is possible to solve a small set of nonlinear equations, but since the algorithm has exponential complexity, the problem may become computationally intractable. The authors report that their attack was successful with as few as 22 injected faults in the inner state, regardless of the value of the 32-bit register after the fault injection (i.e., no assumptions were made on how many bits were flipped or their positions).

B. Attacks on AES

This section provides an insight into the most common fault attacks on AES. These attacks attempt to exploit the byte-wise processing of the state by inserting either single bit or single byte faults during the computation.

A simple and straightforward attack on the AES cipher has been proposed by Bloemer *et al.* in [41] and aims to change a single bit right after the first key addition. The objective is to reset a single bit in the internal state $S^{(0)}$ (in general, $S^{(i)}$ denotes the state at the beginning of the i-th round) and observe whether the value of the ciphertext has changed. If the ciphertext has either changed or was detected as faulty by a fault detection circuit, the attacker knows that the correct value of the bit is 1, otherwise it is 0. Since the altered bit is the result of a xor between the known plaintext and the key, the attacker is able to recover the key one bit at a time. Although this attack can in principle, recover any length of the cipher key, it has been deemed practically infeasible due to the very precise timing required of the fault injection and the strict requirement on the position of the injected fault.

The most straightforward attack, among the practically feasible ones, has been presented by Giraud in [42] and targets directly the state $S^{(i)}$ during the last round. The attack assumes that the injected fault only alters a single bit of the state, prior to the last SubBytes operation. The modified bit then propagates through the last round and results in a single byte corruption in the computed ciphertext. Once the attacker obtains a pair of correct and wrong ciphertext c and $\tilde{c}$, respectively, he can reduce the number of possible key bytes employed to encrypt the corrupted byte by inverting the effect of the last round and checking whether the difference between the two values (obtained with a single byte key hypothesis) is a single byte prior to the SubBytes operation. This kind of attack is thus applicable only to the last AES round that does not include the MixColumns operation, and therefore, a single byte difference in the state will not spread to other bytes.
A limitation to the practical instantiation of this attack is the requirement of a very strict time frame in which the fault must be injected. The authors of [42] were able to achieve the required precise timing while attacking an 8-bit smartcard, by focusing the light emitted by a camera flash through a microscope. Their apparatus was synchronized to the smartcard clock and was able to inject correctly timed faults into the device. Note that the attacker can always determine whether the injected fault has hit the correct point in the circuit, since the fault would corrupt only a single byte.

The attack by Giraud has been successfully extended by Barenghi *et al.* in [43] allowing the exploitation of a single bit fault corrupting the state even in a regular round of the cipher. The diffusion of the fault caused by the MixColumns operation may be coped with by hypothesizing a larger part of the key (namely, a whole 32-bit word) and trying all the possible hypotheses. This is still computationally feasible even with a common desktop. This extended attack is able to reconstruct the full key schedule, thus recovering all the round keys,

Table I
FAULT INJECTION TECHNIQUES SUMMARY

Technique	Accuracy [space]	Accuracy [time]	Technical skill	Cost	Hindered by technological advances	Requires knowledge of the implementation	Damage to the device
Underfeeding	high	none	basic	low	no	no	no
Clock glitch	low	high	moderate	low	yes	yes	no
EM Pulses	low	moderate	moderate	low	no	no	possibly
Heat	low	none	low	low	partial	yes	possibly
Power supply glitch	low	moderate	moderate	low	no	partial	no
Light Radiation	low	low	moderate	low	yes	no	yes
Light Pulse	moderate	moderate	moderate	moderate	yes	yes	possibly
Laser beam	high	high	high	high	yes	yes	possibly
Focused Ion Beam	complete	complete	very high	very high	yes	yes	yes

provided enough faulty ciphertexts are available. The authors of [43] report a practical application of this attack against a software implementation of AES running on an ARM926-based system.

Dusart *et al.* [44] have presented an attack on AES that is based on a more general fault model: it assumes that the injected fault alters the value of a single byte between the (N_r-1)-th and (N_r-2)-th rounds of the encryption primitive, where N_r is the total number of rounds. This attack is able to obtain the last round key, and relies on the key schedule properties to reconstruct the entire AES-128 cipher key. To exploit the injected fault, an hypothesis on a single word of the last round key is made in order to invert the last round and obtain the state right before the last MixColumns operation. After the removal of the last round, the algorithm checks if the key hypothesis made is compatible with a single byte difference in the state through inverting the MixColumns operation (which is linear with respect to the xor-based key addition) and checking whether the difference between the faulty and correct values of the state $S^{(N_r-2)}$ is a single byte. This method of paring down the candidate keys is quite efficient and yields a single candidate with as few as three faulty ciphertexts per key word, thus enabling an attacker to retrieve the complete AES 128-bit key with only 12 injected faults. This attack has been practically carried out against a hardware implementation of AES on a smartcard in [27] and against a software implementation running on an ARM926 system in [26].

It is possible to further generalize the fault model of the above attack to a faulty word in the same position assumed by Dusart *et al.* This extension, proposed by Moradi *et al.* in [45], considers the possible faults occurring in a single word through splitting them into two categories: the ones affecting all four bytes of a word and those that affect fewer than four bytes. For each category the authors provide a bound on how fast they were able to reduce the keyspace. They show that it is possible to recover the key with around 1500 faulty ciphertexts. This key recovery method assumes that the attacker knows to which of the two categories the fault belongs, since having a generic word sized fault, without any hypothesis on the structure, yields no information for the attacker. Moreover, it is impossible to distinguish a-posteriori if the injected fault complies with the model needed to perform the key extraction, thus insisting on the fault injection method to be reliable in terms of the kind of fault induced. Although relying on quite reasonable fault injection hypotheses, this attack has not yet been validated in practice.

Another possible extension of the attack presented by Dusart *et al.* has been proposed in [26] and aims at overcoming the limitation of the attack that allows to retrieve only the last round key. The main difficulty of retrieving a round key before the last one is due to the effect of the MixColumns operation which is present in all the rounds of the cipher except for the last, and provides full diffusion of the injected fault over the whole cipher state after two applications. The key to work around this issue is to compute the difference between the correct and faulty cipher state at the end of the N_r-1 round (thus eliminating the effect of the N_r-1 key), and then invert the effect of the MixColumns on the differential value (which is not difficult since MixColumns is linear with respect to the xor operation). After obtaining the difference between the faulty and the correct states right before the SubBytes step of the N_r-1 round, a guess is made on the correct value of a single word of the state, deriving the value of the faulty word from the difference. After obtaining both values it is possible to roll back the SubBytes operation, which was not possible while holding only differential information due to the non-linear nature of the SubBytes. The last step of the attack checks whether the predicted fault fits the fault model and consequently, discards the inner state hypothesis if it does not. This allows the attacker to fully recover the internal state of the cipher before adding the last key, thus enabling the recovery of this round key. In [26] the authors provide a practical validation of the proposed attack technique against a software implementation of the AES cipher.

A practically feasible attack worth mentioning is the one involving the complete blanking of the S-Box lookup tables of the cipher, achievable through resetting the memory where they are stored. This attack effectively reduces the whole AES cipher to the last AddRoundKey operation, which is performed on a known cipher state, i.e. the null values fetched from the blanked S-Box. This in turn allows the immediate recovery of the last round key by the attacker, although it does not allow to recover the other round keys thus limiting the effectiveness

of the attack to AES-128. This attack has been proved to be feasible on a number of microcontrollers where the S-Box was stored in the internal flash memory. The devices had to be decapsulated before being radiated with ultraviolet light in order to wipe clean the memory. Proper targeting of the memory locations which had to be blanked was achieved using a UV-resistant dye to cover the parts which needed to be protected.

A number of fault injection attacks targeting the key scheduling algorithm (employed to generate round keys from the user supplied key) have been developed. These attacks exploit the highly regular structure of the AES key schedule in order to infer bytes of the key through corrupting one or more bytes during the expansion of the last round key bits. In particular, the attacks proposed by Giraud *et al.* in [42] and by Chen *et al.* in [46] exploit a single byte corruption introduced after the key schedule procedure has been performed, and are thus able to obtain a precise fault that does not propagate to the keys which are derived from it. While this fault model is reasonable whenever the key schedule is precomputed and its result stored in some kind of permanent memory, it is not possible to attack AES implementations which perform key expansion on the fly.

In [47], Peacham *et al.* proposed an attack which takes into account not only a single fault injection in the cipher, but also its effect on the computation of the following parts of the key. This can be done by propagating the fault through the xor and the SubBytes steps since their structure is not dependent on the employed key. The fault model employed by the attack is a single 32-bit word corruption during the computation of the penultimate round key, thus it is employable also in the practical scenario where the key expansion is computed on the fly, even if the attacker is not able to inject a precise fault into a single byte. The authors of [47] describe a successful attack mounted using laser induced fault injection on a commercial grade secure implementation of AES, that did not include any countermeasures against fault attacks, and employed an all-at-once key scheduling strategy. This attack was further enhanced in [48] to reduce the number of faults required to deduce the entire last round key to four instead of more than 10.

Table II summarizes the key charachteristics of different fauly injection attacks on AES that were described in this section.

C. Attacks on RSA

Due to the asymmetric nature of the RSA cipher two kinds of attacks are possible. The first one attempts to recover either the factorization of the public modulus n or the secret exponent d. The second one tries to decrypt the ciphertext c with no knowledge of the secret key whatsoever. This section starts with a description of the former, which can only be applied during the phase that uses the secret exponent d. This can be either the decryption phase of a message that has been encrypted with the public key or the signature phase of a message sent by the private key owner for sender authentication purposes.

The first attack which has been proposed to factor the RSA secret modulus, and actually the very first fault attack technique to be developed, is the so called Bellcore attack [5]. This technique enables the attacker to factor the modulus n through inducing an error during the computation of the exponentiation phase of an RSA implemented using the Chinese Remainder Theorem.

Consider, for example, the signature phase where the signature s is computed as $s = m^d \bmod n$ using a CRT recombination of the two values $s_p = m^d \bmod p$ and $s_q = m^d \bmod q$. The recombination, denoted by $CRT(s_p, s_q)$, is accomplished using the so-called Garner method:

$$s = \big(s_p + p\,((s_q - s_p)(p^{-1} \bmod q) \bmod q)\big) \bmod n$$

The main benefit of this method is that it achieves significant time and area savings by performing the exponentiations with smaller exponents. Since $s_p = m^d \bmod p = m^{d \bmod (p-1)} \bmod p$ and $s_q = m^d \bmod q = m^{d \bmod (q-1)} \bmod q$, the exponents to be used are $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$ which are of order p and q, respectively, instead of n.

Unfortunately, this simpler way to compute the signature also yields an easy path for attackers. The main idea behind the Bellcore attack is to corrupt only one of the two computations, i.e., either s_p or s_q. If, for example, a fault is injected during the computation of s_q while the computation of s_p remains error free, the faulty result may be used to successfully factor the modulus n. Denoting the faulty value of s_q by $\widetilde{s_q} = s_q + \Delta$, we can rewrite the faulty result of the CRT recombination as $\widetilde{s} = CRT(s_p, \widetilde{s_q})$, which is equal to:

$$\widetilde{s} = s + p\,\big(\Delta(p^{-1} \bmod q) \bmod q\big) \bmod n$$

One can now compute the quantity $\widetilde{s} - s$ that shares the factor p with the modulus n. Therefore, it is possible to extract $p = \gcd(\widetilde{s} - s, n)$ efficiently using Euclid's Algorithm, thus factoring n.

Moreover, as shown in [7], the modulus factorization is feasible using only the message m and one faulty computation of the signature $\widetilde{s}$ by exploiting the knowledge of the public exponent e to calculate $p = \gcd(\widetilde{s}^e - m, n)$. This allows an attacker to factor the modulus even in the case the value of the correct signature s is not available.

The simplicity of the fault model assumed by this attack is the most important property of this technique: any random fault perturbing one part of the computation is able to break the cipher. A number of practical implementations of this attack have been attempted. For example, the authors of [49] have induced errors into a smartcard through voltage spikes injected into the power supply line of the device. Even with such a simple setup, the attack was successful in more than 90% of the attempts, further proving the serious threat posed by this attack technique.

The second way to attack RSA is to retrieve the private exponent d while the device is signing messages. This attack is applicable when the RSA implementation performs the exponentiation through a sequence of square-and-multiply

Table II
AES ATTACKS SUMMARY (KS IS THE KEY SCHEDULE)

Attack	Fault model	Spots usable faults	Fault Position	Cipher / KS	Max Key Length	Practically applied
[41]	Single bit	No	First ARK	Cipher	Any	No
[42]	Single bit	No	Last round	Cipher	128	No
[43]	Single bit	Yes	Last 3 rounds	Cipher	Any	Yes
[44]	Single byte	No	Last 2 rounds	Cipher	128	Yes
[45]	Single word	No	Last 2 rounds	Cipher	128	No
[26]	Single byte	Yes	Last 3 rounds	Cipher	Any	Yes
[34]	Whole SBox	Yes	SBox	Cipher	128	Yes
[46]	Exact byte	Yes	9th round key	KS	128	No
[42]	Single byte	No	9th round key	KS	128	No
[47]	Single word	Yes	9th round key	KS	128	Yes
[48]	Single word	Yes	9th round key	KS	128	No

steps. In this attack scenario the attacker has free access to the device and is allowed to choose arbitrary ciphertexts to be fed while injecting faults. It is also assumed that there is no limit to the number of fault injection experiments that the attacker can perform. This assumption restricts the technique to non destructive fault injections.

The key point of the secret exponent recovery attack, first proposed in [6], is to induce a number of faults during the signature process, with each fault leaking the value of a single bit of the exponent. Two types of injected faults can achieve the desired outcome. One is a single transient flip of a bit in d during the computation of the RSA signature. Another way to achieve an analogous effect is to induce a fault that will result in skipping the condition check which determines whether the current intermediate value must be multiplied by the base in a common left-to-right square and multiply exponentiation procedure. As a result of either fault, the corrupted signature $\widetilde{s}$ may assume one of the following two possible values: $\widetilde{s} = s^{d-2^i} \mod n$ or $\widetilde{s} = s^{d+2^i} \mod n$ depending on whether the value of the single bit in position i was flipped up or down. Consequently, either $s/\widetilde{s} \mod n$ or $\widetilde{s}/s \bmod n$ would be equal to $m^{2^i} \mod n$, where $i \in [0, v-1]$ and v is the bit size of the secret exponent d.

To simplify the attack, all the possible values of m^{2^i} (for $i \in [0, v-1]$) can be precomputed and stored in a lookup table, A. After a fault has been injected and the faulty signature $\widetilde{s}$ observed, the lookup table A is searched for a match with either $s/\widetilde{s} \mod n$ or $\widetilde{s}/s \bmod n$. If the first value matches an entry in the table the attacker knows that the i-th bit value is 1 and was changed to 0 by the fault, while if the second value produces a match, the original value of the i-th exponent bit was 0. This procedure can be iterated as necessary.

This attack can also be successful if the injected fault hits two bits of the secret key d. The main difference would be that a bigger lookup table would need to be prepared with v^2 instead of v entries.

To estimate the number of randomly positioned single-bit faults needed to discover the values of v unknown key bits define a random variable X counting the number of faults injected until all the bits have been hit. Assume that j key bits have already been hit and denote by X_j the random variable indicating the number of faults that should be injected in order to increase the number of bit hits to $j+1$. X_j follows a geometric distribution with parameter $\frac{v-j}{v}$. Therefore, the probability that after k injections a yet untouched (single) bit gets hit is given by $\text{Prob}(X_j = k) = \frac{v-j}{v}\left(\frac{j}{v}\right)^{k-1}$. The expected value of X_j is $\mathbb{E}[X_j] = \frac{v}{v-j}$. Since the random variable X satisfies $X = \sum_{j=0}^{v-1} X_j$, its expected value is $\mathbb{E}[X] = v\sum_{j=1}^{v} \frac{1}{j} \leq v\ln(v+1)$.

This implies that on the average $R = v\ln(v+1)$ single-bit faults should be injected in order to retrieve all the bits of the secret key. For example, if the RSA implementation uses a 1024-bit key, the attacker will need approximately $R = 1024\ln(1024) \cong 3083$ restarts of the device and successful fault injections in order to extract the entire secret exponent. As in any such attack, the attacker can stop the fault injections when a brute force search of the remaining key bits becomes feasible.

A variant of this attack has been proposed and applied in practice by Schmidt *et al.* in [29]. The authors caused the skipping of the squaring step in the square and multiply algorithm by introducing glitches into the clock signal of the attacked microcontroller. The employed methodology allows a very precise control of which instruction is skipped and the authors were therefore, successful in recovering all the bits of secret exponent d one bit at a time.

An example of applying the technique proposed by Bao [6] that is worth mentioning is the one reported in [50]. Yen *et al.* have used the technique under a safe error assumption. An error is injected into a single multiply operation during a square and multiply always algorithm and the attacker needs only to observe whether the device is behaving correctly. Any misbehavior (e.g., producing a faulty output value or no output at all) indicates to the attacker that the injected fault has hit a useful multiplication, implying that the bit (of d) driving that multiplication was equal to one. This technique has allowed to bypass all the countermeasures which were known at that time, since the actual faulty output was not needed.

A third way to attack the RSA cryptosystem is to devise a way to extract the e-th root of a number modulo n in a reasonable time. This has been shown to be feasible in [25], by exploiting the knowledge of another power of the same

Algorithm IV.1: e-TH ROOT EXTRACTION

Input: $e_1, e_2 \in \{1, \ldots, \varphi(n) - 1\}$, $e_1 \geq e_2$, $c_1 = m^{e_1} \bmod n$, $c_2 = m^{e_2} \bmod n$

Output: (m, n): either $(m, \perp)$ if the e-th root may be extracted, (p, q) if the modulus can be factored or $(\perp, \perp)$ otherwise

```
1  begin
2      τ ← gcd(c1, n)
3      if τ ≠ 1 then
4          return (τ, n/τ)
5      τ ← gcd(c2, n)
6      if τ ≠ 1 then
7          return (τ, n/τ)
8      if gcd(e1, e2) ≠ 1 then
9          return (⊥, ⊥)
10     γ1, γ2 ← c1, c2
11     ε1, ε2 ← e1, e2
       /* Integer division                    */
12     θ ← ⌊ε1/ε2⌋, ρ ← ε1 mod ε2
13     γ3 ← γ1 γ2^(−θ) mod n
14     while ρ ≠ 0 do
15         γ1, γ2 ← γ2, γ3
16         ε1, ε2 ← ε2, ε1 − θε2
           /* Integer division                */
17         θ ← ⌊ε1/ε2⌋, ρ ← ε1 mod ε2
18         γ3 ← γ1 γ2^(−θ) mod n
19     return (γ2, ⊥)
20 end
```

number. This technique can be used in order to recover the plaintext message without the need to obtain the secret key.

The fault model assumed by this attack is a modification to the value of the public exponent e, leading to two encryptions of the same message sharing the same modulus n. While this does never happen due to an incorrect generation of two public-private key pairs (otherwise the two key holders would be able to mutually read each other's messages), the encryption of a same message through exponentiation by two different public exponents e_1 and e_2 may be forced through proper fault injection.

Once the values of the two different ciphertext resulting from the encryption of the same message are obtained, it is possible to efficiently extract the e-th root by exploiting the following observation. Assuming that $e_1 > e_2$, the value of

$$m^{e_3} = (m^{e_1}) \cdot (m^{e_2})^{-1}$$

can be easily computed. The value of e_3 is lower than that of e_1 and it is possible to lower it further until it becomes lower than e_2. Notice that the attacker knows the values of e_1 and e_2 since they are public, so he knows exactly the value of e_3. In order to further lower the value of the exponent it is possible to compute the value of

$$m^{e_4} = (m^{e_2}) \cdot (m^{e_3})^{-1}$$

and repeat the procedure until either $e_{n+1} = e_n$ or $e_n = 1$. This procedure amounts to computing the greatest common divisor of e_1 and e_2 and employs the descending sequence of remainders as a pivot for the divisions among the two encrypted messages.

Algorithm IV.1 describes an efficient method to retrieve the plaintext of an RSA encryption using Euclid's greatest common divisor algorithm as a pivot to perform operations on the two known ciphertexts.

In order to compute the value of m^{-e_2} from m^{e_2} as required by the algorithm, it is necessary that $\gcd(m, n) = 1$; if this is not the case, it is possible to use m^{e_2} to factor n by simply computing their greatest common divisor. This implies that if the root extraction attack is not applicable, the system may be easily broken otherwise.

Algorithm IV.1 computes $\gcd(e_1, e_2)$ following Euclid's algorithm and calculates at each step the value of $m^{e_1 \bmod e_2} \bmod n$ using the values $c_1 = m^{e_1} \bmod n$ and $c_2 = m^{e_2} \bmod n$ (lines 13 and 18). This, under the assumption that e_1 and e_2 are co-prime, will lead to the computation of m^1.

If e_1 and e_2 are randomly chosen, a known result in number theory [51] states that, provided that two numbers are randomly chosen from a large enough range, the probability of them being co-prime approaches $\frac{6}{\pi^2} \approx 0.61$. This implies that, on the average, two fault injections will be sufficient to successfully extract the encrypted message.

We next estimate the computational complexity of the algorithm. Assuming that $e_1 \geq e_2$, the number of steps that Euclid's algorithm must perform is of the order of $\mathcal{O}(\log(e_1))$ that is equal to $\mathcal{O}(\log \varphi(n))$ (*Lamé's Theorem* [52]). Thus, taking into account the fact that the complexity of performing modular multiplication, exponentiation and inversion is $\mathcal{O}(\log^3 n)$, the complexity of the whole algorithm is $\mathcal{O}(\log^4 n)$ and therefore, tractable even for large values of n.

In order to employ Algorithm IV.1 in a fault attack scenario, the values of e_1 and e_2 must be known: this is equivalent to a precise fault injection assumption regarding the number of faulty exponent bits and their positions. This assumption may be relaxed, at the cost of computing the algorithm for each fault hypothesis and then checking if the recovered plaintext is the correct one through re-encrypting it and comparing it with the correct ciphertext.

Table III summarizes the properties of the attacks on RSA indicating the required precision of the fault injection and the practical applicability of the techniques. Many of the proposed attacks on RSA have been implemented and successfully mounted against real world devices, thus mandating proper incorporation of countermeasures into RSA implementations.

D. Attacks on ECC

Developing fault injection techniques to attack ECC-based ciphers proved to be more difficult than attacking RSA-based ciphers due to the higher complexity of the mathematical operations involved.

Table III
RSA ATTACKS SUMMARY

Attack	Fault model	Required Timing	Enc / Sig	Algorithm	Practically applied
[7]	Anything	Rough	Signature	CRT	Yes
[6]	Single bit flip	Precise	Signature	Plain	Yes
[50]	Safe error attack	Precise	Signature	CRT	No
[29]	Instruction skip	Precise	Signature	Plain	Yes
[25]	Few bits flip	Rough	Encryption	Plain	Yes

Most of the attacks on ECC that have been proposed exploit the structural similarity between the exponentiation through square and multiply used in RSA and the point-scalar multiplication through the double and add method used in ECC. Both ciphers have a common structure where, at each step, an operation (or a set of operations) is executed depending on the value of a single bit of the secret key.

This, in turn, implies that it is possible to apply the same bit flip and check attack which was suggested by Bao *et al.* [6] to recover the secret RSA exponent during the signature operation. Alongside the same attack strategies, also safe error attacks may be employed if a double and add always (the ECC's analog to the RSA's square and multiply always) algorithm is employed. A variation of the safe error attack, relying on the fact that all the computations on elliptic curves are performed on signed values, is the so-called sign flip attack [53]. Through flipping the sign bit of the exponent digit being operated on by the point multiplication algorithm, it is possible to successfully alter the final result, and recover which bit had been flipped.

In addition to the attack techniques that are similar to their RSA counterparts, there are attacks whose goal is lowering the security of the ECC cipher through changing the group of points on which it works. In [54] the authors propose injecting a fault into the base point which gets multiplied k times. This way, the point will no longer belong to the curve selected by the designer, but possibly to another one whose number of points is lower, thus making it possible to attempt a brute force attack on the scheme.

Another attack, directly targeted at the ECC structure, is described in [55], where the authors notice that a fault injected into the point coordinates during the scalar multiplication may move the point into a subgroup of the main group of curve points (called a *twist* of the curve), which has a smaller number of points. The authors show that their attack technique is able to successfully break curves standardized by both NIST [56] and IEEE [57] up to a security level equivalent to the one provided by the AES with a 128-bit key.

A different approach to attack the elliptic curve signature algorithm has been proposed in [58]. The key point is to alter one bit of the inputs of a single-word multiplication during the final multi-word multiplication employed to produce the signature value. The value of the wrong signature is exploited to retrieve one word of the secret key; the attack retrieves the full value of the secret key word by word. This fault attack technique has been extended to multiple bit faults in [59].

V. COUNTERMEASURES

This section describes the basic principles underlying the countermeasures against fault attacks: intrusion detection, algorithmic resistance, and error detection and possibly correction techniques, and will attempt to systematically classify the currently known countermeasures.

One approach to protect an implementation of a cryptographic algorithm against fault attacks relies on making the implementation physically inaccessible. This requires encasing the device in a tamper-proof box and including sensors to detect any attempted tampering with the device. This method has been applied in high-end cryptographic coprocessors such as the IBM 4764 [60].

Other, more cost-effective, approaches to protect against fault injection attacks modify the design of the cryptographic device to allow the detection of the injected faults. One such approach relies on duplicating the encryption or decryption process (using either hardware- or time-redundancy) and comparing the two results. This approach assumes that the injected faults are transient and will not manifest themselves in exactly the same time in these two executions. Although easy to apply, this approach may often impose a overhead too high to be practical. Another approach is based on error detection codes which usually require a smaller overhead compared to straightforward duplication, although possibly at the cost of a lower fault coverage. Thus, a trade-off between the fault coverage and the (hardware and/or time) overhead should be expected.

When using error detecting codes (EDCs) for detecting faults during the encryption/decryption process, check bits are first generated for the input, then, for each operation(s) that the data bits undergo, the check bits of the expected result are predicted. Periodically, check bits for the actual result are generated and compared to the predicted check bits: a fault is detected if the two sets do not match.

The validation checks can be scheduled at various granularities of the cipher, be it after every operation applied to the data, following each round, or only once at the end of the encryption process.

The first step, that of generating the check bits for the input, is straightforward. The non-trivial part is devising the prediction rules for the new values of the check bits following each transformation that the data bits undergo during the encryption/decryption process. The complexity of these prediction rules, combined with the frequency at which the comparison is made, determine the overhead of applying the

EDC, rather than duplication, as a protection against fault attacks.

A. Suggestions for 3G-SNOW

Providing fault attack protection in stream ciphers is particularly challenging due to the fact that these ciphers are commonly used in devices with strict timing and circuit size constraints. This in turn, excludes the use of high overhead techniques. A viable approach to providing moderate error checking capabilities is to employ nonlinear error detecting codes, like the ones described in [61]. These codes check the integrity of the state and provide a moderate error correction capabilities thus enabling a reliable functioning of the circuit even when under attack.

B. Protection options for AES and DES

The countermeasures that have been proposed to protect symmetric block ciphers mainly rely on the introduction of redundancy in the execution, either in the form of error detecting codes (information redundancy) or through duplicated execution (time or hardware redundancy). These schemes are similar to the conventional redundancy techniques that are described in [20].

With temporal duplication the encryption (or decryption) algorithm is executed twice on the same hardware, while with hardware (spatial) duplication the algorithm is executed on two separate circuits. In both cases, the two results are compared and any mismatch indicates an error which may be the result of a maliciously injected fault. These schemes work under the assumption that injecting identical faults during the two independent executions is extremely difficult. Temporal redundancy incurs a performance penalty while spatial redundancy results in a bigger circuit with higher power consumption.

A variation of the above duplication techniques can be applied if the system has a separate hardware unit or software program for executing the inverse of the cryptographic primitive that should be protected. For example, if a device that encrypts a symmetric block cipher also includes an implementation (in hardware or software) of the decryption algorithm, then the calculated ciphertext can be decrypted and if the result of this decryption matches the original plaintext, the ciphertext is considered fault-free and safe to output. In [62], the authors described the application of the above technique at different levels of granularity, i.e., checking against the inverse operation at the operation, round or full cipher level. The proposed scheme allows a precise and early identification of the step during which the fault occurred.

A technique to mask the high latency introduced when temporal duplication is applied to the execution of a cipher has been proposed in [63]. The authors developed a Dual Data Rate (DDR) architecture for AES, allowing to compute twice the same cipher with negligible time overhead by operating at double the frequency of the remaining circuit. The area overhead is reasonable, since the only parts which must be duplicated are the registers holding the cipher state.

Parity-based EDCs were proposed as an effective way to detect faults in the AES in [64], [65] and were previously shown to be useful also for DES [66].

Parity bits can be associated with entire 32-bit words, with individual bytes or even with nibbles (sets of 4 bits), with each such scheme providing a different fault coverage and entailing a different overhead in terms of extra hardware and delay.

As an example, we illustrate the procedure for developing parity prediction rules when using a parity bit for each byte of the AES state. We discuss next the prediction rules for the four steps included in each round.

The prediction of the output parity bits for the ShiftRows transformation is straightforward: it is a rotated version of the input parity bits. Equally simple is the prediction of the output parity bits of the AddRoundKey step: it consists of adding the input parity matrix associated with the state to the parity matrix associated with the current round key.

The SubBytes step commonly uses SBoxes which are 256×8-bit look-up tables. The input to the SBox will already have an associated parity bit. To generate the outgoing parity, a parity bit can be stored with each data byte, increasing the number of bits in each location in the SBox to 9. To make sure that input parity errors are not discarded, we will have to check the parity of the input data, and if an error is detected, stop the encryption process.

A lower overhead solution would be to propagate the input parity errors so that they can be detected later on. This can be achieved by including the incoming parity bit when addressing the SBox, thus further increasing the table size to 512×9. The entries that correspond to input bytes with correct parity will include the appropriate SubBytes transformation result with a correct parity bit. The other entries will contain a deliberately incorrect result, such as an all zeroes byte with an incorrect parity bit.

If fault attacks on the SBox address decoder can be expected, the above scheme is insufficient. Adding a small table that will include the predicted parity bit and one (or more) correct output data bits, as suggested in [64], will allow the detection of most of the addressing circuitry faults.

The prediction of the output parity bits of the MixColumns step is the most complex one. Equations for predicting the parity bits have been derived in [64] and are shown below

$$\begin{aligned} p_{0,j} &= p_{0,j} \oplus p_{2,j} \oplus p_{3,j} \oplus s^{(7)}_{0,j} \oplus s^{(7)}_{1,j} \\ p_{1,j} &= p_{0,j} \oplus p_{1,j} \oplus p_{3,j} \oplus s^{(7)}_{1,j} \oplus s^{(7)}_{2,j} \\ p_{2,j} &= p_{0,j} \oplus p_{1,j} \oplus p_{2,j} \oplus s^{(7)}_{2,j} \oplus s^{(7)}_{3,j} \\ p_{3,j} &= p_{1,j} \oplus p_{2,j} \oplus p_{3,j} \oplus s^{(7)}_{3,j} \oplus s^{(7)}_{0,j} \end{aligned} \quad (3)$$

where $p_{i,j}$ is the parity bit associated with state byte $s_{i,j}$, and $s^{(7)}_{i,j}$ is the most significant bit of $s_{i,j}$.

The question that remains is the granularity at which the comparisons between the generated and predicted parity bits will be made. Scheduling one validation check at the end of the whole encryption process has the obvious advantage of having the lowest overhead in terms of hardware and extra delay.

Theoretically, this could result in the error indication being masked during the encryption procedure, yielding a match between the generated and predicted parity bits in spite of the ciphertext being erroneous. It can be shown, however, that errors injected at any step of the AES encryption procedure will not be masked, and therefore, a single validation check of the final ciphertext is sufficient for error detection purposes.

Still, not every combination of errors can be detected by this scheme. Parity-based EDCs are capable of detecting any fault that consists of an odd number of bit errors. However, an even number of bit errors occurring in a single byte will not be detected. Moreover, if errors are injected in both the state and the round key, some data faults of odd cardinality will not be detected. Although we cannot expect a 100% fault coverage when using a parity-based EDC, the fault coverage has been shown to be very high, even when multiple faults are considered.

In a similar way, EDCs can be developed for other symmetric key ciphers. Several such ciphers which rely on modular addition and multiplication will better match residue codes. Other symmetric ciphers have been shown to require a very expensive implementations of EDCs, leading to the conclusion that the brute force duplication is probably a more suitable solution. The cost of providing protection against fault-based attacks should be taken into account when selecting a cipher for a device.

C. Protection of RSA

Protecting the RSA encryption and decryption primitives without introducing significant overheads, proved to be a challenging task for researchers. The high complexity of the required calculations (relative to those for symmetric block ciphers) results in bigger circuits and/or higher latency, making full temporal or spatial redundancy too costly, especially for resource constrained devices such as smart cards. On the other hand, RSA proved to be very vulnerable to fault attacks, especially when a CRT-based implementation is used.

A lower overhead can be achieved if an EDC is used. Since the RSA cipher is based on modular arithmetic operations, residue codes are a natural choice. At the beginning of the execution, the check bits for the input are generated based on the selected modulus c for the residue check ($m \mod c$ where m is the original message). Then, the operations performed during the RSA algorithm can be applied to the input check bits to obtain the predicted output check bits. The residue check will fail to detect an error when the faulty ciphertext has the same residue check as the correct one. Assuming that the fault injected is random, this match will happen with a probability of $1/c$ and thus, a higher value of c will result in a higher fault coverage (but also a higher overhead).

Another approach, proposed by Shamir in [67] is based on randomizing the computation every time the RSA algorithm is executed. Randomization can serve as a countermeasure not only against fault injection attacks, but also against timing and power attacks since the latency and power profile would depend on some randomly chosen parameters. The proposed scheme targets CRT-based implementations. A random integer r is selected and the following two computations are performed: $s_{rp} = m^d \mod rp$ and $s_{rq} = m^d \mod rq$. Then, the correctness of the computations is checked by verifying that $s_{rp} = s_{rq} \mod r$. Only if no error is detected the final result $s = CRT(s_{rp}, s_{rq})$ is produced. This scheme however, may be subject to a Bellcore-type attack by injecting a fault into either s_{rp} or s_{rq} after the above check has been done but prior to the CRT recombination.

To overcome the above vulnerability, a variation of Shamir's scheme was proposed by Ciet *et al.* [68] employing two different random values which are multiplied with the two prime moduli of the CRT computation. The effect of the random numbers is removed only after the CRT recombination has been performed, thus preventing the Bellcore-type attack.

A generic approach to detecting faults during decryption by executing the inverse process (i.e., encryption) is a viable countermeasure for RSA since the public exponent e is often very small, and consequently, the cost of performing the encryption is negligible. There are however, two issues to be resolved if this countermeasure is to be deployed. The first is that the public exponent e is not always available in the decryption device (e.g., in a smart card). This issue was resolved by Joye [69] who proposed the embedding of the public exponent into the modulus n, thus making it available to the device. The second problem arises when the decrypting device employs the CRT-based algorithm. Such devices are often designed to only perform operations with small moduli rather than the full RSA module n. Since the checking involves a modulo n computation, this may cause a significant slowdown of the process. To solve this problem, Boscher *et al.* [70] proposed a way to employ a CRT-based scheme for the checking thus avoiding the potential slowdown.

To protect RSA encryption and implementations of RSA decryption that are not CRT-based, a strategy to check errors during modular exponentiation is necessary. Modular exponentiation is frequently executed using a "square and multiply" sequence described in Algorithm V.1. The inputs to this algorithm are the encrypted message c, the modulus n and the k-bit private key $d = d_{k-1}, d_{k-2}, \cdots, d_0$.

Algorithm V.1: A straightforward decryption algorithm for RSA.

Input: c, n, $d = [d_{k-1}, d_{k-2}, \cdots, d_0]$: secret exponent
Output: m: plaintext

```
1 begin
2     a ← s
3     for i ← k − 2 to 0 do
4         a ← a^2 mod n
5         if d_i = 1 then
6             a ← c · a mod n
7     return m
8 end
```

This algorithm correctly produces the desired result in k

steps, where a squaring operation is always performed, and a multiplication operation is performed only if the corresponding bit of the private exponent d is equal to one. Unfortunately, this algorithm is vulnerable to simple power analysis techniques, which rely on the different power consumption caused by the presence or lack of the multiplication, to infer the value of the private exponent bits. The most straightforward solution to this issue is to follow a "square and multiply always" strategy, such as the one described in Algorithm V.2.

Algorithm V.2: A modified decryption algorithm for RSA.

Input: c, n, $d = [d_{k-1}, d_{k-2}, \cdots, d_0]$: secret exponent
Output: m: plaintext

```
1  begin
2      a ← s
3      for i ← k − 2 to 0 do
4          a ← a^2 mod n
5          b ← c · a mod n
6          if d_i = 1 then
7              a ← b
8          else
9              a ← a
10     return m
11 end
```

This strategy always performs the multiplication, thus equalizing the power consumption of the otherwise different iterations. Although this strategy is effective in preventing power analysis based attacks, it is possible to attack the aforementioned algorithm through an easy safe error attack. By disturbing the multiplication, through fault injection, it is possible to deduce the corresponding bit of the secret exponent in the following way. If, despite the injection of a fault, the output is correct then the multiplication was unnecessary while if the output is incorrect (or no output is produced due to the fault being internally detected), then the multiplication was needed.

These safe error attacks can be prevented, as was pointed out in [71], by computing the exponentiation using the Montgomery Laddering technique depicted in Algorithm V.3. The technique uses two temporary values a and b whose values are recomputed every iteration and as a result, any fault injected during an inner operation will be detected. This technique has the added benefit that it makes it possible to check at each iteration whether the relationship $ma = b$ holds [72]. The number and positions of such checks may be determined by the designer, according to a trade-off between the computation time and the security level.

Unfortunately, this technique increases the vulnerability to power attacks since the computation now includes a conditional construct that results in an imbalance, albeit slight, in the power consumption. To overcome this problem and obtain a fault and power analysis resistant exponentiation algorithm, Fumaroli *et al.* [73] proposed a variant of the Montgomery laddering technique that employs a randomization scheme

Algorithm V.3: A Montgomery Ladder algorithm for RSA decryption.

Input: c, n, $d = [d_{k-1}, d_{k-2}, \cdots, d_0]$: secret exponent
Output: m: plaintext

```
1  begin
2      a ← s
3      for i ← k − 2 to 0 do
4          a ← a^2 mod n
5          b ← c · a mod n
6          if d_i = 1 then
7              a ← a^2 mod n
8              b ← a · b mod n
9          else
10             a ← a^2 mod n
11             b ← a · b mod n
12     return m
13 end
```

and substitutes the conditional instruction in the previous algorithm by a fast multiplication of the temporary values of the Montgomery ladder with the bit of the exponent. The randomization of the encrypted message is performed through exponentiating rm instead of m, where r is a small random number. In parallel to the exponentiation, r^{-e} is computed and then used to remove the randomization (also known as blinding) effect after the computation.

The randomization introduced by Fumaroli prevents an attacker from knowing the actual result of the exponentiation, and provides a way to check if the computation proceeds properly using the same relationship which held for the original Montgomery ladder. Still, as was later pointed out in [74], it is possible to inject an undetected error in the result, if the attacker only corrupts the calculation of the blinding removal value r^{-e}. Such a fault may constitute a security risk if the above technique is employed in the computation of one of the two halves of the CRT-based decryption. If the undetected faulty value is employed in the CRT reconstruction, it leads to the same scenario as in the Bellcore attack.

To obtain a protected CRT-based low overhead decryption algorithm, Boscher *et al.* [75] combined a checking strategy for the CRT recombination with a fully protected exponentiation.

An alternate strategy to hinder attacks on RSA relies on propagating the effects of the injected fault on the whole computation, leading to a faulty output which is not exploitable by the attacker. This strategy, denominated *fault infective computation* in [76], shifts the focus of the countermeasure from detecting the faults, to directly hindering the attacker without altering the computation flow, thus implicitly preventing safe error attacks. A generalisation of this concept, denominated *fault resilient computation*, has been presented in [77], focusing on the design of a circuit which relies on dual-rail logic to implement the same key concept.

D. Protection of ECC

Elliptic curve cryptosystems can be protected by extending some of countermeasures that were developed for RSA and adapting them to the different group operations performed during ECC encryption/decryption. In [78] the authors present a list of countermeasures for ECC taking into account all the previously proposed attacks. In particular, the authors suggest, as a practical defense against safe error and bit flipping attacks on the double and add ladder, to compute the kP through splitting k into two values. Furthermore, to avoid exploitable deterministic behavior, a random number r (smaller than k) should be selected leading to the computation of $[k \bmod r]P + [\lfloor \frac{k}{r} \rfloor]P$. A second suggested guideline is to avoid decision checks in the same way they are avoided in the Montgomery laddering during RSA exponentiation. This way, an error during the double and add ladder will yield a random result avoiding information leakage. Another information leakage prevention approach, proposed in [79], involves randomizing the base point P used in the kP operation. This countermeasure introduces further randomness into the scheme hindering the recovery of the value of k regardless of the faults induced during the computation.

VI. Power and Fault attacks synergies

Although we have focused in this paper on fault injection attacks, we must keep in mind that there are other types of side channel attacks that a possible attacker may follow in an attempt to breach the security of a system. A commonly used approach is through power analysis attacks, which measure the amount of power consumed while performing encryption or decryption during the normal (fault-free) operation of the attacked device.

An example of these attacks is the differential power attack whose goal is to construct a key-dependent model of power consumption that depends on the switching activity of the circuit, and try to find which one fits best the actual measurements taken on the device. The most common way to protect against this kind of attacks is to design the device such that it has a constant power consumption regardless of the ongoing computation.

Traditionally, fault attacks and power analysis attacks were considered disjoint attack methodologies. This lead to the assumption that protecting the device against each one of these two possible attacks individually, the device is consequently protected against any combination of these two types of attacks. This assumption, however, proved to be false, and it has been shown that it is possible to exploit fault attacks in order to enhance the efficiency of power attacks. The key idea behind the combined attack is that a fault induced during a computation can alter, in addition to the result of the computation, also the power consumed by the device due to a change in the switching activity of the circuit.

The first combined attack technique was reported by Amiel *et al.* in [80]. In this work, the authors showed that it is possible to attack an RSA exponentiation that was protected against power analysis attacks using a balanced algorithm with message randomization, by inducing ad-hoc faults. The technique involves the partial or total blanking of the contents of the register holding the base value of the message m, which in turn, reduces the power consumption of the multiplication by m. Since this multiplication is performed only when the current bit of the secret exponent d is 1, the authors were able to obtain the secret key simply by observing which operations did consume less power after the fault injection. The authors have also pointed out that this kind of attack cannot be prevented by simple countermeasures against faults such as checking the signature at the end of the computation. This is due to the fact that the information leakage happens during the computation and a-posteriori checks cannot prevent it.

One way to protect against these attacks is to detect the injected fault immediately, rather than wait until the computation of the signature is completed, and upon detection, abort the process and thus avoid generating the power profile that divulges the secret key. Well-known examples of this type of countermeasures are the error detection codes based on either parity or residue checking. These codes, through adding redundant check bits to the data processed, are able to detect on the fly an invalid alteration of the data. However, the addition of circuitry to process the added check bits may help differential power analysis attacks since the check bits are highly correlated to the data bits being processed.

In [81], [82] the authors reported the results of correlation power analysis attacks (that are more powerful than differential power analysis) on an implementation of AES where the output buffer of the S-Box has been extended to include an error detection circuit. The considered error detection codes were parity, residue modulo 3 and residue modulo 7. The results of this study showed that the introduction of error checking circuits would help the attacker by reducing the number of power traces needed to discover the bits of the secret key. The experiments in this study were performed with an added measurement noise that must be expected during any practical power analysis attack. A reduction in the required number of traces has been observed even if the attacker is unaware of the presence of check bits in the attacked implementation. Furthermore, the authors observed that the higher the number of check bits is, the easier the power attack becomes.

These observations suggest that, albeit targeting different side channels, substantial synergies between active and passive attacks exist and these may be exploited by a malicious attacker. This in turn implies that novel countermeasures against either power or fault attacks should always take into consideration their impact on the robustness against the other type of side channel attacks.

VII. Future Research Directions

We conclude this survey with a brief description of possible new research directions in the field of fault injection attacks. One such direction will focus on establishing formal models for fault injection attacks and exploring their effectiveness.

This will allow a top-down analysis of possible fault attacks instead of the current bottom-up approach.

The number and types of fault attacks that are being developed for the various ciphers is steadily increasing but no clear classification has emerged. The current trend is to model the injected faults as errors in the underlying mathematical formulae rather than mistakes in the computations performed by a certain computer model. This approach makes it difficult to establish a relationship between the fault model and the existing technology and consequently, its practical relevance is becoming questionable. An alternative approach might be the development of a scientifically rigorous quantification of the robustness of an implementation of a cryptographic primitive (e.g., RSA encryption) against fault attacks by modeling the primitive as a program that runs on a given virtual machine model. One can then determine the fault resilience and cost of the cryptographic primitive implementation where the resilience can be defined in one of the following ways:

- Being able to return the correct result despite a fault, or
- Detecting the fault and discontinuing the process, or
- Returning an incorrect result making it difficult for the attacker to recover either the secret key or the plaintext.

The overhead (cost) of the given implementation can be measured in terms of the execution time or the code size of the program implementing the primitive. A number of possible virtual machines can serve as a framework for the above model ranging from the theoretical Turing Machine to more practical ones.

Clearly, exploring the solution space with respect to these resilience and cost parameters for crypto-primitives implementations that incorporate countermeasures, would have theoretical and practical significance. Such analysis can then be extended to complete implementations of cryptographic protocols consisting of a set of crypto-primitives (e.g., Transport Layer Security [83]).

Determining the resilience versus cost trade-offs in a theoretical way may prove to be difficult. Instead, successive approximations can be obtained by using known attacks procedures. Such procedures could be used to compare primitives and different virtual machine models, and thus progressively refine the analysis.

Besides the above described new research direction, there is still a need to keep developing countermeasures against fault attacks and new ways to make implementations more immune to attacks, and improve the techniques to test the robustness of cryptographic primitives and the robustness provided by technological advancements.

To illustrate the need for the development of new countermeasure consider the following scenario. Assume a message m_1 is encrypted into a ciphertext c_1 using RSA. Injecting a one-bit fault into m_1 during a second encryption will result in the encryption of the message $m_2 = m_1$ xor $2^i = m \pm 2^i = m + b$, for a b predictable by the attacker; hence $z = m_1$ is a common root of the two equations below:

$$\begin{aligned} z^e - c_1 &= 0 \mod n \\ (z+b)^e - c_2 &= 0 \mod n \end{aligned}$$

Thus m_1 will be retrieved with a high probability by the operation below:

$$\gcd\left(z^e - c_1,\ (z+b)^e - c_2\right) = z - m_1 \bmod n$$

The complexity is quadratic in e and the attack will hence work only for small public exponents. This attack is derived from a *non-fault* (i.e., algorithmic) attack on RSA [84] and uncovers an unexpected relationship.

In addition, a number of recently developed crypto-primitives deserve more extensive theoretical and practical investigations. Examples include elliptic curve pairing or lattice-based operations, which are currently of considerable interest.

Furthermore, the number of known countermeasures is as large as the number of different fault attacks that have been developed and quite often, a countermeasure has been finely tuned to a specific attack. There is a need to design more general countermeasures that will be effective against many fault attacks and yet be inexpensive. Developing theoretical generic countermeasures may be too ambitious, and a more promising approach seems to be that of unifying the existing ones (e.g., [70], [85]). The results in this field are still limited to a couple of primitives, thus a deeper insight is required.

A further research direction is represented by the advent of multi-core processors in mobile environments. Since crypto-primitives have been mainly designed and implemented on serial processors, there is an ongoing transition to their implementation on multi-core platforms. Currently, it is unclear how to extend the known fault attack methodologies and techniques to a parallel algorithm, which may be processing several keys simultaneously or several plaintexts using one key. Another related question is whether fault injection attacks (and more generally, side-channel attacks) on parallel implementations will retain their feasibility on these modern platforms.

REFERENCES

[1] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[2] R. J. Anderson and M. G. Kuhn, "Low cost attacks on tamper resistant devices," *in Proc. International Workshop on Security Protocols*, pp. 125–136, 1998.

[3] D. Boneh, R. DeMillo, and R. Lipton, "On the importance of eliminating errors in cryptographic computations," *Journal of Cryptology*, vol. 14, no. 2, pp. 101–119, Nov. 2001.

[4] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," *in Proc. CRYPTO*, pp. 513–525, 1997.

[5] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *in Proc. EUROCRYPT*, pp. 37–51, 1997.

[6] F. Bao, R. H. Deng, Y. Han, A. B. Jeng, A. D. Narasimhalu, and T.-H. Ngair, "Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults," *in Proc. International Workshop on Security Protocols*, pp. 115–124, 1998.

[7] A. K. Lenstra, "Memo on RSA Signature Generation in the Presence of Faults," September 1996, unpublished memo.

[8] L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, Eds., *Proceedings of the Third International Workshop in Fault Diagnosis and Tolerance in Cryptography, 10 October 2006*, ser. Lecture Notes in Computer Science. Springer, 2006, vol. 4236.

[9] L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, Eds., *Proceedings of the Fourth International Workshop on Fault Diagnosis and Tolerance in Cryptography, 10 September 2007*. IEEE Computer Society, 2007.

[10] ——, *Proceedings of the Fifth International Workshop on Fault Diagnosis and Tolerance in Cryptography, 10 August 2008*. IEEE Computer Society, 2008.

[11] *SNOW 3G Specifications*, European Telecommunications Standards Institute Std., September 2006.

[12] P. Ekdahl and T. Johansson, "A New Version of the Stream Cipher SNOW," *in Proc. Selected Areas in Cryptography*, vol. 2595, pp. 47–61, 2003.

[13] *ISO/IEC 18033-4:2005 Information technology – Security techniques – Encryption algorithms – Part 4: Stream ciphers*, ISO Std., 2005.

[14] *FIPS-46-3: Data Encryption Standard (DES)*, National Institute of Standards and Technology (NIST) Std., 1999.

[15] M. Curtin, *Brute force: cracking the data encryption standard*. Springer, 2005.

[16] *FIPS-197: Advanced Encryption Standard*, National Institute of Standards and Technology (NIST) Std., 2001.

[17] J. Daemen and V. Rijmen, *The design of Rijndael: AES–the Advanced Encryption Standard*. Springer Verlag, 2002.

[18] *IEEE P-1619 Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices*, IEEE Std., 2008.

[19] *IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std., 1997.

[20] I. Koren and C. M. Krishna, *Fault Tolerant Systems*. San Francisco, CA, USA: Morgan-Kaufman, 2007.

[21] R. Rivest, A. Shamir, and L. Adleman, "Method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[22] P. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, pp. 519–521, 1985.

[23] STMicroelectronics, "ST23 highly secure smartcard ICs," January 2010. [Online]. Available: http://www.st.com/stonline/products/families/smartcard/sc_st23.htm

[24] D. J. Bernstein, "DNSCurve: Usable security for DNS," January 2009. [Online]. Available: http://dnscurve.org/

[25] A. Barenghi, G. Bertoni, E. Parrinello, and G. Pelosi, "Low Voltage Fault Attacks on the RSA Cryptosystem," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 23–31, 2009.

[26] A. Barenghi, G. M. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi, "Low Voltage Fault Attacks to AES," *in Proc. International Symposium on Hardware-Oriented Security and Trust*, pp. 7–12, 2010.

[27] N. Selmane, S. Guilley, and J.-L. Danger, "Practical Setup Time Violation Attacks on AES," *in Proc. European Dependable Computing Conference*, pp. 91–96, 2008.

[28] A. Barenghi, C. Hocquet, D. Bol, F.-X. Standaert, F. Regazzoni, and I. Koren, "Exploring the Feasibility of Low Cost Fault Injection Attacks on Sub-threshold Devices through an Example of a 65nm AES Implementation," *in Proc. Workshop on RFID Security and Privacy*, pp. 48–60, 2011.

[29] J.-M. Schmidt and C. Herbst, "A Practical Fault Attack on Square and Multiply," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 53–58, 2008.

[30] M. Hutter, T. Plos, and J.-M. Schmidt, "Contact-Based Fault Injections and Power Analysis on RFID Tags," *in Proc. IEEE European Conference on Circuit Theory and Design*, pp. 409–412, 2009.

[31] F. Amiel, C. Clavier, and M. Tunstall, "Fault Analysis of DPA-Resistant Algorithms," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, vol. 4236, pp. 223–236, 2006.

[32] S. Govindavajhala and A. Appel, "Using memory errors to attack a virtual machine," *in Proc. IEEE Symposium on Security and Privacy*, pp. 154–165, 2003.

[33] J.-M. Schmidt and M. Hutter, "Optical and EM Fault-Attacks on CRT-based RSA: Concrete Results," *in Proc. Austrian Workhop on Microelectronics (Austrochip)*, pp. 61–67, 2007.

[34] J.-M. Schmidt, M. Hutter, and T. Plos, "Optical Fault Attacks on AES: A Threat in Violet," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 13–22, 2009.

[35] S. Skorobogatov, "Semi-invasive attacks-a new approach to hardware security analysis," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-630, 2005.

[36] S. P. Skorobogatov and R. J. Anderson, "Optical Fault Induction Attacks," *in Proc. Workshop on Cryptographic Hardware and Embedded Systems*, pp. 2–12, 2002.

[37] M. Agoyan, J.-M. Dutertre, A.-P. Mirbaha, D. Naccache, A.-L. Ribotta, and A. Tria, "How to flip a bit?" *in Proc. IEEE 16th International On-Line Testing Symposium (IOLTS)*, pp. 235 –239, 2010.

[38] R. Torrance and D. James, "The State-of-the-Art in IC Reverse Engineering," *in Proc. Workshop on Cryptographic Hardware and Embedded Systems*, p. 381, 2009.

[39] B. Debraize and I. M. Corbella, "Fault Analysis of the Stream Cipher Snow 3G," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 103–110, 2009.

[40] F. Armknecht and W. Meier, "Fault attacks on combiners with memory," *in Proc. Selected Areas in Cryptography*, vol. 3897, pp. 36–50, 2006.

[41] J. Bloemer and J.-P. Seifert, "Fault Based Cryptanalysis of the Advanced Encryption Standard (AES)," *in Proc. Financial Cryptography*, pp. 162–181, 2003.

[42] C. Giraud, "DFA on AES," *in Proc. International Conference on the Advanced Encryption Standard*, vol. 3373, pp. 27–41, 2005.

[43] A. Barenghi, G. M. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi, "Fault attack on AES with single-bit induced faults," *in Proc. Sixth International Conference on Information Assurance and Security*, pp. 7–13, 2010.

[44] P. Dusart, G. Letourneux, and O. Vivolo, "Differential Fault Analysis on A.E.S." *Applied Cryptography and Network Security*, vol. 2846, pp. 293–306, 2003.

[45] A. Moradi, M. T. M. Shalmani, and M. Salmasizadeh, "A generalized method of differential fault attack against AES cryptosystem," *in Proc. International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 91–100, 2006.

[46] C.-N. Chen and S.-M. Yen, "Differential fault analysis on aes key schedule and some countermeasures," *in Proc. Information Security and Privacy*, pp. 217–217, 2003.

[47] D. Peacham and B. Thomas, "A DFA attack against the AES key schedule," SiVenture Whitepaper, October 2006.

[48] C. H. Kim and J.-J. Quisquater, "New Differential Fault Analysis on AES Key Schedule: Two Faults Are Enough," *in Proc. International Conference on Smart Card Research and Advanced Applications*, pp. 48–60, 2008.

[49] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert, "Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures," *in Proc. Workshop on Cryptographic Hardware and Embedded Systems*, pp. 81–95, 2003.

[50] S.-M. Yen and M. Joye, "Checking before output may not be enough against fault-based cryptanalysis," *IEEE Transaction on Computers*, vol. 49, no. 9, pp. 967–970, 2000.

[51] G. Hardy, *An Introduction to the Theory of Numbers*, 5th ed., ser. Oxford Science Publications. Oxford Press, 1979.

[52] D. E. Knuth, *Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed. Addison-Wesley Professional, November 1997.

[53] J. Blomer, M. Otto, and J. P. Seifert, "Sign change fault attacks on elliptic curve cryptosystems," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 25–40, 2005.

[54] I. Biehl, B. Meyer, and V. Müller, "Differential Fault Attacks on Elliptic Curve Cryptosystems," *in Proc. CRYPTO*, pp. 131–146, 2000.

[55] P.-A. Fouque, R. Lercier, D. Réal, and F. Valette, "Fault attack on elliptic curve montgomery ladder implementation," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 92–98, 2008.

[56] *FIPS-186-3: Digital Signature Standard (DSS)*, National Institute of Standards and Technology (NIST) Std., 2009.

[57] *IEEE Standard Specifications for Password-Based Public-Key Cryptographic Techniques*, IEEE Std., 2009.

[58] A. Barenghi, G. Bertoni, A. Palomba, and R. Susella, "A novel fault attack against ECDSA," *in Proc. IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 161 –166, 2011.

[59] A. Barenghi, G. M. Bertoni, L. Breveglieri, G. Pelosi, and A. Palomba, "Fault attack to the elliptic curve digital signature algorithm with multiple bit faults," *in Proc International Conference on Security of Information and Networks*, pp. 63–72, 2011.

[60] IBM, "Ibm 4764 pci-x cryptographic coprocessor specifications," [Online] http://www.ibm.com/security/cryptocards/pdfs/bs330.pdf.

[61] M. Karpovsky and A. Taubin, "New class of nonlinear systematic error detecting codes," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1818–1820, 2004.

[62] R. Karri, K. Wu, P. Mishra, and Y. Kim, "Fault-based side-channel cryptanalysis tolerant Rijndael symmetric block cipher architecture," *in Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 427–435, 2001.

[63] P. Maistri, P. Vanhauwaert, and R. Leveugle, "A Novel Double-Data-Rate AES Architecture Resistant against Fault Injection," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 54–61, 2007.

[64] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "Error analysis and detection procedures for a hardware implementation of the advanced encryption standard," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 492–505, 2003.

[65] G. Bertoni, L. Breveglieri, I. Koren, and P. Maistri, "An efficient hardware-based fault diagnosis scheme for AES: performances and cost," *in Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 130–138, 2004.

[66] A. Butter, C. Kao, and J. Kuruts, "DES encryption and decryption unit with error checking," US Patent 5,432,848, July 1995.

[67] A. Shamir, "Method and apparatus for protecting public key schemes from timing and fault attacks," US Patent 5,991,415, 1999.

[68] M. Ciet and M. Joye, "Practical Fault Countermeasures for Chinese Remaindering Based RSA," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 124–131, 2005.

[69] M. Joye, "Protecting RSA against Fault Attacks: The Embedding Method," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 41–45, 2009.

[70] A. Boscher, H. Handschuh, and E. Trichina, "Fault Resistant RSA Signatures: Chinese Remaindering in Both Directions," Cryptology ePrint Archive, Report 2010/038, 2010.

[71] M. Joye and S.-M. Yen, "The montgomery powering ladder," *in Proc. Workshop on Cryptographic Hardware and Embedded Systems*, pp. 291–302, 2003.

[72] C. Giraud, "An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis," *IEEE Transactions on Computers*, vol. 55, no. 9, pp. 1116–1120, 2006.

[73] G. Fumaroli and D. Vigilant, "Blinded Fault Resistant Exponentiation," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, p. 62, 2006.

[74] C. H. Kim and J.-J. Quisquater, "How can we overcome both side channel analysis and fault attacks on RSA-CRT?" *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 21–29, 2007.

[75] A. Boscher, H. Handschuh, and E. Trichina, "Blinded Fault Resistant Exponentiation Revisited," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 3–9, 2009.

[76] S.-M. Yen, S. Kim, S. Lim, and S.-J. Moon, "Rsa speedup with chinese remainder theorem immune against hardware fault cryptanalysis," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 461 – 472, april 2003.

[77] S. Guilley, L. Sauvage, J.-L. Danger, and N. Selmane, "Fault injection resilience," 2010, pp. 51–65.

[78] M. Ciet and M. Joye, "Elliptic curve cryptosystems in the presence of permanent and transient faults," *Designs, Codes and Cryptography*, vol. 36, no. 1, pp. 33–43, 2005.

[79] A. Dominguez-Oviedo and M. Hasan, "Error Detection and Fault Tolerance in ECSM Using Input Randomization," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 3, pp. 175–187, 2009.

[80] F. Amiel, K. Villegas, B. Feix, and L. Marcel, "Passive and Active Combined Attacks: Combining Fault Attacks and Side Channel Analysis," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 92–102, 2007.

[81] F. Regazzoni, T. Eisenbarth, J. Großschädl, L. Breveglieri, P. Ienne, I. Koren, and C. Paar, "Power attacks resistance of cryptographic s-boxes with added error detection circuits," *in Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI*, pp. 508–516, 2007.

[82] F. Regazzoni, T. Eisenbarth, L. Breveglieri, P. Ienne, and I. Koren, "Can knowledge regarding the presence of countermeasures against fault attacks simplify power attacks on cryptographic devices?" *in Proc. IEEE International Symposium on Defect and Fault-Tolerancein VLSI Systems*, pp. 202–210, 2008.

[83] *The Transport Layer Security (TLS) Protocol Version 1.2*, Internet Engineering Task Force (IETF) Std., 2008.

[84] D. Coppersmitsh, M. Franklin, J. Patarin, and M. Reiter, "Low-exponent rsa with related messages," *in Proc. EUROCRYPT*, pp. 1–9, 1996.

[85] M. Joye, "On the security of a unified countermeasure," *in Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 87–91, 2008.