Fault Attack to the Elliptic Curve Digital Signature Algorithm with Multiple Bit Faults

Alessandro Barenghi
DEI - Dipartimento di Elettronica e Informazione
Politecnico di Milano
Via Ponzio 34/5,
20133 Milano, Italy
barenghi@elet.polimi.it

Guido M. Bertoni
STMicroelectronics
Via Olivetti 2,
20041 Agrate Brianza
(MB), Italy
guido.bertoni@st.com

Luca Breveglieri
DEI - Dipartimento di Elettronica e Informazione
Politecnico di Milano
Via Ponzio 34/5,
20133 Milano, Italy
brevegli@elet.polimi.it

Andrea Palomba
DEI - Dipartimento di Elettronica e Informazione
Politecnico di Milano
Via Ponzio 34/5,
20133 Milano, Italy
palomba@elet.polimi.it

Gerardo Pelosi
DEI - Dipartimento di Elettronica e Informazione
Politecnico di Milano
Via Ponzio 34/5,
20133 Milano, Italy
pelosi@elet.polimi.it

ABSTRACT

Elliptic curve cryptosystems proved to be well suited for securing systems with constrained resources like embedded and portable devices. In a fault attack, errors are induced during the computation of a cryptographic primitive, and the faulty results are collected to derive information about the secret key stored into the device in a non-readable way. Scenarios where the secure devices are seized by an opponent are quite common. Consequently, it is possible for an attacker to induce changes in the working environment of the device to cause alterations in the computation of the cryptographic primitive. We introduce a new fault model and attack methodology to recover the secret key employed in implementations of the Elliptic Curve Digital Signature Algorithm. Our attack exploits the information leakage induced through altering the execution of the modular arithmetic operations used in the signature primitive and does not rely on the properties of the underlying elliptic curve mathematical structure. The attack is easily reproducible with low cost fault injection technologies relying on transient errors placed within a single datapath width of the target architecture.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application Based Systems**]: Microprocessor/microcomputer applications;
C.5.3[**Computer System Implementation**]: Microcomputers[portable devices];
E.3[**Data Encryption**]: Standards (ECDSA)

for consideration of *SIN'11,* Nov. 14–19, 2011, Sydney, Australia.

General Terms

Security

Keywords

Fault Attacks, Elliptic Curve Digital Signature Algorithm

1. INTRODUCTION

In the last few years there has been a rapidly growing interest for digital signature frameworks from both public institutions and private enterprises to facilitate the adoption of large-scale IT applications. Digital signature schemes guarantee the detection of forgery or tampering of transmitted data through providing data integrity, data origin authentication, and non-repudiation assurances of previous actions or commitments. Indeed, digital signature schemes represent an essential building block of many cryptographic protocols that provide other services including entity authentication, and authenticated key agreement. Currently, the most innovative and widely used scheme for digital signatures is the Elliptic Curve Digital Signature Algorithm (ECDSA) [1,17]. The same standards certify also the ECDSA key sizes advised by NSA, in its public cryptographic suite, for SECRET and TOP SECRET grade security documents. The basis for the security of ECDSA is the intractability of the elliptic curve discrete logarithm problem (ECDLP), which appears to be harder than both the discrete logarithm problem in finite fields and the problem of factoring a composite integer. Assuming a predetermined security level, the parameters and operands involved in the ECDSA algorithm are smaller than the ones employed in other systems, with the important consequence of obtaining resource-saving and low-power consumption implementations while keeping high security margins. Most signature-creation and signature-verification systems are currently based on embedded and portable devices, which keep all necessary private information (f.i., keys and certificates) in a non-volatile storage either to prove their authenticity to other integrated systems or to accept only firmware/software updates from valid issuers. Usually, implementation attacks aim either at com-

promising or recovering the private information manipulated through the cryptographic primitive in secure devices. In principle, the only option of the potential attackers to guess the secret key value should be an exhaustive search of the full key space, without any information leakage through observing or manipulating the inputs/outputs of the device.

1.1 Fault induction techniques

Semi-invasive fault attacks may find a way around physical hardening techniques employed on tamper resistant devices since it is possible to apply some techniques even without tampering with the physical barriers built around the device. Common techniques to induce controlled faults include: supplying noisy power or clock signals, providing an insufficient feeding voltage during the execution, irradiating the device with either coherent light or X-rays. Alterations in external clock and supply voltage glitches are known to cause either incorrect data loading from memory or instruction skips [19], while the constant underfeeding may induce temporary stuck-at faults due to setup time violations [3,4,21]. Selective irradiation of sensitive area of the circuit, such as the registers or the arithmetic-logical units, induces either single local bit flips or multiple bit flips in a contiguous part of a value [23]. Depending on the precision of the fault induction technique, it is possible for an attacker to induce more or less controlled faults in a computing device (in timing and/or location). The most precise fault induction techniques allow the insertion of single bit faults aimed at a specific operation of the algorithm, thus allowing the attacker to alter its behavior in a fully controlled way. Coarser fault induction techniques may result in the corruption of a single byte or in the modification of the full value held by one register during the execution. The security margin effectively provided by a cryptographic token depends on both the physical means available to the attacker and the structure and implementation of the cryptographic algorithm. In particular the implementation and structure play a crucial role in determining the model of the faults which must be employed to obtain exploitable output values from the device.

1.2 Contributions

In this work we will present an extension of the single bit fault attack to ECDSA proposed in [5] aimed at recovering the secret key through inducing faults in the signature generation primitive. The extension concerns the introduction of multiple single bit faults placed within a single datapath width of the target architecture. This extension effectively lowers the security margin of the algorithm since the precise induction of single bit faults usually requires expensive equipment such as Focused Ion Beam stations. In particular, it is possible to significantly broaden the fault induction methodologies to obtain a word-sized fault in a computing device: even the cheapest techniques induce faults fitting this model. However, a less precise fault model implies that the information leaked from the erroneous results will be less precise, thus leading to an increased computational effort to exploit it. We provide a complexity analysis detailing the running time of the secret key retrieval algorithm and the average number of faults required to retrieve the secret key. Our attack relies on the fact that multiple precision multiplications are implemented in an operand scanning fashion [15] in the underlying architecture. This can be achieved either in software, where the word length of the operand scanning algorithm is determined by the architecture, or in a dedicated ASIC implementation, where the datapath width represents a design parameter. A notable example is the latest OpenSSL implementation[1], conforming to ANSI X9.62 [1], which employs the aforementioned multiplication strategy and is available for a large variety of CPU architectures covering the full spectrum of current computing devices. The most common hardware implementation for high speed multiple precision multipliers, the "Coarsely Integrated Operand Scanning" [24], also relies on the same operative methodology. We consider architecture datapath widths ranging from 8 to 32 bits to cover the full range of current implementations of ECDSA for embedded and mobile platforms. The attack workflow relies on collecting the erroneous results produced by the device and recognizing whether they are effectively exploitable to retrieve a word of the secret key, or if they are to be discarded immediately.

1.3 Organization of the paper

The remainder of the paper is organized as follows. Section 2 resumes the mathematical background on elliptic curve cryptography and the ECDSA algorithm. Section 3 introduces the fault model assumed by the proposed attack. Section 4 describes the secret-key retrieval algorithm designed to operate on the faulty outputs of the ECDSA signature generation primitive. Section 5 presents the performance results. Section 6 describes related work through surveying the main results on fault attacks aimed at Elliptic Curve Cryptosystems (ECC) and the attacks at the current state-of-the-art that specifically target the ECDSA primitive. Finally, Section 7 draws our conclusions.

2. PRELIMINARIES

2.1 Elliptic curves

Let $\mathbb{F}_p$ denote the finite field built over the equivalence classes generated by integers $\mathbb{Z}$ modulo p, where p is a prime number. Assuming $p \notin \{2, 3\}$ we denote as $\mathbb{E}(\mathbb{F}_p)$ the elliptic curve represented by the set of points $\mathrm{P}(x_\mathrm{P}, y_\mathrm{P}) \in \mathbb{E}, x_\mathrm{P}, y_\mathrm{P} \in \mathbb{F}_p$ satisfying equation (1), plus the point at infinity $\mathcal{O}$ that represents the directions parallel to the y-axis in the projective plane.

$$y^2 = x^3 + a\,x + b, \quad 4a^3 + 27b^2 \neq 0, \quad a,\, b \in \mathbb{F}_p \qquad (1)$$

Let P, Q be two points of $\mathbb{E}(\mathbb{F}_p)$, and let $\mathrm{R} \in \mathbb{E}(\mathbb{F}_p)$ be the third point of intersection of $\mathbb{E}$ with the straight line joining P and Q (or with the tangent line at P if P=Q). The point S derived as the third point of intersection between $\mathbb{E}$ and the vertical line joining R and $\mathcal{O}$ is defined to be the outcome of a commutative internal composition law (a.k.a. "secant-&-tangent rule") between P, Q and denoted as S=P+Q. The set of points of an elliptic curve with the previous internal composition law constitutes an algebraic commutative group $(\mathbb{E}(\mathbb{F}_p), +)$ [25] where $\mathcal{O}$ is the neutral element (i.e., $\mathrm{P}+\mathcal{O}=\mathrm{P}\ \forall \mathrm{P} \in \mathbb{E}(\mathbb{F}_p)$). Given an integer $k \in \mathbb{Z}_n$ (the set of canonical representatives of residue classes modulo n) and a point $\mathrm{P} \in \mathbb{E}(\mathbb{F}_p)$, the "scalar-point multiplication" operation is defined as the iterated sum: $\mathrm{P}+\ldots+\mathrm{P}=[k]\mathrm{P}$, with $[k]\mathrm{P}=\mathcal{O}$ if $k=0$. In the following we will denote as G the generator of the group $(\mathbb{E}(\mathbb{F}_p), +)$, and as $n=|\langle \mathrm{G} \rangle|=|\mathbb{E}((\mathbb{F}_p), +)|$ the order of the group. The result of the following lemma will

[1]Mark J. Cox et al., The OpenSSL Project, ver.1.0.0d. `http://www.openssl.org/`

be employed to design the attack methodology described in the next sections.

LEMMA 1. *Let* $\mathrm{P}(x_\mathrm{P}, y_\mathrm{P})$ *be a point of the elliptic curve* $\mathbb{E}(\mathbb{F}_p)$*, and* $r{=}x_\mathrm{P} \bmod n$*. There are at most three other points belonging to the curve* $\mathbb{E}(\mathbb{F}_p)$ *having an* `x`*-coordinate in the same equivalence class modulo* n *of* r*.*

PROOF. Hasse's theorem [25] is a well known result in elliptic curve theory that bounds the number of points on a curve, $n{=}|\mathbb{E}(\mathbb{F}_p)|$: $p{+}1{-}2\sqrt{p} \leq n \leq p{+}1{+}2\sqrt{p}$. Making use of simple algebraic equivalences, it is easy to see how the previous relation implies $n{+}1{-}2\sqrt{n} \leq p \leq n{+}1{+}2\sqrt{n}$, and then also $p{<}2n$. Therefore, given a generic point $\mathrm{Q}(x_\mathrm{Q}, y_\mathrm{Q})$, $\mathrm{Q}{\neq}\mathrm{P}$, $x_\mathrm{Q}, y_\mathrm{Q}{\in}\mathbb{F}_p$, the relation $r{=}x_\mathrm{Q} \bmod n \Leftrightarrow r{=}x_\mathrm{Q}{-}\lambda\, n$ with $\lambda{\in}\mathbb{N}$, may be valid only when $\lambda{\in}\{0,1\}$. Considering that a generic point Q and its opposite, $-\mathrm{Q}$, have the same `x`-coordinate, we are able to infer that there are at most three points on $\mathbb{E}(\mathbb{F}_p)$, other than P, with an `x`-coordinate in the same equivalence class modulo n of r. Given a value r, the candidate points can be obtained via replacing $r{+}\lambda\, n$, $\lambda{\in}\{0,1\}$ as the `x`-coordinate value at the second member of equation (1), and checking whether such a value is a quadratic residue modulo p. In case a valid coordinate pair, $\mathrm{Q}{=}(x_\mathrm{Q}, y_\mathrm{Q})$, $x_\mathrm{Q}{=}r{+}\lambda n$, is found, the computation of the opposite $-\mathrm{Q}$ allows to collect either two different points (whereas, $\mathrm{Q}{\neq}{-}\mathrm{Q}$) or one point with double multiplicity. □

2.2 Digital Signature

Standards on ECDSA [1, 12, 17] provide a list of recommended elliptic curves $\mathbb{E}(\mathbb{F}_p)$, each of which has a specified prime finite field $\mathbb{F}_p$, group generator G and group order $n{=}|\langle \mathrm{G} \rangle|$. The ECDSA specification defines three algorithms for the *key generation*, the *signature generation* and the *signature verification*, respectively.

The *key generation* algorithm selects a cryptographically strong random integer $d{\in}\mathbb{Z}_n{\setminus}\{0\}$ as private key, and a corresponding public key $(\mathbb{E}(\mathbb{F}_p), \mathrm{G}, n, \mathrm{Y})$ where $\mathrm{Y}{=}\,[d]\mathrm{G}$.

Algorithm 2.1: ECDSA SIGNATURE GENERATION

Globals: $\langle \mathrm{G} \rangle{=}(\mathbb{E}(\mathbb{F}_p), +)$, $n{=}|\langle \mathrm{G} \rangle|$, $\mathcal{H}$: hash function
Input: message, m; secret key, $d{\in}\mathbb{Z}_n{\setminus}\{0\}$
Output: signature token, (r, s) with $r, s{\in}\mathbb{Z}_n{\setminus}\{0\}$

```
1 begin
2     repeat
3         e←H(m), k ←Rand {1,...,n−1} /* e, k ∈ Z_n */
4         r←x-coord([k]G) mod n
5         s←(e + r d) k^{-1} mod n
6     until r=0 OR s=0
7 return (r, s)
```

The *signature generation* algorithm (see Algorithm 2.1) takes as input the private key, a message m, and produces a signature token (r, s), with $r, s{\in}\mathbb{Z}_n{\setminus}\{0\}$. The algorithm first obtains a hashed version e of m and a cryptographically strong random number $k{\in}\mathbb{Z}_n{\setminus}\{0\}$ that must be different in every run of the primitive. Subsequently, the scalar-point multiplication $[k]\mathrm{G}$ is performed and the `x`-coordinate of the resulting point is reduced modulo the order of the curve n to obtain the first part of the signature token, r (line **4**). The second part of the signature token, s, is computed through combining together the hash of the message e, the value r and the extracted random number k through computing one modular inversion, one modular addition, and two modular multiplications (line **5**). In case either $r{=}0$ or $s{=}0$ the procedure is re-run with a different k until an admissible signature is obtained.

The *verification algorithm* takes as input the message m, the signature token (r, s) and the public key $(\mathbb{E}(\mathbb{F}_p), \mathrm{G}, n, \mathrm{Y})$, $\mathrm{Y}{=}[d]\mathrm{G}$. The procedure first verifies that $r, s{\in}\mathbb{Z}_n{\setminus}\{0\}$, then computes $u_1{\leftarrow}\mathcal{H}(m)\, s^{-1} \bmod n$, $u_2{\leftarrow}r\, s^{-1} \bmod n$, and $v{\leftarrow}$`x-coord`$([u_1]\mathrm{G} + [u_2]\mathrm{Y}) \bmod n$ to return a positive validation of the signature token if and only if $v{=}r$.

2.3 Discrete Logarithm Problem

The mathematical security of the ECDSA signature generation algorithm is based on the hardness of the underlying ECDLP. The complexity of the logarithm problem largely depends on the considered algebraic group structure. Indeed, the best method to solve the DLP in the multiplicative group of a finite field, $\mathbb{F}_p^*$, is the "index calculus" method. This technique finds a relatively small *factor base* to express most of the group elements as products of elements in the factor base. The group of points on an elliptic curve $\mathbb{E}(\mathbb{F}_p)$ does not have the same "smoothness" of $\mathbb{F}_p^*$ thus, the factor base strategy cannot be applied. As a security measure, all the standardized curves were defined taking care of having a prime group order n. The best algorithms to solve the discrete logarithm problem in a generic finite cyclic group with prime order, like $(\mathbb{E}(\mathbb{F}_p), +)$, are the Baby-Step/Giant-Step (BSGS) method [22] and the Pollard's rho method [18].

Informally, the Pollard's rho algorithm involves guessing a random sequence of powers of the logarithm base, until two of them give the same value, while the BSGS method pre-computes an ordered list of powers and compares the value of another ordered sequence of powers against the former list to find a match. The spatial complexity of the BSGS method ($\mathcal{O}(\sqrt{n})$) makes this technique inconvenient when compared with the Pollard's rho algorithm, assuming no further information regarding the expected value of the logarithm is available.

In the next sections we will employ a DLP extraction routine to find a logarithm value which ranges in a predetermined interval. We note that, this a-priori knowledge about the range limits of the discrete logarithm is of no use with the Pollard's rho method (since its random walk among the powers of the logarithm base B is uniformly spread over the entire set of group elements), while the table lookup scheme of a BSGS strategy can be easily tailored to sweep a bounded range of values. Assuming to solve the DLP $\mathrm{Q}{=}[\delta]\mathrm{B}$, $0{\leq}\delta{\leq}M$, where Q, $\mathrm{B}{\in}\mathbb{E}(\mathbb{F}_p)$, $0{<}M{\ll}\sqrt{n}$, $n{=}|(\mathbb{E}(\mathbb{F}_p), +)|$, the optimized BSGS strategy considers $\delta{=}a\lceil\sqrt{M}\rceil{+}b$, $0{\leq}a, b{\leq}\lceil\sqrt{M}\rceil$ and formulates the problem as:

$$\underbrace{\mathrm{Q} - [b]\,\mathrm{B}}_{\mathrm{Baby-Step}} = \underbrace{[a]\,\big(\big[\lceil\sqrt{M}\rceil\big]\,\mathrm{B}\big)}_{\mathrm{Giant-Step}} \qquad (2)$$

A list of Baby-Steps is first computed and stored to be searched through a hash-based lookup strategy. Subsequently, the Giant-Steps are sequentially computed for each value $a{\in}\{0, \ldots, \lceil\sqrt{M}\rceil\}$ and checked against the table of Baby-Steps values. If a match occurs then the logarithm does exist and the values of a, b, and δ are easily recovered with an overall cost bounded by $\mathcal{O}(\sqrt{M})$ group operations.

2.4 Modular Arithmetic

In an elliptic curve cryptosystem the modular multiplication operations among the values of the point coordinates

Algorithm 2.2: Operand Scanning Multiplication

Input: $a=(a_{t-1},\ldots,a_0)_{2^w}$, $b=(b_{t-1},\ldots,b_0)_{2^w}$
Output: $c=a\,b=(c_{2t-1},\ldots,c_0)_{2^w}$

```
1  begin
2      (c_{2t-1},...,c_0)_{2^w} ← (0,...,0)_{2^w}
3      for j ← 0 to t−1 do
4          carry ← 0
5          for i ← 0 to t−1 do
6              (hi, lo)_{2^w} ← a_i × b_j
7              lo ← lo + carry
8              hi ← hi + (lo < carry)
9              lo ← lo + c_{i+j}
10             hi ← hi + (lo < c_{i+j})
11             c_{i+j} ← lo
12             carry ← hi
13         c_{j+t} ← carry
14     return c
```

account for the majority of the total execution time. Therefore, the performances of any implementation of this scheme heavily depend on the underlying speed of the finite field arithmetic operations. To achieve an efficient modular multiplication, the ECDSA standards [17] specify a prime number for the generation of the finite field of a set of recommended curves (e.g., $\mathrm{P}-192$). The primes are chosen with a specific form so that it is possible to execute a fast reduction procedure, i.e., $p=p_{t-1}(2^w)^{t-1}\pm,\ldots,\pm p_0(2^w)^0$, $p_i\in\{0,1\}$, $0\leq i\leq t-1$, where t is the number of w-bit processor's words composing the multiprecision integer, p. Indeed, the reduction operation is implemented as a few single precision additions among the words of the input operand. After performing the scalar-point multiplication (line **4** in Algorithm 2.1), the ECDSA signature primitive performs all the subsequent computations modulo the order of the curve n, which is a generic prime without any particular form. However, there are only a small number of operations to be performed modulo n, namely two multiplications, one addition and one inversion (line **5**, Algorithm 2.1). The field inversion is performed via Euclid's extended algorithm, thus avoiding the need to employ Montgomery's representation to compute it via exponentiation. This, in turn, results in the modular multiplications being done via a common multiple precision multiplication followed by a reduction made via trivial division algorithm. The operand scanning method reported in Algorithm 2.2 is the common multiprecision-multiplication strategy employed in the most adopted software libraries[2]. The algorithm outputs the product from the least significant word to the most significant word, one at each outer iteration via summing the outcomes of equal order single-precision products, $(\mathtt{hi},\mathtt{lo})_{2^w}$ (see line **6**) and properly propagating the single-precision carry values.

3. FAULT MODEL

A fault induction technique not spatially precise enough to limit the impact of the alteration in the computation will most likely cause a multiple bit flip in one of the intermediate values. This kind of hazard in a computation is commonly attainable through a number of technical means, for instance the ones described [3, 19].

We consider the effects of faults injected into the ECDSA signature generation primitive (see Algorithm 2.1) targeting the multiprecision modular multiplication executed during the computation of the second part, s, of the signature token (see line **5** in Algorithm 2.1).

The considered faults are modeled as a random change in one of the two single precision operands employed during the execution of the operand scanning multiplication strategy, within a single iteration of the nested-loop structure. The faulted multiplication outcome and the relative multiplication error exploited in our attack are more formally stated as follows.

Definition 1 (Faulted Multiplication). *Let a, b be two multiprecision integers composed by t processor words with w-bit size: $a=(a_{t-1},\ldots,a_0)_{2^w}$, $b=(b_{t-1},\ldots,b_0)_{2^w}$, and $c=a\,b=(c_{2t-1},\ldots,c_0)_{2^w}$ be the result of a multiprecision multiplication computed following Algorithm 2.2.*

*A faulted multiplication is defined as the value computed through Algorithm 2.2 when a change is induced in one word of an input factor during a single iteration (i,j) of the loop nest, with $i,j\in\{0,\ldots,t-1\}$, just before the execution of the single-precision multiplication operation (line **6** in Algorithm 2.2).*

The knowledge of the loop indices of the nested-loop structure where the fault is injected enables the attacker to deduce a precise characterization of the multiplication error.

Definition 2 (Multiplication Error). *Let a, b be two multiprecision integers composed by t processor words with w-bit size: $a=(a_{t-1},\ldots,a_0)_{2^w}$, $b=(b_{t-1},\ldots,b_0)_{2^w}$, and $c=a\,b=(c_{2t-1},\ldots,c_0)_{2^w}$ be the result of a multiprecision multiplication computed following Algorithm 2.2.*

A multiplication error is defined as the integer value given by the difference between the faulty ($\tilde{c}$) and faulty-free (c) multiprecision multiplication outcomes:

$$\tilde{c} = c \pm \mathtt{MulError}$$

If the multiprecision multiplication algorithm is faulted during the specific (i,j) loop nest iteration, with $i,j\in\{0,\ldots,t-1\}$, the multiplication error is expressed as

$$\mathtt{MulError}=\begin{cases}(\mathtt{emf}\times a_i)\,(2^w)^{i+j}, & \text{when } b_j \text{ is altered}\\ (\mathtt{emf}\times b_j)\,(2^w)^{i+j}, & \text{when } a_i \text{ is altered}\end{cases}$$

where $\mathtt{emf}\in\{1,\ldots,2^w-1\}$ is a random multiplication factor.

In our attack scenario the ECDSA signature generation routine is considered. In particular, we will refer to the multiprecision multiplication employed to compose the second part of the signature (line **5** in Algorithm 2.1) combining $r=(r_{t-1},\ldots,r_0)_{2^w}$ with the secret key $d=(d_{t-1},\ldots,d_0)_{2^w}$.

The faulty signature obtained, when the operation $r\,d$ is affected by an hazard on an operand of a specific single precision multiplication, can be expressed as the pair $(r,\tilde{s})$, where $\tilde{s}=s\pm\mathtt{MulError}\,k^{-1} \bmod n$; in particular:

$$\tilde{s}=s\pm\big((\mathtt{emf}\times d_i)\,(2^w)^{i+j}\big)\,k^{-1}\bmod n,\quad i,j\in\{0,\ldots,t-1\} \qquad (3)$$

$$\tilde{s}=s\pm\big((\mathtt{emf}\times r_i)\,(2^w)^{i+j}\big)\,k^{-1}\bmod n,\quad i,j\in\{0,\ldots,t-1\} \qquad (4)$$

depending on whether the fault damaged either r, (3) or d, (4). The faults exploitable for the secret key retrieval process, are only the ones described by equation (3). In the next section we will see how to distinguish them from the ones described by equation (4), via observing that the whole value of r is known to the attacker, since it is a part of the signature.

[2]In the following sections, for the sake of clarity, we will denote a single-precision multiplication between factors with w-bit size as $\times$, as reported at line **6** of Algorithm 2.2

We note that in case any word among d_i and r_i in the former equations is equal to zero, the output of the signature generation routine will be correct instead of erroneous, i.e. $\tilde{s}{=}s$. However, none of the two possibilities pose an issue to the recovery of the whole key. On one hand, the fact that the value of r changes at each signature generation avoids the possibility of having an r_i taking always a zero value. On the other hand, the possible zero values taken by some words, d_i, of the secret key can be dealt with via initializing the guessed secret key words to zero and checking at the retrieval of each word of the secret key whether the value of the whole key d generates a valid public key.

4. ATTACK DESCRIPTION

The attack is formulated with an *on-line* strategy which extracts information about one word d_i of the secret key $d{=}(d_{t-1},\ldots,d_0)_{2^w}$ at a time, and repeatedly collects faults until the whole value is revealed. However, the actual physical collection of the faulty signatures may be easily decoupled from the key retrieval procedure, thus reducing the time during which the opponent needs to seize the secure device. For the sake of clarity, the attack will be described as collecting faulty signatures from the same message. Since this hypothesis is not used in the attack, it is possible to employ signatures coming from different messages without any penalty. Moreover, the key recovery procedure does not rely on knowing the value of a correct signature of the message m. This is particularly appropriate, since the ECDSA signature generation algorithm mandates the use of a random nonce k for every run of the signature routine, thus effectively yielding a different signature every time.

4.1 Secret Key Retrieval Algorithm

Algorithm 4.1 acts by recovering secret key related information through injecting faults during a specific iteration of the loop nest of the multiprecision multiplication, $r\,d$, involved in the signature generation process (see line **5** of Algorithm 2.1).

Information related to one word of the secret key is then extracted from the analysis of any faulty result. Through collecting a number of faults related to each word, it is eventually possible to reveal the whole value d. The end is reached when the guessed d correctly yields the known associated public key, i.e., when $[d]\mathrm{G}{=}\mathrm{Y}$. Through injecting a fault during a single-precision multiplication within the nested-loop iteration (i, j) with $i{=}j{=}\mathtt{ind}$, $\mathtt{ind}{\in}\{0,\ldots,t{-}1\}$, it is possible to obtain a faulty result, δ, carrying information about either the word $d_{\mathtt{ind}}$ or $r_{\mathtt{ind}}$, depending on the position of the fault, as explained in Section 3. A subsequent check on it will distinguish the two cases and keep the value δ only when it is recognized as a damage on the $r_{\mathtt{ind}}$ factor, in such a way to exploit equation (3) and, subsequently, derive the key word $d_{\mathtt{ind}}$. Assuming that a change of value in the word $r_{\mathtt{ind}}$ has been caused, a careful rewriting of equation (3) allows a "reduced" ECDLP to be formulated. Indeed, starting from the following equation

$$\left((\mathtt{emf}\times d_{\mathtt{ind}})\,(2^w)^{2\mathtt{ind}}\right)\tilde{s}^{-1} = \pm(k - e\tilde{s}^{-1} - rd\tilde{s}^{-1}) \bmod n$$

and considering both members as coefficients in a scalar-point multiplication by the curve generator G, we obtain:

$$\underbrace{[\mathtt{emf}\times d_{\mathtt{ind}}]}_{\text{discrete log }\delta}\underbrace{[(2^w)^{2\mathtt{ind}}\,\tilde{s}^{-1}]\,\mathrm{G}}_{\text{base point B}} = \underbrace{\pm(\hat{\mathrm{P}} - [e\tilde{s}^{-1}]\mathrm{G} - [r\tilde{s}^{-1}]\mathrm{Y})}_{\text{discrete log argument point Q}}$$

Algorithm 4.1: SECRET-KEY RETRIEVAL

Globals: n: order of the group, $\mathrm{n}{=}\langle\mathrm{G}\rangle{=}|\,(\mathbb{E}(\mathbb{F}_p),+)\,|$; w: processor word size; t: number of words to represent $\mathbb{Z}_n$ elements, $t{=}\left\lceil\frac{\lceil \lg_2 n\rceil}{w}\right\rceil$
Input: public key, $\mathrm{Y}{=}[d]\mathrm{G}{\in}\mathbb{E}(\mathbb{F}_p)$
Output: value of the private key, $d{\in}\mathbb{Z}_n$ $d{=}(d_{t-1},\ldots,d_0)_{2^w}, 0{\le}d_{\mathtt{ind}}{\le}2^w{-}1,\ 0{\le}\mathtt{ind}{\le}t{-}1$

```
1  begin
2      d=(d_{t-1},...,d_0)_{2^w} ← (0,...,0)_{2^w}
3      m ←^{Rand} {0,1}*;  e ← H(m)/* e ∈ Z_n          */
4      ind ←0
5      while [d]G≠Y do
6          (r, s̃) ← FAULTED SIGN(m, ind)
7          foreach P̂∈{(x,y)∈⟨G⟩ : x mod n=r} do
8              Q ← P̂−[e s̃^{-1}]G−[r s̃^{-1}]Y
9              B ← [(2^w)^{2·ind} s̃^{-1}]G
10             δ ← Optimized BSGS(B, Q) /* δ=emf×d_ind   */
11             if δ ≠ ⊥ AND r_ind ∤ δ then
12                 d_ind ← GCD(d_ind, δ)
13                 break
14         ind ←(ind+1) mod t
15     return d
```

where $\hat{\mathrm{P}}$ is one of the possible curve points having the same $\mathtt{x}$-coordinate of the unknown point $[k]\mathrm{G}$ (see line **4** in Algorithm 2.1), as described by Lemma 1.

The reduced ECDLPs formulated above (one for each value of the right-hand side of the relation) can be efficiently solved, employing a BSGS strategy, thanks to the observation that the discrete logarithm value δ is an integer in the range $\{0,\ldots,M{-}1\}$, with $M{=}2^{2w}$, since $M{-}1$ is the maximum value that a single-precision multiplication may yield (see Section 2). An optimized implementation of the BSGS method can try to solve the two ECDLP instances (depending on the sign of Q) through coupling them together. Starting from the basic description of the BSGS in Section 2.3, we can rewrite equation (2) as: $[b]\,\mathrm{B}{=}\mathrm{Q}{-}[a]\,([\sqrt{M}]\,\mathrm{B})$, with $a,b{\in}\{0,\ldots,\sqrt{M}\}$. This form of the equation is amenable to be computed faster since, it is possible to match each Baby-Step value with the Giant-Step values corresponding to both $\mathrm{Q}{-}[a]\,([\sqrt{M}]\,\mathrm{B})$ and $-\mathrm{Q}{-}[a]\,([\sqrt{M}]\,\mathrm{B})$, at the cost of only one additional elliptic curve point operation. After obtaining a discrete logarithm δ, the computation of $d_{\mathtt{ind}}$ is carried out through a sequence of GCD operations among the set of values of the form $\delta{=}\mathtt{emf}\times d_{\mathtt{ind}}$, with $\mathtt{emf}$ being a random w-bit value.

The secret key recovery procedure is detailed in Algorithm 4.1, which takes as input the public key Y and the public parameters of the employed elliptic curve, and outputs the value of the secret key d. As a first initialization step, the algorithm sets the value of all the words $d_{t-1},\ldots,d_0$ of the key hypothesis d to zero (line **2**). Subsequently, it draws a random message m from the acceptable message space and computes its hash e (line **3**). The algorithm will recover every word of the secret key through injecting a fault in the single-precision multiplication between two words indexed by the same value, $\mathtt{ind}$, which will take all the values from 0 to $t{-}1$ (line **4**). While the value of the key hypothesis is not correct (line **5**) the Algorithm gathers new information about the secret key via obtaining a flawed signature respecting the fault model defined in Section 3. The FAULTED SIGN primitive (line **6**) takes as inputs the message m and the index $\mathtt{ind}$ to inject a fault in the single-precision mul-

tiplication $r_{\mathtt{ind}} \times d_{\mathtt{ind}}$, computed during the execution of the signature generation procedure when the multiplication $r\,d$, implemented following Algorithm 2.2, occurs. Given a faulty signature, $(r, \tilde{s})$, the attack algorithm seeks the value of the multiplication error occurred in the computation (lines **8**–**10**) through evaluating each possible value for the point $\hat{\mathrm{P}}$ (line **7**). The first step is to compute the value of both the logarithm argument Q (line **8**) and the base point B (line **9**) from the collected signature and the elliptic curve public parameters. Employing such points, the OPTIMIZED BSGS subroutine computes the discrete logarithm value δ which contains the useful information regarding the secret key word $d_{\mathtt{ind}}$. If the logarithm exists (i.e., $\delta \neq \perp$), to distinguish whether the hazard caused a multiplication error ($\mathtt{MulError} = \delta\,(2^w)^{\mathtt{ind}+\mathtt{ind}}$) with $d_{\mathtt{ind}}$ as a factor or not, it is sufficient to check the divisibility of δ by $r_{\mathtt{ind}}$ (line **11**). Note that, if $r_{\mathtt{ind}}$ divides $d_{\mathtt{ind}}$ the algorithm discards an otherwise exploitable value of δ; however, this does not hinder the key recovery process as $r_{\mathtt{ind}}$ will change at each faulty signature generation. The hypothesis for the $d_{\mathtt{ind}}$ word is updated with the greatest common divisor between the current $d_{\mathtt{ind}}$ value and δ (line **12**). In such a way, as we will demonstrate in the following section, the algorithm is able to recover, with high probability, the actual value of the secret key d, through collecting only 3 or 4 exploitable faults for each value of `ind` (i.e., for each secret key word). Before checking if the updated hypothesis for the secret key d is correct (see loop condition at line **5**), the procedure increments the index of the targeted single-precision multiplication (line **14**) preparing the state for the retrieval of another secret key word. When the public key is correctly derived from the current hypothesis of the secret key d, the algorithm ends (line **15**).

4.2 Complexity Analysis

The computational cost required to lead the fault attack previously described is formally expressed by the following propositions.

PROPOSITION 1 (KEY WORD RECOVERY). *Given a fault injection technique able to correctly put the fault model in Section 3 into effects, and given $\eta_{\mathtt{ind}}$ as the number of different faults injected on the w-bit word with index* `ind` *of the targeted single-precision multiplication of the ECDSA signature generation primitive (Algorithm 2.1) the recovery of the correct value of the corresponding secret key word is obtained in Algorithm 4.1, with a probability of 99%, employing $\eta_{\mathtt{ind}}$=4 faults.*

PROOF. To obtain the correct value of the secret key word d_{ind}, Algorithm 4.1 computes the GCD among different discrete logarithms $\delta = \mathtt{emf} \times d_{\mathtt{ind}} \in \{0, \ldots, (2^w)^2 - 1\}$, thus eliminating the random value `emf`. The correct value of d_{ind} will be therefore recovered when at least two values of `emf` are co-prime. A well known result in number theory [10], asserts that the probability $p_{\mathtt{co}}$ that two positive integers (≥ 2), chosen uniformly at random, are co-prime, ranges in the interval $[\frac{1}{2}, \frac{6}{\pi^2})$. This, in turn, implies that the probability of obtaining at least one pair of co-prime values, after $\eta_{\mathtt{ind}}$ faults have been collected, amounts to $p_{\mathtt{ok}} = 1 - (1 - p_{\mathtt{co}})^{\binom{\eta_{\mathtt{ind}}}{2}}$. Willing to make a conservative assumption, choosing $p_{\mathtt{co}} = \frac{1}{2}$, the original value of $d_{\mathtt{ind}}$ can be obtained with only $\eta_{\mathtt{ind}}$=4 faults with a probability p_{ok}=0.99. However, for numbers greater than 15 (i.e., w>4) the probability that two of them have no common factors quickly increases up to $p_{co} \geq 0.6$. Thus, keeping $p_{co} = \frac{6}{\pi^2}$ gives a fault number $\eta_{\mathtt{ind}}$=3, which is better suited in practice for any realistic architecture word size. □

PROPOSITION 2 (COMPLEXITY). *Given a fault injection technique able to correctly put the fault model in Section 3 into effects, the secret key retrieval procedure described in Algorithm 4.1 recovers the t w-bit words of the private key used in an ECDSA signature primitive (on average) in $\mathcal{O}(t\,\eta_{\mathtt{ind}}\,(3\,2^w + \frac{3}{2}w))$ elliptic curve operations.*

PROOF. Each call to the OPTIMIZED BSGS subroutine (line **10**) has an average case complexity of $\mathcal{O}(3\,2^w + \frac{3}{2}w)$ elliptic curve point operations (see Section 4.1).

Indeed, the OPTIMIZED BSGS subroutine is called as many times as the values for the candidate points $\hat{\mathrm{P}}$ having their reduced `x`-coordinate equal to the first part of the faulty signature, $(r, \tilde{s})$ (line **7**). The set of candidate points includes at most four values for $\hat{\mathrm{P}}$, as mentioned in Lemma 1. However, the probability that there are actually four candidates depends on how much smaller the curve order n can be with respect to the modulus p of the corresponding finite field. The actual probability of having four points, when considering the elliptic curves recommended in the ECDSA standard is $\frac{\sqrt{p}-1}{n} \approx \frac{1}{\sqrt{p}} < 2^{-96}$. Therefore, it is safe to consider the number of candidate points to be always 2, and the probability of choosing the correct $\hat{\mathrm{P}}$ at first to be 1/2.

As stated in Proposition 1, the algorithm collects $\eta_{\mathtt{ind}}$ values for each of the t words of the secret key prior to finish, thus the computational complexity of the whole procedure is $\mathcal{O}(t\,\eta_{\mathtt{ind}}\,(3\,2^w + \frac{3}{2}w))$.

We omitted the complexity of the calls to the GCD subroutine as this is negligible with respect to the computational complexity of a single call of the BSGS algorithm. □

5. EXPERIMENTAL EVALUATION

We implemented a serial version of the secret key retrieval procedure shown in Algorithm 4.1 on an Intel Core i7 920 CPU clocked at 2.66 GHz, with 12 GB DDR3-1600 main memory, employing the Sage toolkit[3] running on a x86-64 Gentoo Linux 2011.3 with a 2.6.37 kernel.

Tables 1 and 2 report the performance figures of the key retrieval algorithm applied to the elliptic curves recommended by the ECDSA standards. The standard curves are referred to by the size of the underlying finite field which also indicates the order of magnitude of the cyclic group represented by the set of elliptic curve points (e.g., the descriptive parameters of the P−192 curve are: a finite field $\mathbb{F}_p$ with $p = 2^{192} - 2^{64} - 1$, and a group order $n \approx 2^{192}$).

The tables refer to architectures with processor word size w=8-bit and w=16-bit, respectively. These word-sizes are commonly used in embedded systems where the ECDSA provides strong authentication of the device, such as anti-counterfeiting RFID tags [11].

The experimental evaluation has been lead averaging the performance figures out of 30 runs for each combination of processor word size w and elliptic curve. The first column of the tables report the average number of faults needed to recover the whole secret key. The results match the expectations reported in the complexity analysis section and are proportional to the number of words composing the multi-precision value of the secret key. The second column shows

[3]William A. Stein et al. Sage Mathematics Software (ver.4.6.1), The Sage Development Team, 2011. `http://www.sagemath.org`

Employed Curve, $\mathbb{E}(\mathbb{F}_p)$	Avg. No. of Faults	BSGS Time [ms]	Total Attack Time [s]	Storage [kiB]
P−192	72	50.6	10.9	6.144
P−224	84	57.0	14.4	7.168
P−256	96	61.7	17.8	8.192
P−384	144	95.0	41.0	12.288
P−521	198	137.0	81.4	16.672

Table 1: Attack performance when considering a target architecture with processor word size w=8-bit. The number of words, t, depends on the selected elliptic curves, with $t \in \{24, 28, 32, 48, 66\}$

Employed Curve, $\mathbb{E}(\mathbb{F}_p)$	Avg. No. of Faults	BSGS Time [s]	Total Attack Time [s]	Storage [MiB]
P−192	36	6.17	666	1.573
P−224	42	6.46	814	1.835
P−256	48	6.76	973	2.097
P−384	72	8.45	1830	3.146
P−521	99	7.27	2160	4.268

Table 2: Attack performance when considering a target architecture with processor word size w=16-bit. The number of words, t, depends on the selected elliptic curves, with $t \in \{12, 14, 16, 24, 33\}$

the time required by a single call to the BSGS routine to possibly solve an ECDLP, while the third column reports the average execution time of the entire attack procedure. Finally, the last column provides the memory fingerprint required by the attack, where the main part of it is taken by the tables computed in the BSGS subroutine.

We note that carrying out a key recovery attack against an implementation of the ECDSA employing any standard curve on 8- and 16-bit architectures can be managed within negligible time on a common desktop. It is interesting to notice that the finite field size, i.e. the usual parameter considered for the security level of the cryptosystem, has only a minimal effect on the feasibility of the attacks, as it occurs as a linear term in the computational complexity of the presented fault attack. This in turn implies that raising the field size, and consequentially the key length, has only a minimal effects in terms of mitigation of our attack.

When considering platforms with processor word size w=32-bit, the computational complexity of the attack raises significantly as a consequence of the exponential complexity of the BSGS method. In such a case, our current implementation would give projected space and time performance between 18.7 Ms and 68.4 Ms as running time and between 103.1 GiB and 279.7 GiB as storage. These timings and storage requirements are still within acceptability, when considering a single commodity desktop. However, due to the nature of the BSGS algorithm, it is possible for an attacker to speed up the recovery procedure through exploiting the native parallelism offered by it. In particular modern Graphics Processing Units (GPU) offer a cheap and easily available many-core environment, particularly well suited to this kind of computational effort. Open literature reports speedups of an order of magnitude for each GPU employed in the computation [6,14], provided that the brute force tasks do not need to communicate with each other. The presented attack can be parallelized both at secret key word level, i.e. assigning the computation of a single word (out of t secret key words) to a different computation node, and splitting the BSGS steps equally among the available GPU cards. Consequentially, it is possible for the attacker to achieve a speedup factor of $(10\,t\,\#\texttt{GPUcards})$, bringing the attack against the ECDSA primitive implemented on a 32-bit architecture within feasibility in a few days, while exploiting commodity hardware. For instance, through employing $\#\texttt{GPUcards}$=8 GPU cards, the required time for the attack can be reduced to ten days for the P−521 curve.

6. RELATED WORK

Developing fault injection techniques to attack Elliptic Curve Cryptosystems (ECC) proved to be more difficult than attacking factoring-based ciphers due to the higher complexity of the mathematical operations involved. Most of the proposed attacks exploit the structural similarity between the modular exponentiation computed through square-&-multiply used in RSA and the scalar-point multiplication computed through double-&-add method [2]. In [7, 8] the authors propose injecting a fault into the public parameter of the elliptic curve cryptosystem. This way, the intermediate computations involving the secret key d will possibly operate in the set of points of another elliptic curve, whose cardinality may be lower than the original one, thus making it possible to attempt a brute force approach. Another attack, directly targeted at the ECC algebraic structure, is described in [9], 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, thus simplifying a brute force approach to the ECDLP. The open literature provides few examples of fault attacks aimed at the secret key retrieval from the ECDSA signature generation algorithm. The attack presented in [13] relies on faulting the modulus used in the arithmetic computations, but needs a thousands of faulty results to be successful against a system employing even a small-sized curve. A glitch attack is used in [16] to recover the DSA secret key from a set of faulty signatures obtained through zeroing some of the least significant bytes of the random nonce k. A lattice reduction technique (based on the so-called Hidden Number Problem) is employed to recover the secret key from only 27 faulty signatures having the least significant bytes reset to zero. In [20] an instruction skip fault is used to recover some bits of the random nonce k, employed by the ECDSA signature generation routine. Subsequently, the collected faulty signatures with the bits of the corresponding nonces, are intended to be used in a lattice-attack to recover the secret key. A few dozens of faulty results are again sufficient for a small-sized curve, but an efficient countermeasure is also presented. In our attack, we exploit an information leakage caused by the multiplication operations performed modulo n, without disturbing any of the parts of the scalar point multiplication, or the known parameters of the elliptic curve, with the interesting point to shift the security considerations about the cryptosystem to the underlying architectural feature related to the size of the processor word size. Indeed, the exploited fault model can be easily put into effects through low-cost fault injection techniques as clock glitching or the underfeeding of the power-supply.

7. CONCLUSION

The presented attack effectively exploits the faulty computation of an ECDSA cryptosystem implemented on an

embedded platform to reveal the secret key securely stored inside it. The defined fault model allows the underlying ECDLP of the cryptosystem to be mapped on a sequence of discrete logarithms defined on a much smaller domain. The proposed key retrieval algorithm has a complexity that mainly depends on the processor word size and logarithmically on the size of the underlying finite field. Indeed, the experimental results show how the choice of elliptic curve parameters with a greater security level does not bring any significant benefit with respect to the proposed attack (whereas the main security parameter is the bit-size of the word processor). There are several low-cost fault injection technologies described in open literature that can put into effect the defined fault model.

Acknowledgment

This work was supported in part by the ENIAC Joint Undertaking, within the Trusted Computing for European Embedded Systems (TOISE) project, call ENIAC-2010-1, proposal number 282557-2.

8. REFERENCES

[1] American National Standards Institute (ANSI). Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). Standard ANSI ANSI X9.62:2005, 2005.

[2] 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*, pages 115–124, 1998.

[3] A. Barenghi, G. Bertoni, E. Parrinello, and G. Pelosi. Low Voltage Fault Attacks on the RSA Cryptosystem. In *Fault Diagnosis and Tolerance in Cryptography*, pages 23–31. IEEE CS, 2009.

[4] A. Barenghi, G. M. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi. Low Voltage Fault Attacks to AES. In *Proc. of the 3rd Annual IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2010.

[5] A. Barenghi, G. M. Bertoni, A. Palomba, and R. Susella. A Novel Fault Attack Against ECDSA. In *4th Annual IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2011.

[6] A. D. Biagio, A. Barenghi, G. Agosta, and G. Pelosi. Design of a parallel AES for graphics hardware using the CUDA framework. In *IPDPS*, pages 1–8. IEEE, 2009.

[7] I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. *In Proc. CRYPTO*, pages 131–146, 2000.

[8] M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Des. Codes Cryptography*, 36(1):33–43, 2005.

[9] 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*, pages 92–98, 2008.

[10] G. Hardy. *An Introduction to the Theory of Numbers.* Oxford Science Publications. Oxford Press, fifth edition, 1979.

[11] M. Hutter, M. Feldhofer, and J. Wolkerstorfer. A Cryptographic Processor for Low-Resource Devices: Canning ECDSA and AES like Sardines. In Springer, editor, *Information Security Theory and Practices - WISTP 2011, 5th International Workshop, Heraklion, Greece, June 1-3, 2011, Proceedings.*, volume 6633 of *Lecture Notes in Computer Science*, pages 144 – 159, 2011.

[12] Institute of Electrical and Electronics Engineers. Specifications For Public Key Cryptography. Standard IEEE 1363-2000, 2000, `http://grouper.ieee.org/groups/1363/P1363/index.html`.

[13] M. Kara-Ivanov, E. Iceland, and A. Kipnis. Attacks on Authentication and Signature Schemes Involving Corruption of Public Key (Modulus). In *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 108–115. IEEE CS, 2008.

[14] D. Kirk, W. Hwu, and W. Hwu. *Programming massively parallel processors: a hands-on approach.* Applications of GPU Computing Series. Morgan Kaufmann Publishers, 2010.

[15] I. Koren. *Computer arithmetic algorithms.* A K Peters, Ltd., 2002.

[16] D. Naccache, P. Nguyễn, M. Tunstall, and C. Whelan. Experimenting with Faults, Lattices and the DSA. In S. Vaudenay, editor, *Public Key Cryptography - PKC 2005*, volume 3386 of *LNCS*, pages 16–28. Springer Berlin / Heidelberg, 2005.

[17] National Institute of Standards and Technology (NIST) - U.S. Department of Commerce. Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-3, National Technical Information Service, Springfield, Virginia, USA 2009, `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`.

[18] J. M. Pollard. Theorems on factorization and primality testing. *Proc. of the Cambridge Philosophical Society*, 76:521–528, 1974.

[19] J.-M. Schmidt and C. Herbst. A Practical Fault Attack on Square and Multiply. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *FDTC*, pages 53–58. IEEE CS, 2008.

[20] J.-M. Schmidt and M. Medwed. A Fault Attack on ECDSA. In D. Naccache and E. Oswald, editors, *6th Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2009, Proc.*, pages 93 – 99. Verlag IEEE-CS Press, 2009.

[21] N. Selmane, S. Guilley, and J.-L. Danger. Practical Setup Time Violation Attacks on AES. In *Seventh European Dependable Computing Conference*, pages 91–96, Washington, DC, USA, 2008. IEEE CS.

[22] D. Shanks. Class number, a theory of factorization and genera. *Proc. of Symposia on Pure Mathematics, American Mathematical Society*, 20:415–440, 1971.

[23] S. P. Skorobogatov and R. J. Anderson. Optical Fault Induction Attacks. In *CHES '02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 2–12, London, UK, 2003. Springer-Verlag.

[24] C. D. Walter. Systolic Modular Multiplication. *IEEE Trans. Computers*, 42(3):376–378, 1993.

[25] L. C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition.* Chapman & Hall/CRC, 2 edition, 2008.