Fault Attacks on the Elliptic Curve ElGamal Cryptosystem

Martin Kreuzer
Fakultät für Informatik und Mathematik
Universität Passau
Passau, Germany
Martin.Kreuzer@uni-passau.de

Nourhan Elhamawy
Institut für Technische Informatik
Universität Stuttgart
Stuttgart, Germany
NourhanElhamawy@hotmail.com

Maël Gay
Institut für Technische Informatik
Universität Stuttgart
Stuttgart, Germany
Mael.Gay@informatik.uni-stuttgart.de

Ange-S. Messeng Ekossono
Fakultät für Informatik und Mathematik
Universität Passau
Passau, Germany
Ange-Salome.MessengEkossono@uni-passau.de

Ilia Polian
Institut für Technische Informatik
Universität Stuttgart
Stuttgart, Germany
Ilia.Polian@informatik.uni-stuttgart.de

ABSTRACT

Hardware implementations of advanced cryptographic schemes gain in importance for emerging cyber-physical and autonomous systems, and their resistance against physical attacks is becoming a central requirement. This paper studies fault-injection attacks against the private key of the Elliptic Curve ElGamal cryptosystem. It extends previously introduced bit and byte fault models by models that assume faults in arbitrary s-bit portions (subtuples) of the key. We provide a mathematical proof that characterizes the set of subtuple candidates after a fault injection affecting an arbitrary number of bits s. The proof reinforces earlier findings for $s = 8$ and implies that the number of key subtuple candidates grows exponentially in s. We also report on fault-injection experiments, both on the software level and using an optimized hardware implementation for NIST-recommended elliptic curves.

KEYWORDS

Fault attack, fault injection, elliptic curve cryptography, ElGamal cryptosystem

ACM Reference Format:
Martin Kreuzer, Nourhan Elhamawy, Maël Gay, Ange-S. Messeng Ekossono, and Ilia Polian. 2019. Fault Attacks on the Elliptic Curve ElGamal Cryptosystem. In *Proceedings of Preprint 2019*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The ongoing transition to cyber-physical and autonomous systems has led to a strong need for high-performance and yet low-cost security solutions for such systems. As a consequence, hardware implementations of cryptographic primitives are becoming increasingly attractive due to their high speed and low energy requirements. A central requirement for cryptographic hardware is its resistance against physical attacks, both passive side-channel analysis [28, 32] and active fault-injection attacks [7, 35]. To design appropriate countermeasures and validate their effectiveness, it is necessary to accurately understand the threat and the potential of an adversarial party to mount an attack.

In this paper, we investigate fault-injection attacks on the *Elliptic Curve ElGamal* (ECEG) cryptosystem. Elliptic curve cryptosystems are attractive for hardware implementations thanks to their reduced key size compared to other, more widely used public-key cryptosystems, such as RSA. The majority of prior works on fault-based cryptoanalysis focused on symmetric schemes and were able to break many popular ciphers using very few fault injections, e.g., [11]. However, asymmetric schemes were attacked as well, requiring a larger number of injections (as we will discuss further below).

The first fault attacks on ECEG [8] extended the ideas developed in [10, 12, 31] for RSA to the elliptic curve setting. The main idea of these attacks was to feed invalid points into the decryption algorithm such that the output values reveal information about the secret key. A similar attack was also proposed in [21]. Dottax [17] extended the attack of flipping a single bit of the secret key from RSA [5] to ECEG. The attacks in [14] generalized the findings of [8] to random and unknown errors in the elliptic curve parameters and the base point. The fault attack proposed by Fouque et al. in [21] was tailored to the classical Montgomery ladder method for scalar multiplication.

The attacks mentioned above produce intermediate points which are not contained in the given elliptic curve, or move from a secure elliptic curve to a weaker curve. A simple countermeasure against these attacks is to check whether the input and output points are all on the given elliptic curve. A number of attacks circumvent this countermeasure by using faults that result in manipulated values that still remain on the original elliptic curve. Blömer et al. [9] propose to change the sign of an intermediate variable during scalar multiplication. (This attack does not apply if the decryption

Preprint 2019, July 17, 2019, Passau

algorithm employs Montgomery's scalar multiplication algorithm [33] which does not use the y-coordinate.) Giraud and Knudsen [23] inject faults directly into the secret key; their analysis was subsequently improved in [24]. Kim et al. [27] manipulate the public key. Romailler and Pelissier [37] propose an attack against Edwards-curve based signature schemes.

This paper proposes an extension of the attacks based on fault injections into the ECEG cryptosystem's secret key [17, 23, 24], improves their theoretical analysis, and discusses their practical realization in hardware. These attacks are relevant because they are immune to the above-mentioned detection countermeasures, since all affected values stay on the original elliptic curve. The attacks discussed in this work are based on fault models with different assumptions on the adversary's capabilities to inject faults. One can generally distinguish between low-cost, low-precision injection techniques (like under-powering or inducing clock glitches [6]) and elaborate optical [40] and electromagnetic [16] approaches with a much better spatial and temporal resolution.

Previous papers considered two fault models: the bit fault model [17] where precisely one bit of the word changed its value, and the byte fault model [23, 24] where random and unknown modifications were restricted to one byte. Faults according to the second model are easier to achieve using practical equipment, but they require more elaborate mathematical post-processing and a larger number of injections. In this paper, we also consider a generalization of the byte fault model where random and unknown fault effects are restricted to an s-bit portion of the word under attack. (Here $s = 8$ corresponds to the byte fault model.) We provide an analysis for the number of fault injections needed to obtain one candidate for an s-bit portion of the secret key. It turns out that this number rises exponentially in s, yielding an interesting trade-off: choosing a smaller s requires (linearly) more chunks to be attacked, but the required number of injections is exponentially smaller. This suggests that as small an s as can be reliably supported by the fault-injection equipment should be used. As an interesting side-result, our analysis reinforces the bound shown in [24] for $s = 8$ and confirms that this bound is tight.

The theoretical analysis is complemented by fault-injection experiments performed by simulation on the hardware level. For the purpose of these experiments, we implemented a circuit for ECEG, including efficient realizations of point multiplication, inversion and division. The outcomes of the simulated fault injections (in hardware and in software) for some NIST-recommended elliptic curves over finite fields of large prime order confirm the theoretical findings.

The remainder of the paper is organized as follows. The next section provides an overview of the ECEG cryptosystem. Section 3 introduces the two fault models considered in this paper. Section 4 describes the attacks, models them and studies their properties by mathematical means. Results for the software implementation and for a specially designed hardware circuit are reported in Section 5. Finally, Section 6 concludes the paper.

2 THE ELLIPTIC CURVE ELGAMAL CRYPTOSYSTEM

Throughout this paper, let $p > 3$ be a prime number, and let $\mathbb{F}_p$ be the finite field having p elements. An *elliptic curve* E over $\mathbb{F}_p$ is a smooth cubic curve in $\mathbb{P}^2(\mathbb{F}_p)$ which has at least one $\mathbb{F}_p$-rational point. After introducing a suitable system of coordinates, the curve E is given by a short Weierstraß equation

$$E:\ y^2 \ =\ x^3 + \bar{a}x + \bar{b}$$

where the discriminant $\Delta = -4\bar{a}^3 - 27\bar{b}^2$ is non-zero and where we consider E as embedded in the affine plane $D_+(x_0)$ and having one point at infinity, namely $O = (0:0:1)$. Notice that the use of the short Weierstraß form is not mandatory for the contents of this paper, and that everything can be easily adjusted to other standard forms, e.g., to Edwards or Montgomery curves.

The set of $\mathbb{F}_p$-rational points of E is given by

$$E(\mathbb{F}_p) \ =\ \{(x,y) \in \mathbb{F}_p^2 \mid y^2 = \bar{a}x + \bar{b}\} \ \cup\ O.$$

It is a finite abelian group with neutral element O, with the inverse of a point $P = (x,y)$ given by $-P = (x,-y)$, and with the addition $P_3 = P_1 + P_2$ given by

$$x_3 = \lambda^2 - x_1 - x_2 \quad \text{and} \quad y_3 = (x_1 - x_3)\lambda - y_1$$

where $P_i = (x_i, y_i)$ and $P_2 \neq -P_1$. Here we let $\lambda = \frac{y_1 - y_2}{x_1 - x_2}$ if $P_2 \neq P_1$ and $\lambda = \frac{3x_1^2 + \bar{a}}{2y_1}$ if $P_2 = P_1$. Recall that the group $E(\mathbb{F}_p)$ is the basis of the following cryptosystem.

Definition 2.1. The **Elliptic Curve ElGamal Cryptosystem (ECEG)** is a public key cryptosystem which consists of the following parts:

(1) For the *key generation*, we find a group $E(\mathbb{F}_p)$ as above and a point P in $E(\mathbb{F}_p)$ whose order is a large number q, we choose a random number $a \in \{2, \dots, q-2\}$, and we calculate $Q = aP$. All information is public except for the secret key a.
(2) For *encryption*, we use the message space $E(\mathbb{F}_p)$ and we encrypt a message unit $M \in E(\mathbb{F}_p)$ by first choosing an ephemeral key $k \in \{2, \dots, q-2\}$ randomly and then calculating the ciphertext $C = (C_1, C_2) = (kP,\ M + kQ)$ in the ciphertext space $E(\mathbb{F}_p)^2$.
(3) For the *decryption* of a ciphertext unit $C = (C_1, C_2)$, we calculate $C_2 - aC_1$ and get M.

To compute the multiples aP, kP, kQ and aC_1 in this cryptosystem, fast algorithms, usually based on double-and-add techniques, are known (see [36]). Moreover, for the setup phase, there exist efficient algorithms for computing the order of the group $E(\mathbb{F}_p)$ and of a point P in this group (see for instance [15]). The security of the ECEG cryptosystem rests on the following problem:

Elliptic Curve Diffie-Hellman Problem (ECDHP): *Given the points $C_1 = kP$ and $Q = aP$, compute the point akP.*

It is quite clear that this problem can be solved if we are able to solve the following stronger problem.

Elliptic Curve Discrete Logarithm Problem (ECDLP): *Given the points P and $Q = aP$, find a.*

For more details about elliptic curves and elliptic curve cryptography, we refer the reader to [38] and [25].

3 FAULT ATTACKS AND FAULT MODELS FOR THE ECEG CRYPTOSYSTEM

Given a physical realization of the above ECEG cryptosystem, a fault attack is an intentional manipulation of the electronic circuit of this implementation in order to provoke miscalculations. Clearly, for a public key cryptosystem, the only meaningful way to mount such an attack is to manipulate the decryption device. As explained in the introduction, our proposed attack neither changes the elliptic curve nor does it produce points which are not contained in the given curve, and thus may be more difficult to detect and prevent. The central idea is to attack the secret key a.

For this purpose, we assume that a is stored in secure write-protected memory (typically EEPROMs) and then transferred piece-wise into the device performing the decryption algorithm. Depending on the way this transfer is achieved and on the capabilities of the attacker, we distinguish two basic fault models.

Fault Model 1 (FM1): The secret key a is transferred bitwise, or, for some other reason, we are able to inject a fault which flips a single bit of a. In this fault model, we do not assume control over which bit of a has been flipped. Of course, the attack still applies if we are able to restrict to a range of bits which may or may not be affected. Notice that we assume that we can also decrypt without fault injection. Thus we can easily detect if no fault was injected. As we shall see, we will also notice if more than one bit of a was affected by the fault injection.

Fault Model 2 (FM2): The secret key is transferred in chunks of s bits. In this scenario we assume that we can inject a random fault into a chosen s-bit subtuple of a and repeat this kind of injection a number of times. To underscore the applicability of this fault model, we point out that it applies to common implementations of ECEG on 8-bit microprocessors (see for instance [20], [19], [29], and [39]) and 8-bit smartcards (see for instance [2]). Moreover, with additional effort, it may be extended to cases in which a is transferred in 16-bit or even 32-bit chunks (see for instance [22]).

The two fault models FM1 and FM2 defined above relate to different levels of attacker capabilities and we require two distinct precisions for the injected faults.

In FM1, we consider a powerful attacker capable of flipping a single bit. Single bit-flips are not easily achievable by conventional means. However, they can be achieved by using a more sophisticated equipment, such as a laser. For example, the authors of [1] provide a description of their setup and details on how to inject a single bit-flip into an Advanced Encryption Standard (AES) implementation and by extension realizing a Differential Fault Attack (DFA). This supports the feasibility of FM1, as, even though we consider a strong attacker and require high precision fault injection, single bit-flip faults can be achieved and we do not assume any control over the position of the bit-flip.

For our second fault model, FM2, we consider a random s-bit modification of the secret key a. The required precision is therefore much smaller, and as such we consider FM2 a weaker attack scenario than FM1. As shown in [26], it is possible to inject random nibble or byte or faults when using a clock based fault injector. Hence, a fault injection following FM2 can be achieved using a low cost setup. Additionally, enough fault injections following FM2 will result in a successful attack, as we detail in Subsection 4.2.

In this paper, we focus on the simulation of both fault models and present software and hardware based simulations of these attacks in Section 5.

4 MATHEMATICAL DESCRIPTION OF THE FAULT ATTACK

In the setting of Section 2, we represent the numbers a, k, and so on, by bit-tuples $a = (a_0, \ldots, a_{r-1})$, $k = (k_0, \ldots, k_{r-1})$, and so on, where $a_i, k_j \in \{0,1\}$. In other words, we have $a = a_0 + a_1\, 2 + \cdots + a_{r-1}\, 2^{r-1}$, etc. Here r has to be chosen such that $2^r > \max\{p, q\}$ where q is the order of P.

4.1 The Attack for the Fault Model FM1

In the following we assume that we have produced a ciphertext point pair (C_1, C_2), where $C_1 = kP$ with an (unknown) random number k, and with $C_2 = M + kQ$, where M is the plaintext point and $Q = aP$. We assume that we can run the decryption algorithm and compute $M = C_2 - aC_1$ without fault injection, and that we can produce faulty plaintext points $M_i = C_2 - (a + f_i)C_1$ for $i = 1, \ldots, r$, where the i-th fault f_i is given by $f_{i\,\nu(i)} \in \{1, -1\}$ in position $\nu(i)$ and this position is chosen uniformly at random from $\{0, \ldots, r-1\}$.

From these data the secret key can be recovered as follows.

Proposition 4.1. *In the above setting, the following claims hold.*

(a) *We have* $M - M_i = (1 - 2a_{\nu(i)})\, 2^{\nu(i)}\, C_1$. *Hence* $a_{\nu(i)} = 0$ *if* $M - M_i = 2^{\nu(i)}\, C_1$ *and* $a_{\nu(i)} = 1$ *if* $M - M_i = -2^{\nu(i)}\, C_1$.

(b) *The expected number of successful fault injections needed to recover the full secret key* a *is* $r\,(1 + \frac{1}{2} + \cdots + \frac{1}{r}) \approx r\, \ln(r)$.

Proof. To prove (a), we note that

$$M - M_i = f_i C_1 = f_{i\,\nu(i)}\, 2^{\nu(i)}\, C_1 = (1 - 2a_{\nu(i)})\, 2^{\nu(i)}\, C_1.$$

For the proof of (b), we note that, after having found i bits of the secret key, the probability of discovering a new bit using one fault injection is $\frac{r-i}{r}$. Repeating injections until a new bit is affected is distributed geometrically, and thus has an expected value of $\frac{r}{r-i}$ injections. Altogether, we expect $\frac{r}{r} + \frac{r}{r-1} + \cdots + \frac{r}{1}$ injections until we have found the full secret key. □

This proposition leads us immediately to the FM1 Fault Attack Algorithm 1.

Clearly, this algorithm requires the calculation of $r - 1$ point doublings and $n \approx r\, \ln(r)$ point additions on the elliptic curve E. A further speed-up can be achieved by choosing $M = \mathcal{O}$, in which case no point additions are necessary.

4.2 The Attack for the Fault Model FM2

In this section, we continue to assume that the numbers a, k, etc., are represented by r-bit tuples $a = (a_0, \ldots, a_{r-1})$, etc. In FM2, we attack s-bit subtuples of a. For simplicity, let us assume that s divides r. (This is merely used to keep the indices well laid out.) Hence we represent a by $(a^{(1)}, \ldots, a^{(t)})$, where $r = st$. Here $a^{(\ell)} = (a_{\ell 0}, \ldots, a_{\ell\, s-1})$ is the ℓ-th subtuple of a, with $a_{\ell i} \in \{0,1\}$ for $\ell = 1, \ldots, t$ and $i = 0, \ldots, s-1$. Fault model FM2 assumes that we inject a random error in a chosen s-bit subtuple of a. Thus we assume that this is the ℓ-th subtuple, and we write

Algorithm 1 (FM1 Fault Attack Algorithm)

Input: A ciphertext pair $C = (C_1, C_2) \in E(\mathbb{F}_p)^2$, the correct plaintext point M and faulty plaintext points M_i for $i = 1, \dots, n$.
Output: A bittuple $(a_0, \dots, a_{r-1})$ representing the secret key a.

$L_1 := \{2^i\, C_1 \mid i = 0, \dots, r-1\}$;
$L_2 := \{-2^i\, C_1 \mid i = 0, \dots, r-1\}$;
for $i = 1$ **to** n **do**
  if $M - M_i = 2^{\nu(i)}\, C_1 \in L_1$ **then**
    $a_{\nu(i)} := 0$;
  else if $M - M_i = -2^{\nu(i)}\, C_1 \in L_2$ **then**
    $a_{\nu(i)} := 1$;
  end if
end for
if $a_j \in \{0, 1\}$ **for** $j = 0, \dots, r-1$ **then**
  return $(a_0, \dots, a_{r-1})$
else
  return "not enough injections"
end if

$(f_{\ell 0}, \dots, f_{\ell\, s-1}) \in \{0,1\}^s$ for the *fault pattern*, i.e., for marking the positions of the flipped bits.

The faulty point returned by the i-th injection is then given by $M_i = C_2 - (a + \tilde{g}_i)\, C_1$ where the number $\tilde{g}_i$ corresponds to the tuple $(g_i^{(1)}, \dots, g_i^{(t)})$ and the affected s-bit tuple can be written in the form $g_i^{(\ell)} = (b_{\ell 0}, \dots, b_{\ell\, s-1})$ with $b_{\ell j} = f_{\ell j}\,(1 - 2a_{\ell j})$ for $j = 0, \dots, s-1$. Consequently, the difference $M - M_i$ is equal to the point

$$b_{\ell 0}\, C_1^{(\ell)} + \dots + b_{\ell\, s-1}\, 2^{s-1}\, C_1^{(\ell)} \;=\; (b_{\ell 0} + \dots + 2^{s-1}\, b_{\ell\, s-1})\, C_1^{(\ell)}$$

where $C_1^{(\ell)} = 2^{s(\ell-1)}\, C_1$. Thus, for the i-th injection, we observe the number $g_i = b_{\ell 0} + \dots + 2^{s-1}\, b_{\ell\, s-1}$, where $(b_{\ell 0}, \dots, b_{\ell\, s-1})$ is contained in $\{-1, 0, 1\}^s$. Clearly, the observation of g_i will, in general, not be sufficient to deduce the tuple $(b_{\ell 0}, \dots, b_{\ell\, s-1})$. Knowledge of this tuple allows us to determine those bits of the ℓ-th subtuple $a^{(\ell)} = (a_{\ell 0}, \dots, a_{\ell\, s-1})$ of the secret key for which $b_{\ell j} \neq 0$, namely via $a_{\ell j} = (1 - b_{\ell j})/2$.

As a consequence of this discussion, we introduce the following concepts.

Definition 4.2. Let $g_i \in \{-2^s + 1, -2^s + 2, \dots, 2^s - 1\}$.

(a) A tuple $(b_0, \dots, b_{s-1}) \in \{-1, 0, 1\}^s$ such that $g_i = b_0 + 2b_1 + \dots + 2^{s-1} b_{s-1}$ is called a **signed s-bit representation** of g_i.
(b) Given a signed s-bit representation $B = (b_0, \dots, b_{s-1})$ of g_i, we let $K_B(g_i)$ be the set of all s-bit tuples $(a_0, \dots, a_{s-1})$ such that $a_j = (1 - b_j)/2$ whenever $b_j \neq 0$. Then the union of all sets $K_B(g_i)$, where B traverses the signed s-bit representations of g_i, is called the **key subtuple candidate set** of g_i and is denoted by $K(g_i)$.

Unfortunately, the signed s-bit representation of a number g_i is, in general, not uniquely determined. For instance, Figure 1 plots the number of signed 8-bit representations for the integers in the range $\{-255, -254, \dots, 255\}$.

Let $\lambda(g_i, s)$ denote the number of signed s-bit representations of g_i. The function $\lambda(g_i, s)$ was studied in [18]. In particular, Lemmas 3 and 4 of this paper provide recursive formulas which allow

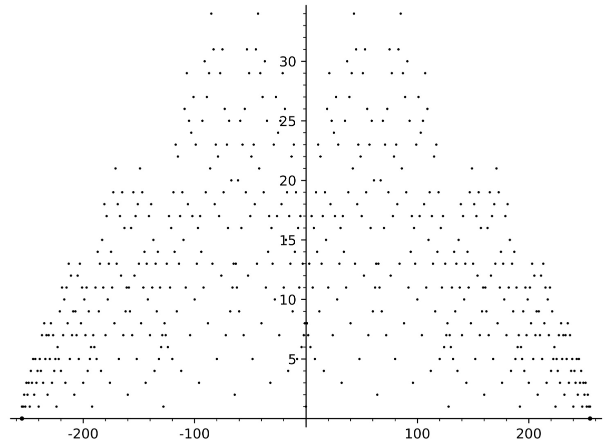

Figure 1: Number of signed 8-bit representations

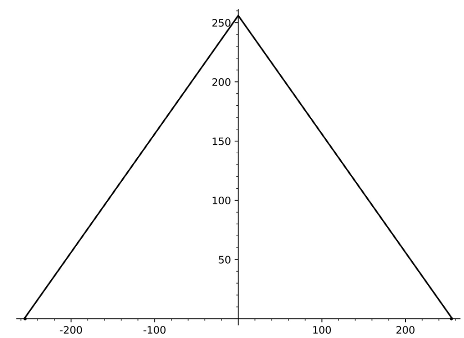

Figure 2: Number of key byte candidates

us to compute the values of $\lambda(g_i, s)$ easily. Moreover, by evaluating all 3^s signed s-bit tuples, we may precompute for each number g_i the list $R(g_i)$ of all signed s-bit representations of g_i. Finally, by taking the union of all sets $K_B(g_i)$ for $B \in R(g_i)$, we may precompute the list of all key subtuple candidate sets $K(g_i)$, where $g_i \in \{-2^s + 1, \dots, 2^s - 1\}$.

Figure 2 illustrates the number of key subtuple candidates as a function of g_i in the case $s = 8$.

In fact, we can describe the key subtuple candidate sets depending on g_i as follows.

Proposition 4.3. *Let $g_i \in \{-2^s + 1, -2^s + 2, \dots, 2^s - 1\}$. Then the key subtuple candidate set $K(g_i)$ is given by*

$$K(g_i) \;=\; \{c \in \{0,1\}^s \mid 0 \le g_i + N(c) \le 2^s - 1\}.$$

where $N(c) = c_0 + 2c_1 + \dots + 2^{s-1} c_{s-1}$ for $c = (c_0, \dots, c_{s-1}) \in \{0,1\}^s$.

Proof. First we prove the inclusion $\subseteq$. Given $c \in K(g_i)$, there exists a signed s-bit representation $B = (b_0, \dots, b_{s-1})$ of g_i such that $c \in K_B(g_i)$. Writing $c = (c_0, \dots, c_{s-1})$, we have $N(c) = c_0 + 2c_1 + \dots + 2^{s-1} c_{s-1}$ and $c_j = (1 - b_j)/2$ for all j with $b_j \neq 0$. Therefore we obtain

$$g + N(c) \;=\; \sum_{\{j \mid b_j \neq 0\}} (b_j + (1 - b_j)/2)\, 2^j \;+\; \sum_{\{j \mid b_j = 0\}} c_j\, 2^j$$

and then $b_j+(1-b_j)/2 = (1+b_j)/2 \in \{0,1\}$ leads to $0 \le g+N(c) \le 2^s - 1$.

Thus the main task is to prove the inclusion $\supseteq$. For $g_i = 0$, we may use the signed binary representation $B = (0,\dots,0)$ and conclude that all s-bit tuples are key subtuple candidates.

Next we consider the case $g_i < 0$. We only need to show $g_i + N(c) \ge 0$, since $g_i + N(c) \le 2^s - 1$ is trivially true. First we determine the s-bit tuple c for which $N(c)$ has the smallest possible value and show that this value is $-g_i$.

Given a signed binary representation $B = (b_0,\dots,b_{s-1})$ of g_i, we let $\nu(B) = \{j \mid b_j = -1\}$, $\pi(B) = \{j \mid b_j = 1\}$, and $\zeta(B) = \{j \mid b_j = 0\}$. As $b_j = 1$ forces $c_j = 0$ and $b_j = -1$ forces $c_j = 1$, the number $N(c)$ satisfies $N(c) = \sum_{j\in\nu(B)} 2^j + \sum_{j\in\zeta(B)} c_j 2^j$ for every $c \in K_B(g_i)$. Hence the number $g+N(c)$ equals $\sum_{j\in\pi(B)} 2^j + \sum_{j\in\zeta(B)} c_j 2^j$. Its minimum value is achieved when $\pi(B) = \emptyset$ and $c_j = 0$ for $j \in \zeta(B)$, and this minimum value is zero. In fact, this value is attained for $B = (-\beta_0,\dots,-\beta_{s-1})$, where $-g_i = \sum_{j=0}^{s-1} \beta_j 2^j$, and for the tuple $c = (\beta_0,\dots,\beta_{s-1})$.

Now we prove by induction that all numbers $N(c)$, starting from this minimum number $N(c) = -g_i > 0$ and up to $N(c) = 2^s - 1$, are realized by a tuple $c \in K_B(g_i)$ for some signed binary representation B of g_i. Let $(c_0',\dots,c_{s-1}')$ be the tuple representing the preceding number. We distinguish two cases: $c_0' = 0$ and $c_0' = 1$.

In the fist case $c_0' = 0$ we want to show that $c = (1, c_1',\dots,c_{s-1}')$ is in $K_B(g_i)$ for a suitable B. Since we can use $B = B'$ if $b_0' = 0$, we may assume that $b_0' = 1$. Notice that the fact that B' represents $g_i < 0$ implies that B' ends in $(-1,0,\dots,0)$ with a non-negative number of zeros. Hence there exists a number $2 \le k \le s-1$ such that the first $k+1$ columns of the matrix $A' = \begin{pmatrix} 1 & b_1' & \cdots & b_{s-1}' \\ 0 & c_1' & & c_{s-1}' \end{pmatrix}$ are of the form $S = \begin{pmatrix} 1 & \bar{c}_1' & \cdots & \bar{c}_{k-1}' & b_k' \\ 0 & c_1' & & c_{k-1}' & c_k' \end{pmatrix}$ where $\bar{c}_i' = 1 - c_i'$ for $i < k$ and $b_k' \ne 1 - c_k'$. For the pair (b_k', c_k'), there exist two subcases.

The first subcase is $c_k' = 0$. Here we necessarily have $b_k' = 0$. Then the last part of the matrix S has the form $T = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & & 1 & 0 \end{pmatrix}$ where we have a non-negative number of columns $\binom{0}{1}$ and where the column $\binom{1}{0}$ has some column index $j < k$. Now we replace the first row of T by $(-1,-1,\dots,-1,1)$ and get $T' = \begin{pmatrix} -1 & -1 & \cdots & -1 & 1 \\ 0 & 1 & & 1 & 0 \end{pmatrix}$. Notice that the first row of T' is a signed binary representation of the same number as the first row of T. We replace the corresponding columns of A' by the columns of T' and get a new matrix A'' with first row B''. Then the conditions for $c' \in K_{B''(g_i)}$ are satisfied except for the j-th column $\binom{-1}{0}$.

If $j = 1$, we find that $c = (1, c_1',\dots,c_{s-1}')$ is contained in $K_{B''}(g_i)$ and we are finished with the subcase. If $j > 1$, we look at the part of the matrix A'' which ends with the j-th column and is of the form $U = \begin{pmatrix} 1 & 0 & \cdots & 0 & -1 \\ 0 & 1 & & 1 & 0 \end{pmatrix}$. As before, we replace the first row with a tuple which represents the same number and get $U' = \begin{pmatrix} -1 & -1 & \cdots & -1 & 0 \\ 0 & 1 & & 1 & 0 \end{pmatrix}$. After that, we replace the corresponding columns of A'' by the columns of U' and get a matrix A'''. Let B''' be the first row of A'''. The tuple B''' still represents g_i and the condition $c' \in K_{B'''}(g_i)$ is violated only in the column $\binom{-1}{0}$. Continuing in this way, we move the column $\binom{-1}{0}$ to the left until it is the first column. Then the resulting first row B continues to represent g_i and we have $c = (1, c_1',\dots,c_{s-1}') \in K_B(g_i)$, as we wanted to show.

In the second subcase $c_k' = 1$, the k-th column of S is $\binom{-1}{1}$. Then the last part of S is of the form $T = \begin{pmatrix} 1 & 0 & \cdots & 0 & -1 \\ 0 & 1 & & 1 & 1 \end{pmatrix}$ with a non-negative number of columns of the form $\binom{0}{1}$. Once again we replace the first row of T with a row which represents the same number and get $T' = \begin{pmatrix} -1 & -1 & \cdots & -1 & 0 \\ 0 & 1 & & 1 & 0 \end{pmatrix}$. By inserting T' for the corresponding columns of A', we get a matrix A'' whose first row B'' still represents g_i. The condition $c' \in K_{B''}(g_i)$ is only violated by the column $\binom{-1}{0}$ corresponding to the first column of T'. If this is the first column of A'' then we have $c = (1, c_1',\dots,c_{s-1}') \in K_{B''}(g_i)$, as desired. Otherwise we continue to move the column $\binom{-1}{0}$ to the left as in the last part of the proof of the first subcase.

Next we examine the second case $c_0' = 1$. There exists an index $1 \le k \le s-1$ such that $(c_0',\dots,c_k') = (1,\dots,1,0)$. We want to show that the tuple $c = (c_0,\dots,c_{s-1}) = (1,0,\dots,0,c_{k+1},\dots,c_{s-1})$ is in $K_B(g_i)$ for a suitably chosen signed binary representation B of g_i. Thus our goal is to show that we can replace $(b_0',\dots,b_k')$ by a tuple $(b_0,\dots,b_k)$ which represents the same number, but satisfies $b_j \in \{0,1\}$ for $j < k$ and $b_k \in \{-1,0\}$. Notice that we have $b_j' \in \{-1,0\}$ for $j < k$ and $b_k' \in \{0,1\}$. Let $\tilde{b} = (\tilde{b}_0,\dots,\tilde{b}_k)$ be the current tuple, and assume that it still contains elements $b_j = -1$ for some $j < k$. Let $(\tilde{b}_\mu,\dots,\tilde{b}_m)$ be the initial streak of elements -1 in $\tilde{b}$. Depending on the next element of $\tilde{b}$, there are two subcases.

Firstly, if we have $\tilde{b}_{m+1} = 0$ then we replace $(-1,\dots,-1,0)$ by $(1,0,\dots,0,-1)$. Secondly, if we have $\tilde{b}_{m+1} = 1$, then we replace $(-1,\dots,-1,1)$ by $(1,0,\dots,0)$. In the first case, the first element -1 will appear further to the right, and in the second case, the first streak of elements -1 is eliminated altogether. After finitely such replacements, we obtain a tuple $b = (b_0,\dots,b_{s-1})$ with $b_j \in \{0,1\}$ for $j = 1,\dots,k-1$ and $b_k \in \{-1,0\}$. This tuple continues to represent the same number g_i, and in addition we have $c \in K_B(g_i)$, as desired.

Finally, in the case $g_i > 0$, it clearly suffices to prove the inequality $g_i + N(c) \le 2^s - 1$. This follows from the previous case, since $-g_i + N(\bar{c}) \ge 0$, where the complementary bit tuple $\bar{c}$ corresponds to the number $2^s - 1 - N(c)$. $\square$

Now the FM2 Fault Attack Algorithm 2 is easy to formulate.

For every fault injection, only one point addition on E, and some easy fixed-cost operations are necessary. Notice that Step 7 can be carried out efficiently using Proposition 4.3 by checking if the remaining candidates in K satisfy the given inequalities.

The number of necessary fault injections until we have $\#K = 1$ was experimentally found to be around $r \approx 1.5 \cdot 2^s$. More detailed values for different values of s are given in the next section. Of course, since we are actually interested in the full secret key, we have to repeat this attack t times, so that the full key recovery requires approximately $1.5\, 2^s\, t$ fault injections.

5 EXPERIMENTAL RESULTS

5.1 Software Simulations

The following data were obtained with a prototype implementation of the proposed fault attacks using the interpreted top-level

Algorithm 2 (FM2 Fault Attack Algorithm)

Input: A ciphertext pair $C = (C_1, C_2) \in E(\mathbb{F}_p)^2$, the correct plaintext point M and faulty plaintext points M_i for $i = 1, \ldots, n$, a number $\ell \in \{1, \ldots, t\}$, and a precomputed list of points of $E(\mathbb{F}_p)$.

Output: An s-bit tuple $a^{(\ell)} = (a_{\ell 0}, \ldots, a_{\ell\, s-1})$ representing the ℓ-th s-bit subtuple of the secret key a.

1: $C_1^{(\ell)} := 2^{s(\ell-1)}\, C_1$;
2: $L := \{j\, C_1^{(\ell)} \mid j = -2^s + 1, \ldots, 2^s - 1\}$;
3: $K := \{0, 1\}^s$;
4: **for** $i = 1$ **to** n **do**
5: Find $g_i \in \{-2^s+1, \ldots, 2^s-1\}$ such that $M - M_i = g_i\, C_1^{(\ell)} \in L$.
6: $K := \{c \in K \mid 0 \le g_i + N(c) \le 2^s - 1\}$;
7: **if** $\#K = 1$ **then**
8: **return** the element $a^{(\ell)}$ of K.
9: **end if**
10: **end for**
11: **return** "not enough injections"

language of the computer algebra system ApCoCoA (cf. [3]). Since the mathematical components of the attacks are very efficient, the computational power of a laptop having an Intel i5-4300M microprocessor and 16 GB of RAM was fully sufficient.

5.1.1 Simulating Fault Model FM1. Both the generation of a randomly located bit flip in the bit-tuple $(a_0, \ldots, a_{r-1})$ representing the secret key a and the recovery of the corresponding key bit from the faulty point M' take only a fraction of a second. Also the pre-computation of the lists L_1 and L_2 is a matter of a few seconds, even in the case of the largest curves under consideration.

For the various values of r, the following NIST curves were used:

(1) The curve *secp128r1* is given in [13].
(2) The curves *secp192r1*, *secp224r1*, *secp256r1*, and *secp384r1* are also given in [13]. They are called P-192, P-224, P-256, and P-385, resp., in NIST FIPS 186-4 (see [34]) and recommended for official usage there.

Table 1 collects the experimentally found values for the number of necessary fault injections. In each case, we performed 500 sequences of simulated injections, each lasting until the secret a was fully recovered. We list the minimum and maximum number of injections needed, called $N_{\min}$ and $N_{\max}$, the average number N_{avg} of injections needed, as well as the average time T_{avg} to recover a from one injection sequence.

Table 1: Number of injections for fault model FM1

curve	$N_{\min}$	$N_{\max}$	N_{avg}	T_{avg} (sec)
secp128r1	373	1270	670	0.092
secp192r1	624	2657	1129	0.143
secp224r1	751	3022	1352	0.179
secp256r1	856	3360	1589	0.212
secp384r1	1589	5001	2487	0.368

Note that the number N_{avg} approximates the theoretical value $r(1 + \frac{1}{2} + \cdots + \frac{1}{r})$ very well in each case, but that the standard deviation is rather large.

5.1.2 Simulating Fault Model FM2. For the fault model FM2, we assume that we are able to inject a random fault into an s-bit subtuple of the secret key. Thus a fault is an s-bit tuple which characterizes the positions of the bit flips in the given subtuple $(a_{\ell 0}, \ldots, a_{\ell\, s-1})$ of the secret key. For the software simulation, we suppose that the fault pattern $(f_{\ell 0}, \ldots, f_{\ell\, s-1})$ is equidistributed in the set of all s-bit tuples, where $f_{\ell j} = 1$ if and only if $a_{\ell j}$ has been flipped.

Then we let $b_{\ell j} = f_{\ell j}(1 - 2a_{\ell j})$ for $j = 0, \ldots, s-1$ and note that the resulting number $g_i = b_{\ell 0} + \cdots + 2^{s-1} b_{\ell\, s-1}$ which is observed via $M - M_i = g_i C_1^{(\ell)}$ is not equidistributed anymore in the range $\{-2^s + 1, -2^s + 2, \ldots, 2^s - 1\}$. According to Algorithm 2, we repeat such fault injections and intersect with the set $K(g_i)$ until the remaining key candidate set has only one element. This leads to the natural question how many fault injections are needed on average.

Table 2 lists the results of the following experiment: we inject faults into a fixed s-bit segment of a, where $s \ge 3$. Depending on s, i.e., depending on our spacial resolution, we tabulate the average number of injections N_{avg} needed.

Table 2: Number of injections for the s-bit version of FM2

s	3	4	5	6	7	8
N_{avg}	10.91	23.05	47.51	95.21	190.77	381.88

This table suggest the conjecture that $N_{\text{avg}} \approx 1.5 \cdot 2^s$. In the case $s = 8$, our experimental results agree very well with the theoretically derived $N_{\text{avg}} = 381.5$ given in [24]. Since we have equality in Proposition 4.3, whereas [24] merely uses the trivial containment $\subseteq$, we can conclude that these bounds represent the best possible values. Notice that the average number of injections needed depends on the actual value of the s-bit subtuple $a^{(\ell)} = (a_{\ell 0}, \ldots, a_{\ell s-1})$ of the secret key. For instance, in the case $s = 3$, these average numbers are distributed as given in Table 3.

Table 3: Number of 3-bit injections per secret key

$a^{(\ell)}$	(0,0,0)	(1,0,0)	(0,1,0)	(1,1,0)	(0,0,1)	(1,0,1)	(0,1,1)	(1,1,1)
N_{avg}	7.54	12.01	12.29	11.81	11.81	12.29	12.01	7.54

Clearly, this table is unchanged if we replace (a_0, a_1, a_2) by $(1 - a_0, 1 - a_1, 1 - a_2)$, since the probabilities of a bit flip from 0 to 1 and from 1 to 0 are identical in our setting.

5.2 Hardware Implementation and Simulation

5.2.1 Hardware Implementation. The main operation of the ECEG cryptosystem, as already mentioned in Section 4, is the point multiplication. Each point multiplication, which consists of point doublings and additions, is based on four modular arithmetic operations. These are inversion, multiplication, subtraction and addition. Modular addition and modular subtraction are easily implemented and out of scope for this paper. The inversion and modular multiplication are time-consuming and required in each point doubling and point addition. Therefore efficient implementations are needed. In our implementation, we use the left-to-right double-and-add algorithm for the point multiplication. This algorithm is well-known and, as such, we do not describe it here and rather discuss more relevant implementation choices that we have made.

As previously mentioned, we also need an inversion module. An efficient inversion exists using the Extended Euclidean Algorithm as described in [30]. We present an adaptation in Algorithm 3. The algorithm computes $gcd(x, p)$. Since p is a prime number, we have $gcd(x, p) = 1$. Starting with $s = p$ and $t = x$, division with remainder yields $s = q \cdot t + r$, where q and r are the quotient and the remainder, respectively. Meanwhile, $s = m_s \cdot p + n_s \cdot x$ and $t = m_t \cdot p + n_t \cdot x$, describe s and t using multiples of the original values x and p. Therefore the remainder r is given as

$$\begin{aligned} r &= s - q \cdot t \\ &= (m_s \cdot p + n_s \cdot x) - q \cdot (m_t \cdot p + n_t \cdot x) \\ &= (m_s - q \cdot m_t) \cdot p + (n_s - q \cdot n_t) \cdot x \\ r &= m_r \cdot p + n_r \cdot x \end{aligned}$$

Finally, when $r = 1$, we only have one further check to get x^{-1}. Namely, if $n_r < 0$, then $x^{-1} = n_r + p$, and otherwise $x^{-1} = n_r$.

Algorithm 3 (HW Inversion via Extended Euclidean Algorithm)

Input: x , p
Output: x^{-1}
$s := p$; $t := x$; $q := 0$; $r := 0$
$m_s := 1$; $n_s := 0$; $m_t := 0$; $n_t := 1$
while $r \neq 1$ **do**
  division(s, t, q, r)
  $m_r := m_s - q \cdot m_t$
  $n_r := n_s - q \cdot n_t$
  $s := t$
  $t := r$
  $m_s := m_t$; $n_s := n_t$; $m_t := m_r$; $n_t := n_r$
end while
if $n_r > 0$ **then**
  $x^{-1} := n_r$
else
  $x^{-1} := n_r + p$
end if
return x^{-1}

To calculate $q = \lfloor \frac{s}{t} \rfloor$, we need a division component. The equation $q = \sum_{i=n-1}^{0} c_i 2^i$ describes the idea presented in Algorithm 4. The quotient q is broken into powers of two. It requires only a shift register and an adder to obtain the result. The amount of the shift is calculated as $shift = MSB(a) - MSB(t)$, where MSB indicates the position of the most significant bit which is equal to 1, and the starting value of a is the dividend s. Intermediate results are saved in the registers a, b, and c, where $b = t \cdot 2^{shift}$, $a = a - b$, and $c = c + 2^{shift}$. One condition which must always be satisfied is $a > b$, so that the subtraction does not generate negative values. Finally, when $a < t$, the value q has been calculated and the remainder r is given by $r = a$.

Algorithm 4 (HW Division Algorithm)

Input: dividend s, divisor t
Output: quotient q, remainder r
$a := s$; $b := t$; $c := 0$
while $a \geq t$ **do**
  $i_A := MSB(a)$
  $i_B := MSB(b)$
  $b := b \ll (i_A - i_B)$
  $c := c + (1 \ll (i_A - i_B))$
  if $b > a$ **then**
    $b := b \gg 1$
    $c := c \gg 1$
  end if
  $a := a - b$
  $b := t$
end while
$r := a$
$q := c$
return q, r

For a fast multiplication algorithm, we implemented the Montgomery Modular Multiplication Algorithm as described in [4]. It had to be extended to include a conversion, since the output of the algorithm with inputs a, b, p, and the bit length k, is equal to $a \cdot b \cdot 2^{-k} \pmod{p}$. The extension aims to convert the result back to $a \cdot b \pmod{p}$. The description of the algorithm is given in Algorithm 5.

Algorithm 5 (HW Modular Multiplication Algorithm)

Input: a,b,p,k
Output: $c = a \cdot b \bmod p$
/*Montgomery Modular Multiplication*/
$u = 0$
for $i = 0$ **to** $k - 1$ **do**
  $u = u + a[i] \cdot b$
  if $u_0 = 1$ **then**
    $u = u + p$
  end if
  $u = u \gg 1$
end for
if $u \geq p$ **then**
  $u = u - p$
end if
/*Conversion*/
$c = u \cdot (2^k \bmod p)$
$c = c \bmod p$
return c

Notice that taking a value $\pmod{p}$ is achieved by Algorithm 4, where we use $t = p$, $i_B = k - 1$, and the result is the *remainder* while the *quotient* is neglected.

5.2.2 Hardware Simulation. As mentioned in Section 3, we focused on hardware simulation of fault injections. For this purpose, we used an Intel(R) Core(TM) i7-8550U processor with 16GB of RAM and ran behavioral simulation of the previously described hardware implementation using the *Xilinx Vivado Design Suite*.

The simulation setup includes a point multiplication module, an encryption module, and a decryption module. The point multiplication module was used to generate the public point $Q = aP$. The result, together with the generating point P, forms the input of the encryption module, which generates the encrypted message pair (C_1, C_2). The encrypted message pair and the secret key a are then decrypted to output the message M. Next, the secret key transfer is disturbed by XORing the secret key a with a random fault f, resulting in a faulty secret key a_f. Therefore the decrypted message M_f is also faulty.

After gathering multiple ciphertext-plaintext pairs from our simulation, we ran the attack for the fault model FM2 presented in Section 4. We considered a single s-bit subtuple of the secret key a for our simulation. (More specifically, we used the least significant s-bit subtuple $a^{(1)}$.) Fully recovering the complete secret key is only a matter of repeating this attack several times, with varying s-bit subtuples, where the number of these repetitions depends on the key length.

Table 4 presents our results for the curve *secp128r1*, using various values of s and different numbers of fault injections. As expected, these results are close to the theoretical value of the average number of fault injections needed to have only a single s-bit key subtuple candidate, as described in Table 2. In a few instances several fault injections resulted in the same faulty points, in particular for $s = 3$. In one specific case we had only four distinct faulty plaintext points out of 13 fault injections, and therefore we only managed a reduction to three key subtuple candidates. However, in most other cases, only a single s-bit key subtuple candidate remained after performing the attack.

In the table, the number n denotes the number of fault injections, $\#K$ is the number of key subtuple candidates left, and T represents the timing of the FM2 Fault Attack Algorithm including the preprocessing step.

Table 4: Hardware simulation for fault model FM2

s	n	$\#K$	T (sec)
3	13	3	0.037
4	28	1	0.275
5	58	1	0.947
6	116	1	3.659
7	230	1	4.931
8	400	2	38.906
8	425	1	40.143
8	450	1	38.670

In addition to recovering a single s-bit key subtuple, we present in Table 5 timings for complete key recovery attacks for different values of s. Again we considered the curve *secp128r1*. Similarly to Table 4, we do not always have a single key subtuple candidate for each subtuple, which leads to several key candidates. However, the number of key candidates stays low and the remaining candidates can easily be brute forced. The table also shows that, as discussed previously, the smaller s is, the fewer overall fault injections are required, and the solving time decreases in a similar fashion.

Table 5: FM2 complete key recovery hardware simulation

s	n	$\#K$	T (sec)
3	602	4609	36.10
5	1508	385	202.95
7	4370	1	1535.10

6 CONCLUSIONS

Extending the attack from $s = 1$ and $s = 8$ to an arbitrary length s of the affected key subtuple gives the adversary a range of interesting options. Our results suggest that the adversary should use the smallest s compatible with the precision of the available fault-injection equipment, because the number of subtuples of one key grows linearly, but the number of required fault injections per subtuple decreases exponentially when we lower s. The proof we presented gives new insights into the complexity of the fault attack for different values of s. Our future work will concentrate on attacking hardware implementations equipped with protective mechanisms.

ACKNOWLEDGMENTS

This research was supported by DFG (German Research Foundation) project "Algebraische Fehlerangriffe" grants PO 1220/7-2 and KR 1907/6-2.

REFERENCES

[1] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. 2010. How to flip a bit?. In *16th Int. On-Line Testing Symposium (IOLTS)*. IEEE, Piscataway, NJ, USA, 235–239. https://doi.org/10.1109/IOLTS.2010.5560194

[2] Harald Aigner, Holger Bock, Markus Hütter, and Johannes Wolkerstorfer. 2004. A low-cost ECC coprocessor for smartcards. In *Cryptographic Hardware and Embedded Systems - CHES 2004 (LNCS 3156)*. Springer-Verlag, Berlin, Heidelberg, 107–118. https://doi.org/10.1007/978-3-540-28632-5_8

[3] The ApCoCoA Team. 2013. ApCoCoA: Applied Computations in Computer Algebra. Retrieved June 24, 2019 from http://apcocoa.uni-passau.de

[4] Rashidi Bahram, Reza Rezaeian Farashahi, and Sayed Masoud Sayedi. 2017. High-speed hardware implementations of point multiplication for binary Edwards and generalized Hessian curves. Cryptology ePrint Archive, Report 2017/005.

[5] Feng Bao, Robert H. Deng, Yongfei Han, Albert B. Jeng, A. Desai Narasimhalu, and Teow-Hin Ngair. 1997. Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults. In *5th Int. Workshop on Security Protocols (LNCS 1361)*, Bruce Christianson, Bruno Crispo, T. Mark A. Lomas, and Michael Roe (Eds.). Springer-Verlag, Berlin, Heidelberg, 115–124. https://doi.org/10.1007/BFb0028164

[6] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. 2006. The sorcerer's apprentice guide to fault attacks. *IEEE Proc.* 94, 2 (2006), 370–382.

[7] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. 2012. Fault injection attacks on cryptographic devices: theory, practice and countermeasures. *Proc. IEEE* 100, 11 (2012), 3056–3076.

[8] Ingrid Biehl, Bernd Meyer, and Volker Müller. 2000. Differential fault attacks on elliptic curve cryptosystems (extended abstract). In *Advances in Cryptology – CRYPTO 2000 (LNCS 1880)*, Mihir Bellare (Ed.). Springer-Verlag, Berlin, Heidelberg, 131–146. https://doi.org/10.1007/3-540-44598-6_8

[9] Johannes Blömer, Martin Otto, and Jean-Pierre Seifert. 2006. Sign change fault attacks on elliptic curve cryptosystems. In *Fault Diagnosis and Tolerance in Cryptography (LNCS 4236)*. Springer-Verlag, Berlin, Heidelberg, 36–52. https://doi.org/10.1007/11889700_4

[10] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. 1997. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology – EUROCRYPT '97 (LNCS 1233)*. Springer-Verlag, Berlin, Heidelberg, 37–51. https://doi.org/10.1007/3-540-69053-0_4

[11] Jan Burchard, Mael Gay, Ange Messeng Ekossono, Jan Horáček, Bernd Becker, Tobias Schubert, Martin Kreuzer, and Ilia Polian. 2017. AutoFault: towards automatic construction of algebraic fault attacks. In *2017 Workshop on Fault*

Diagnosis and Tolerance in Cryptography (FDTC). IEEE, Piscataway, NJ, USA, 65–72. https://doi.org/10.1109/FDTC.2017.13

[12] Mike Burmester. 1990. A remark on the efficiency of identification schemes. In *Advances in Cryptology – EUROCRYPT '90 (LNCS 473)*. Springer-Verlag, Berlin, Heidelberg, 493–495. https://doi.org/10.1007/3-540-46877-3_47

[13] Certicom Research Group. 2000. SEC 2: Recommended Elliptic Curve Domain Parameters. Retrieved June 24, 2019 from https://www.secg.org/SEC2-Ver-1.0.pdf

[14] Mathieu Ciet and Marc Joye. 2005. Elliptic curve cryptosystems in the presence of permanent and transient faults. *Des. Codes Cryptogr.* 36, 1 (July 2005), 33–43. https://doi.org/10.1007/s10623-003-1160-8

[15] Henri Cohen. 1993. *A Course in Computational Algebraic Number Theory* (3 ed.). GTM, Vol. 138. Springer-Verlag, Berlin, Heidelberg.

[16] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. 2012. Electromagnetic transient faults injection on a hardware and a software implementation of AES. In *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, Piscataway, NJ, USA, 7–15.

[17] Emmanuelle Dottax. 2002. Fault attacks on NESSIE signature and identification schemes. www.cosic.esat.kuleuven.be/nessie/nessie/reports/phase2/SideChan_1.pdf NESSIE Public Report.

[18] Nevine Ebeid and M. Anwar Hasan. 2007. On binary signed digit representations of integers. *Des. Codes Cryptogr.* 42, 1 (Jan. 2007), 43–65. https://doi.org/10.1007/s10623-006-9014-9

[19] Hans Eberle, Arvinderpal Wander, Nils Gura, Sheueling Chang-Shantz, and Vipul Gupta. 2005. Architectural extensions for elliptic curve cryptography over $\mathrm{GF}(2^m)$ on 8-bit microprocessors. In *2005 IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP '05)*. IEEE, Piscataway, NJ, USA, 343–349. https://doi.org/10.1109/ASAP.2005.15

[20] Markus Ernst, Michael Jung, Felix Madlener, Sorin A. Huss, and Rainer Blümel. 2003. A reconfigurable system on chip implementation for elliptic curve cryptography over $\mathrm{GF}(2^n)$. In *Cryptographic Hardware and Embedded Systems – CHES 2002 (LNCS 2523)*. Springer-Verlag, London, UK, 381–399. https://doi.org/10.1007/3-540-36400-5_28

[21] Pierre-Alain Fouque, Reynald Lercier, Denis Réal, and Frédéric Valette. 2008. Fault attack on elliptic curve Montgomery ladder implementation. In *Fifth International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, Piscataway, NJ, USA, 92–98. https://doi.org/10.1109/FDTC.2008.15

[22] Lubos Gaspar. 2012. *Cryptoprocessor-architecture, programming and evaluation of the security*. Ph.D. Dissertation. Université Jean Monnet, Saint-Etienne, France.

[23] Christophe Giraud and Erik Woodward Knudsen. 2004. Fault attacks on signature schemes. In *9th Australasian Conf. on Information Security and Privacy – ACISP 2004 (LNCS 3108)*, Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan (Eds.). Springer-Verlag, Berlin, Heidelberg, 478–491. https://doi.org/10.1007/978-3-540-27800-9_41

[24] Christophe Giraud, Erik Woodward Knudsen, and Michael Tunstall. 2010. Improved fault analysis of signature schemes. In *9th Int. Conf. on Smart Card Research and Advanced Appl. – CARDIS 2010 (LNCS 6035)*, Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny (Eds.). Springer-Verlag, Berlin, Heidelberg, 164–181. https://doi.org/10.1007/978-3-642-12510-2_12

[25] Darrel Hankerson, Alfred Menezes, and Scott Alexander Vanstone. 2004. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, New York, USA.

[26] Batya Karp, Mael Gay, Osnat Keren, and Ilia Polian. 2018. Detection and correction of malicious and natural faults in cryptographic modules. In *7th Int. Workshop on Security Proofs for Embedded Systems – PROOFS 2018 (Kalpa Publ. in Computing)*, Vol. 7. Easychair, Manchester, UK, 68–82. https://doi.org/10.29007/w37p

[27] Chong Hee Kim, Philippe Bulens, Christophe Petit, and Jean-Jacques Quisquater. 2008. Fault attacks on public key elements: application to DLP-based schemes. In *5th European Workshop on Public Key Infrastructure: Theory and Practice – EuroPKI 2008 (LNCS 5057)*, Stig Fr. Mjølsnes, Sjouke Mauw, and Sokratis K. Katsikas (Eds.). Springer-Verlag, Berlin, Heidelberg, 182–195. https://doi.org/10.1007/978-3-540-69485-4_13

[28] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Advances in Cryptology – CRYPTO'99 (LNCS 1666)*. Springer-Verlag, Berlin, Heidelberg, 388–397.

[29] Manuel Koschuch, Joachim Lechner, Andreas Weitzer, Johann Großschädl, Alexander Szekely, Stefan Tillich, and Johannes Wolkerstorfer. 2006. Hardware/software co-design of elliptic curve cryptography on an 8051 microcontroller. In *Cryptographic Hardware and Embedded Systems – CHES 2006 (LNCS 4249)*. Springer-Verlag, Berlin, Heidelberg, 430–444. https://doi.org/10.1007/11894063_34

[30] Mun-Kyu Lee, Keon Tae Kim, Howon Kim, and Dong Kyue Kim. 2006. Efficient hardware implementation of elliptic curve cryptography over $\mathrm{GF}(p^m)$. In *Int. Workshop on Information Security Applications – WISA 2005 (LNCS 3786)*. Springer-Verlag, Berlin, Heidelberg, 207–217. https://doi.org/10.1007/11604938_16

[31] Chae Hoon Lim and Pil Joong Lee. 1997. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *Advances in Cryptology – CRYPTO '97 (LNCS 1294)*. Springer-Verlag, Berlin, Heidelberg, 249–263. https://doi.org/10.1007/BFb0052240

[32] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. 2007. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer-Verlag, Berlin, Heidelberg.

[33] Peter L. Montgomery. 1987. Speeding the Pollard and elliptic curve methods of factorization. *Math. of Comp.* 48, 177 (Jan. 1987), 243–264. https://doi.org/10.1090/S0025-5718-1987-0866113-7

[34] National Institute of Standards and Technology (NIST). 2013. Digital Signature Standard (DSS), FIPS PUB 186-4. Retrieved June 24, 2019 from https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[35] Ilia Polian and Francesco Regazzoni. 2017. Counteracting malicious faults in cryptographic circuits. In *22nd IEEE European Test Symp. (ETS)*. IEEE, Piscataway, NJ, USA, 10. https://doi.org/10.1109/ETS.2017.7968230

[36] Matthieu Rivain. 2011. Fast and regular algorithms for scalar multiplication over elliptic curves. *IACR Cryptology ePrint Archive,* Report 2011/338 (2011), 25 p. https://eprint.iacr.org/2011/338.pdf

[37] Yolan Romailler and Sylvain Pelissier. 2017. Practical fault attack against the Ed25519 and EdDSA signature schemes. In *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC 2017*. IEEE Computer Society, Piscataway, NJ, USA, 17–24. https://doi.org/10.1109/FDTC.2017.12

[38] Joseph H. Silverman. 1996. *The Arithmetic of Elliptic Curves* (2 ed.). Graduate Texts in Math., Vol. 106. Springer-Verlag, New York, USA.

[39] Eric Simpson and Patrick Schaumont. 2006. Offline hardware/software authentication for reconfigurable platforms. In *Cryptographic Hardware and Embedded Systems – CHES 2006 (LNCS 4249)*. Springer-Verlag, Berlin, Heidelberg, 311–323. https://doi.org/10.1007/11894063_25

[40] Jasper G. J. van Woudenberg, Marc F. Witteman, and Federico Menarini. 2011. Practical optical fault injection on secure microcontrollers. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, Piscataway, NJ, USA, 91–99.