Algorithm-level Error Detection for ECSM

Agustin Dominguez-Oviedo*
Department of Mechatronics
ITESM Campus Queretaro, Mexico

M. Anwar Hasan
Department of Electrical and Computer Engineering
University of Waterloo, Canada

January 12, 2009

Abstract

For some applications, elliptic curve cryptography (ECC) is an attractive choice because it achieves the same level of security with a much smaller key size in comparison with other schemes such as those that are based on integer factorization or discrete logarithm. Unfortunately, cryptosystems including those based on elliptic curves have been subject to attacks. For example, fault-based attacks have been shown to be a real threat in today's cryptographic implementations. For security reasons, especially to provide resistance against fault-based attacks, it is very important to verify the correctness of computations in ECC applications. We deal with protections to fault attacks against ECSM at algorithm level. To this end, we use the concepts of point verification (PV) and coherency check (CC). We investigate the error detection coverage of PV and CC for the Montgomery ladder ECSM algorithm. Additionally, we propose two algorithms based on the double-and-add-always method that are resistant to the safe error (SE) attack. We demonstrate that one of these algorithms also resists the sign change fault (SCF) attack.

1 Introduction

Elliptic curve cryptography (ECC) was independently proposed by both Miller [33] and Koblitz [27] in 1985. Since then, ECC has been a subject of extensive research and standardization efforts that have led it to be widely known and accepted. Some of the ECC standards include: FIPS 186 [20], IEEE P1363 [23], ANSI X9.62 [4], and ISO 15946 [24]. Cryptosystems including those based on elliptic curves have been subject to attacks. Cryptoanalytic attacks may reveal system vulnerabilities, which then need to be addressed with countermeasures. Various researchers have emphasized the significance of cryptographic applications being resistant to side-channel analysis (e.g., reconstruction of a secret key from

*This work was done when the first author was with the University of Waterloo.

analysis of timing [28], power consumption signals [29], and electromagnetic emanations [1] during cryptographic operations).

Another type of attacks that has received considerable attention is the fault analysis attack. Introduced by Boneh et al. [11], this attack is based on producing malfunctions in cryptosystems to leak sensitive information (i.e., secret keys). They have shown how some cryptographic schemes, such as the RSA and Rabin digital signatures, are vulnerable to induced computational errors. In order to avoid such attacks, they suggest verifying the correctness of computations in cryptographic applications.

Anderson and Kuhn reported a practical fault attack [2]. It is based on producing faults in instructions rather than in data. The underlying idea does not appear to be new; in fact, it seems that this technique was used by amateur hackers on satellite television smart cards [3]. It consists of applying a high frequency glitch into the clock or power supply signals. Due to different delays in the processor's internal signal paths, this glitch might affect only some signals. Varying the timing and duration of the glitch, the attacker can possibly enforce to execute different wrong instructions which might compromise some sensitive information (e.g., a stored cryptographic key). Recently, Kim and Quisquater [26] showed how general propose microcontrollers can be targets of a so called *double*-fault attack, i.e., one attack to the RSA signature generation and the other to part of the status register (i.e., zero flag). Their fault injection method is based on inducing a glitch which makes a transient fault with a voltage spike. These glitches are used to corrupt data transferred between registers and memory or to prevent the execution of the code. They mount successfully this attack on a microcontroller computing the Chinese remainder theorem (CRT) based RSA signature generation algorithm.

Biehl et al. [7] extended fault-based attacks to cryptosystems using elliptic curves [7]. They proposed two attacks. The basic idea behind the first attack is to enforce, by a fault, a computation in a weaker group where solving the elliptic curve discrete logarithm problem (ECDLP) is feasible. Using this principle, they show how it is possible to derive secret information (e.g., a secret key) from a device that computes the elliptic curve scalar multiplication (ECSM). They assume that it is possible for an attacker to select the input point P or to induce a fault in that point. The second proposed attack is known as differential fault analysis (DFA) attack. In fact the latter is an extension of the attack presented by Boneh et al. [11] for RSA cryptosystems. This attack assumes that the adversary knows the implementation details, i.e., the underlying ECSM algorithm, the curve parameters, and the internal variables representation. Also, they consider that the result of the error-free computation of an EC scalar multiplication $Q = kP$ is known, where scalar k is the secret. In this case if the attacker injects a single-bit fault (i.e., bit flip) in a register that holds an ECSM partial result, and the faulty result $\widetilde{Q}$ is released, it is possible to reduce the exhaustive search space. If this process is repeated varying the timing of the attack, then the scalar bits can be retrieved in small blocks.

In order to resist the attacks presented by Biehl et al. [7], one can simply verify that the output is on the valid elliptic curve. This process is known as point verification (PV). However, this basic countermeasure is not sufficient for two other fault based attacks: the safe error (SE) attack proposed by Yen and Joye [39] and Yen et al. [40], and the sign change fault (SCF) attack presented by Blömer et al. [10]. The former shows the vulnerability of algorithms that utilize dummy instructions for making a uniform execution flow such as the double-and-add-always method (e.g., [16]). On the other hand, the SCF attack is applicable to elliptic curves over prime fields, where a sign change in a point implies only a change of sign of its y-coordinate. An interesting aspect of this attack is that the elliptic curve operations do not need to leave the original group $E(\mathbb{F}_p)$. They assume that the attacker can induce a fault that produces a sign change into an intermediate point during the ECSM operation. After having a set of erroneous results due to SCF attacks and the correct result Q, it is possible to recover the scalar k for a input pair (k, P).

In general, fault attacks take advantage of errors that occur while a cryptographic device is performing a private-key operation. Such errors may be induced by a malicious adversary who has physical access to the device or may occur because of hardware failure. An adversary may derive sensitive information from the incorrect output. Thus, error detection is an essential process from a security point of view. In the case of ECC, PV has been shown to be an important countermeasure against fault attacks. However, since there exist attacks where PV is not sufficient, it is necessary to include other protections.

As described above, fault-based attacks against cryptosystems are a real threat and should be taken into account. Accordingly, the design of cryptosystems should include some countermeasures against fault-based attacks. In this report we present our work on algorithm-level error detection in ECSM. First, we analyze the error detection coverage of PV and coherency check (CC) for the Montgomery ladder ECSM algorithm over the binary field. Then, we provide left-to-right and right-to-left double-and-add-always ECSM methods that will resist an SE attack. We show that the right-to-left version will also resist an SCF attack. Next, we discuss the case where two faults could be injected in one run of the ECSM, the first where sensitive information is used, and the second for skipping conditional tests. Finally, we provide a countermeasure to this strong attack model.

The organization of the remainder of this report is as follows. In Section 2, we give a brief overview of ECC including some algorithms to compute the ECSM. Additionally, we give the error detection strategies and assumptions that are utilized. In Section 2, we consider error detection in the Montgomery ladder ECSM. Section 3 presents error detection in the double-and-add-always method. In Section 4, we give a countermeasure that can be used against the double-fault attack. Finally, we make some concluding remarks in Section 5.

2 Background

2.1 Elliptic curve cryptography overview

An *elliptic curve* E over a finite field $GF(q)$ is defined by the following equation:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \quad a_i \in GF(q). \tag{1}$$

For binary finite fields, from Equation (1), one can perform an admissible change of variables that transforms E to the non-supersingular curve:

$$y^2 + xy = x^3 + ax^2 + b. \tag{2}$$

The points (x, y) that satisfy Equation (2), along with the point at infinity ($\mathcal{O}$), and a particular operation known as point addition ($\uplus$), form an abelian group. The number of points on the elliptic curve E, denoted as $\#E$, is know as the *order* of E. If P is a point on the curve

E and k is a positive integer, then the *elliptic curve scalar multiplication* (ECSM) is given as follows $kP = P \uplus P \uplus \cdots \uplus P$ (k times).

The operations needed to perform the ECSM operation can be computed utilizing the well-known *double-and-add* method as shown in Algorithm 1. This algorithm scans bits of scalar k from left to right (i.e., from the most significant bit to the least significant bit), one bit at a time. In every iteration, a point doubling is performed. Additionally, depending on the scanned bit value, a point addition is performed.

Algorithm 1. Left-to-right ECSM by double-and-add

```
Input: $P \in E(\mathbb{F}_q)$, $k = (k_{t-1} \cdots k_1\ k_0)_2$.
Output: $Q = kP$.

1. $Q \leftarrow \mathcal{O}$.
2. For $i = t-1$ downto 0 do
    2.1 $Q \leftarrow 2Q$.
    2.2 If ($k_i = 1$) then
        2.2.1 $Q \leftarrow Q \uplus P$.
3. Return($Q$).
```

Coron [16] has shown that algorithms with a non-homogeneous operation flow, such as Algorithm 1, are vulnerable to a simple power analysis (SPA) attack. As a countermeasure he proposed a method called double-and-add-always. The idea is to add a *dummy* point addition operation whenever the bit scalar is equal to zero during the main loop. The corresponding method is presented in Algorithms 2 and 3 for the left-to-right and right-to-left versions, respectively. The included dummy operation permits to have a uniform execution flow, i.e., one point addition and one point doubling are executed in every iteration during the loop. As a result, there is a performance penalty since the required point operations are t doublings and t additions. Moreover, Yen and Joye [39] have observed that algorithms with dummy operations might be susceptible to a special fault attack called *safe-error* (SE) attack. This is an example that in some cases a countermeasure against one attack may benefit another attack.

2.2 Montgomery's ECSM method

Montgomery [34] presented a method to compute multiples of points for a special type of elliptic curve over prime fields. His technique has been generalized to other curves of cryptographic interests [31] [35] [13]. Utilizing a variant of the binary method known as *binary ladder* [17], Montgomery's idea is based on the fact that the addition of two points can be obtained without the y-coordinates of such points knowing the difference between them.

The binary ladder method follows the next observation [25][1]. Let $k = \sum_{j=0}^{t-1} k_j 2^j$ and $Q = kP$ be the scalar and ECSM result, respectively. Let us define two integers as $L_i = \sum_{j=i}^{t-1} k_j 2^{j-i}$ and $M_i = L_i + 1$. Then we can obtain L_{i+1} as

[1] Joye and Yen described this idea in the context of modular exponentiation.

Algorithm 2. Left-to-right ECSM by double-and-add-always

Input: $P \in E(\mathbb{F}_q)$, $k = (k_{t-1} \cdots k_1\ k_0)_2$.
Output: $Q = kP$.

```
1. $Q_0 \leftarrow \mathcal{O}$.
2. For $i = t-1$ downto 0 do
    2.1 $Q_0 \leftarrow 2Q_0$.
    2.2 $Q_1 \leftarrow Q_0 \uplus P$.
    2.3 $Q_0 \leftarrow Q_{k_i}$.
3. Return($Q_0$).
```

Algorithm 3. Right-to-left ECSM by double-and-add-always

Input: $P \in E(\mathbb{F}_q)$, $k = (k_{t-1} \cdots k_1\ k_0)_2$.
Output: $Q = kP$.

```
1. $Q_0 \leftarrow \mathcal{O}$.
2. For $i = 0$ to $t-1$ do
    2.1 $Q_1 \leftarrow Q_0 \uplus P$.
    2.2 $P \leftarrow 2P$.
    2.3 $Q_0 \leftarrow Q_{k_i}$.
3. Return($Q_0$).
```

$$L_{i+1} = \sum_{j=i+1}^{t-1} k_j 2^{j-i-1} = \frac{1}{2}(L_i - k_i).$$

We can write expressions for L_i and M_i as follows,

$$\begin{aligned} L_i &= 2L_{i+1} + k_i = L_{i+1} + M_{i+1} + k_i - 1, \\ M_i &= 2M_{i+1} + k_i - 1 = L_{i+1} + M_{i+1} + k_i. \end{aligned}$$

Using the above equations, let us define the pair (L_i, M_i) as

$$(L_i, M_i) = \begin{cases} (2L_{i+1}, L_{i+1} + M_{i+1}) & \text{if } k_i = 0, \\ (L_{i+1} + M_{i+1}, 2M_{i+1}) & \text{if } k_i = 1. \end{cases} \tag{3}$$

Now, let $Q_{0,i} = L_i P$ and $Q_{1,i} = M_i P$ be points for $i \in \{0, t-1\}$. Utilizing Equation (3), we can obtain the pair $(Q_{0,i}, Q_{1,i})$ as follows

$$(Q_{0,i}, Q_{1,i}) = \begin{cases} (2Q_{0,i+1}, Q_{0,i+1} \uplus Q_{1,i+1}) & \text{if } k_i = 0, \\ (Q_{0,i+1} \uplus Q_{1,i+1}, 2Q_{1,i+1}) & \text{if } k_i = 1. \end{cases} \tag{4}$$

Note that if $L_{t-1} = 1$ and $M_{t-1} = 2$ (i.e., $Q_{0,t-1} = P$ and $Q_{1,t-1} = 2P$), and we use Equation (4) repeatedly for i from $t-2$ to 0, then $Q_{0,0}$ and $Q_{1,0}$ will be kP and $(k+1)P$, respectively. The complete procedure that uses this idea is presented as Algorithm 4. This algorithm is referred to as *basic* Montgomery's ladder ECSM. Let us use the word "basic" to distinguish this algorithm among others that do not utilize the y-coordinate of the intermediate points Q_0 and Q_1 during the ECSM computation.

Algorithm 4. Basic Montgomery's ladder ECSM

Input: $P \in E(\mathbb{F}_q)$, $k = (k_{t-1} \cdots k_1\ k_0)_2$ with $k_{t-1} = 1$.
Output: $Q = kP$.

```
1. $Q_0 \leftarrow P$, $Q_1 \leftarrow 2P$.
2. For $i = t-2$ downto 0 do
    2.1 If $(k_i = 0)$ then
        2.1.1 $Q_1 \leftarrow Q_0 \uplus Q_1$, $Q_0 \leftarrow 2Q_0$;
    2.2 Else
        2.2.1 $Q_0 \leftarrow Q_0 \uplus Q_1$, $Q_1 \leftarrow 2Q_1$.
3. Return$(Q_0)$.
```

This algorithm keeps the difference between Q_1 and Q_0 equal to P at any value of i during the loop. Also, one point doubling and one point addition are performed in every iteration which make this algorithm attractive against attacks such as timing [28] and SPA [29]. Additionally, since it does not have dummy instructions, the SE fault attack does not apply. Furthermore, due to the usage of all point coordinates, it is possible to have an improved error detection using coherency check among involved variables.

An extension of the ECSM Montgomery idea for non-supersingular elliptic curves over the binary finite field was presented by López and Dahab [31]. They showed algorithms for both affine and projective coordinate systems. Let us present some resulting expressions from lemmas given by López and Dahab [31]. Let $P_0 = (x_0, y_0)$ and $P_1 = (x_1, y_1)$ be points that belong to the elliptic curve defined by Equation (2). The x-coordinate of $P_0 \uplus P_1$, x_2, can be obtained as follows:

$$x_2 = \frac{x_0 y_1 + x_1 y_0 + x_0 x_1^2 + x_0^2 x_1}{(x_0 + x_1)^2}. \tag{5}$$

Suppose that $P = (x, y)$ is the difference between P_1 and P_0, i.e., $P_1 - P_0 = P$. If P is known, then the x-coordinate of $P_0 \uplus P_1$ can be obtained by the following function:

$$\mathtt{x}(P_0 \uplus P_1) = \begin{cases} x_0^2 + \dfrac{b}{x_0^2} & \text{if } P_0 = P_1, \\ x + \dfrac{x_0}{x_0 + x_1} + \left(\dfrac{x_0}{x_0 + x_1}\right)^2 & \text{if } P_0 \neq P_1. \end{cases} \tag{6}$$

Additionally the y-coordinate of P_0, y_0, can be obtained from $P = (x, y)$, and the x-coordinates of P_0 and P_1 (i.e., x_0 and x_1, respectively) as follows:

$$y_0 = \frac{(x_0 + x)\left[(x_0 + x)(x_1 + x) + x^2 + y\right]}{x} + y. \tag{7}$$

Based on Algorithm 4 and Equations (6) and (7), Algorithm 5 implements the affine version of Montgomery's ECSM. During each interaction of the algorithm, a point doubling and a point addition are performed without y-coordinate. This is possible due to the difference of the two intermediate points, namely Q_0 and Q_1, being known (i.e., $= P$). After the final interaction the x-coordinates of $Q_0 = Q = kP$ and $Q_1 = (k+1)P$ are obtained, i.e., Q_{0_x} and Q_{1_x}. Using these values and P, the y-coordinate of the result is computed in Step 3. Note that in Algorithm 5 $\mathtt{x}(\cdot)$ corresponds to the x-coordinate of the point given in the argument.

Algorithm 5. Montgomery's ladder ECSM in affine coordinates

Input: $P = (x, y) \in E(\mathbb{F}_{2^m})$, $k = (k_{t-1} \cdots k_1\ k_0)_2$ with $k_{t-1} = 1$.
Output: $Q = kP$.

```
1. $Q_{0_x} \leftarrow x$, $Q_{1_x} \leftarrow \mathtt{x}(2P)$.
2. For $i = t-2$ downto 0 do
   2.1 If $(k_i = 0)$ then
      2.1.1 $Q_{1_x} \leftarrow \mathtt{x}(Q_0 \uplus Q_1)$, $Q_{0_x} \leftarrow \mathtt{x}(2Q_0)$;
   2.2 Else
      2.2.1 $Q_{0_x} \leftarrow \mathtt{x}(Q_0 \uplus Q_1)$, $Q_{1_x} \leftarrow \mathtt{x}(2Q_1)$.
3. $Q_{0_y} = (Q_{0_x} + x)\left[(Q_{0_x} + x)(Q_{1_x} + x) + x^2 + y\right]/x + y$.
4. Return$(Q_{0_x}, Q_{0_y})$.
```

In Algorithm 5 the intermediate points and their related operations are in the affine coordinate system. In applications where the multiplicative inverse is relatively expensive, it might be more attractive to use a projective coordinate system. To this end, López and Dahab [31] presented the projective version of Algorithm 5. This particular algorithm represents an attractive option because it gives a computational advantage over other algorithms that do not use pre-computation such as that the binary [32] and addition-subtraction [23] methods.

Let $P_0 = (X_0, Y_0, Z_0)$ and $P_1 = (X_1, Y_1, Z_1)$ be points represented in the López and Dahab projective coordinates system. Suppose that $P = (x, y)$ is the difference in affine coordinates between P_1 and P_0. Then, the Z- and X-coordinates of the point doubling and addition can be obtained by the following functions:

$$\mathtt{Z}(P_0 \uplus P_1) = Z_2 = \begin{cases} X_0^2 Z_0^2 & \text{if } P_0 = P_1, \\ (X_0 Z_1 + X_1 Z_0)^2 & \text{if } P_0 \neq P_1. \end{cases} \tag{8}$$

$$\mathtt{X}(P_0 \uplus P_1) = X_2 = \begin{cases} X_0^4 + b Z_0^4 & \text{if } P_0 = P_1, \\ x Z_2 + X_0 X_1 Z_0 Z_1 & \text{if } P_0 \neq P_1. \end{cases} \tag{9}$$

Using these group formulas, it is possible to have an efficient algorithm to compute the ECSM. Algorithm 6 presents the resulting method, where $\mathtt{X}(\cdot)$ and $\mathtt{Z}(\cdot)$ correspond to the X- and Z-coordinate, respectively, of the point given in the arguments. Similar to the algorithm of the affine system, point operations are performed without Y-coordinates since the difference between Q_1 and Q_0 is known. Here, after the last interaction the X- and Z-coordinates of $Q_0 = Q = kP$ and $Q_1 = (k+1)P$ are obtained, i.e., Q_{0_X}, Q_{0_Z}, Q_{1_X}, and Q_{1_Z}. Using these values, the affine representation of $Q = (x_2, y_2)$ is computed in Steps 5.1-5.3. In comparison with Algorithm 4, not handling the Y-coordinates helps to have fewer memory requirements and a faster method since some finite field multiplications are saved.

Algorithm 6. Montgomery's ladder ECSM in projective coordinates

Input: $P = (x, y) \in E(\mathbb{F}_{2^m})$, $k = (k_{t-1} \cdots k_1\ k_0)_2$ with $k_{t-1} = 1$.
Output: $Q = kP$.

```
1. $Q_{0_X} \leftarrow x$, $Q_{0_Z} \leftarrow 1$, $Q_{1_X} \leftarrow \mathtt{X}(2P)$, $Q_{1_Z} \leftarrow \mathtt{Z}(2P)$.
2. For $i = t-2$ downto 0 do
   2.1 If $(k_i = 0)$ then
      2.1.1 $T \leftarrow \mathtt{Z}(Q_0 \uplus Q_1)$, $Q_{1_X} \leftarrow \mathtt{X}(Q_0 \uplus Q_1)$, $Q_{1_Z} \leftarrow T$,
      2.1.2 $T \leftarrow \mathtt{X}(2Q_0)$, $Q_{0_Z} \leftarrow \mathtt{Z}(2Q_0)$, $Q_{0_X} \leftarrow T$;
   2.2 Else
      2.2.1 $T \leftarrow \mathtt{Z}(Q_0 \uplus Q_1)$, $Q_{0_X} \leftarrow \mathtt{X}(Q_0 \uplus Q_1)$, $Q_{0_Z} \leftarrow T$,
      2.2.2 $T \leftarrow \mathtt{X}(2Q_1)$, $Q_{1_Z} \leftarrow \mathtt{Z}(2Q_1)$, $Q_{1_X} \leftarrow T$.
3. If $(Q_{1_Z} = 0)$ then return($\mathcal{O}$);
4. Else if $(Q_{0_Z} = 0)$ then return$(x, x+y)$;
5. Else
   5.1 $T \leftarrow 1/(xQ_{0_Z}Q_{1_Z})$,
   5.2 $x_2 \leftarrow xQ_{0_X}Q_{1_Z}T$,
   5.3 $y_2 \leftarrow (x + x_2)\left[(x^2 + y)Q_{0_Z}Q_{1_Z} + (Q_{0_X} + xQ_{0_Z})(Q_{1_X} + xQ_{1_Z})\right]T + y$,
   5.4 Return$(x_2, y_2)$.
```

2.3 Error detection strategies

In this subsection we give an overview of some techniques that can be used for error detection in ECSM. First, a method that is specific for applications using elliptic curves, namely PV process, is described. Next we describe an error detection technique using coherency check (CC) among the involved variables that can be utilized for having protections at the algorithm level.

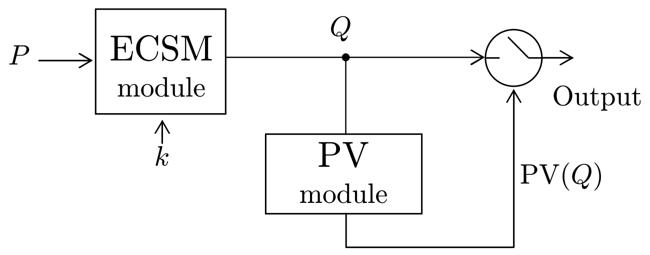

Figure 1: Point verification (PV) module after the ECSM module

2.3.1 Error detection using point verification (PV)

For ECC, PV can be used to detect errors during the ECSM. This is a basic countermeasure to prevent some fault attacks on ECC and it has been considered by Biehl et al. [7], Ciet and Joye [15], Antipa et al. [5], Blömer et al. [10], and Domínguez and Hasan [18]. A PV module takes a point Q as its input and checks whether the point is in $E(\mathbb{F}_q)$ (see Figure 1). This checking is done simply by verifying whether the coordinates of Q satisfy the governing elliptic curve equation, e.g., Equation (2) for curves defined over binary fields and represented with affine coordinates. If they do, only then an affirmative signal (say 'ok') is generated as output, i.e.,

$$\mathrm{PV}(Q) = \begin{cases} \mathrm{ok} = 1 & \text{if } Q \in E(\mathbb{F}_q), \\ 0 & \text{otherwise.} \end{cases}$$

The PV process requires only a few finite field operations, and hence, its implementation is relatively easy. Although, PV is an important countermeasure against some ECC fault-based attacks, this process alone is not sufficient for some cases, let us describe two examples:

- *SCF attack*: The SCF attack is based on changing the sign into an targeted intermediate point during the ECSM operation. Then, the faulty points never leave the original elliptic curve E and the PV module alone cannot provide resistance against this attack.
- *SE attack*: Let us assume that we have a PV module after the ECSM module as illustrated in Figure 1. Consider that the ECSM algorithm uses dummy instructions for having a uniform execution (e.g., Algorithm 3). If an SE attack is mounted as described in [39] and [40], the attacker can perform a similar oracle attack. Consequently, PV does not provide the sufficient protection.

Since some attacks are not prevented with PV module alone, it is necessary to add other protections.

2.3.2 Error detection using coherency check (CC)

Consistency or coherency check (CC) process verifies the intermediate or final results with respect to a valid pattern. In the context of public key cryptography, this technique has been used for detecting errors during the RSA signature generation. As shown by Boneh et al. [11], the RSA digital signature scheme using the CRT is particularly vulnerable to fault attacks, i.e., with only one faulty signature and its corresponding error-free version it is possible to efficiently factor the modulus used. A natural countermeasure for this attack is to verify the signature utilizing the public exponent. However, in some applications this value might not be available. Additionally, if the public exponent is not small, then the verification could be costly. For these reasons, a number of countermeasures that include protections inside the modular exponentiation algorithm have been proposed in the literature (e.g., [36], [42], [9], [14], [21], and [12][2]). The countermeasures proposed by Giraud [21] and Boscher et al. [12] use CC for detecting errors during the modular exponentiation operation. Let us describe both approaches.

Giraud [21] has shown how the SPA-and-SE resistant algorithm published by Joye and Yen [25] can be utilized to prevent fault analysis (FA) attacks. This modular exponentiation algorithm is based on the Montgomery ladder method. Let N be the product of two large primes. Let $(d_{n-1} \cdots d_1\, d_0)_2$ be the binary representation of the exponent d. Consider that the signature $S = m^d \mod N$ of a message m is computed with Giraud's FA-resistant modular exponentiation algorithm (Algorithm 7). The basic idea for detecting errors in this algorithm is based on the fact that the pair (a_0, a_1) is of the form $(m^\beta, m^{\beta+1})$ after each iteration during the loop. If there is an error in either the modular multiplication (Step 2.1) or the squaring (Step 2.2), then the coherency between a_0 and a_1 will be lost. In such a case, since for each iteration the values of a_0 and a_1 are used for obtaining the next pair (a_0, a_1), it is expected that the final pair is not of the form (m^d, m^{d+1}). In this way, a coherency verification step can be included after the main loop to check if $a_0 \cdot m \equiv a_1 \pmod{N}$. Whether or not the computed signature S is returned depends on this test.

Recently, Boscher et al. [12] proposed a new FA-resistant modular exponentiation algorithm (Algorithm 8). Their idea is very close to Giraud's one. The main difference is that they use the square-and-multiply-always method instead of the Montgomery ladder. This algorithm permits the verification of coherency among three variables, namely a_0, a_1, and A. At the end of the loop the expected value for these variables is:

$$\begin{aligned} a_0 &= m^d \mod N, \\ a_1 &= m^{2^n - d - 1} \mod N, \\ A &= m^{2^n} \mod N. \end{aligned}$$

In fact, in an error-free computation this relation holds

$$a_0 \cdot a_1 \cdot m = m^{2^n} = A.$$

In this way, errors can be detected at the end of the modular exponentiation if the above expression does not hold as illustrated in Step 3 of Algorithm 8.

[2]Many of them have been shown vulnerable when they are analyzed under a different fault model (e.g., [6], [37], [8], and [26]).

Algorithm 7. Giraud's FA-resistant modular exponentiation

Input: $m \neq 0$, $d = (d_{n-1} \cdots d_1\ d_0)_2$, N.
Output: $S = m^d \mod N$.

```
1. $a_0 \leftarrow 1$, $a_1 \leftarrow m$.
2. For $i = n-1$ downto 0 do
    2.1 $a_{\overline{d_i}} \leftarrow a_{\overline{d_i}} \cdot a_{d_i} \mod N$.
    2.2 $a_{d_i} \leftarrow a_{d_i}^2 \mod N$.
3. $t \leftarrow a_0 \cdot m \mod N$.
4. If $(t = a_1)$ then
    4.1 Return($a_0$).
5. Return("Error detected").
```

Algorithm 8. Boscher-Naciri-Prouff's modular exponentiation

Input: $m \neq 0$, $d = (d_{n-1} \cdots d_1\ d_0)_2$, N.
Output: $S = m^d \mod N$.

```
1. $a_0 \leftarrow 1$, $a_1 \leftarrow 1, A \leftarrow m$.
2. For $i = 0$ to $n-1$ do
    2.1 $a_{\overline{d_i}} \leftarrow a_{\overline{d_i}} \cdot A \mod N$.
    2.2 $A \leftarrow A^2 \mod N$.
3. $t \leftarrow a_0 \cdot a_1 \cdot m \mod N$.
4. If $(t = A)$ and $(A \neq 0)$ then
    4.1 Return($a_0$);
5. Return("Error detected").
```

2.4 Basic assumptions

This report we present error detection at the algorithm level for ECSM. Here, we add protections inside the ECSM algorithm in order to detect errors caused either by naturally occurred or deliberately injected faults. For this purpose, we use point verification (PV) and coherency check (CC) among selected variables utilized for the ECSM. The CC functions that we define are algorithm specific. On the contrary, PV can be applied to any ECSM method.

In the following sections we investigate the error detection capability of different methods in ECSM. For the remainder of this report, the following assumptions are made:

- Any variable utilized in the underlying ECSM algorithm can be a target of natural or deliberately injected by an attacker. For simplicity in the analysis we assume that variables such as the loop counter i and the scalar k can be checked for integrity in order to prevent disturbance of their values.
- Faults might occur directly in the registers containing the variables using a flip-bit fault model or at a finite field arithmetic level. In the latter, any error might spread into one or more variables.
- In Sections 3 and 4 we assume that decisional tests (e.g., "if (PV(Q) = 1) then") are not susceptible to faults. This might be the case for secure microcontrollers utilized in today's smart cards where hardware protections are added for not permitting fault injection into sensitive registers (e.g., CPU's status register). In Section 5 we relax this assumption considering the *double*-fault attack proposed by Yen et al. [42] and refined by Kim and Quisquater [26] for RSA cryptosystems.

- In this report a number of algorithm specific functions for CC are defined. These are labelled as CCi, where $i \in [1, 4]$ (i.e., CC1-CC4).

Let us define a vector $(V_0, V_1, \dots V_{j-1})$ which is composed of the variables utilized in the ECSM algorithm. As a consequence of faults occurred naturally or injected deliberately the vector might be changed to $(\widetilde{V}_0, \widetilde{V}_1, \dots \widetilde{V}_{j-1})$. Depending on the resultant vector, the error-detecting scheme may detect the presence of an error caused by the fault. Whether or not the error is detected depends on the rules utilized for error detection (e.g., PV(Q)) and the actual values of the specific variables to be tested. We can see this by analogy as a binary code of coding theory. For this case the length of the code n is the size (in bits) of the vector $(V_0, V_1, \dots V_{j-1})$. The codewords are those cases where no error is detected, i.e., the error-detecting scheme cannot distinguish between an error-free computation and a faulty one. For comparing the error-detecting schemes presented in this report we use the following definition [30]:

Definition 1 *The ratio $c_R = \log_2(r)/n$ is called the* code rate*, where r and n are the number of codewords and length of the code, respectively.*

The error detection capability of a particular coding scheme is correlated to its code rate. A higher code rate can be seen as high information content and low coding overhead. However, the fewer bits used for coding redundancy, the less error protection is provided [38].

3 Error detection in the Montgomery ladder algorithm

In this section we present our work on error detection in the Montgomery ladder ECSM algorithm for non-supersingular elliptic curves over the binary finite field proposed by López and Dahab [31]. First, we consider the case where a PV process is placed at the end of the ECSM. Then, we use CC among the variables involved for error detection. We give a comparison between both approaches that suggests the use of an integrity check (IC) with either a PV or CC process.

Let $k = (k_{t-1} \cdots k_1\ k_0)_2$ and $Q = kP$ be the scalar and the ECSM result, respectively, where $P = (x, y) \in E(\mathbb{F}_{2^m})$ and $n = \texttt{ord}(P)$. First, let us define the "exceptional" cases for the ECSM result be $Q = \pm P$, $Q = \mathcal{O}$, and $Q = (0, \sqrt{b})$. In the Montgomery ladder algorithm (Algorithm 4), for fault-free computations the exceptional cases arise from the following values of k:

$$k = \begin{cases} n+1 & \text{(i.e., } Q_0 = P,\ Q_1 = 2P\text{)},\\ n-1 & \text{(i.e., } Q_0 = -P,\ Q_1 = \mathcal{O}\text{)},\\ n & \text{(i.e., } Q_0 = \mathcal{O},\ Q_1 = P\text{)},\\ n/2 & \text{(i.e., for } n \text{ even } Q_0 = (0, \sqrt{b})\text{)}. \end{cases}$$

In the algorithms presented in this section let us restrict these exceptional cases for error-free computations, i.e., simply by restricting the input k from being $n \pm 1$ and n, and $n/2$ if n is even.

3.1 PV process at the end of the ECSM

The Montgomery ladder algorithm in affine coordinates proposed by López and Dahab [31] (Algorithm 5) is shown in Algorithm 9 with a PV process at the end of the ECSM. This algorithm restricts the occurrence of the exceptional cases described above by excluding certain values for the input scalar. In Step 3.1, as defined by Equation (7), the y-coordinate of the output (i.e., Q_{0_y}) is computed using the following function:

$$g(Q_{0_x}, Q_{1_x}, x, y) = \frac{(Q_{0_x} + x)\left[(Q_{0_x} + x)(Q_{1_x} + x) + x^2 + y\right]}{x} + y. \tag{10}$$

For Algorithm 9 let us obtain the cases where $\mathrm{PV}(Q_0) = 1$ in Step 3.2, i.e., where Q_0 is released as the ECSM output. Clearly this occurs always in an error-free computation. However, this is not likely to be the case when errors are produced by faults occurring naturally or injected deliberately by an attacker. Let us assume that an adversary can induce fault(s) during the execution of the ECSM. Consider that this produces an incorrect result $\widetilde{Q} \neq Q$ which will be checked by the PV process.

Algorithm 9. Montgomery's ladder ECSM with PV at the end

Input: $P = (x, y) \in E(\mathbb{F}_{2^m})$ of order n, where n is an odd prime. A positive integer $k = (k_{t-1} \cdots k_1 k_0)_2$, where $k_{t-1} = 1$, $k \neq n$, and $k \neq n \pm 1$.
Output: $Q = kP$.

```
1. $Q_{0_x} \leftarrow x$, $Q_{1_x} \leftarrow \mathtt{x}(2P)$.
2. For $i = t - 2$ downto 0 do
    2.1 If $(k_i = 0)$ then
        2.1.1 $Q_{1_x} \leftarrow \mathtt{x}(Q_0 \uplus Q_1)$, $Q_{0_x} \leftarrow \mathtt{x}(2Q_0)$;
    2.2 Else
        2.2.1 $Q_{0_x} \leftarrow \mathtt{x}(Q_0 \uplus Q_1)$, $Q_{1_x} \leftarrow \mathtt{x}(2Q_1)$.
3. If $((Q_{0_x} \neq x)$ and $(Q_{0_x} \neq 0)$ and $(x \neq 0))$ then
    3.1 $Q_{0_y} = g(Q_{0_x}, Q_{1_x}, x, y)$.
    3.2 If $(\mathrm{PV}(Q_0) = 1)$ then return$(Q_{0_x}, Q_{0_y})$;
    3.3 Else return("Error detected").
4. Else return("Error detected").
```

Let us consider that any variable utilized by the ECSM algorithm can be affected by errors due to faults. For the case of Algorithm 9 instead of having an error-free vector (Q_{0_x}, Q_{1_x}, x, y) we have the erroneous vector $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$. From now on, let us refer to Q_{0_x}, Q_{1_x}, x, and y as the final value of these variables after the main loop. Assume that the adversary is able to inject faults during the main loop, i.e., where the sensitive information (i.e., scalar k) is utilized. With the above considerations let us obtain the cases where $\mathrm{PV}(\widetilde{Q}_0) = 1$. We can substitute the point $\widetilde{Q}_0 = (\widetilde{Q}_{0_x}, \widetilde{Q}_{0_y})$ in the governing elliptic curve equation (Equation (2)) to obtain:

$$\widetilde{Q}_{0_y}^2 + \widetilde{Q}_{0_x}\widetilde{Q}_{0_y} = \widetilde{Q}_{0_x}^3 + a\widetilde{Q}_{0_x}^2 + b. \tag{11}$$

Using Equation (10), in Step 3.1 $\widetilde{Q}_{0_y}$ is computed as a function of $\widetilde{Q}_{0_x}$, $\widetilde{Q}_{1_x}$, $\widetilde{x}$, and $\widetilde{y}$. Replacing this value of $\widetilde{Q}_{0_y}$ in Equation (11) we can get the following quadratic expression for $\widetilde{Q}_{1_x}$

$$\left(\frac{\widetilde{Q}_{0_x}^4}{\widetilde{x}^2} + \widetilde{x}^2\right)\widetilde{Q}_{1_x}^2 + \left(\frac{\widetilde{Q}_{0_x}^3}{\widetilde{x}} + \widetilde{x}\widetilde{Q}_{0_x}\right)\widetilde{Q}_{1_x} + \widetilde{Q}_{0_x}^2\left(\widetilde{Q}_{0_x}^2 + \frac{\widetilde{y}^2}{\widetilde{x}^2} + \widetilde{x}^2 + \frac{\widetilde{y}}{\widetilde{x}} + \widetilde{x} + a\right) + b = 0.$$

The above equation has two solutions if and only if $\mathtt{Tr}(w_1) = 0$, where

$$w_1 = \widetilde{Q}_{0_x}^2 + \frac{\widetilde{y}^2}{\widetilde{x}^2} + \widetilde{x}^2 + \frac{\widetilde{y}}{\widetilde{x}} + \widetilde{x} + a + \frac{b}{\widetilde{Q}_{0_x}^2}. \tag{12}$$

In such a case the two solutions for $\widetilde{Q}_{1_x}$ are

$$\widetilde{Q}_{1_x(a)} = \frac{\widetilde{x}\widetilde{Q}_{0_x}}{\widetilde{x}^2 + \widetilde{Q}_{0_x}^2}\mathtt{Ht}(w_1), \tag{13}$$

$$\widetilde{Q}_{1_x(b)} = \frac{\widetilde{x}\widetilde{Q}_{0_x}}{\widetilde{x}^2 + \widetilde{Q}_{0_x}^2}(\mathtt{Ht}(w_1) + 1). \tag{14}$$

If $\mathtt{Tr}(w_1) = 0$ and the relation among $\widetilde{Q}_{0_x}$, $\widetilde{Q}_{1_x}$, $\widetilde{x}$, and $\widetilde{y}$ satisfies either Equation (13) or (14), then $\mathrm{PV}(\widetilde{Q}_0) = 1$. In this scenario PV fails to detect such errors.

Lemma 1 *For an arbitrary vector $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$, the number of combinations where $PV(\widetilde{Q}_0) = 1$ in Step 3.2 of Algorithm 9 is about 2^{3m}.*

Proof For this case for each possible value of $\widetilde{Q}_{0_x}$ (i.e., $\#E(\mathbb{F}_{2^m})/2-2$) there are two solutions for $\widetilde{Q}_{1_x}$. For fixed $\widetilde{Q}_{0_x}$ and $\widetilde{Q}_{1_x}$, the number of possible values for $\widetilde{x}$ and $\widetilde{y}$ is $2^m - 2$ and 2^m, respectively (i.e., $\widetilde{x} \neq \widetilde{Q}_{0_x}$, and $\widetilde{x} \neq 0$). Thus, the number of combinations where $\mathrm{PV}(\widetilde{Q}_0) = 1$ in Step 3.2 of Algorithm 9 is

$$2^m(\#E(\mathbb{F}_{2^m}) - 2)(2^m - 2) \approx 2^{3m}.$$

□

Using Lemma 1 we can obtain the code rate c_R for this case as

$$c_R = \frac{\log_2(2^m(\#E(\mathbb{F}_{2^m}) - 2)(2^m - 2))}{4m} \approx \frac{3}{4}. \tag{15}$$

In Step 3.1 of Algorithm 9 Q_{0_y} is obtained as a function of Q_{0_x}, Q_{1_x}, x, and y. This computation assumes that the difference between Q_1 and Q_0 is P. If due to a fault this

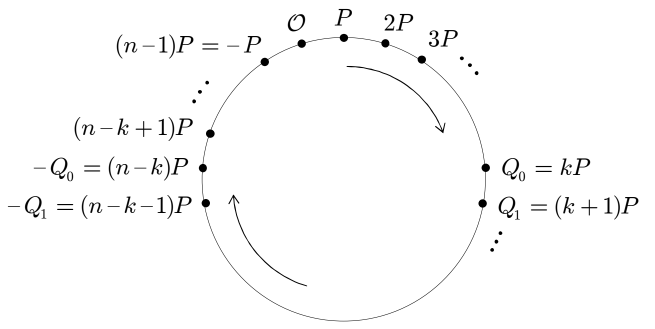

Figure 2: Multiples of point P of order n

difference is lost, then presumably the corresponding $\widetilde{Q}_0$ will become a finite field pair that does not belong to $E(\mathbb{F}_{2^m})$. Then PV process at the end of ECSM will detect such errors. This is not the case for the combinations obtained in Lemma 1.

3.2 CC process at the end of the ECSM

Instead of obtaining Q_{0_y} assuming that $Q_1 - Q_0 = P$, we can verify first if the coherency among Q_{0_x}, Q_{1_x}, x, and y does exist. Only if it does, the corresponding ECSM output is released. For checking the coherency among a given vector (Q_{0_x}, Q_{1_x}, x, y) we can proceed as follows. First, we can search for a point $\widehat{Q}_0 \in E(\mathbb{F}_{2^m})$ of the form $(Q_{0_x}, \widehat{Q}_{0_y})$ for some $\widehat{Q}_{0_y} \in E(\mathbb{F}_{2^m})$. This step involves the solution of a quadratic equation for $\widehat{Q}_{0_y}$ from the elliptic curve equation. Let us consider an error-free computation for which $\widehat{Q}_0$ will always exist. In fact, for $Q_{0_x} \neq 0$ there are two solutions of the quadratic equation. Let us set $\widehat{Q}_{0_y}$ to one of these solutions. In this way, $\widehat{Q}_0$ will be either Q_0 or $-Q_0$, depending on which solution is selected. With $\widehat{Q}_0$, we can perform $\widehat{Q}_0 \uplus P$ which will result in either $Q_1 = (k+1)P$ or $(n-k+1)P$, where $n = \texttt{ord}(P)$ (see Figure 2). Adding $-\widehat{Q}_0 \uplus P$ also permits identifying which one of $\widehat{Q}_0$ or $-\widehat{Q}_0$ corresponds to Q_0. For CC purposes, since the only information available about Q_1 is its x-coordinate, we can perform only $\texttt{x}(\widehat{Q}_0 \uplus P)$ and $\texttt{x}(-\widehat{Q}_0 \uplus P)$ and compare the results with Q_{1_x}. Let us define the CC function that defines the error detection rules for this scheme as:

$$\text{CC1}(Q_{0_x}, Q_{1_x}, x, y) = \begin{cases} \text{ok} = 1 & \text{if } \texttt{x}(\widehat{Q}_0 \uplus P) = Q_{1_x} \text{ or } \texttt{x}(-\widehat{Q}_0 \uplus P) = Q_{1_x}, \\ 0 & \text{otherwise.} \end{cases}$$

Algorithm 10 implements this function. Additionally, if the CC passes, then the corresponding Q_{0_y} is also returned.

Algorithm 10. Computing CC1 and Q_{0_y}

Input: Q_{0_x}, Q_{1_x}, x, y.
Output: CC1(Q_{0_x}, Q_{1_x}, x, y), Q_{0_y}.

```
1. If $((Q_{0_x} \neq x)$ and $(Q_{0_x} \neq 0)$ and $(x \neq 0))$ then
    1.1 $w_2 \leftarrow Q_{0_x} + b/Q_{0_x}^2 + a$.
    1.2 If $(\mathtt{Tr}(w_2) = 0)$ then
        1.2.1 $\widehat{Q}_{0_y} \leftarrow Q_{0_x} \cdot \mathtt{Ht}(w_2)$
        1.2.2 $T_1 \leftarrow 1/(Q_{0_x} + x)$.
        1.2.3 $T_2 \leftarrow T_1 \cdot (\widehat{Q}_{0_y} + y)$.
        1.2.4 $T_2 \leftarrow T_2^2 + T_2 + Q_{0_x} + a$.
        1.2.5 If $(T_2 = Q_{1_x})$ then return$(1, \widehat{Q}_{0_y})$;
        1.2.6 Else
                $T_2 \leftarrow T_1 \cdot (Q_{0_x} + \widehat{Q}_{0_y} + y)$.
                $T_2 \leftarrow T_2^2 + T_2 + Q_{0_x} + a$.
                If $(T_2 = Q_{1_x})$ then return$(1, Q_{0_x} + \widehat{Q}_{0_y})$;
                Else return(0, "Error detected").
    1.3 Else Return(0,"Error detected").
2. Else return(0, "Error detected").
```

The Montgomery ladder ECSM algorithm that uses function CC1 at the end for error detection is presented as Algorithm 11. Now let us examine the cases where Algorithm 11 releases Q_0, i.e., CC1$(Q_{0_x}, Q_{1_x}, x, y) = 1$. This is always the case for error-free computations. Similar to PV in Algorithm 9, let us assume an erroneous vector $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$. Let us define $\varepsilon_P \in \mathbb{F}_{2^m}$ as $\varepsilon_P = \widetilde{y}^2 + \widetilde{x}\widetilde{y} + \widetilde{x}^3 + a\widetilde{x}^2 + b$. Based on this definition, $\widetilde{P} \in (\mathbb{F}_{2^m})$ iff $\varepsilon_P = 0$. It can be shown that $\widehat{Q}_0 = (\widetilde{Q}_{0_x}, \widehat{Q}_{0_y}) \in E(\mathbb{F}_{2^m})$ exists if and only if $\mathtt{Tr}(w_2) = 0$, where

$$w_2 = \widetilde{Q}_{0_x} + \frac{b}{\widetilde{Q}_{0_x}^2} + a. \tag{16}$$

In such a case, $\widehat{Q}_{0_y}$ is set to one of the two quadratic equation solutions, i.e., $\widehat{Q}_{0_y} = \widetilde{Q}_{0_x}\mathtt{Ht}(w_2)$. Utilizing the same group formulas as Algorithm 10, we can obtain $\mathtt{x}(\widehat{Q}_0 \uplus \widetilde{P})$ and $\mathtt{x}(-\widehat{Q}_0 \uplus \widetilde{P})$. Whenever any of these results is equal to $\widetilde{Q}_{1_x}$, function CC1 will fail to detect such errors.

The corresponding values for $\widetilde{Q}_{1_x}$, where this condition is satisfied, are:

$$\widetilde{Q}_{1_x(a)} = \frac{\widetilde{x}\widetilde{Q}_{0_x}}{\widetilde{x}^2+\widetilde{Q}_{0_x}^2}\left(\mathtt{Ht}(w_2)+\widetilde{x}+\frac{\widetilde{y}}{\widetilde{x}}+\widetilde{Q}_{0_x}+\frac{\varepsilon_P}{\widetilde{x}\widetilde{Q}_{0_x}}\right), \tag{17}$$

$$\widetilde{Q}_{1_x(b)} = \frac{\widetilde{x}\widetilde{Q}_{0_x}}{\widetilde{x}^2+\widetilde{Q}_{0_x}^2}\left(\mathtt{Ht}(w_2)+\widetilde{x}+\frac{\widetilde{y}}{\widetilde{x}}+\widetilde{Q}_{0_x}+\frac{\varepsilon_P}{\widetilde{x}\widetilde{Q}_{0_x}}+1\right). \tag{18}$$

Algorithm 11. The Montgomery ladder ECSM in affine coordinates with CC

Input: $P=(x,y)\in E(\mathbb{F}_{2^m})$ of order n, where n is an odd prime. A positive integer $k=(k_{t-1}\cdots k_1k_0)_2$, where $k_{t-1}=1$, $k\neq n$, and $k\neq n\pm 1$.
Output: $Q=kP$.

```
1. $Q_{0_x} \leftarrow x$, $Q_{1_x} \leftarrow \mathtt{x}(2P)$.
2. For $i = t-2$ downto 0 do
    2.1 If $(k_i = 0)$ then
        2.1.1 $Q_{1_x} \leftarrow \mathtt{x}(Q_0 \uplus Q_1)$, $Q_{0_x} \leftarrow \mathtt{x}(2Q_0)$;
    2.2 Else
        2.2.1 $Q_{0_x} \leftarrow \mathtt{x}(Q_0 \uplus Q_1)$, $Q_{1_x} \leftarrow \mathtt{x}(2Q_1)$.
3. Use Algorithm 10 to compute $c = \mathrm{CC1}(Q_{0_x}, Q_{1_x}, x, y)$ and $Q_{0_y}$.
4. If $(c = 1)$ then return$(Q_{0_x}, Q_{0_y})$;
5. Else return("Error detected").
```

It can be shown that for an arbitrary $(\widetilde{Q}_{0_x},\widetilde{Q}_{1_x},\widetilde{x},\widetilde{y})$ the count of combinations where Algorithm 11 fails to detect errors is the same as Algorithm 9 obtained in Lemma 1. Consequently, c_R for Algorithm 11 is also $\approx 3/4$. In the next subsection we compare the error detection coverage of these two algorithms.

3.3 Error detection comparison between PV and CC1

In the following lemmas we show some similarities and differences in terms of error detection between $\mathrm{PV}(Q_0)$ and $\mathrm{CC1}(Q_{0_x},Q_{1_x},x,y)$. In particular, the next lemma shows that if one of these error-detecting approaches fails to detect a vector with a specific value of $\widetilde{Q}_{0_x}$, then the other approach also fails to detect some vectors with the same $\widetilde{Q}_{0_x}$.

Lemma 2 *Let (r,s,u,v) be a particular vector of $(\widetilde{Q}_{0_x},\widetilde{Q}_{1_x},\widetilde{x},\widetilde{y})$, where $r,u\in\mathbb{F}^*_{2^m}$, $s,v\in\mathbb{F}_{2^m}$, and $r\neq u$ (i.e., not considering the exceptional cases). Let $\widetilde{Q}_{0_y}=l$ be the value obtained by function g in Step 3.1 of Algorithm 9 as function of $r,s,u,$ and v.*

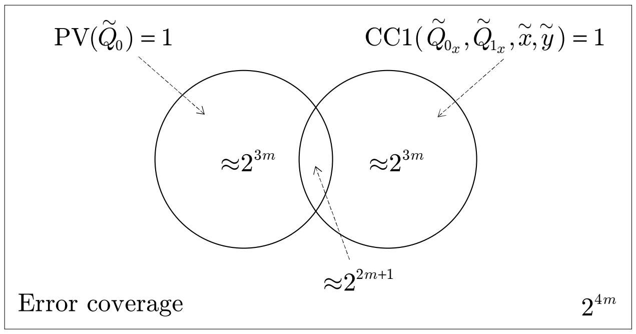
$\mathrm{PV}(\widetilde{Q}_0) = 1$
$\mathrm{CC1}(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y}) = 1$
$\approx 2^{3m}$
$\approx 2^{3m}$
$\approx 2^{2m+1}$
Error coverage
2^{4m}

Figure 3: Error detection coverage for arbitrary $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$

(i) If $PV((r,l)) = 1$, then there exists a vector (r, s', u', v') for which CC1$(r, s', u', v') = 1$ for some $s', v' \in \mathbb{F}_{2^m}$, and $u' \in \mathbb{F}^*_{2^m}$.

(ii) If CC1$(r, s, u, v) = 1$, then there exists a pair (r, l') for which $PV((r, l')) = 1$, for some $l' \in \mathbb{F}_{2^m}$.

Proof For PV$(\widetilde{Q}_0)$, to obtain solutions to $\widetilde{Q}_{1_x}$ using Equations (13) and (14) we should have $\mathtt{Tr}(w_1) = 0$, where w_1 is defined by Equation (12). Similarly for CC1(Q_{0_x}, Q_{1_x}, x, y), $\widehat{Q}_0 \in E(\mathbb{F}_{2^m})$ exists if and only if $\mathtt{Tr}(w_2) = 0$, where w_2 is defined by Equation (16). From Equation (12) we can obtain

$$\begin{aligned}\mathtt{Tr}(w_1) &= \mathtt{Tr}\left(\widetilde{Q}_{0_x}^2 + \frac{\widetilde{y}^2}{\widetilde{x}^2} + \widetilde{x}^2 + \frac{\widetilde{y}}{\widetilde{x}} + \widetilde{x} + a + \frac{b}{\widetilde{Q}_{0_x}^2}\right), \\ &= \mathtt{Tr}(\widetilde{Q}_{0_x}) + \mathtt{Tr}(a) + \mathtt{Tr}\left(\frac{b}{\widetilde{Q}_{0_x}^2}\right),\end{aligned}$$

which corresponds to $\mathtt{Tr}(w_2)$, i.e., $\mathtt{Tr}(w_1) = \mathtt{Tr}(w_2)$. Then (i) and (ii) are true since $\mathtt{Tr}(w_1)$ (and $\mathtt{Tr}(w_2)$) depends on $\widetilde{Q}_{0_x}$ and not on $\widetilde{Q}_{1_x}$, $\widetilde{x}$, or $\widetilde{y}$. □

In some cases, these error-detecting approaches fail to detect the same vectors. As illustrated in Figure 3, this happens in about 2^{2m+1} of the possible combinations of arbitrary $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$. This is explained in the following lemma.

Lemma 3 *Let l, r, s, u, and v be defined as in Lemma 2.*

(i) There exists a vector (r, s, u, v) for which $PV((r,l)) = 1$ and CC1$(r, s, u, v) = 1$, i.e., cases where both Algorithms 9 and 11 fail in detecting the same vector $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$ (the overlapping area in Figure 3).

(ii) For all cases where (i) is satisfied, ε_P is either 0 or $r \cdot u$ (i.e., 0 or $\widetilde{x}\widetilde{Q}_{0_x}$).

(iii) The number of combinations that satisfies (i) is about 2^{2m+1}.

Proof For having the same vector $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$ for which both $\mathrm{PV}(\widetilde{Q}_0) = 1$ and $\mathrm{CC1}(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y}) = 1$, it is necessary from Equations (13-5) and (17-9) that either

$$\widetilde{Q}_{1_x(a)\mathrm{PV}} = \widetilde{Q}_{1_x(a)\mathrm{CC1}} \quad \text{or} \quad \widetilde{Q}_{1_x(a)\mathrm{PV}} = \widetilde{Q}_{1_x(b)\mathrm{CC1}}.$$

Clearly, if one of these conditions is satisfied, then either $\widetilde{Q}_{1_x(b)\mathrm{PV}} = \widetilde{Q}_{1_x(a)\mathrm{CC1}}$ or $\widetilde{Q}_{1_x(b)\mathrm{PV}} = \widetilde{Q}_{1_x(b)\mathrm{CC}}$. If we equate Equations (13) and (17), and Equations (13) and (18) we obtain

$$\widetilde{x}\widetilde{Q}_{0_x}\mathtt{Tr}\left(\widetilde{Q}_{0_x} + \frac{\widetilde{y}}{\widetilde{x}} + \widetilde{x}\right) = \varepsilon_P, \tag{19}$$

$$\widetilde{x}\widetilde{Q}_{0_x}\left[\mathtt{Tr}\left(\widetilde{Q}_{0_x} + \frac{\widetilde{y}}{\widetilde{x}} + \widetilde{x}\right) + 1\right] = \varepsilon_P, \tag{20}$$

respectively. Depending on the value of $\mathtt{Tr}(\widetilde{Q}_{0_x} + \frac{\widetilde{y}}{\widetilde{x}} + \widetilde{x})$, Equations (19) and (20) are reduced, one to

$$\varepsilon_P = 0, \tag{21}$$

and the other to

$$\varepsilon_P = \widetilde{x}\widetilde{Q}_{0_x}, \tag{22}$$

which shows that (i) and (ii) are true. In summary, for a given vector $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$, if $\widetilde{Q}_{0_x}$, $\widetilde{Q}_{1_x}$, $\widetilde{x}$, and $\widetilde{y}$ satisfy either Equation (13) or (14), and either Equation (21) or (22), then $\mathrm{PV}(\widetilde{Q}_0)$ and $\mathrm{CC1}(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$ do not detect such an erroneous vector. Now let us obtain the number of combinations that satisfies (i). Let us consider separately the cases where $\varepsilon_P = 0$ from those with $\varepsilon_P = \widetilde{x}\widetilde{Q}_{0_x}$:

- *Case 1:* $\varepsilon_P = 0$. Here $\widetilde{P} \in E(\mathbb{F}_{2^m})$, where $\widetilde{x} \neq \widetilde{Q}_{0_x}$, and $\widetilde{x} \neq 0$. For each $\widetilde{P}$ there are $(\#E(\mathbb{F}_{2^m})/2 - 1)$ possible values of $\widetilde{Q}_{0_x}$ with two solutions for each one. Accordingly, for this case there are $(\#E(\mathbb{F}_{2^m}) - 2)(\#E(\mathbb{F}_{2^m}) - 4) \approx 2^{2m}$ combinations.

- *Case 2:* $\varepsilon_P = \widetilde{x}\widetilde{Q}_{0_x}$. From the definition of ε_P we have:

$$\widetilde{y}^2 + \widetilde{x}\widetilde{y} + \widetilde{x}^3 + a\widetilde{x}^2 + b + \widetilde{x}\widetilde{Q}_{0_x} = 0. \tag{23}$$

  The resultant quadratic expression for $\widetilde{y}$ will have a solution if and only if:

$$\mathtt{Tr}\left(\widetilde{x} + a + \frac{b}{\widetilde{x}^2}\right) = \mathtt{Tr}\left(\frac{\widetilde{Q}_{0_x}}{\widetilde{x}}\right). \tag{24}$$

  For arbitrary $\widetilde{x}$, where $\widetilde{x} \in \mathbb{F}^*_{2^m}$ and $\widetilde{x} \neq \widetilde{Q}_{0_x}$ the left side of Equation (24) is 0 in $\#E(\mathbb{F}_{2^m}) - 4$ cases, i.e., it is the same condition for solving the quadratic on the elliptic curve equation (Equation (2)). On the other hand, for arbitrary $\widetilde{Q}_{0_x}$ and $\widetilde{x}$ the right side of this equation is expected to be 0 for one half of the cases and 1 for the other half. As a consequence, Equation (24) will be satisfied in about half of the combinations for which two solutions for $\widetilde{y}$ are obtained for arbitrary $\widetilde{Q}_{0_x}$ and $\widetilde{x}$. Accordingly, for each possible value of $\widetilde{Q}_{0_x}$ we can have about 2^m pairs of $(\widetilde{x}, \widetilde{y})$ that satisfy Equation (23). Then the number of combinations for which both Algorithms 9 and 11 fail when $\varepsilon_P = \widetilde{x}\widetilde{Q}_{0_x}$ is $(\#E(\mathbb{F}_{2^m}) - 2)2^m \approx 2^{2m}$.

Adding the counts obtained when $\varepsilon_P = 0$ and $\varepsilon_P = \widetilde{x}\widetilde{Q}_{0_x}$ we obtain the value given in (iii) (i.e., $\approx 2^{2m+1}$). $\square$

From Lemma 3 we can deduce that if we combine PV(Q_0) and CC1(Q_{0_x}, Q_{1_x}, x, y) we can obtain a code rate of

$$c_R \approx \frac{2m+1}{4m} = \frac{1}{2} + \frac{1}{4m}. \tag{25}$$

However, from this lemma we can also see that if $\varepsilon_P = 0$ both error detection approaches have the same error detection coverage. This means that if we add a point verification process to P (i.e., PV(P)), we can use either PV(Q_0) or CC1(Q_{0_x}, Q_{1_x}, x, y) and have an improved c_R of about $\approx \frac{1}{2}$. This code rate of about 0.5 means that the number of redundant bits utilized for error detection is about half of the length of the code.

3.4 PV and integrity check (IC) at the end of the ECSM

In the previous two subsections we have provided an analysis of two different approaches to detect errors in the Montgomery ladder ECSM. The first consists of only verifying whether or not the result Q_0 lies on $E(\mathbb{F}_{2^m})$ which is a basic countermeasure against fault-based attacks and has been considered by Biehl et al. [7], Ciet and Joye [15], Antipa et al. [5], Blömer et al. [10], and Domínguez and Hasan [18]. The second approach consists of checking the coherency between the involved variables which is an extension of the work presented by Giraud [21] in the context of RSA cryptosystems. Even when both approaches (and their combination) give a very good code rate, it is possible to have a further improvement with practically no cost. The idea is to have an integrity check (IC) of P. That is, a verification after the main loop to check whether or not the register containing $P = (x, y)$ corresponds to the original input point. Hence, IC does not involve computations with other variables as required for CC. This idea is illustrated in Figure 4. The IC process can be implemented using duplication and/or the well-known cyclic redundancy check. Let us assume that this mechanism permits the detection of any alteration on the register containing P.

Let us obtain the cases where the error-detecting scheme presented in Figure 4 will not detect an erroneous output $\widetilde{Q}_0 \neq kP$. Using function g, $\widetilde{Q}_{0_y}$ is computed as a function of $\widetilde{Q}_{0_x}$, $\widetilde{Q}_{1_x}$, x, and y. Replacing this value of $\widetilde{Q}_{0_y}$ in Equation (11) we get the following quadratic expression for $\widetilde{Q}_{1_x}$

$$\left(\frac{\widetilde{Q}_{0_x}^4}{x^2} + x^2\right)\widetilde{Q}_{1_x}^2 + \left(\frac{\widetilde{Q}_{0_x}^3}{x} + x\widetilde{Q}_{0_x}\right)\widetilde{Q}_{1_x} + \widetilde{Q}_{0_x}^2\left(\widetilde{Q}_{0_x}^2 + \frac{y^2}{x^2} + x^2 + \frac{y}{x} + x + a\right) + b = 0.$$

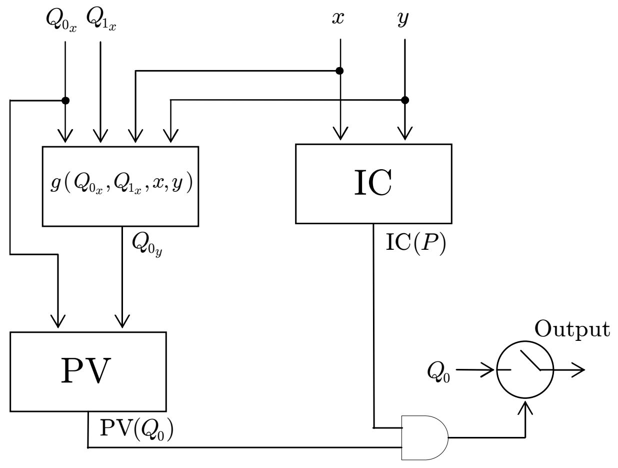

Figure 4: Error detection utilizing PV and IC processes

The above equation has two solutions if and only if $\mathtt{Tr}(w_3) = 0$, where

$$w_3 = \widetilde{Q}_{0_x}^2 + x^2 + \frac{b}{x^2} + \frac{b}{\widetilde{Q}_{0_x}^2}. \tag{26}$$

In such a case the two solutions for $\widetilde{Q}_{1_x}$ are

$$\widetilde{Q}_{1_x(a)} = \frac{x\widetilde{Q}_{0_x}}{x^2 + \widetilde{Q}_{0_x}^2}\mathtt{Ht}(w_3), \tag{27}$$

$$\widetilde{Q}_{1_x(b)} = \frac{x\widetilde{Q}_{0_x}}{x^2 + \widetilde{Q}_{0_x}^2}\left(\mathtt{Ht}(w_3) + 1\right). \tag{28}$$

Since $P \in E(\mathbb{F}_{2^m})$ the following relation holds

$$x\,\mathtt{Ht}\left(x + \frac{b}{x^2} + a\right) = \begin{cases} y & \text{or} \\ x + y. \end{cases}$$

Utilizing these values, Equations (27) and (28) can be rewritten as follows

$$\widetilde{Q}_{1_x(a)} = \frac{x\widetilde{Q}_{0_x}}{x^2 + \widetilde{Q}_{0_x}^2}\left[\mathtt{Ht}(w_3') + \frac{y}{x} + x + \mathtt{Tr}(x)\right], \tag{29}$$

$$\widetilde{Q}_{1_x(b)} = \frac{x\widetilde{Q}_{0_x}}{x^2 + \widetilde{Q}_{0_x}^2}\left[\mathtt{Ht}(w_3') + \frac{y}{x} + x + \mathtt{Tr}(x) + 1\right]. \tag{30}$$

where $w'_3 = \widetilde{Q}_{0_x}^2 + \frac{b}{\widetilde{Q}_{0_x}^2} + a$. Here for each possible value of $\widetilde{Q}_{0_x}$ (from a total of $(\#E(\mathbb{F}_{2^m})/2 - 2)$) there are two solutions to $\widetilde{Q}_{1_x}$. Thus, the number of combinations where this scheme fails in detecting an erroneous $\widetilde{Q}_0$ is $\#E(\mathbb{F}_{2^m}) - 4$.

For an arbitrary $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$, IC verifies any change in the register that contains point P. This permits having the following c_R:

$$c_R = \frac{\log_2(\#E(\mathbb{F}_{2^m}) - 4)}{4m} \approx \frac{1}{4}, \tag{31}$$

which represents the least value of c_R obtained so far. This is mainly based on the simple observation of checking the integrity of P. The overhead of this error-detecting mechanism is quite low in comparison with the main ECSM procedure. Assuming a random vector $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$, a code rate of 0.25 is equivalent of having a theoretical probability of undetected error of $1/2^{3m}$ which is zero for any practical scenario. However, the assumption of having a random vector of $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$ might not be true for faults injected by a sophisticated attacker (e.g., SCF attack for applications using elliptic curves defined over $\mathbb{F}_p$).

3.5 Basic Montgomery's ladder ECSM algorithm

We have shown that by using simple techniques such as PV and IC it is possible to detect errors efficiently in the variables involved. Now, let us consider the basic Montgomery ladder ECSM (Algorithm 4). As discussed in Subsection 2.2 this algorithm uses the y-coordinate of the intermediate points Q_0 and Q_1 during the ECSM computation. Thus, after the main loop we have all the coordinates of $Q_0 = kP$ and $Q_1 = (k+1)P$. Let us add a PV process for Q_0 and one IC process to P at the end of the ECSM, i.e., PV(Q_0) and IC(P). Now let us define a CC function as

$$\mathrm{CC2}(Q_0, Q_1, P) = \begin{cases} \text{ok} = 1 & \text{if } Q_0 \uplus P = Q_1, \\ 0 & \text{otherwise.} \end{cases}$$

Since $Q_0, Q_1, P \in E(\mathbb{F}_{2^m})$, for each pair (Q_0, P) there is only one value of Q_1 for which $\mathrm{CC2}(Q_0, Q_1, P) = 1$. Then, for an arbitrary vector $(\widetilde{Q}_0, \widetilde{Q}_1, \widetilde{P})$ the value for c_R is

$$c_R \approx \frac{m}{6m} = \frac{1}{6}. \tag{32}$$

Even though this approach theoretically has an improved error detection capability, it has an important drawback in terms of performance as illustrated in Table 1. This table compares the operation counts for ECSM among the Montgomery ladder algorithms proposed by López and Dahab [31] (Algorithms 5 and 6), the Montgomery ladder with PV at the end, the basic Montgomery ladder (Algorithm 4), and the basic Montgomery ladder (Algorithm 4) with PV(Q_0) and CC2(Q_0, Q_1, P). This table shows that the basic Montgomery ladder with PV(Q_0) and CC2(Q_0, Q_1, P) requires about double the number of finite field multiplications in comparison with the Montgomery ladder with PV at the end for the algorithms that use the affine system, i.e., $4tM$ vs. $(2t+4)M$. For the case of algorithms that use the projective coordinate system, the basic Montgomery ladder algorithm with PV(Q_0) and CC2(Q_0, Q_1, P)

requires about triple more finite field multiplications than the Montgomery ladder with PV at the end, i.e., $(18t - 6)M$ vs. $(6t + 6)M$.

Coordinate system	**ECSM Method**	**Operations required for the ECSM**
Affine ($\mathcal{A}$)	Montgomery ladder [1]	$(2t - 1)I + (2t + 2)M + 2tS$
	Montgomery ladder with PV(Q_0) [1,2]	$(2t - 1)I + (2t + 4)M + (2t + 2)S$
	Basic Montgomery ladder [3]	$(2t - 2)I + (4t - 4)M + (2t - 2)S$
	Basic Montgomery ladder with PV(Q_0) and CC2(Q_0, Q_1, P) [3,4]	$(2t - 1)I + 4tM + (2t + 1)S$
Projective ($\mathcal{LD}$)	Montgomery ladder [5]	$1I + (6t + 4)M + (5t - 2)S$
	Montgomery ladder with PV(Q_0) [5,2]	$1I + (6t + 6)M + 5tS$
	Basic Montgomery ladder [3]	$1I + (18t - 18)M + (9t - 9)S$
	Basic Montgomery ladder with PV(Q_0) and CC2(Q_0, Q_1, P) [3,4]	$1I + (18t - 6)M + (9t - 1)S$

[1]Using Algorithm 5. [2]With error detection scheme of Figure 4. [3]Using Algorithm 4.

[4]With PV(Q_0) and CC2(Q_0, Q_1, P). [5]Using Algorithm 6.

Table 1: Operation counts for computing the ECSM utilizing error detection at the algorithm level for Montgomery's ladder method

3.6 Discussion on Montgomery's ladder method

In this section we have considered error detection in Montgomery's ladder methods for non-supersingular elliptic curves over the binary finite field. Note that the SCF attack proposed by Blömer et al. [10] only applies to applications using curves over $\mathbb{F}_p$, and not for those using curves defined over $\mathbb{F}_{2^m}$. Additionally, because of its uniformity this method is resistant to attacks based on timing [28] and simple power analysis [29].

For invalid-curve attacks (e.g., [7], [15], and [5]) verifying whether the output is in $E(\mathbb{F}_{2^m})$ is crucially important. For all the approaches presented in this section that utilize PV and/or CC the output always relies on the original elliptic curve. In DFA, the attacker needs to inject faults in a given location during a number of runs of the ECSM algorithm and obtain erroneous results. However, as we have shown in this section the error detection coverage of the methods presented is quite high. This makes DFA impractical. However, it is well known that a security system is only as strong as its weakest link [19]. For applications utilizing elliptic curves over $\mathbb{F}_{2^m}$, from the attacker point of view, PV (or CC1 or CC2) can be seen as a strong protection that does not permit producing any faulty results as output. Then, the question is how this type of protections can be bypassed. And, in such a case what protections should be added against these strong adversaries. In Section 5 we consider the scenario where the attacker can inject a fault in the main algorithm and then bypass an error detection mechanism based on decisional tests (e.g., "if (PV(Q) = 1) then").

4 Error detection in ECSM by double-and-add-always

In this section we consider error detection in the double-and-add-always ECSM method proposed by Coron [16] for preventing the SPA attack. This method is based on adding dummy instructions to make the number of point operations constant during the main loop. However, Yen and Joye [39] and Yen et al. [40] have shown that adding dummy instructions makes an SE attack possible.

Here we first present a left-to-right version of the double-and-add-always that permits the use of mixed coordinates and is resistant to attacks such as SE and DFA attacks. Thereafter, we present an extension of the right-to-left version presented in the context of RSA cryptosystems by Boscher et al. [12] (Algorithm 8). For this approach we use the concepts of PV, CC, and IC presented in the previous section. We show that this ECC version of this algorithm can prevent not only SE and DFA attacks but also the SCF attack. This is an interesting result, since to the best of our knowledge the only countermeasures proposed against SCF attack are to use a called combined curve [10], utilizing the Montgomery ladder algorithm without the y-coordinates, and those that use scalar randomization [18]. For this section, let us consider elliptic curves defined over either $\mathbb{F}_{2^m}$ or $\mathbb{F}_p$, i.e., $\mathbb{F}_q$.

4.1 Left-to-right ECSM by double-and-add-always

Compared to Algorithm 1, Algorithm 2 adds a dummy instruction in Step 2.3 whenever the scanned bit of the scalar is 0, i.e., if $k_l = 0$ Step 2.3 becomes $Q_0 \leftarrow Q_0$ for any $l \in [0, t-1]$. Since the value of Q_1 is not utilized when $k_l = 0$ and it is overwritten at $i = l + 1$ with $Q_1 \leftarrow Q_0 \uplus P$, it can be a target of the SE attack.

The idea is to modify this algorithm in such a way that even during dummy instructions the alteration of the related registers could be detected. The resultant method is presented as Algorithm 12. Here, Q_0 follows the same sequence of operations as the original double and add method (Algorithm 1), i.e., each interaction Q_0 is doubled and if $k_i = 1$ then it is added with P. On the other hand, Q_1 will change only during the dummy instructions, i.e., whenever $k_i = 0$, Q_1 will be added to P. In this way at the end of the loop $Q_1 = H(\overline{k})P$, where $H(\overline{k})$ denotes the Hamming weight of the binary complement of k, i.e., $\overline{k} = 2^t - k - 1$. Note that the output Q_0 does not depend on Q_1. However, for defending against an SE attack we can check any alteration on Q_1 utilizing the following CC function:

$$\text{CC3}(k, P, Q_1) = \begin{cases} \text{ok} = 1 & \text{if } Q_1 = H(\overline{k})P, \\ 0 & \text{otherwise.} \end{cases}$$

Since $H(\overline{k}) \in [0, t-1]$, in the worst case the scalar multiplication $H(\overline{k})P$ takes $2\lceil \log_2(t-1) \rceil$ point operations. For example, for an elliptic curve defined over $\mathbb{F}_{2^{163}}$ and $t = m$, $H(\overline{k})P$ takes at the most 16 point operations. If P is known a priori, we can precompute jP for $j \in [2, t-1]$ and store those results in a table. Since k and P are verified for integrity, and assuming that the computation of $H(\overline{k})P$ is error free, CC3 function will detect any alteration in Q_1 resulting in an effective defense against the SE attack.

Algorithm 12. Left-to-right double-and-add-always ECSM with PV and CC

Input: $P = (x, y) \in E(\mathbb{F}_{2^m})$ of order n, where n is an odd prime. A positive integer $k = (k_{t-1} \cdots k_1 k_0)_2$.
Output: $Q = kP$.

```
1. $Q_0 \leftarrow \mathcal{O}$, $Q_1 \leftarrow \mathcal{O}$.
2. For $i = t-1$ downto 0 do
    2.1 $Q_0 \leftarrow 2Q_0$.
    2.2 $Q_{\overline{k_i}} \leftarrow Q_{\overline{k_i}} \uplus P$.
3. If ((PV($Q_0$) = 1) and (IC($P$) = 1) and (CC3($k, P, Q_1$) = 1)) then
    3.1 Return($Q_0$);
4. Else return("Error detected").
```

In addition to the SE attack protections, Algorithm 12 includes PV and IC processes. This permits for an arbitrary vector $(\widetilde{Q}_0, \widetilde{Q}_1, \widetilde{P})$ to have a code rate equivalent to the basic Montgomery ladder ECSM, i.e., from Equation (32) $c_R \approx \frac{1}{6}$. Even when this code rate might be quite good for some cases (e.g., random errors), for curves defined over $\mathbb{F}_p$ Algorithm 12 is insecure against the SCF attack. In a similar way as described by Blömer et al. [10], if the attacker can change the sign of the register containing the intermediate value of Q_0 at random values of i, then the attacker can retrieve the scalar. The problem of Algorithm 12 is that PV is not sufficient to avoid the SCF attack, i.e., faulty points never leave $E(\mathbb{F}_p)$. Additionally, the CC3 function does not verify alterations in Q_0, i.e., CC3 verifies alterations in Q_1 that permits to resist the SE attack. In the next subsection we show that using another CC function it is possible to resist the SCF attack for the right-to-left version of ECSM by double-and-add-always.

4.2 Right-to-left ECSM by double-and-add-always

Similar to the left-to-right version, Algorithm 3 performs a dummy instruction during the main loop (i.e., $Q_0 \leftarrow Q_0$) when $k_l = 0$ at Step 2.3, for $l \in [0, t-1]$. Extending the idea from Boscher et al. [12] to ECC, it is possible that Q_1 could hold a computation, different than kP, that could be checked at the end of the ECSM. The resultant method is presented as Algorithm 13. After the main loop the error free value of the following variables are:

$$\begin{aligned} Q_0 &= kP, \\ Q_1 &= \overline{k}P, \\ Q_2 &= 2^t P, \end{aligned}$$

where $\overline{k} = 2^t - k - 1$. Note that if we add Q_0, Q_1, and P we obtain

$$Q_0 \uplus Q_1 \uplus P = kP \uplus (2^t - k - 1)P \uplus P = 2^t P = Q_2.$$

Hence, for verifying the coherency among these points we can use the following CC function

$$\mathrm{CC4}(Q_0, Q_1, Q_2, P) = \begin{cases} \mathrm{ok} = 1 & \text{if } Q_2 = Q_0 \uplus Q_1 \uplus P, \\ 0 & \text{otherwise.} \end{cases}$$

In addition to CC, Algorithm 13 includes PV and IC processes. This permits for an arbitrary vector $(\widetilde{Q}_0, \widetilde{Q}_1, \widetilde{Q}_2, \widetilde{P})$ to have a code rate $c_R \approx \frac{1}{4}$.

Algorithm 13. Right-to-left double-and-add-always ECSM with PV and CC

Input: $P = (x, y) \in E(\mathbb{F}_{2^m})$ of order n, where n is an odd prime. A positive integer $k = (k_{t-1} \cdots k_1 k_0)_2$.
Output: $Q = kP$.

```
1. $Q_0 \leftarrow \mathcal{O}$, $Q_1 \leftarrow \mathcal{O}$, $Q_2 \leftarrow P$.
2. For $i = 0$ to $t - 1$ do
    2.1 $Q_{\overline{k_i}} \leftarrow Q_{\overline{k_i}} \uplus Q_2$.
    2.2 $Q_2 \leftarrow 2Q_2$.
3. If (($\mathrm{PV}(Q_0) = 1$) and ($\mathrm{PV}(Q_1) = 1$) and ($\mathrm{CC4}(Q_0, Q_1, Q_2, P) = 1$) and ($\mathrm{IC}(P) = 1$))
   then
    3.1 Return($Q_0$);
4. Else return("Error detected").
```

4.2.1 Security Analysis on Algorithm 13

As we discussed earlier, a DFA attack becomes impractical when PV of the output is performed. However there are two attacks, namely SE and SCF, for which PV does not provide enough protection. Let us consider these two attacks on Algorithm 13.

SE Attack: Let us assume that the attacker can inject one fault, single or multiple-bit, in one variable during the ECSM. In fact, since the output Q_0 does not depend on intermediate values of Q_1, the register holding the latter point might be exploited for mounting the SE attack. Suppose that the attacker injects a fault in Q_1 at a specific iteration i, i.e., $Q_{1,i}$ becomes $\widetilde{Q}_{1,i}$. This error will propagate for subsequent values of the variable Q_1. At the end of the main loop we will have $\widetilde{Q}_1 \neq \overline{k}P$. Here Algorithm 13 provides a two-level protection. The first is with PV of variable Q_1. For a random error in the corresponding coordinates of $Q_{1,i}$, $\mathrm{PV}(Q_1)$ might be sufficient to make SE impractical. However, suppose the attacker can inject a fault in such a way that $\widetilde{Q}_{1,i} \in E(\mathbb{F}_q)$ (e.g., SCF attack). For this case $\mathrm{PV}(\widetilde{Q}_1) = 1$ and CC4 provides a second protection. Here, CC4 will compute

$$Q_0 \uplus \widetilde{Q}_1 \uplus P = (k+1)P + \widetilde{Q}_1.$$

The latter value will be equal to Q_2 (i.e., CC4$(Q_0, \widetilde{Q}_1, Q_2, P) = 1$) if, and only if,

$$\widetilde{Q}_1 = (2^t - k - 1 + jn)P, \tag{33}$$

where j is an integer and $n = \texttt{ord}(P)$. For these exceptional cases PV and CC4 will fail detecting such errors. However in our opinion this is unlikely to happen in practice for the following reasons. First, the only known attack where $\widetilde{Q}_1 \in E(\mathbb{F}_q)$ is the SCF attack. Secondly, even if an SCF is injected on $Q_{1,i}$, the attacker is unlikely to have the precision for obtaining a final result $\widetilde{Q}_1$ that satisfies Equation (33) since he/she does not know k. As a result, Algorithm 13 can be considered as SE resistant.

SCF Attack: Let us assume that the attacker can inject one SCF into an intermediate point of Algorithm 13 at some random and unknown loop iteration $i \in [0, t-1]$. This fault model is similar to the one used by Blömer et al. [10]. Since the intermediate values of Q_0 and Q_2 are used for computing the final result of the ECSM, these two points might be susceptible to an SCF attack. We can obtain the error free output of Algorithm 13 $Q_0 = kP$ as

$$\begin{aligned} Q_0 &= k_{t-1}2^{t-1}P \uplus k_{t-2}2^{t-2}P \cdots \uplus k_{i+1}2^{i+1}P \uplus \underbrace{k_i 2^i P \cdots \uplus k_1 2P \uplus k_0 P}_{Q_{0,i}}, \\ &= \sum_{j=i+1}^{t-1} k_j 2^j P \uplus Q_{0,i}, \end{aligned} \tag{34}$$

where $Q_{0,i}$ is the intermediate value of Q_0 during some loop iteration $i \in [0, t-1]$, i.e., $Q_{0,i} = \sum_{j=0}^{i} k_j 2^j P$. Now, let us consider separately an SCF attack on intermediate values of Q_0 and Q_2.

- *SCF attack targeting Q_0 in Step 2.1.* Assume that the attacker mounts an SCF attack on Q_0 at some loop iteration $i \in [0, t-1]$, such that $\widetilde{Q}_{0,i} = -Q_{0,i}$. From Equation (34) we have

  $$\widetilde{Q}_0 = \sum_{j=i+1}^{t-1} k_j 2^j P - Q_{0,i} = \sum_{j=i+1}^{t-1} k_j 2^j P - \sum_{j=0}^{i} k_j 2^j P.$$

  The above expression can be rewritten as

  $$\widetilde{Q}_0 = Q_0 - 2\sum_{j=0}^{i} k_j 2^j P.$$

  Note that if the attacker knows $\widetilde{Q}_0$ and Q_0, and i is small enough for doing an exhaustive search, then he/she can obtain the $i+1$ least significant bits of k. This can be repeated for different and ascending values of i utilizing the known bits of k until the retrieval of the complete scalar. However, CC4 does not permit to release any $\widetilde{Q}_0 \neq Q_0$ since

$$\widetilde{Q}_0 \uplus Q_1 \uplus P = -2\sum_{j=0}^{i} k_j 2^j P \uplus 2^t P,$$

which does not match with $Q_2 = 2^t P$ for $\sum_{j=0}^{i} k_j \neq 0$. Hence, CC4 protects Algorithm 13 for the SCF attack in $Q_{0,i}$.

- *SCF attack targeting Q_2 in Step 2.2.* For this case the SCF attack is on Q_2 at some loop iteration $i \in [0, t-1]$, such that $\widetilde{Q}_{2,i} = -Q_{2,i}$. Since $Q_{2,i} = 2^{i+1}P$, $\widetilde{Q}_{2,i} = -2^{i+1}P$, and at the end of the loop we have

  $$\widetilde{Q}_2 = -2^t P.$$

  From Equation (34) we can obtain the faulty value $\widetilde{Q}_0$ at the end of the loop as

  $$\widetilde{Q}_0 = Q_{0,i} - \sum_{j=i+1}^{t-1} k_j 2^j P = \sum_{j=0}^{i} k_j 2^j P - \sum_{j=i+1}^{t-1} k_j 2^j P,$$

  which can be expressed as

  $$\widetilde{Q}_0 = 2\sum_{j=0}^{i} k_j 2^j P - kP.$$

  Similarly, it can be shown that the final value of $\widetilde{Q}_1$ is

  $$\widetilde{Q}_1 = 2\sum_{j=0}^{i} \overline{k}_j 2^j P - \overline{k}P.$$

  Then, with these values CC4 will compute

  $$\begin{aligned}\widetilde{Q}_0 \uplus \widetilde{Q}_1 \uplus P &= 2\sum_{j=0}^{i} k_j 2^j P \uplus \sum_{j=0}^{i} \overline{k}_j 2^j P - kP - \overline{k}P,\\ &= 2\sum_{j=0}^{i} 2^j P \uplus 2P - 2^t P,\\ &= 2^{i+2}P - 2^t P,\end{aligned}$$

  which matches $\widetilde{Q}_2 = -2^t P$ if, and only if,

  $$2^{i+2}P = \mathcal{O}.$$

  The latter cannot be true if the order of P is prime. As a result, CC4 will detect any SCF attack in $Q_{2,i}$.

Under the same fault model as the utilized by Blömer et al. [10], Algorithm 13 is SCF attack resistant. This is an interesting result because this algorithm does not use a combined curve [10] or randomization as RC and PC schemes presented in [18]. This protection is based on CC among the involved variables and it resists both the SE and the SCF attacks.

4.3 Costs for ECSM by double-and-add-always

In this section we have presented double-and-add-always ECSM methods that are resistant to both DFA attack and SE attack. The former is prevented simply by performing a PV of the output. For the algorithms presented in this section the SE attack is prevented with a CC of the variables involved. In Table 2 we present the cost in terms of finite field operations for the double-and-add-always algorithms utilizing elliptic curves defined over $\mathbb{F}_{2^m}$. The first four rows present the cost of Algorithms 2 and 3 which do not include any error-detecting processes. In contrast, the last four rows shows their counterparts that includes PV and CC (Algorithms 12 and 13). Table 3 shows estimates of these costs for a specific value of $t = m = 163$. In this table we can notice an overhead in terms of finite field operations for the algorithms that include PV and CC for error detection in comparison with the algorithms without error detection. For affine coordinates, these overheads are approximately $4.91\%I + 5.06\%M + 5.32\%S$ and $0.61\%I + 0.92\%M + 1.23\%S$ for Algorithms 12 and 13, respectively. On the other hand, for the projective coordinates case, the overheads are of about $4.95\%M + 5.08\%S$ and $1.00\%M + 1.07\%S$ for Algorithms 12 and 13, respectively. Additionally, we can notice a speed advantage of using the left-to-right versions over a their right-to-left counterparts when using projective coordinates, e.g., $1I + 1959M + 1142S$ vs. $1I + 2611M + 1305S$ for Algorithms 2 and 3, respectively; and $1I + 2056M + 1200S$ vs. $1I + 2637M + 1319S$ for Algorithms 12 and 13, respectively. This is mainly for the use of mixed coordinates.

Double-and-add-always method	**Coordinate system**	**SE-DFA-resistant**	**Operations required for the ECSM**
Left-to-right [1]	$\mathcal{A}$	No	$2tI + 4tM + 2tS$
	$\mathcal{LD}$	No	$1I + (12t + 2)M + (10t + 1)S$
Right-to-left [2]	$\mathcal{A}$	No	$2tI + 4tM + 2tS$
	$\mathcal{LD}$	No	$1I + (18t + 2)M + (9t + 1)S$
Left-to-right with PV and CC [3]	$\mathcal{A}$	Yes	$2t'I + (4t' + 2)M + (2t' + 2)S$ [5]
	$\mathcal{LD}$	Yes	$1I + (12t' + 4)M + (10t' + 3)S$ [5]
Right-to-left with PV and CC [4]	$\mathcal{A}$	Yes	$(2t + 2)I + (4t + 8)M + (2t + 6)S$
	$\mathcal{LD}$	Yes	$1I + (18t + 31)M + (9t + 18)S$

[1]Using Algorithm 2. [2]Using Algorithm 3. [3]Using Algorithm 12 to obtain kP and Algorithm 2 for $H(\overline{k})P$. [4]Using Algorithm 13. [5]$t' = t + \lceil \log_2(t-1) \rceil$.

Table 2: Operation counts for the double-and-add-always ECSM method for curves defined over $\mathbb{F}_{2^m}$

Similarly, Table 4 shows the count of finite field operations for the double-and-add-always algorithms utilizing elliptic curves defined over $\mathbb{F}_p$. In addition, Table 5 gives estimates of these costs for a specific value of t, i.e., $t = 192$. For the case of affine coordinates, in this table we can notice an overhead of approximately $4.17\%I + 4.30\%M + 4.51\%S$ and $0.52\%I + 0.78\%M + 1.04\%S$ for Algorithms 12 and 13, respectively. For the projective coordinates case, the overheads are of about $4.20\%M + 4.31\%S$ and $0.85\%M + 0.91\%S$ for Algorithms

Double-and-add-always method	**Coordinate system**	**Field operations**		
		I	M	S
Left-to-right [1]	$\mathcal{A}$	326	652	489
	$\mathcal{LD}$ [2]	1	1959	1142
Right-to-left [3]	$\mathcal{A}$	326	652	489
	$\mathcal{LD}$	1	2611	1305
Left-to-right with PV and CC [3]	$\mathcal{A}$	342	685	515
	$\mathcal{LD}$ [2]	1	2056	1200
Right-to-left with PV and CC [5]	$\mathcal{A}$	328	658	495
	$\mathcal{LD}$	1	2637	1319

[1]Using Algorithm 2. [2]Using mixed coordinates for point addition. [3]Using Algorithm 3. [4]Using Algorithm 12 to obtain kP and Algorithm 2 for $H(\overline{k})P$, assuming $H(\overline{k}) = \lceil t/2 \rceil = 82$. [5]Using Algorithm 13.

Table 3: Number of finite field operations for the double-and-add-always ECSM method for $t = m = 163$ for curves over $\mathbb{F}_{2^{163}}$

12 and 13, respectively. Additionally, we have shown that the right-to-left version of the double-and-add-always ECSM can be equipped with PV and CC in order to resist the SCF attack. The overhead if the ECSM is computed in affine coordinates is of about 0.53%[3]. For the case of Algorithm 13 utilizing projective coordinates, let us compare it with the left-to-right version without error detection. The main reason is due to the penalty for not using mixed coordinates. With this consideration, the overhead of Algorithm 13 utilizing projective coordinates is about 27.4%[1]. This contrasts with the $30-40\%$ reported by Blömer et al. [10] or about the 100% for RC or PC schemes of [18].

5 Double-fault attack resistant ECSM

Yen et al. [42] noted that error detection schemes based on decisional tests should be avoided. Their observation relies on the fact that decisional tests depend on the use of the zero flag of the CPU. Then, if the attacker can inject a fault in this bit of the status register, the conditional test may be bypassed. Today's smart cards are equipped with countermeasures that protect sensitive registers with robust mechanisms [21]. In such a case this attack might not be possible. However, Kim and Quisquater [26] showed that general propose microcontrollers can be a target of the so called *double*-fault attack, i.e., one attack to the RSA signature generation and the other to status register (i.e., zero flag). Since there is a growing use of portable and embedded systems that may not be protected for this type of attack, some protections should be added. In this section we present algorithm level countermeasures against this type of attacks for ECSM.

[3]We assume that $I = 80M$ and $S = 0.85M$ as [22].

Double-and-add-always method	**Coordinate system**	**SE-DFA-resistant**	**SCF-resistant**	**Operations required for the ECSM**
Left-to-right [1]	$\mathcal{A}$	No	No	$2tI + 4tM + 3tS$
	$\mathcal{J}$	No	No	$1I + (12t + 3)M + (7t + 1)S$
Right-to-left [2]	$\mathcal{A}$	No	No	$2tI + 4tM + 3tS$
	$\mathcal{J}$	No	No	$1I + (16t + 3)M + (8t + 1)S$
Left-to-right with PV and CC [3]	$\mathcal{A}$	Yes	No	$2t'I + (4t' + 1)M + (3t' + 2)S$ [5]
	$\mathcal{J}$	Yes	No	$1I + (12t' + 4)M + (7t' + 3)S$ [5]
Right-to-left with PV and CC [4]	$\mathcal{A}$	Yes	Yes	$(2t + 2)I + (4t + 6)M + (3t + 6)S$
	$\mathcal{J}$	Yes	Yes	$1I + (16t + 29)M + (8t + 15)S$

[1]Using Algorithm 2. [2]Using Algorithm 3. [3]Using Algorithm 12 to obtain kP and Algorithm 2 for $H(\overline{k})P$.
[4]Using Algorithm 13. [5]$t' = t + \lceil \log_2(t-1) \rceil$.

Table 4: Operation counts for the double-and-add-always ECSM method for curves defined over $\mathbb{F}_p$

Fault Model. The attacker can mount two attacks during one run of the ECSM algorithm. The first to the main algorithm in order to corrupt operations that depend on sensitive information (i.e., scalar k). The second to any decisional test used even on those with error detection procedures (e.g., "if (PV(Q) = 1) then").

Let us base our countermeasure on Montgomery's ladder ECSM for non-super- singular elliptic curves over the binary finite field proposed by López and Dahab [31] (Algorithm 5). However, these concepts can generally be extended to the corresponding Montgomery's ladder ECSM for prime fields presented by Brier and Joye [13]. Since SCF attack does not apply to Montgomery's ladder ECSM methods that do not use the y-coordinate for computing the ECSM, under this double-fault model the most dangerous threat is the DFA attack. An effective defensive measure against this attack can be achieved adding randomness to the computation in such a way that the attacker cannot obtain useful information from a faulty output. In fact, if a projective coordinate system is utilized one can simply apply base point randomization [16] for making DFA impractical. However, for the affine system we can use the concept of point "blinding" presented by Coron [16]. The idea is to add a random point R to the initial values of Q_0 and Q_1. Let a scalar r be defined as $r = \log_P R$, i.e., $R = rP$. Using Equation (3), the original Montgomery ladder algorithm sets $L_{t-1} = 1$ and $M_{t-1} = 2$ (i.e., $Q_0 = P$ and $Q_1 = 2P$), and we use Equation (4) repeatedly for i from $t-2$ to 0 to obtain $Q_0 = kP$ and $Q_1 = (k+1)P$. Now, we can set $L_{t-1} = 1 + r$ and $M_{t-1} = 2 + r$ (i.e., $Q_0 = P \uplus R$ and $Q_1 = 2P \uplus R$). Note that since R is added initially to both Q_0 and Q_1, at each iteration during the loop their corresponding part dependent on R is doubled. Thus, after the main loop the following is obtained:

$$Q_0 = kP \uplus 2^{t-1}R,$$
$$Q_1 = (k+1)P \uplus 2^{t-1}R.$$

Double-and-add-always method	**Coordinate system**	**Field operations**		
		I	M	S
Left-to-right [1]	$\mathcal{A}$	384	768	576
	$\mathcal{J}$ [2]	1	2307	1345
Right-to-left [3]	$\mathcal{A}$	384	768	576
	$\mathcal{J}$	1	3075	1537
Left-to-right with PV and CC [3]	$\mathcal{A}$	400	801	602
	$\mathcal{J}$ [2]	1	2404	1403
Right-to-left with PV and CC [5]	$\mathcal{A}$	386	774	582
	$\mathcal{J}$	1	3101	1551

[1]Using Algorithm 2. [2]Using mixed coordinates for point addition. [3]Using Algorithm 3. [4]Using Algorithm 12 to obtain kP and Algorithm 2 for $H(\overline{k})P$, assuming $H(\overline{k}) = \lceil t/2 \rceil = 96$. [5]Using Algorithm 13.

Table 5: Number of finite field operations for the double-and-add-always ECSM method for $t = 192$ for curves over $\mathbb{F}_{p_{192}}$

Then, to obtain $Q = kP$ we need to compute $Q = Q_0 - 2^{t-1}R$. This is depicted in Figure 5. The complete procedure that implements this version of Montgomery's ladder is presented as Algorithm 14. In this way, if the attacker can inject a fault to avoid the conditional test of Step 10, then the output's finite field pair will be released regardless of whether it is or not in $E(\mathbb{F}_{2^m})$. However, if the attacker also injects a fault during the main loop, where the sensitive information is utilized, the output might be not useful since he/she does not know R. In fact, a requirement for mounting the DFA proposed by Biehl et al. [7] is to know the details of the implementation, such as the parameters, the algorithm utilized, and the representation of internal variables. The latter is not satisfied by adding a random point R to the initial values of Q_0 and Q_1. The overhead of Algorithm 14 in comparison with Algorithm 5 that does not include any fault attack protections is about 50% more field multiplicative inverses and squarings, and about 100% more field multiplications, i.e., $(3t + 3)I + (4t + 6)M + (3t + 7)S$ vs. $(2t + 1)I + (2t + 1)M + (2t + 1)S$.

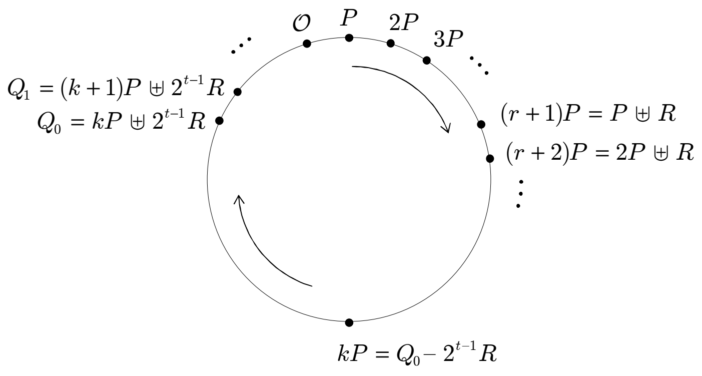

Figure 5: Blinding point P for the Montgomery ladder ECSM

Algorithm 14. Double-fault attack resistant Montgomery ladder ECSM

Input: $P = (x, y) \in E(\mathbb{F}_{2^m})$ of order n, where n is an odd prime. A positive integer $k = (k_{t-1} \cdots k_1\ k_0)_2$, where $k_{t-1} = 1$, $k \neq n$, and $k \neq n \pm 1$.
Output: $Q = kP$.

```
1. Pick a random point $R = (R_x, R_y) \in E(\mathbb{F}_{2^m})$ with odd prime order.
2. $Q_2 \leftarrow -R$.
3. $T \leftarrow (y + R_y)/(x + R_x)$.
4. $Q_{0_x} \leftarrow T^2 + T + x + R_x + a$.
5. $T \leftarrow x/(x + Q_{0_x})$.
6. $Q_{1_x} \leftarrow T^2 + T + R_x$
7. For $i = t - 2$ downto 0 do
    7.1 $T \leftarrow \mathtt{x}(Q_0 \uplus Q_{\overline{k_i}})$, $Q_{1_x} \leftarrow \mathtt{x}(Q_1 \uplus Q_{\overline{k_i}})$, $Q_{0_x} \leftarrow T$, $Q_2 \leftarrow 2Q_2$.
8. $Q_{0_y} = g(Q_{0_x}, Q_{1_x}, x, y)$.
9. $Q_0 \leftarrow Q_0 \uplus Q_2$.
10. If ((PV($Q_0$) = 1) and (PV($Q_2$) = 1) and (IC($P$) = 1) and ($Q_{0_x} \neq 0$) and ($Q_{0_x} \neq x$)) then
    10.1 Return($Q_0$);
11. Else return("Error detected").
```

6 Conclusion

In this report we have presented error-detecting schemes at the algorithm level for ECSM. We have used PV in conjunction with CC functions that are algorithm dependent. In the Montgomery ladder algorithm, we have considered the use of PV of the output and the concept of CC. In fact, we have shown that if we verify the integrity of the input point P (i.e., IC(P)), these two approaches are equivalent with respect to their error detection coverage. In this way, we can use PV of the output along with IC of P to have an improved error detection coverage with negligible cost. The double-and-add-always ECSM method presented by Coron [16] have been shown to be susceptible to the SE attack [39] [40]. In this report, we have presented the left-to-right and the right-to-left methods which provide resistance to the SE attack by utilizing CC among selected variables. Additionally, we have shown that for the right-to-left ECSM by double-and-add-always it is possible to resist the SCF attack presented by Blömer et al. [10]. This result is interesting since this algorithm do not use the combined curve [10], or the use randomization as RC and PC schemes. For applications utilizing affine coordinates it has a negligible cost in terms of additional finite field operations needed (i.e., less that 1% for $t \geq 192$). And even, for the case of projective coordinates, we have noted that the cost is of about 27.4% for $t = 192$. This value is less than the $30 - 40\%$ reported by Blömer et al. [10], or about 100% for schemes like RC or PC. Finally, we have considered the case where an attacker could mount a double-fault attack. Even with this strong fault model, it is possible to avoid fault attacks by utilizing some type of randomization. In this case we have utilized the concept of point blinding on the Montgomery ladder ECSM method.

References

[1] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM side-channel(s)," in *CHES 2002: Cryptographic Hardware and Embedded Systems*, ser. LNCS 2523. Springer-Verlag, 2002, pp. 29–45. 2

[2] R. Anderson and M. Kuhn, "Low cost attacks on tamper resistant devices," in *Security Protocols, 5th International Workshop Proceedings*, ser. LNCS 1361. Springer-Verlag, 1997, pp. 125–136. 2

[3] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems.* New York, NY, USA: John Wiley & Sons, Inc., 2001. 2

[4] ANSI X9.62, *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).* American National Standards Institute, 1999. 1

[5] A. Antipa, D. R. L. Brown, A. Menezes, R. Struik, and S. A. Vanstone, "Validation of elliptic curve public keys," in *PKC 2003: Public Key Cryptography*, ser. LNCS 2567. Springer-Verlag, 2003, pp. 211–223. 9, 20, 23

[6] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert, "Fault attacks on RSA with CRT: Concrete results and practical countermeasures," in *CHES 2002: Crypto-*

graphic Hardware and Embedded Systems, ser. LNCS 2523. Springer-Verlag, 2002, pp. 260–275. 10

[7] I. Biehl, B. Meyer, and V. Müller, "Differential fault attacks on elliptic curve cryptosystems," in *CRYPTO 2000: Advances in Cryptology*, ser. LNCS 1880. Springer-Verlag, 2000, pp. 131–146. 2, 9, 20, 23, 32

[8] J. Blömer and M. Otto, "Wagners attack on a secure CRT-RSA algorithm reconsidered," in *FDTC 2006: Workshop on Fault Diagnosis and Tolerance in Cryptography*, ser. LNCS 4236, 2006, pp. 13–23. 10

[9] J. Blömer, M. Otto, and J.-P. Seifert, "A new CRT-RSA algorithm secure against Bellcore attacks," in *ACM Conference on Computer and Communications Security.* ACM, 2003, pp. 311–320. 10

[10] ——, "Sign change attacks on elliptic curve cryptosystems," in *FDTC 2005: Fault Diagnosis and Tolerance in Cryptography*, ser. LNCS 4236. Springer-Verlag, 2006, pp. 36–42. 2, 9, 20, 23, 24, 25, 27, 28, 30, 34

[11] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of eliminating errors in cryptographic computations," *Journal of Cryptology*, vol. 14, no. 2, pp. 101–119, 2001. 2, 10

[12] A. Boscher, R. Naciri, and E. Prouff, "CRT RSA algorithm protected against fault attacks," in *WISTP 2007: Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems, International Workshop*, ser. LNCS 4462. Springer-Verlag, 2007, pp. 229–243. 10, 24, 25

[13] E. Brier and M. Joye, "Weierstraß elliptic curves and side-channel attacks," in *PKC 2002: Public Key Cryptography*, ser. LNCS 2274. Springer-Verlag, 2002, pp. 335–345. 4, 31

[14] M. Ciet and M. Joye, "Practical fault countermeasures for Chinese remaindering based RSA," in *FDTC 2005: Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2005, pp. 124–132. 10

[15] ——, "Elliptic curve cryptosystems in the presence of permanent and transient faults," *Designs, Codes and Cryptography*, vol. 36, no. 1, pp. 33–43, 2005. 9, 20, 23

[16] J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *CHES 1999: Cryptographic Hardware and Embedded Systems*, ser. LNCS 1717. Springer-Verlag, 1999, pp. 292–302. 2, 4, 24, 31, 34

[17] R. E. Crandall, "Method and apparatus for public key exchange in a cryptographic system," United States Patent 5,159,632, October 1992. 4

[18] A. Domínguez-Oviedo and M. A. Hasan, "Improved error-detection and fault-tolerance in ECSM using input randomization," CACR Technical Reports CACR 2006-41, University of Waterloo, Tech. Rep., 2006, a revised version to appear in IEEE Transactions on Dependable and Secure Computing. 9, 20, 24, 28, 30

[19] N. Ferguson and B. Schneier, *Practical Cryptography.* Wiley, 2003. 23

[20] FIPS 186 Digital Signature Standard (DSS), *Federal Information Processing Standards Publication 186.* National Institute for Standards and Technology, 1994. 1

[21] C. Giraud, "An RSA implementation resistant to fault attacks and to simple power analysis," *IEEE Transactions on Computers*, vol. 55, no. 9, pp. 1116–1120, 2006. 10, 20, 30

[22] D. Hankerson, A. Menezes, and S. A. Vanstone, *Guide to Elliptic Curve Cryptography.* Springer-Verlag, 2003. 30

[23] IEEE P1363, *IEEE Standard Specifications for Public Key Cryptography.* IEEE, 2000. 1, 7

[24] ISO/IEC 15946, *Information Technology - Security Techniques - Cryptographic techniques based on elliptic curves*, 2002, Parts 1-4. 1

[25] M. Joye and S.-M. Yen, "The Montgomery powering ladder," in *CHES 2002: Cryptographic Hardware and Embedded Systems*, ser. LNCS 2523. Springer-Verlag, 2002, pp. 291–302. 4, 10

[26] C. H. Kim and J.-J. Quisquater, "Fault attacks for CRT based RSA: New attacks, new results, and new countermeasures," in *WISTP 2007: Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems, International Workshop*, ser. LNCS 4462. Springer-Verlag, 2007, pp. 215–228. 2, 10, 11, 30

[27] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203–209, 1987. 1

[28] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman," in *CRYPTO 1996: Advances in Cryptology*, ser. LNCS 1109. Springer-Verlag, 1996, pp. 104–113. 2, 6, 23

[29] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *CRYPTO 1999: Advances in Cryptology*, ser. LNCS 1666. Springer-Verlag, 1999, pp. 388–397. 2, 6, 23

[30] S. Lin and D. J. Costello, *Error Control Coding.* Prentice-Hall, 2004. 12

[31] J. López and R. Dahab, "Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation," in *CHES 1999: Cryptographic Hardware and Embedded Systems*, ser. LNCS 1717. Springer-Verlag, 1999, pp. 316–327. 4, 6, 7, 12, 13, 22, 31

[32] A. Menezes, *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1994. 7

[33] V. S. Miller, "Use of elliptic curves in cryptography," in *CRYPTO 1985: Advances in Cryptology*, ser. LNCS 218. Springer-Verlag, 1986, pp. 417–426. 1

[34] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Mathematics of Computation*, vol. 48, pp. 243–264, 1987. 4

[35] K. Okeya and K. Sakurai, "Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery-form elliptic curve," in *CHES 2001: Cryptographic Hardware and Embedded Systems*, ser. LNCS 2162. Springer-Verlag, 2001, pp. 126–141. 4

[36] A. Shamir, "Method and apparatus for protecting public key schemes from timing and fault attacks," United States Patent 5,991,415, November 1999. 10

[37] D. Wagner, "Cryptanalysis of a probable secure CRT-RSA algorithm," in *ACM Conference on Computer and Communications Security.* ACM, 2004, pp. 82–91. 10

[38] R. B. Wells, *Applied Coding and Information Theory for Engineers.* Prentice-Hall, 1999. 12

[39] S.-M. Yen and M. Joye, "Checking before output may not be enough against fault-based cryptanalysis," *IEEE Transactions on Computers*, vol. 49, no. 9, pp. 967–970, 2000. 2, 4, 9, 24, 34

[40] S.-M. Yen, S. Kim, S. Lim, and S. Moon, "A countermeasure against one physical cryptanalysis may benefit another attack," in *ICISC 2001: International Conference Seoul on Information Security and Cryptology*, ser. LNCS 2288. Springer-Verlag, 2001, pp. 414–427. 2, 9, 24, 34

[41] ——, "RSA speedup with residue number system immune against hardware fault cryptanalysis," in *ICISC 2001: International Conference on Information Security and Cryptology*, ser. LNCS 2288. Springer-Verlag, 2001, pp. 397–413. 37

[42] ——, "RSA speedup with residue number system immune against hardware fault cryptanalysis," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 461–472, 2003, an earlier version appears in [41]. 10, 11, 30