On Fault-based Attacks and Countermeasures for Elliptic Curve Cryptosystems

by

Agustín Domínguez Oviedo

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

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. In this thesis, we consider fault-based attacks and countermeasures for ECC. We propose a new fault-based attack against the Montgomery ladder elliptic curve scalar multiplication (ECSM) algorithm. 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 two levels: module and algorithm. For protections at the module level, where the underlying scalar multiplication algorithm is not changed, a number of schemes and hardware structures are presented based on re-computation or parallel computation. It is shown that these structures can be used for detecting errors with a very high probability during the computation of ECSM. For protections at the algorithm level, 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.

Acknowledgements

First at all, my greatest debt of gratitude is to my supervisor, Professor M. Anwar Hasan, for his support, kindness, insightful guidance, and encouragement throughout the course of this research. I would also like to thank Professor M. Oussama Damen, Professor Guang Gong, and Professor Urs Hengartner for taking part in my PhD Committee. I would like to extend my gratitude to Professor Andrew Kennings and Professor Alfred Menezes for serving on the committee during the comprehensive exam. I am also very grateful to Professor Howard Heys from Memorial University for agreeing to be the external examiner, and for taking his valuable time to review this thesis. I would also thank Bijan Ansari for sharing and discussing with me initial ideas on how to mount a fault attack on the Montgomery ladder algorithm. I would like to thank my colleagues at the University of Waterloo, especially Abdulaziz Alkhoraidly and Nevine Ebeid for their cooperation, discussions and suggestions.

I would like to thank foremost my wife Liliana for her love and patience during this work. She has been supportive and understanding throughout the long hours. My sons Agustín and Mauricio also have been an inspiration for me. A special thanks must be given to my parents, Héctor and Imelda, who have pushed me to do my best and have always been extraordinary examples for me. Thanks to my brother Héctor and my sister Mariana for their moral support. I would like to thank Pepe, Hortencia, Odón, and Tania for giving love and support to my family. I would like to extend my gratitude to my wife's family, Guillermo, Tere, Doña Aurora, Jesús, and Mary for all their support and prayers. I would like to thank the friends I made in Waterloo, especially Julio, Lourdes, Félix, Lorena, Isaías, and

Malena.

I also want to thank CONACYT, and especially the ITESM Campus Querétaro México for sponsoring my studies at the University of Waterloo, allowing me to achieve this goal in my life and reinforcing my commitment to the development and progress of my country.

To Lila, my beloved wife

Contents

List of Tables

List of Figures

List of Acronyms

ANSI	American National Standards Institute
CC	Coherency Check
CRT	Chinese Remainder Theorem
DES	Data Encryption Standard
DFA	Differential Fault Analysis
DMR	Dual Modular Redundancy
DMR_PV	Dual Modular Redundant system with Point Verification
DPA	Differential Power Analysis
DSA	Digital Signature Algorithm
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
ECMQV	Elliptic Curve Menezes-Qu-Vanstone
ECSM	Elliptic Curve Scalar Multiplication
FA	Fault Analysis
FIPS	Federal Information Processing Standards
FPGA	Field-Programmable Gate Array
gcd	Greatest Common Divisor
IC	Integrity Check
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers

ISO	International Standards Organization
ISO-IEC	International Standards Organization - International Electrotechnical Commission
lcm	Least Common Multiple
NIST	National Institute of Standards and Technology
PC	Parallel Computation
PRC	Parallel and Re-Computation
PV	Point Verification
RC	Re-Computation
RC_partial	Partial Re-Computation
RSA	Rivest Shamir Adleman
SCF	Sign Change Fault
SE	Safe Error
SPA	Simple Power Analysis
TMR	Triple Modular Redundancy

Chapter 1

Introduction

Cryptography refers to design principles, means and methods for rendering plain information unintelligible to unauthorized parties. In the past, cryptography was used only for secret communications between powerful security entities such as military and intelligence agencies. Today, with the widespread use of computers and the Internet, secure communications are more than a privilege; they are a priority requirement even for the general public. In 1976, with the introduction of public key cryptography by Diffie and Hellman [25], secure communications were made practical. E-commerce and smart cards are examples of how cryptographic applications have become a part of everyday life.

Elliptic curve cryptography (ECC) was independently proposed by both Miller [67] and Koblitz [50] 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 [33], IEEE P1363 [42], ANSI X9.62 [5], and ISO 15946 [43]. Today ECC is an attractive choice because it achieves

the same level of security with a much smaller key size in comparison with other methods such as integer factorization or discrete logarithm based cryptosystems. Smaller key sizes generate smaller signatures, require less memory for storage, and use less bandwidth for communications. Because of this, for devices such as smart cards, personal digital assistants (PDAs), and mobile telephones, ECC appears to be an attractive choice for providing the public key services required for secrecy, authentication and non-repudiation purposes. In secrecy terms, ECC could provide the key distribution privacy needed for symmetric algorithms. For authentication and non-repudiation, ECC could be used to provide digital signatures.

Unfortunately, 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 analysis of timing [52], power consumption signals [53], and electromagnetic emanations [2] during cryptographic operations).

Another type of attacks that has received considerable attention is the fault analysis attack. Introduced by Boneh et al. [15], 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. Particularly, for an implementation of RSA based on the Chinese Remainder Theorem (CRT), they have demonstrated that with only two signatures of the same message (one computed correctly and one produced after some fault), it is possible to efficiently factor the modulus used. In order to avoid such attacks, they suggest verifying

the correctness of computations in cryptographic applications. In 1997, based on Boneh's work, Biham and Shamir generalized this technique for any secret key cryptographic algorithm proposed in the open literature [10].

Anderson and Kuhn reported a more practical fault attack just few months after Biham and Shamir's publication in 1997 [3]. 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 [4]. 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 [48] 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.

Skorobogatov and Anderson [84] introduced a new way to induce faults into a single bit using a laser beam. This is called *optical fault induction attack*. They used a low-cost laser in order to change the contents of any single RAM bit. Using this with the principles of differential fault analysis, it is possible to mount an inexpensive attack against many microcontrollers used today in constrained devices.

Biehl et al. [9] extended fault-based attacks to cryptosystems using elliptic curves [9]. 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. [15] 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. [9], 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 [91] and Yen et al. [92], and the sign change fault (SCF) attack presented by Blömer et al. [14]. 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., [23]). 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).

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 thesis we present our work on fault-based attacks and countermeasures for ECC.

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.

While error detection is a sufficient countermeasure for preventing fault-based attacks, fault-tolerant characteristics enable a system to perform its normal operation in the presence of some faults. This will result in more reliable systems where faults may occur due to natural causes such as, abnormal temperature, electromagnetic interference (EMI) or power supply changes. Error detection plays an

important role in the context of fault-tolerant system design. In detecting system failure it indicates the necessity for remedial actions.

1.1 Thesis organization

The organization of the remainder of this thesis is as follows: In Chapter 2, we present a brief overview of finite fields and elliptic curve cryptography (ECC). We also describe important fault attacks reported in the open literature for ECC. Additionally, in this chapter we present the error detection strategies that are used in this thesis. Chapters 3, 4, and 5 present the major research contributions of this thesis.

In Chapter 3, we present invalid-curve attacks that apply to the Montgomery ladder ECSM algorithm. An elliptic curve over the binary field is defined using two parameters, namely a and b. We show that with a different "value" for curve parameter a, there exists a cryptographically weaker group in nine of the ten NIST-recommended elliptic curves over $\mathbb{F}_{2^m}$. Thereafter, we present two attacks that are based on the observation that parameter a is not utilized for the Montgomery ladder algorithms proposed by López and Dahab [58]. We also present the probability of success of such attacks for general and NIST-recommended elliptic curves. At the end of the chapter, we give some countermeasures to resist this attack.

In Chapter 4, we present error detection and fault tolerance in ECSM by working with EC scalar multiplication modules and without making changes in the underlying scalar multiplication algorithm. To that end, first we describe a number of encoding techniques that rely on properties of elliptic curves. Thereafter, we give error-detecting and fault-tolerant structures for ECSM based on re-computation

and parallel computation. We show that it is possible to have fault-tolerant schemes utilizing two ECSM modules. This contrasts with the three modules needed by the well-known triple modular redundancy (TMR) based scheme. Then we give overhead costs and experimental results for the probability of undetected errors.

Algorithm-level error detection in ECSM is presented in Chapter 5. 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.

Chapter 6 provides concluding remarks and future research work. In the bibliography which begins on page 173, each reference ends with numbers that correspond to the page number in this thesis where that reference appears.

1.2 Summary of research contributions

The main contributions in this thesis are as follows:

- New invalid-curve attack on the Montgomery ladder ECSM algorithm over the binary field.
- Error detection in ECSM using repeated and parallel computation.
- Fault-tolerant ECSM using re-computation, parallel computation, and PV.

- Analysis of error detection of the Montgomery ladder ECSM algorithm over the binary field that reveals the advantage of performing an integrity check (IC) of the input point P.
- New double-and-add-always algorithms that resist the SE attack. The right-to-left version also resists the SCF attack.

Chapter 2

Background

In this chapter, we first present an overview of the mathematical background pertaining to elliptic curve cryptography (ECC). The fundamental operation used for ECC is the elliptic curve scalar multiplication (ECSM). We present this operation and some algorithms used for computing it. We also provide an overview of recent fault attacks on ECC. Error detection plays an important role as a countermeasure against such attacks. Moreover, it is a basic task in the context of fault-tolerant system design. Hence, we give some error detection strategies that are utilized in the remainder of this thesis.

For a better understanding of the materials to be presented in the following chapters, we give below some background related to finite fields and ECC. For more on these topics, the reader is referred to the following references [51], [62], [11], [40], [64], and [22].

2.1 Finite fields

Definition 2.1 An *abelian group* is a set G with a binary operation $*$ on G that satisfies the following five properties:

1. The operation $*$ is closed (i.e., $a * b \in G$ for all a, $b \in G$).
2. The operation $*$ is associative (i.e., $a * (b * c) = (a * b) * c$ for all $a, b, c \in G$).
3. The operation $*$ is commutative (i.e., $a * b = b * a$ for all a, $b \in G$).
4. There exists an identity element $e \in G$ called identity such that $a*e = e*a = a$ for any $a \in G$.
5. For every $a \in G$, there exists an inverse element $a^{-1} \in G$ such that $a*a^{-1} = e$.

Sometimes, we denote the group as a triple $(G, *, e)$. An example of an abelian group is the integers under the addition operation, with an identity element $e = 0$ and an inverse $a^{-1} = -a$. It could be referred to as $(\mathbb{Z}, +, 0)$. In cryptography the used groups typically have a finite number of elements. The number of elements in any group G is called the *order* of G, denoted as $|G|$ or $\texttt{ord}(G)$.

Definition 2.2 A finite group G is said to be *cyclic* if all elements of the group can be generated by applying the group operation repeatedly to an element $\alpha \in G$ which is denoted as a *generator* of the group G.

Definition 2.3 For a finite group $(G, *, e)$, *the order of an element* a (denoted $\texttt{ord}(a) = b$) is the smallest positive integer b such that $\underbrace{a * a * \cdots * a}_{b \text{ times}} = e$.

Theorem 2.1 Let G be a cyclic group of order n and $d|n$. Then G has $\phi(d)$ elements of order d and $\phi(n)$ generators, where the function ϕ is called the *Euler phi function*[1] [66].

Fact 2.1 Some properties of the Euler phi function include:

1. If p is prime, then $\phi(p) = p - 1$.
2. If m and n are relatively prime, then $\phi(mn) = \phi(m) \cdot \phi(n)$.
3. If $n = \prod_{i=0}^{j-1} p_i^{r_i}$ is the prime factorization of n, then $\phi(n) = n \cdot \prod_{i=0}^{j-1}(1 - \frac{1}{p_i})$.

By utilizing the group concept, we can define a field as follows:

Definition 2.4 A *field* F is a set of elements with two binary operators, denoted as $+$ and $\cdot$, which exhibits the following properties:

1. F is an abelian group under the operation $+$.
2. The non-zero elements of F form an abelian group under the operation $\cdot$.
3. The distributive laws apply (i.e., $a \cdot (b+c) = a \cdot b + a \cdot c$ and $(b+c) \cdot a = b \cdot a + c \cdot a$ for all a, b, $c \in F$).

A finite field is a field with a finite number of elements. It is referred to as Galois field, named after its inventor, Évariste Galois (1811-1832). The finite or Galois field with q elements is denoted by $\mathbb{F}_q$.

Example 2.1 (a) The set of real numbers under multiplication and addition is a field (infinite field). (b) Let p be a prime. The set $\{0, 1, \ldots, p-1\}$ forms a finite

[1] $\phi(n)$ corresponds to the number of positive integers $< n$ and relatively prime to n.

field of order p under modulo-p addition and multiplication. It is commonly referred to as *prime field* $\mathbb{F}_p$.

Theorem 2.2 For p prime and $m \geq 2$, there is a unique finite field of order p^m, denoted as $\mathbb{F}_{p^m}$. It is called the *extension field of the prime field*. A proof of this theorem is presented by Golomb and Gong [38].

Definition 2.5 Let $m \geq 2$. The field $\mathbb{F}_{2^m}$ is called *characteristic-two finite field* or *binary finite field*. It can be seen as a vector space of dimension m over the field $\mathbb{F}_2$ which has only the elements 0 and 1. There are m elements $(\alpha_{m-1}, \alpha_{m-2}, \ldots, \alpha_1, \alpha_0)$ in $\mathbb{F}_{2^m}$ such that each element $a \in \mathbb{F}_{2^m}$ can be represented in the following form:

$$a = a_{m-1}\alpha_{m-1} + a_{m-2}\alpha_{m-2} + \cdots + a_1\alpha_1 + a_0\alpha_0, \text{ where } a_i \in \mathbb{F}_2.$$

The set $\{\alpha_{m-1}, \alpha_{m-2}, \ldots, \alpha_1, \alpha_0\}$ is called a basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. Every element in the field can be represented as a bit string of the form $(a_{m-1}a_{m-2}\ldots a_1a_0)$. The field addition is simply the bit-wise XOR operation, and the field multiplication depends on the field basis chosen. The selection of a particular basis may depend on the used platform (e.g., hardware or software). This choice frequently influences implementation cost and the complexity of finite field computations. The two most common types of bases used in conventional software and hardware applications are polynomial and normal bases. Others, like dual, redundant and triangular bases, are less commonly used but have some advantages in specific implementations. The work discussed in this thesis uses polynomial basis.

In *polynomial basis* the elements of $\mathbb{F}_{2^m}$ are represented as linear combinations of the set $\{\alpha^{m-1}, \alpha^{m-2}, \ldots, \alpha^2, \alpha, 1\}$, where α is a root of an irreducible polynomial $f(z)$ of degree m over $\mathbb{F}_2$. Polynomial basis is in many cases referred to as *canonical*

Binary finite field	**Reduction polynomial**
$\mathbb{F}_{2^{163}}$	$f(z) = z^{163} + z^7 + z^6 + z^3 + 1$
$\mathbb{F}_{2^{233}}$	$f(z) = z^{233} + z^{74} + 1$
$\mathbb{F}_{2^{283}}$	$f(z) = z^{283} + z^{12} + z^7 + z^5 + 1$
$\mathbb{F}_{2^{409}}$	$f(z) = z^{409} + z^{87} + 1$
$\mathbb{F}_{2^{571}}$	$f(z) = z^{571} + z^{10} + z^5 + z^2 + 1$

Table 2.1: NIST-recommended binary finite fields and their reduction polynomials

or *standard* basis. For U.S. Federal Government usages, the National Institute of Standards and Technology (NIST) has recommended five binary fields along with their corresponding reduction polynomials (see Table 2.1 [32]). It is important to note two aspects of these polynomials. First, these polynomials are trinomial or pentanomial. Second, the degree of the second leading term is a small number in comparison with the extension degree of the binary field (i.e., m). Both aspects are important in terms of efficiency of finite field operations.

Binary finite fields are very attractive to implementers due to their "carry-free" arithmetic, and the availability of different representations of the field (i.e., basis and/or polynomial selection) which can be suited to and optimized for the computational environment [11]. Some cryptographic applications, such as elliptic curve cryptosystems, permit the use of either a prime field $\mathbb{F}_p$ or a binary finite field $\mathbb{F}_{2^m}$. For hardware implementation, in order to reduce the complexity of the design, a binary finite field may be selected [1] [78].

Solving quadratic equations over $\mathbb{F}_{2^m}$

Solving second-degree equations over $\mathbb{F}_{2^m}$ has a number of applications in ECC. For example, to obtain the value(s) of $y \in \mathbb{F}_{2^m}$ that might satisfy an elliptic curve

equation from a given $x \in \mathbb{F}_{2^m}$. Also, it plays a crucial step in the context of ECSM utilizing point halving [49]. In this thesis, some results from this subsection are used in Chapters 3 and 5. For more details on this topic the reader is referred to [8] and [18].

First, let us define two functions that are important for solving quadratic equations over the binary field, the trace and half-trace functions:

Definition 2.6 *The trace function* of an element $\beta \in \mathbb{F}_{2^m}$, denoted by $\mathtt{Tr}(\beta)$, can be defined as

$$\mathtt{Tr}(\beta) = \sum_{i=0}^{m-1} \beta^{2^i}.$$

Definition 2.7 Let m be an odd integer. *The half-trace function* of an element $\beta \in \mathbb{F}_{2^m}$, denoted by $\mathtt{Ht}(\beta)$, can be defined as

$$\mathtt{Ht}(\beta) = \sum_{i=0}^{(m-1)/2} \beta^{2^{2i}}.$$

The trace and the half-trace functions are linear, i.e., $\mathtt{Tr}(\alpha+\beta) = \mathtt{Tr}(\alpha)+\mathtt{Tr}(\beta)$ and $\mathtt{Ht}(\alpha + \beta) = \mathtt{Ht}(\alpha) + \mathtt{Ht}(\beta)$, for all α, $\beta \in \mathbb{F}_{2^m}$. Also it can be shown that $\mathtt{Tr}(\beta) = \mathtt{Tr}(\beta^2) = \mathtt{Tr}^2(\beta)$, $\mathtt{Ht}(\beta^2) = \mathtt{Ht}^2(\beta)$, and $\mathtt{Ht}(\beta^2) + \mathtt{Ht}(\beta) = \mathtt{Tr}(\beta) + \beta$, for all $\beta \in \mathbb{F}_{2^m}$.

Let us solve an equation over $\mathbb{F}_{2^m}$ of the form $M^2 + uM + v = 0$. First consider the trivial case where $u = 0$ in which case the solution is $M = \sqrt{v} = v^{2^{m-1}}$. Then for $u \neq 0$, we can perform a change of variables $M \leftarrow Mu$ that yields the following simplified equation

$$M^2 + M + w = 0, \text{where } w = v/u^2. \tag{2.1}$$

Theorem 2.3 Let m be an odd integer. Equation (2.1) has a solution over $\mathbb{F}_{2^m}$ if and only if $\mathtt{Tr}(w) = 0$. In such a case one solution corresponds to $z = \mathtt{Ht}(w)$ and the other to $\mathtt{Ht}(w) + 1$.

Proof Let z be a solution of Equation (2.1). Obtaining the trace function of this equation, we have

$$\mathtt{Tr}(z^2 + z + w) = \mathtt{Tr}(z^2) + \mathtt{Tr}(z) + \mathtt{Tr}(w) = 0.$$

This will be valid if and only if $\mathtt{Tr}(w) = 0$. To show that $z = \mathtt{Ht}(w)$ is a solution we can obtain that

$$z^2 + z = \mathtt{Ht}^2(w) + \mathtt{Ht}(w) = w + \mathtt{Tr}(w),$$

in which case z is a solution as claimed. It can be easily verified that $\mathtt{Ht}(w)+1$ also satisfies Equation (2.1). □

2.2 Elliptic curve cryptography (ECC)

Definition 2.8 Let $\mathbb{F}_q$ be a finite field. An *elliptic curve* E over $\mathbb{F}_q$ is formed by the points (x, y) that satisfy the following equation:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \quad a_i \in \mathbb{F}_q. \tag{2.2}$$

Equation (2.2) is referred to as the affine form of the Weierstrass equation. The resulting set of points, plus an additive identity defined as the point at infinity ($\mathcal{O}$), together with a particular operation, denoted as point addition ($\uplus$), form an abelian group.

Definition 2.9 Two elliptic curves are *isomorphic* over $\mathbb{F}_q$ if they have defining equations which are the same under some *admissible changes of variables*.

2.2.1 Non-supersingular elliptic curves of characteristic two

From Equation (2.2) we can perform the following change of variables:

$$x = a_1^2 x_1 + \frac{a_3}{a_1} \qquad y = a_1^3 y_1 + \frac{a_1^2 a_4 + a_3^2}{a_1^3},$$

with $a_1 \neq 0$ the resulting relationship is

$$y_1^2 + x_1 y_1 = x^3 + a_2' x_1^2 + a_6',$$

where a_2' and a_6' are appropriate functions of a_1, a_2, a_3, a_4, and a_6. Now we can write a simplified affine Weierstrass equation for binary finite fields for non-supersingular elliptic curves as follows,

$$y^2 + xy = x^3 + ax^2 + b. \tag{2.3}$$

Theorem 2.4 Let E and $\overline{E}$ be non-supersingular elliptic curves defined over $\mathbb{F}_{2^m}$. E and $\overline{E}$ given by the equations

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

are *isomorphic* over $\mathbb{F}_{2^m}$ if and only if $\mathtt{Tr}(a) = \mathtt{Tr}(\bar{a})$ and $b = \bar{b}$. If the last conditions are met, then there is an admissible change of variables $(x, y) \rightarrow (x, y + tx)$ that converts E into $\overline{E}$ for some $t \in \mathbb{F}_{2^m}^*$ that satisfies $\bar{a} = t^2 + t + a$.

2.2.2 Elliptic curves of characteristic $p > 3$

From Equation (2.2) we can complete the square in the left side,

$$\left(y + \frac{a_1 x}{2} + \frac{a_3}{2}\right)^2 = x^3 + \left(a_2 + \frac{a_1^2}{4}\right) x^2 + \left(a_4 + \frac{a_1 a_3}{2}\right) x + \left(\frac{a_3^2}{4} + a_6\right),$$

which can be rewritten as

$$y_1^2 = x^3 + a_2' x^2 + a_4' x + a_6',$$

where $y_1 = y + \frac{a_1 x}{2} + \frac{a_3}{2}$; and a_2', a_4', and a_6' are appropriate functions of a_1, a_2, a_3, a_4, and a_6. Now we can perform a change of variable in the right side as $x_1 = x + \frac{a_2'}{3}$ obtaining,

$$y_1^2 = x_1^3 + a x_1 + b,$$

for some constants a and b. Now we can write a simplified affine Weierstrass equation for prime finite fields ($p > 3$) as follows,

$$y^2 = x^3 + ax + b, \tag{2.4}$$

where $a, b \in \mathbb{F}_p$, and $4a^3 + 27b^2 \neq 0 \pmod{p}$.

2.2.3 Group law

Let P and Q be any two distinct points on E, which is defined by either Equation (2.3) or (2.4). The rules for point addition are given as follows:

1. The point $\mathcal{O}$ is used as the *identity element*. For any point P, $P \uplus \mathcal{O} = P$ and $\mathcal{O} \uplus P = P$.

2. The *negative* of the point $P = (x, y)$, denoted as $-P$, such that $P \uplus (-P) = \mathcal{O}$ is,

$$-P = \begin{cases} (x, x + y) & \text{if } E \text{ is defined over } \mathbb{F}_{2^m}, \\ (x, -y) & \text{if } E \text{ is defined over } \mathbb{F}_p. \end{cases}$$

3. Let $P = (x_1, y_1) \neq \mathcal{O}$ and $Q = (x_2, y_2) \neq \mathcal{O}$, where $P \neq \pm Q$. The *point addition*, $R = P \uplus Q = (x_3, y_3)$, is defined as follows,

$$x_3 = \begin{cases} \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)^2 + \dfrac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a & \text{if } E \text{ is defined over } \mathbb{F}_{2^m}, \\ \left(\dfrac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 & \text{if } E \text{ is defined over } \mathbb{F}_p. \end{cases}$$

$$y_3 = \begin{cases} \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + x_3 + y_1 & \text{if } E \text{ is defined over } \mathbb{F}_{2^m}, \\ \left(\dfrac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1 & \text{if } E \text{ is defined over } \mathbb{F}_p. \end{cases}$$

4. Let $P = (x_1, y_1) \neq \mathcal{O}$ and $P \neq -P$. The *point doubling*, $R = P \uplus P = 2P = (x_3, y_3)$, is defined as follows,

$$x_3 = \begin{cases} \left(x_1 + \dfrac{y_1}{x_1}\right)^2 + \left(x_1 + \dfrac{y_1}{x_1}\right) + a & \text{if } E \text{ is defined over } \mathbb{F}_{2^m}, \\ \left(\dfrac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 & \text{if } E \text{ is defined over } \mathbb{F}_p. \end{cases}$$

$$y_3 = \begin{cases} (x_1 + x_3)\left(x_1 + \dfrac{y_1}{x_1}\right) + x_3 + y_1 & \text{if } E \text{ is defined over } \mathbb{F}_{2^m}, \\ \left(\dfrac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1 & \text{if } E \text{ is defined over } \mathbb{F}_p. \end{cases}$$

2.2.4 Group order and structure

Definition 2.10 The *order of an elliptic curve* E defined over $\mathbb{F}_q$, denoted as $\#E(\mathbb{F}_q)$, is defined as the number of points in $E(\mathbb{F}_q)$.

Definition 2.11 The *order of a point* $P \in E(\mathbb{F}_q)$, denoted as $\texttt{ord}(P)$, is the smallest positive integer e such that $eP = \mathcal{O}$.

Using a result of Lagrange's theorem from group theory [89], we can affirm that the order of any point always divides the order of the group. As a result, if $\#E(\mathbb{F}_q)$ is prime, then the order of any point is $\#E(\mathbb{F}_q)$ and it can be used as a generator. For cryptographic applications, the order of a selected point P should be divisible by a sufficiently large prime [42].

In order to learn more about the structure of the group $E(\mathbb{F}_q)$, it is important to know the value of $\#E(\mathbb{F}_q)$. The next well-known theorem gives us a bound for this parameter.

Theorem 2.5 (*Hasse's Theorem*). Let E be an elliptic curve defined over $\mathbb{F}_q$. Then

$$\#E(\mathbb{F}_q) = q + 1 - t, \quad \text{where } |t| \leq 2\sqrt{q}.$$

The exact value of $\#E(\mathbb{F}_q)$ can be efficiently obtained using some point counting algorithms such as the Schoof-Elkies-Atkin algorithm [11] for curves over prime fields, or the Satoh-Skjernaa-Taguchi algorithm [77] for curves over the binary field.

Let $\mathbb{Z}_n$ be a cyclic group of order n. The following theorem is about the group structure of $E(\mathbb{F}_q)$.

Theorem 2.6 (*Rück Theorem* [75]). Let E be an elliptic curve over $\mathbb{F}_q$. Then $E(\mathbb{F}_q)$ is isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$ where $n_2 | n_1$ and $n_2 | (q-1)$.

E over	Coordinate system	Points	Correspondence
$\mathbb{F}_{2^m}$ (non-super-singular)	Affine ($\mathcal{A}$)	(x, y)	
	Standard projective ($\mathcal{P}$)	(X, Y, Z)	$(X/Z, Y/Z)$
	López and Dahab ($\mathcal{LD}$)	(X, Y, Z)	$(X/Z, Y/Z^2)$
	Jacobian ($\mathcal{J}$)	(X, Y, Z)	$(X/Z^2, Y/Z^3)$
$\mathbb{F}_p$ ($p > 3$)	Affine ($\mathcal{A}$)	(x, y)	
	Standard projective ($\mathcal{P}$)	(X, Y, Z)	$(X/Z, Y/Z)$
	Jacobian ($\mathcal{J}$)	(X, Y, Z)	$(X/Z^2, Y/Z^3)$
	Chudnovsky Jacobian ($\mathcal{J}^c$)	(X, Y, Z, Z^2, Z^3)	$(X/Z^2, Y/Z^3)$

Table 2.2: Examples of coordinate systems

Based on Theorem 2.6, $E(\mathbb{F}_q)$ can be either a cyclic group or the direct sum of two cyclic groups. With $\#E(\mathbb{F}_q) = n_1 n_2$, if $n_2 = 1$, then $E(\mathbb{F}_q)$ is a cyclic group. If $n_2 > 1$, then $E(\mathbb{F}_q)$ is said to have rank 2 [40].

2.2.5 Coordinate systems

The elliptic curves obtained from Equations (2.3) and (2.4) with their respective point addition rules are for affine coordinates (i.e., (x, y)). In order to reduce computational cost, a number of projective coordinate systems (i.e., (X, Y, Z)) have been suggested in the literature. In Table 2.2 some of these systems with their correspondence with the affine system are presented.

For each coordinate system, the total number of field operations is different resulting in different time cost for elliptic curve point addition and doubling. Previously in Subsection 2.2.3, the EC group law using affine coordinates has been given. Using relations between some other systems with the affine, it is possible to obtain the point addition and doubling equations for those systems.

Standard projective ($\mathcal{P}$) [64], Jacobian ($\mathcal{J}$) [42], Chudnovsky Jacobian ($\mathcal{J}^c$)

[19], and López and Dahab's ($\mathcal{LD}$) [57] coordinate systems avoid the necessity to compute one multiplicative inverse in each point operation. This is achieved at the expense of more multiplications and storage space. The decision regarding whether to use affine coordinate system or other is based primarily on implementation aspects such as the availability of memory for storing temporary values and the relative performance of the field inversion and multiplication algorithms used to implement the EC group operations. Table 2.3 shows the number of finite field operations needed to perform point addition and doubling for each system [40] [22]. The symbols M, S, and I denote, respectively, the cost of finite field multiplication, squaring, and inversion. These symbols are used for the remainder of this thesis.

2.3 Elliptic curve scalar multiplication (ECSM)

Definition 2.12 Let k be a positive integer and P be a point in $E(\mathbb{F}_q)$, then the *elliptic curve scalar multiplication (ECSM)* is given as follows,

$$kP = \underbrace{P \uplus P \uplus \cdots \uplus P}_{k \quad \text{times}}.$$

To carry out ECSM, it is necessary to perform point doublings and additions. As described above in the EC group law, such operations require some finite field arithmetic (i.e., multiplication, addition, squaring, and multiplicative inverse). The dependency of ECSM on EC point doubling is explained below.

Let $(k_{t-1} \cdots k_1 \; k_0)_2$ be the t-bit binary representation of k. The scalar multiplication kP can be computed as follows,

$$Q = kP = \left(\sum_{i=0}^{t-1} k_i 2^i \right) P,$$

E over	Point addition		Point doubling	
$\mathbb{F}_{2^m}$ (non-super-singular)	$\mathcal{A}+\mathcal{A}\rightarrow\mathcal{A}$	$2M+1S+1I$	$2\mathcal{A}\rightarrow\mathcal{A}$	$2M+1S+1I$
	$\mathcal{P}+\mathcal{P}\rightarrow\mathcal{P}$	$13M+1S$	$2\mathcal{P}\rightarrow\mathcal{P}$	$7M+5S$
	$\mathcal{LD}+\mathcal{LD}\rightarrow\mathcal{LD}$	$14M+4S$	$2\mathcal{LD}\rightarrow\mathcal{LD}$	$4M+5S$
	$\mathcal{J}+\mathcal{J}\rightarrow\mathcal{J}$	$14M+4S$	$2\mathcal{J}\rightarrow\mathcal{J}$	$5M+5S$
	$\mathcal{P}+\mathcal{A}\rightarrow\mathcal{P}$	$12M+2S$		
	$\mathcal{LD}+\mathcal{A}\rightarrow\mathcal{LD}$	$8M+5S$		
	$\mathcal{J}+\mathcal{A}\rightarrow\mathcal{J}$	$11M+3S$		
$\mathbb{F}_p$ ($p>3$)	$\mathcal{A}+\mathcal{A}\rightarrow\mathcal{A}$	$2M+1S+1I$	$2\mathcal{A}\rightarrow\mathcal{A}$	$2M+2S+1I$
	$\mathcal{P}+\mathcal{P}\rightarrow\mathcal{P}$	$12M+2S$	$2\mathcal{P}\rightarrow\mathcal{P}$	$7M+3S$
	$\mathcal{J}+\mathcal{J}\rightarrow\mathcal{J}$	$12M+4S$	$2\mathcal{J}\rightarrow\mathcal{J}$	$4M+4S$
	$\mathcal{J}^c+\mathcal{J}^c\rightarrow\mathcal{J}^c$	$11M+3S$	$2\mathcal{J}^c\rightarrow\mathcal{J}^c$	$5M+4S$
	$\mathcal{J}+\mathcal{A}\rightarrow\mathcal{J}$	$8M+3S$		
	$\mathcal{J}+\mathcal{J}^c\rightarrow\mathcal{J}$	$11M+3S$		
	$\mathcal{J}^c+\mathcal{A}\rightarrow\mathcal{J}^c$	$7M+3S$		

Table 2.3: Field operations count for point addition and doubling using various coordinate systems

$$Q=\left(k_{t-1}2^{t-1}P\right)\uplus\left(k_{t-2}2^{t-2}P\right)\uplus\cdots\uplus\left(k_1 2P\right)\uplus\left(k_0 P\right). \tag{2.5}$$

By factoring out 2 we obtain,

$$Q=2\left(\left(k_{t-1}2^{t-2}P\right)\uplus\left(k_{t-2}2^{t-3}P\right)\uplus\cdots\uplus\left(k_1 P\right)\right)\uplus\left(k_0 P\right).$$

Now we can repeat this operation until we have

$$Q=2(2(\cdots 2\left(2\left(k_{t-1}P\right)\uplus k_{t-2}P\right)\uplus\cdots)\uplus k_1 P)\uplus k_0 P. \tag{2.6}$$

The operations in Equations (2.5) and (2.6) can be performed utilizing the well-known *double-and-add* method. To that end, Algorithm 2.1 implements the operations of Equation (2.6). 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. On the other hand, the right-to-left counterpart of Algorithm 2.1 is illustrated as Algorithm 2.2. The operations in this algorithm are performed corresponding to Equation (2.5). These algorithms use a method that is commonly referred to as *binary method* [22]. The expected number of point operations performed in Algorithms 2.1 and 2.2 is t point doublings and $t/2$ point additions on average.

Algorithm 2.1. 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).

Algorithm 2.2. Right-to-left 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 $i = 0$ to $t - 1$ do
   2.1 If $(k_i = 1)$ then
      2.1.1 $Q \leftarrow Q \uplus P$.
   2.2 $P \leftarrow 2P$.
3. Return(Q).

For ECC applications that use a projective coordinate system for point representation, Algorithm 2.1 is usually preferred over Algorithm 2.2. The main reason is that the point addition required in Algorithm 2.1 uses a fixed operand (i.e., P). This aspect permits the use of mixed coordinates for point addition, i.e., one point to be added is given in some projective coordinate system, and point P in the affine

system. As illustrated in Table 2.3, the use of mixed coordinates saves some finite field multiplications.

Coron [23] has shown that algorithms with a non-homogeneous operation flow, such as Algorithms 2.1 and 2.2, 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 2.3 and 2.4 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 [91] have observed that algorithms with dummy operations might be susceptible to a special fault attack

Algorithm 2.3. 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 2.4. 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).

called *safe-error* (SE) attack. This is an example that in some cases a countermeasure against one attack may benefit another attack.

2.3.1 Montgomery's ECSM method

Montgomery [68] 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 [58] [69] [17]. Utilizing a variant of the binary method known as *binary ladder* [24], 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 [47][2]. 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

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

We can write expressions for L_i and M_i as follows,

$$L_i = 2L_{i+1} + k_i = L_{i+1} + M_{i+1} + k_i - 1,$$
$$M_i = 2M_{i+1} + k_i - 1 = L_{i+1} + M_{i+1} + k_i.$$

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

$$(L_i, M_i) = \begin{cases} (2L_{i+1}, L_{i+1} + M_{i+1}) & \text{if } k_i = 0, \\ (L_{i+1} + M_{i+1}, 2M_{i+1}) & \text{if } k_i = 1. \end{cases} \tag{2.7}$$

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

Now, let $Q_{0,i} = L_iP$ and $Q_{1,i} = M_iP$ be points for $i \in \{0, t-1\}$. Utilizing Equation (2.7), we can obtain the pair $(Q_{0,i}, Q_{1,i})$ as follows

$$(Q_{0,i}, Q_{1,i}) = \begin{cases} (2Q_{0,i+1}, Q_{0,i+1} \uplus Q_{1,i+1}) & \text{if } k_i = 0, \\ (Q_{0,i+1} \uplus Q_{1,i+1}, 2Q_{1,i+1}) & \text{if } k_i = 1. \end{cases} \tag{2.8}$$

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 (2.8) 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 2.5. 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 2.5. 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 [52] and SPA [53]. 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. We discuss this in Chapter 5.

An extension of the ECSM Montgomery idea for non-supersingular elliptic curves over the binary finite field was presented by López and Dahab [58]. They showed algorithms for both affine and projective coordinate systems. Let us present some resulting expressions from lemmas given by López and Dahab [58]. 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 (2.3). The x-coordinate of $P_0 \uplus P_1$, x_2, can be obtained as follows:

$$x_2 = \frac{x_0 y_1 + x_1 y_0 + x_0 x_1^2 + x_0^2 x_1}{(x_0 + x_1)^2}. \tag{2.9}$$

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:

$$\mathtt{x}(P_0 \uplus P_1) = \begin{cases} x_0^2 + \dfrac{b}{x_0^2} & \text{if } P_0 = P_1, \\ x + \dfrac{x_0}{x_0 + x_1} + \left(\dfrac{x_0}{x_0 + x_1}\right)^2 & \text{if } P_0 \neq P_1. \end{cases} \tag{2.10}$$

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:

$$y_0 = \frac{(x_0 + x)\left[(x_0 + x)(x_1 + x) + x^2 + y\right]}{x} + y. \tag{2.11}$$

Based on Algorithm 2.5 and Equations (2.10) and (2.11), Algorithm 2.6 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 2.6 $\mathtt{x}(\cdot)$ corresponds to the x-coordinate of the point given in the argument.

Algorithm 2.6. 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 2.6 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 [58] presented the projective version of Algorithm 2.6. 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 [64] and addition-subtraction [42] 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:

$$\mathtt{Z}(P_0 \uplus P_1) = Z_2 = \begin{cases} X_0^2 Z_0^2 & \text{if } P_0 = P_1, \\ (X_0 Z_1 + X_1 Z_0)^2 & \text{if } P_0 \neq P_1. \end{cases} \tag{2.12}$$

$$\mathtt{X}(P_0 \uplus P_1) = X_2 = \begin{cases} X_0^4 + bZ_0^4 & \text{if } P_0 = P_1, \\ xZ_2 + X_0 X_1 Z_0 Z_1 & \text{if } P_0 \neq P_1. \end{cases} \tag{2.13}$$

Using these group formulas, it is possible to have an efficient algorithm to compute the ECSM. Algorithm 2.7 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 2.5, not handling the Y-coordinates helps to have fewer memory

requirements and a faster method since some finite field multiplications are saved.

Algorithm 2.7. 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).

2.3.2 Elliptic curve discrete logarithm problem (ECDLP)

The ECSM is the base operation used for ECC. In fact, the elliptic curve discrete logarithm problem (ECDLP) is based on the difficulty of obtaining k given P and

$Q(= kP)$ for some integer k and P, $Q \in E(\mathbb{F}_q)$. This principle has led to schemes equivalent to DLP-based cryptosystems, such as: Diffie-Hellman key exchange [25], ElGamal public key encryption [30], ElGamal digital signatures [30], and DSA [33].

In practice for the ECDLP to be intractable, it is important to select appropriate domain parameters such as the finite field $\mathbb{F}_q$ where the curve E is defined, the curve E itself, and the base point P. When the order n of the base point P is a large prime, the fastest known algorithms to solve the ECDLP, namely the baby-step giant-step [82] and the Pollard's rho [73] algorithms, need $O(\sqrt{n})$ steps. Consequently, for security purposes it is necessary that the size of the underlying finite field be at least the double of the security level in bits. Security level of L bits is referred to as the best algorithm for breaking the system that takes approximately 2^L steps [40]. For example, for achieving an 80-bit security level, the cryptosystem would require an elliptic curve defined over a finite field $\mathbb{F}_q$, where $q \approx 2^{160}$. With respect to the selection of the elliptic curve E, some types of curves should be avoided for cryptographic applications since the ECDLP can be reduced. These curves include supersingular curves [65], anomalous curves [76] [80], and curves over $\mathbb{F}_{2^m}$ for some non-prime values of m [34] [36] [61].

If the order of the base point P does not contain at least a large prime factor, then it is possible to use an extension for ECC of the Silver-Pohlig-Hellman algorithm [72] to solve the ECDLP as presented in Algorithm 2.8. This algorithm reduces the problem to subgroups of prime order. Let n be the order of the base point P with a prime factorization $n = \prod_{i=0}^{j-1} p_i^{e_i}$, where $p_i < p_{i+1}$. Suppose that $Q = lP$, where $P,\ Q \in E(\mathbb{F}_q)$ and $l \in [0, n-1]$. This algorithm obtains during the outer loop, the value of $l \bmod {p_i}^{e_i}$ for each $0 \leq i \leq j-1$. With these values $l \bmod n$ can be uniquely computed using the CRT. It is important to note that at Step

1.3.2 one EC discrete logarithm needs to be computed. However, this operation is in a subgroup at the most of order p_{j-1}. It can be performed with the fastest known algorithms for ECDLP such as the Pollard's rho algorithm with an expected running time of $O(\sqrt{p_m})$, where p_m is the largest prime divisor of $\texttt{ord}(P_i)$.

Algorithm 2.8. Silver-Pohlig-Hellman's algorithm for solving the ECDLP

Input: $P \in E(\mathbb{F}_q)$, $Q \in \langle P \rangle$, $n = \texttt{ord}(P) = \prod_{i=0}^{j-1} p_i^{e_i}$, where $p_i < p_{i+1}$.
Output: $l \bmod n$.

---

1. For $i = 0$ to $j - 1$ do

   1.1 $Q' \leftarrow \mathcal{O}$, $l_i \leftarrow 0$.

   1.2 $P_i \leftarrow (n/p_i)P$.

   1.3 For $t = 0$ to $(e_i - 1)$ do

      1.3.1 $Q_{t,i} \leftarrow (n/p_i^{t+1})(Q \uplus Q')$.

      1.3.2 $W_{t,i} \leftarrow \log_{P_i} Q_{t,i}$. {ECDLP in a subgroup of order $\texttt{ord}(P_i)$.}

      1.3.3 $Q' \leftarrow Q' - W_{t,i} p^t P$.

      1.3.4 $l_i \leftarrow l_i + p^t W_{t,i}$.

2. Use the CRT to solve the system of congruences $l \equiv l_i \pmod{p_i^{e_i}}$. This gives us $l \bmod n$.

3. Return(l).

Example 2.2 Let E be the curve $y^2 + xy = x^3 + 1$ over the field $\mathbb{F}_{2^{11}}$ given by the polynomial $f(z) = z^{11} + z^2 + 1$. Let us represent the elements of $\mathbb{F}_{2^{11}}$ in hexadecimal form. Consider the point $P = (\texttt{0x10F},\texttt{0x27A})$ whose order is $n = 92 = 2^2 \cdot 23$. Let $Q = (\texttt{0x1FB},\texttt{0x2C6})$. We can use Algorithm 2.8 to obtain $l = \log_P Q$ as follows:

- During the first loop for $i = 0$ we can obtain $l_0 = l \bmod 2^2$. We can find that $l_0 = W_{0,0} + 2W_{2,0} = 1 + 2 \cdot 0 = 1$.
- For the second loop for $i = 1$ we determine $l_1 = l \bmod 23$. It can be shown that $l_1 = W_{1,0} = 18$.
- Finally we have the following pair of congruences: $l \bmod 4 = 1$ and $l \bmod 23 = 18$. Solving using the CRT we have $l = 41$.

To resist the Silver-Pohlig-Hellman attack one can simply select an elliptic curve E such that its group order, $\#E(\mathbb{F}_{2^m})$, is prime or *almost prime*, i.e., $\#E(\mathbb{F}_{2^m}) = hn$, where n is a prime and h is small [40] (e.g., $h \in [1, 4]$). However, in Chapter 3 we will show how Algorithm 2.8 could be useful under an invalid-curve attack to the ECSM algorithm based on Montgomery's ladder.

2.4 ECC fault-based attacks

In 1996 fault analysis attack was introduced by Boneh et al. [15]. This attack is also known as Bellcore and is based on fault injection in a device performing an RSA or Rabin digital signature. Biehl et al. [9] proposed the first fault-based attack on ECC. Their basic idea is to force, through a fault, a computation in a weaker group where solving the ECDLP is feasible. They consider that the attacker can either select or modify the input point $P \in E(\mathbb{F}_q)$ of the ECSM (i.e., $P \leftarrow \widetilde{P}$, $\widetilde{P} \in \widetilde{E}(\mathbb{F}_q)$). A basic assumption for this attack is that the parameter a_6 from Equation (2.2) is not involved for point operations formulas. In this way, the computation could be performed in a cryptographically less secure elliptic curve $\widetilde{E}(a_1, a_2, a_3, a_{4,}, \widetilde{a}_6)$,

which differs from the original elliptic curve $E(a_1, a_2, a_3, a_4, a_6)$ only in the curve parameter a_6. The relation between $\widetilde{a}_6$ and $\widetilde{P} = (\widetilde{x}, \widetilde{y})$ can be obtained from Equation (2.2) as follows:

$$\widetilde{a}_6 = \widetilde{y}^2 + a_1\widetilde{x}\widetilde{y} + a_3\widetilde{y} - \widetilde{x}^3 - a_2\widetilde{x}^2 - a_4\widetilde{x}. \tag{2.14}$$

If the attacker can choose $\widetilde{P}$, then $\widetilde{a}_6$ could be selected in a such way that the order of $\widetilde{E}(\mathbb{F}_q)$, $\#\widetilde{E}(\mathbb{F}_q)$, has small enough prime factors for solving the ECDLP in such a group. For a given $\widetilde{a}_6$, the point $\widetilde{P}$ can be chosen by selecting values of $\widetilde{x}$ and $\widetilde{y}$ that satisfy Equation (2.14). Let r be the order of $\widetilde{P}$, i.e., $r = \texttt{ord}(\widetilde{P})$. Since the computation is performed in $\widetilde{E}(\mathbb{F}_q)$ instead of $E(\mathbb{F}_q)$, each ECSM computation with $\widetilde{P}$ can supply the value $k \bmod r$ from $k\widetilde{P}$. Repeating this procedure with sufficiently different chosen points $\widetilde{P}_i$ could give us a unique solution for k using the CRT. The same idea can be applied to the short versions of the Weierstrass equation (i.e., Equations (2.3) and (2.4)). If the curve operations do not consider parameter b, then this attack can be mounted by selecting a point on a weaker curve $\widetilde{E}(a, \widetilde{b})$, which differs from the original elliptic curve $E(a, b)$ only in parameter b.

Ciet and Joye [21] have shown how to recover the secret key k by having an unknown but fixed faulty input point $\widetilde{P}$. This is a more realistic scheme because, excluding EC ElGamal encryption, in most EC cryptosystems the point P is predetermined and might be fixed. The attacker cannot select this point which is usually stored in the memory of a device. Their fault model considers that only one coordinate x or y is faulty. If it is assumed that the x-coordinate of a point $P = (x, y)$ is altered, the resultant point will be $\widetilde{P} = (\widetilde{x}, y)$. The scalar multiplication algorithm computes $\widetilde{Q} = k\widetilde{P}$. The faulty result $\widetilde{Q} = k(\widetilde{x}, y) = (\widetilde{x}_k, \widetilde{y}_k)$, is over $\widetilde{E}(\mathbb{F}_q)$ on which

the operation was performed, defined as $\widetilde{E}(a_1, a_2, a_3, a_4, \widetilde{a}_6)$, where $\widetilde{a}_6$ is obtained as follows

$$\widetilde{a}_6 = \widetilde{y}_k^2 + a_1\widetilde{x}_k\widetilde{y}_k + a_3\widetilde{y}_k - \widetilde{x}_k^3 - a_2\widetilde{x}_k^2 - a_4\widetilde{x}_k.$$

Using Equation (2.2) for $\widetilde{E}$ we obtain

$$\widetilde{E} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + \widetilde{a}_6. \tag{2.15}$$

Since $\widetilde{P} \in \widetilde{E}(\mathbb{F}_q)$, it needs to follow Equation (2.15). Substituting its coordinates $(\widetilde{x}, y)$ and grouping the terms with $\widetilde{x}$ we have the following expression

$$\widetilde{x}^3 + a_2\widetilde{x}^2 + (a_4 - a_1y)\widetilde{x} + (\widetilde{a}_6 - y^2 - a_3y) = 0.$$

By solving the above equation for $\widetilde{x}$, we can have up to three different roots, where one is the fault generated x-coordinate. If the point P is known (usually this parameter is public), $\widetilde{x}$ can be distinguished easily since it will differ from x just for the faulty bits. Having $\widetilde{P}$ and $\widetilde{Q}$, the procedure described by Biehl et al. [9] for obtaining k can be followed. Additionally, the authors consider other attacks where the underlying finite field or the elliptic curve parameters are disturbed by a fault. They assume that it is possible to inject a transient fault into these parameters. For example, in a smart card system parameters are usually stored in non-volatile memory (e.g., EEPROM) and they are transferred into RAM for executing a specific algorithm. It is possible to induce an error in such data before or during this transfer.

Antipa et al. [6] state the importance of checking whether the received point for any EC key agreement and public-key encryption protocols are on the proper elliptic

curve. Specifically, they show the vulnerability under this scenario of the one-pass EC Diffie-Hellman protocol, the EC integrated encryption scheme (ECIES), the one-pass ECMQV protocol and the EC digital signature algorithm. Some of these protocols or algorithms are defined in standards such as ANSI X9.62, ANSI X9.63, IEEE 1363-2000, ISO/IEC 15946-3, FIPS 186-2, and ISO 15946-2. They mention the significance for implementers of taking into account point verification, even when some standards do not mandate that. A possible countermeasure considered by Antipa et al. [6] is to use other formulas for the addition law that use both elliptic curve parameters, a and b. In fact, some standards such as IEEE 1363 [42] include group formulas for curves over $\mathbb{F}_{2^m}$ that involve both parameters (i.e., point doubling formula uses b while point addition uses a). In this scenario, the invalid-curve attacks described by Biehl et al. [9] and Ciet and Joye [21] might not constitute a threat.

It is important to note that the invalid-curve attacks presented by Biehl et al. [9] and Ciet and Joye [21] apply to applications where parameter b is not used for the group formulas. However for Algorithms 2.6 and 2.7, it is not the case since parameter b is utilized. In Chapter 3 we present an attack that takes advantage of parameter a not being used in these algorithms.

Biehl et al. [9], in addition to the invalid-curve attack, proposed an ECC differential fault analysis (DFA) attack. Their work was an extension of the attack presented by Boneh et al. [15] for RSA cryptosystems. They assume that the adversary knows the details of the implementation, such as the parameters, the utilized algorithm, and the representation of internal variables. They consider that the attacker can precisely flip a bit in a register that holds a partial result during the ECSM operation. They also assume that the attacker can determine that the

fault was induced v iterations before the ECSM computation finished. Consider a left-to-right algorithm where the scalar bits are processed from $i = t - 1$ to 0 (e.g., Algorithm 2.1). If v is small enough for doing an exhaustive search, then the attacker can obtain the v least significant bits as follows:

1. Compute the normal ECSM, i.e., $Q = kP$.
2. Repeat the operation inducing a single bit-flip fault into a variable that holds a partial result (e.g., Step 2.1 of Algorithm 2.1) at $i = v - 1$ to obtain $\widetilde{Q}$.
3. For all v-bit integers d compute $Q - dP$ and $\widetilde{Q} - dP$.
4. Look for pairs of results that differ only in one bit. Biehl et al. [9] show that the probability that more than one pair satisfies this condition is low. If only one pair exists, then they are the original and the faulty intermediate results, respectively. The associated value of d gives the v least significant bits of k.

The full value of k can be recovered by repeating Steps 1-4 for different (and ascending) values of v and utilizing the known bits of the scalar.

One countermeasure against the attacks presented by Biehl et al. [9], Ciet and Joye [21], and Antipa et al. [6] is to verify whether or not the output point lies on the original elliptic curve. This process is often called point verification (PV). This basic countermeasure is not sufficient for all cases. Blömer et al. [14] presented an ECC fault-based attack called sign change fault (SCF) attack in which PV is not a sufficient protection. This attack is based on changing the sign of an intermediate point during the ECSM. It is important to note that this attack only applies to cryptosystems that use elliptic curves over prime fields where the change of a point involves only a change of the sign of the y-coordinate.

An interesting aspect of the SCF attack is that the elliptic curve operations do not leave the original group $E(\mathbb{F}_p)$ in contrast with those attacks described by Biehl et al. [9], Ciet and Joye [21], and Antipa et al. [6]. Blömer et al. assume that the attacker can induce a fault that produces a sign change into an intermediate point Q', such that $Q' \leftarrow -Q'$ during the ECSM operation. They show the applicability of this attack for applications utilizing conventional crypto co-processors. The SCF attack can be mounted in some ECSM algorithms such as the non-adjacent form (NAF) based ECSM by double-and-add (e.g., Algorithm 1 of [14]). In this case the attacker needs to induce a SCF in the point Q' during the execution of specific steps (e.g., Steps 3 and 4 of Algorithm 1 of [14]). The error can be induced for some unknown values of i during the loop. They show that having a number of $c = (t/m)\log(2t)$ attacks on the same input pair (k, P) and the correct result Q, it is possible to recover the scalar k with a probability of at least $1/2$, where t is the length of NAF(k) and m is a parameter related with the amount of offline work. Otto [70] has extended the SCF attack to other algorithms such that the ECSM by double-and-add (Algorithm 2.1) and the basic Montgomery ladder ECSM (Algorithm 2.5).

A straightforward countermeasure against an SCF attack is to use a version of Montgomery's ladder algorithm that does not use the y-coordinate for computing ECSM (e.g., [17]). Another countermeasure presented by Blömer et al. uses a second elliptic curve whose order is a small prime number. This curve is utilized to define a named "combined curve" that is used to verify the final result to avoid this SCF attack. One disadvantage of this approach is that elliptic curve crypto-processors might have fixed parameters (e.g., those recommended by NIST). Hence, it might not be possible to redefine the curve parameters for having such

a countermeasure. In Chapter 4 we present some structures that permit detection of errors in ECSM without modifying the curve parameters. These are based on re-computation and parallel computation. In Chapter 5, we present an algorithm-level countermeasure against this attack that utilizes coherency check among the involved variables.

Another fault-based attack that has received considerable attention is the safe error (SE) attack proposed separately by Yen and Joye [91] and Yen et al. [92]. Even when the idea behind both attacks is in essence the same (i.e., attack during dummy operations), the most important difference between them is their fault models. Yen and Joye [91] assume that the attacker can modify the value of some operand(s) stored in a register (i.e., memory-safe error). On the other hand, Yen et al. [92] consider that the adversary can damage the computed result of some operation (i.e., computational-safe error). In both cases algorithms with dummy instructions can be a potential target (e.g., Algorithms 2.3 and 2.4).

These attacks were originally presented in the context of RSA signature generation. However, they can generally be extended to ECSM. For example, let us consider the right-to-left ECSM by double-and-add-always algorithm (Algorithm 2.4). Assume that the attacker, during the main loop at $i = l$, can modify the value of Q_1 just after the computation of Step 2.1. If the scalar bit k_l is 0, there is a dummy instruction at Step 2.3 (i.e., $Q_0 \leftarrow Q_0$) that will not be affected for the faulty value of Q_1. In the next round, Q_1 will be overwritten when $Q_1 = Q_0 \uplus P$ is computed. Accordingly, the final result will be correct even when a SE fault was injected. In contrast, if $k_l = 1$, the value of Q_0 will be infected by Q_1 at Step 2.3 (i.e., $Q_0 \leftarrow Q_1$). This is likely to produce an incorrect result. This scenario allows an oracle attack, i.e., $k_l = 1$ if the final result is faulty and $k_l = 0$ otherwise. The

last example was for a memory-safe error fault attack. However, it can be easily extended to a computational-safe attack if the attacker changes the result of the operation performed in Step 2.1 ($Q_1 = Q_0 \uplus P$) and follows in similar way the procedure described above. Note that SE attack depends only on the availability of injecting a fault during loop iterations where a dummy instruction is computed.

One countermeasure against SE attack is to utilize randomization. In fact, randomization of the data processed is essential in resisting other attacks such as SPA and differential power analysis (DPA) attacks [28]. Another way to prevent the SE attack is to use an algorithm that does not have dummy operations (e.g., the Montgomery ladder algorithm). Another countermeasure is to verify, even in dummy operations, that an error has occurred. This approach will be utilized in Chapter 5 for detecting errors during the ECSM using coherency check among the involved variables.

2.5 Error detection strategies

In this section, 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 present general re-computation and parallel computation schemes that can also be used for detecting errors in ECSM. Additionally, 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.

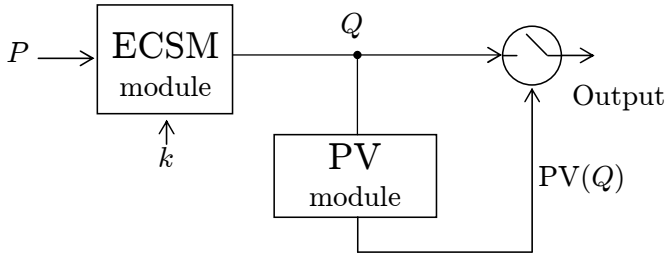

Figure 2.1: Point verification (PV) module after the ECSM module

2.5.1 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. [9], Ciet and Joye [21], Antipa et al. [6], Blömer et al. [14], and Domínguez and Hasan [27]. A PV module takes a point Q as its input and checks whether the point is in $E(\mathbb{F}_q)$ (see Figure 2.1). This checking is done simply by verifying whether the coordinates of Q satisfy the governing elliptic curve equation, e.g., Equation (2.3) 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. For example, for curves defined over the binary field using affine coordinates it takes two finite field multiplications, two squarings, and three additions[3]. If implemented in hardware as a separate module, assuming a

[3]Assume $a = 1$. If $a = 0$, one less addition is required. If $a \notin \{0, 1\}$, one more multiplication is needed.

pipelined implementation i.e., while the PV module is verifying a current output, the ECSM module executes the next operation, the finite field multiplier in PV module could be designed to run at a much slower speed since PV is needed just once for each scalar multiplication. On the other hand, the multiplier in the ECSM module is often optimized for speed since its operation is repeated many times. For instance, the ECSM using Algorithm 2.1 with affine coordinates over $\mathbb{F}_{2^{163}}$ will need approximately[4] 489 finite field multiplications, 244 squarings, 1630 additions, and 244 inverses, on average.

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 2.1. Consider that the ECSM algorithm uses dummy instructions for having a uniform execution (e.g., Algorithm 2.4). If an SE attack is mounted as described in previous section the attacker can perform a similar oracle attack, i.e., $k_l = 1$ if the final result is not given and $k_l = 0$ if the correct result is released by the PV module. Consequently, PV does not provide the sufficient protection.

[4]Utilizing the addition formulas given by Avanzi et al. [22].

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

2.5.2 Error detection using re-computation and parallel computation

For applications where the time required for error detection is not critical, it is possible to compute the result twice (i.e., once at time t_0 and then again at time t_1) and then compare the two results to detect a possible error. This technique and its variants are used in various computing applications (e.g., integer multiplication); however, to the best of our knowledge there has been no precedence of our extension of these techniques to ECSM. Under this technique, a transient error occurring in any one of the two ECSM computations (i.e., at time t_0 or t_1), but not in both will be detected. However, if the computation module is permanently in a faulty state, the errors produced will remain undetected. Because of this, the re-computation at time t_1 is better done with a different method.

A general re-computation based scheme that uses only one computing module is shown in Figure 2.2. The *compute* module is multiplexed between the upper and the lower data paths at time t_0 and t_1, respectively. The result of the upper data path is stored in a register. The encoder in the bottom data path of the figure produces a different set of inputs to the compute module and the decoder transforms the output of the module back to the original domain for the final comparison. The encoding and the decoding processes enable us to detect errors caused by a permanent fault in the compute module. Encoding and decoding can be carried out through a number of approaches. For conventional number systems, the most popular ones include:

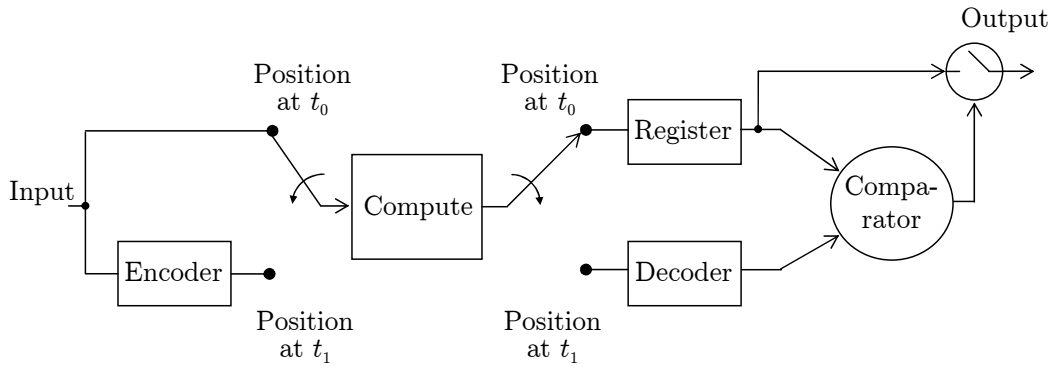

Figure 2.2: General re-computation based scheme

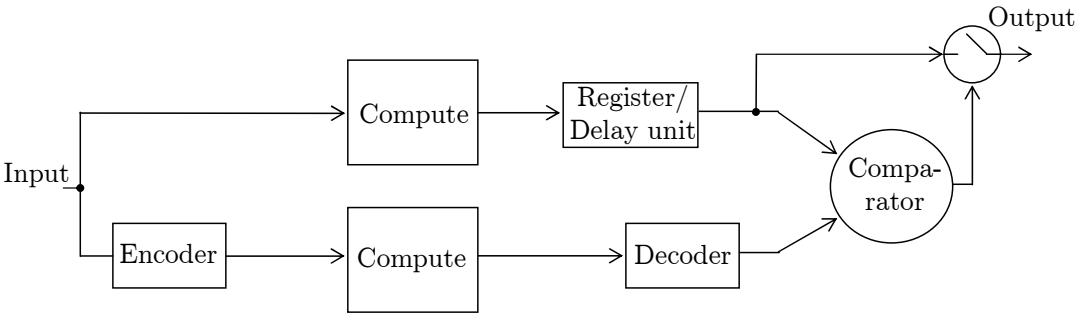

Figure 2.3: General parallel computation based scheme

alternating logic [74], re-compute with shifted operands [71] and re-compute with swapped operands [45]. In Chapter 4, we provide encoding and decoding processes that are suitable for ECSM.

Similar to re-computation, where we have one module that performs the computation twice, we can have two independent modules working in parallel. A general parallel computation based scheme is given in Figure 2.3, where the register or the delay unit synchronize the inputs to the comparator.

2.5.3 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 fact, this technique is widely used in areas such as telecommunications, computers, and media storage. For example, some computers use address and op-code coherence checking to avoid accessing an invalid memory register or executing an invalid instruction.

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. [15], 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., [81], [94], [13], [20], [37], and [16][5]). The countermeasures proposed by Giraud [37] and Boscher et al. [16] use CC for detecting errors during the modular exponentiation operation. Let us describe both approaches.

Giraud [37] has shown how the SPA-and-SE resistant algorithm published by Joye and Yen [47] 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

[5]Many of them have been shown vulnerable when they are analyzed under a different fault model (e.g., [7], [88], [12], and [48]).

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 2.9). 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 $a_0 \cdot m \equiv a_1 \ (\texttt{mod}\ N)$. Whether or not the computed signature S is returned depends on this test.

Recently, Boscher et al. [16] proposed a new FA-resistant modular exponentiation algorithm (Algorithm 2.10). 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 2.10. In Chapter 5 we extend some of the concepts utilized by Giraud [37] and Boscher et al. [16] for detecting errors in ECSM. For example, for the Montgomery ladder ECSM we investigate the difference in terms of error detection between PV and CC.

Algorithm 2.9. 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 2.10. 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").

2.6 Conclusion

In this chapter, we have presented background related to ECC. This includes the fundamental operation for elliptic curve cryptosystems, i.e., the scalar multiplication (ECSM). We have given some algorithms that can be utilized for computing

this operation. Additionally, we have presented an overview of the recent ECC fault-based attacks. Finally, we have introduced the error detection strategies that are utilized for the remainder of this thesis.

Chapter 3

Fault-Based Attack on Montgomery's Ladder Algorithm

In Section 2.4, we have presented important fault-based attacks reported in the open literature for ECC. The invalid-curve attacks reported by Biehl et al. [9], and Ciet and Joye [21] apply only to applications where parameter b is not utilized for the group formulas. However, this is not the case for the Montgomery ladder ECSM algorithm proposed by López and Dahab [58] (Algorithms 2.6 and 2.7) where parameter b is considered specifically for point doubling formulas.

In this chapter, we present fault-based attacks that apply to Montgomery's ladder ECSM algorithm. First, we present some preliminary work with examples for the NIST-recommended curves over the binary field. Next, we present two invalid-curve based attacks on the target algorithm. Finally, we present some possible countermeasures to the attacks presented in this chapter.

3.1 Preliminaries

By Theorem 2.4 we can state that the number of isomorphism classes for elliptic curves defined by Equation (2.3) is $2^{m+1}-2$. The latter comes from the number of possible values for parameter b (i.e., 2^m-1) times the possible values of the trace function of parameter a (i.e., 2). With the last observation, for a fixed value of parameter b there are only two isomorphic classes of curves, one for each value of $\gamma \in \{0,1\}$, where $\mathtt{Tr}(a) = \gamma$. Let us define two representative elliptic curves, E_0 and E_1, one for each of these isomorphic classes:

$$E_0 : y^2 + xy = x^3 + b \qquad (a = 0), \tag{3.1}$$

$$E_1 : y^2 + xy = x^3 + x^2 + b \quad (a = 1). \tag{3.2}$$

Lemma 3.1 Let E_0 and E_1 be two elliptic curves over $\mathbb{F}_{2^m}$ defined by Equations (3.1) and (3.2), respectively.

(i) The only points that $E_0(\mathbb{F}_{2^m})$ and $E_1(\mathbb{F}_{2^m})$ share are $\mathcal{O}$ and $(0, \sqrt{b})$.

(ii) Let $(u, v) \in E_j(\mathbb{F}_{2^m})$, where $u \in \mathbb{F}_{2^m}^*$, $v \in \mathbb{F}_{2^m}$, and $j \in \{0,1\}$. Then, there does not exist any point in $E_{\overline{j}}(\mathbb{F}_{2^m})$ of the form (u, w) for any $w \in \mathbb{F}_{2^m}$, where $\overline{j} = 1 - j$.

(iii) There exist two points of the form (u, v) and $(u, u + v)$ in either $E_0(\mathbb{F}_{2^m})$ or $E_1(\mathbb{F}_{2^m})$ for each $u \in \mathbb{F}_{2^m}^*$ and some $v \in \mathbb{F}_{2^m}$.

(iv) The orders of $E_0(\mathbb{F}_{2^m})$ and $E_1(\mathbb{F}_{2^m})$ satisfy the following

$$\#E_0(\mathbb{F}_{2^m}) + \#E_1(\mathbb{F}_{2^m}) = 2^{m+1} + 2. \tag{3.3}$$

Proof First, if we solve the quadratic expressions resulting from Equations (3.1) and (3.2) with $x = 0$, we obtain a unique solution $y = \sqrt{b}$. For $x \neq 0$, by Theorem 2.3 Equation (2.3) has a solution for y if and only if

$$\mathtt{Tr}(x) + \mathtt{Tr}(a) + \mathtt{Tr}\left(\frac{b}{x^2}\right) = 0. \tag{3.4}$$

Since the only difference between Equations (3.1) and (3.2) is the value of parameter a, we can conclude from Equation (3.4) that if any value of $x \in \mathbb{F}_{2^m}^*$ does not have a solution with $a = j$, then it does with $a = \bar{j}$ for $j = 0$ or 1. Also this equation shows that it is not possible to have a solution for both E_0 and E_1 with the same $x \neq 0$.

Additionally, by Theorem 2.3 we know that for a given value of $x \neq 0$ we have two distinct solutions that represent two elliptic curve points (i.e., a point and its negative). To this end, for $x \neq 0$, $\#E_0(\mathbb{F}_{2^m}) + \#E_1(\mathbb{F}_{2^m})$ consider exactly $2^{m+1} - 2$ points on both curves. In addition, the points $\mathcal{O}$ and $(0, \sqrt{b})$ are common and are counted twice in the sum of both orders, bringing the total up to $2^{m+1}+2$ as shown in Equation (3.3). □

Example 3.1 Let us consider $\mathbb{F}_{2^5}$ as represented by the irreducible polynomial $f(z) = z^5 + z^2 + 1$. Let us represent the elements of $\mathbb{F}_{2^5}$ in hexadecimal form. Let E_0 and E_1 be the curves $y^2 + xy = x^3 + 1$ and $y^2 + xy = x^3 + x^2 + 1$, respectively, defined over $\mathbb{F}_{2^5}$. $E_0(\mathbb{F}_{2^5})$ has an order of 44 with the following set of points:

```
{(0x00,0x01),(0x01,0x00),(0x01,0x01),(0x02,0x1F),(0x02,0x1D),(0x03,0x0C),
 (0x03,0x0F),(0x04,0x12),(0x04,0x16),(0x05,0x1A),(0x05,0x1F),(0x07,0x1F),
 (0x07,0x18),(0x09,0x1D),(0x09,0x14),(0x0B,0x16),(0x0B,0x1D),(0x0C,0x05),
 (0x0C,0x09),(0x0D,0x0B),(0x0D,0x06),(0x0F,0x19),(0x0F,0x16),(0x10,0x09),
 (0x10,0x19),(0x11,0x03),(0x11,0x12),(0x12,0x14),(0x12,0x06),(0x15,0x12),
 (0x15,0x07),(0x17,0x0B),(0x17,0x1C),(0x18,0x0F),(0x18,0x17),(0x1A,0x11),
```

(0x1A,0x0B),(0x1B,0x0F),(0x1B,0x14),(0x1C,0x09),(0x1C,0x15),(0x1F,0x06),
(0x1F,0x19),$\mathcal{O}$}.

On the other hand, $E_1(\mathbb{F}_{2^5})$ has an order of 22 with the following set of points:

{(0x00,0x01),(0x06,0x10),(0x06,0x16),(0x08,0x17),(0x08,0x1F),(0x0A,0x18),
(0x0A,0x12),(0x0E,0x07),(0x0E,0x09),(0x13,0x1C),(0x13,0x0F),(0x14,0x0D),
(0x14,0x19),(0x16,0x02),(0x16,0x14),(0x19,0x04),(0x19,0x1D),(0x1D,0x1B),
(0x1D,0x06),(0x1E,0x15),(0x1E,0x0B),$\mathcal{O}$}

Example 3.2 *Examples for NIST-recommended curves*: Let $E(a, b)$ be a NIST-recommended elliptic curve defined over the binary field $\mathbb{F}_{2^m}$ with curve parameters a and b. In Table 3.1 each NIST-recommended randomly chosen elliptic curve over $\mathbb{F}_{2^m}$ is presented, where $m = 163$, 233, 283, 409 and 571. Then, for each of these curves its corresponding curve $\widehat{E}(\widehat{a}, b)$ is shown, where $\widehat{a} = 1 - \mathtt{Tr}(a)$. Similarly, Table 3.2 gives the NIST-recommended Koblitz curves. For each curve the "values" of m, $f(z)$, a, b, and $\#E(\mathbb{F}_{2^m})$ are listed, where $f(z)$ is the irreducible trinomial or pentanomial used as the reduction polynomial. For the random curves, parameter b is shown in hexadecimal form. For each case the group order $\#E(\mathbb{F}_{2^m})$ is given in decimal, followed by its prime factorization.

We notice that for each listed NIST-recommended curve E, the group $\widehat{E}(\mathbb{F}_{2^m})$ is cryptographically weaker, i.e., all the prime factors of $\#\widehat{E}(\mathbb{F}_{2^m})$ are smaller than the larger prime factor of $\#E(\mathbb{F}_{2^m})$, with only one exception for the case of $m = 283$ for Koblitz curves, where the orders of both $E(\mathbb{F}_{2^m})$ and $\widehat{E}(\mathbb{F}_{2^m})$ are *almost* prime. In Table 3.3, the size of each prime factor of the group orders of these elliptic curves is presented. Additionally, it can be shown by Theorem 2.6 that $E(\mathbb{F}_{2^m})$ and $\widehat{E}(\mathbb{F}_{2^m})$, where $m \in \{163, 233, 283, 409, 571\}$, are cyclic groups for all the curves in Tables 3.1 and 3.2.

Example for $m = 163$: $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$,
$b =$ `0x 00000002 0A601907 B8C953CA 1481EB10 512F7874 4A3205FD`

Standard Curve B-163. $a = 1$
$\#E(\mathbb{F}_{2^{163}}) =$ `116920130986472233456294848857527813785136864031 74`
`= (2)(58460065493236116728147424428763906892568432015 87)`

Weaker Curve. $\widehat{a} = 0$
$\#\widehat{E}(\mathbb{F}_{2^{163}}) =$ `116920130986472233456294724377077469359812342844 44`
`= (2)^2 (31)(907)(18908293)(192478327)(285644694766939633075451013 53)`

Example for $m = 233$: $f(z) = z^{233} + z^{74} + 1$,
$b =$ `0x 00000066 647EDE6C 332C7F8C 0923BB58 213B333B 20E9CE42 81FE115F 7D8F90AD`

Standard Curve B-233. $a = 1$
$\#E(\mathbb{F}_{2^{233}}) =$ `13803492693581127574869511724554051111679625474690027110758767268970926`
`= (2)(69017463467905637874347558622770255558398127373450135553793836 34485463)`

Weaker Curve. $\widehat{a} = 0$
$\#\widehat{E}(\mathbb{F}_{2^{233}}) =$ `13803492693581127574869511724554050698124810413991519109891329626226260`
`= (2)^2 (5)(283)(541)(584818873)(783195327693846094609)(98420105436969060152144124234193 03)`

Example for $m = 283$: $f(z) = z^{283} + z^{12} + z^7 + z^5 + 1$,
$b =$ `0x 027B680A C8B8596D A5A4AF8A 19A0303F CA97FD76 45309FA2 A581485A F6263E31 3B79A2F5`

Standard Curve B-283. $a = 1$
$\#E(\mathbb{F}_{2^{283}}) =$ `15541351137805832567355695254588151253139251848753809778218393053540088555574757385742`
`= (2)(77706755689029162836778476272940756265696259243769048891091965267700442777787378692871)`

Weaker Curve. $\widehat{a} = 0$
$\#\widehat{E}(\mathbb{F}_{2^{283}}) =$ `15541351137805832567355695254588151253139257576080422561810605502282380007708578585076`
`= (2)^2 (7)(19)^2(5942982169)(48758898298463720443)(45527407299960753170946983)(11654464127519441963117752 7)`

Table 3.1: Examples for NIST-recommended randomly chosen curves over $\mathbb{F}_{2^m}$

Example for $m = 409$: $f(z) = z^{409} + z^{87} + 1$,
$b =$ 0x 0021A5C2 C8EE9FEB 5C4B9A75 3B7B476B 7FD6422E F1F3DD67 4761FA99 D6AC27C8 A9A197B2 72822F6C D57A55AA 4F50AE31 7B13545F

Standard Curve B-409. $a = 1$
$\#E(\mathbb{F}_{2^{409}}) =$ 132211193758049719790383061606554207965680936592856243856929759660831554965474961041628744752435822193195973457673313505354 2
$=$ (2)(661055968790248598951915308032771039828404682964281219284648798304157774827374805208143723762179110965979867288366567526771)

Weaker Curve. $\widehat{a} = 0$
$\#\widehat{E}(\mathbb{F}_{2^{409}}) =$ 132211193758049719790383061606554207965680936592856243856929758448930761529049577288446939423450291745840511352336029816348 4
$=$ (2)^2(13)(43)(599)(1867)(4201)(10711)(378828133699627599347)(3101704082899971294666512289235259940780107395876742769754357060357968277689511477292 9)

Example for $m = 571$: $f(z) = z^{571} + z^{10} + z^5 + z^2 + 1$,
$b =$ 0x 02F40E7E 2221F295 DE297117 B7F3D62F 5C6A97FF CB8CEFF1 CD6BA8CE 4A9A18AD 84FFABBD 8EFA5933 2BE7AD67 56A66E29 4AFD185A 78FF12AA 520E4DE7 39BACA0C 7FFEFF7F 2955727A

Standard Curve B-571. $a = 1$
$\#E(\mathbb{F}_{2^{571}}) =$ 772907504603451668939070378186397468859785465941286999731447050290303828457912084907228799877883154616626776224385388897249374492563362614046905657660666482278638221057140 6
$=$ (2)(386453752301725834469535189093198734429892732970643499865723525145151914228956042453614399938941577308313388112192694448624687246281681307023452828830333241139319110528570 3)

Weaker Curve. $\widehat{a} = 0$
$\#\widehat{E}(\mathbb{F}_{2^{571}}) =$ 772907504603451668939070378186397468859785465941286999731447050290303828457912084907248706754885876568358670188215481973696656971853832448250257811726165817200154104872229 2
$=$ (2)^2(7)(1153)(99262049966063)(641043691173743374578683)(36502311411080739536696 03)(562516514411236993734142229508523209240999366989)(1832372106849886832903758716153488484939785889992701131641)

Table 3.1: (Contd.) Examples for NIST-recommended randomly chosen curves over $\mathbb{F}_{2^m}$

Example for $m = 163$: $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$, $b = 1$

Standard Curve K-163. $a = 1$

$\#E(\mathbb{F}_{2^{163}}) =$ 11692013098647223345629483507196896696658237148126
$=$ (2)(5846006549323611672814741753598448348329118574063)

Weaker Curve. $\widehat{a} = 0$

$\#\widehat{E}(\mathbb{F}_{2^{163}}) =$ 11692013098647223345629473816263631617836683539492
$=$ $(2)^2$(653)(6521)(34101072914026637)(20129541232727197849723433)

Example for $m = 233$: $f(z) = z^{233} + z^{74} + 1$, $b = 1$

Standard Curve K-233. $a = 0$

$\#E(\mathbb{F}_{2^{233}}) =$ 13803492693581127574869511724554051042283763955449008505312348098965372
$=$ $(2)^2$(345087317339528189371737793113851276057094098886225212632808702474134 3)

Weaker Curve. $\widehat{a} = 1$

$\#\widehat{E}(\mathbb{F}_{2^{233}}) =$ 13803492693581127574869511724554050767520671933232537715337748796231814
$=$ (2)(92269)(114861079)(130034039)(5062109767067236109)(9893311373906301287655774909 07)

Example for $m = 283$: $f(z) = z^{283} + z^{12} + z^7 + z^5 + 1$, $b = 1$

Standard Curve K-283. $a = 0$

$\#E(\mathbb{F}_{2^{283}}) =$ 15541351137805832567355695254588151253139246935172245297183499990119263318817690415492
$=$ $(2)^2$(3885337784451458141838923813647037813284811733793061324295874997529815829704422603873)

Other Curve. $\widehat{a} = 1$

$\#\widehat{E}(\mathbb{F}_{2^{283}}) =$ 15541351137805832567355695254588151253139262489661987042845498565703205244465645555326
$=$ (2)(7770675568902916283677847627294075626569631244830993521422749282851602622232822777663)

Table 3.2: Examples for NIST-recommended Koblitz curves over $\mathbb{F}_{2^m}$

Example for $m = 409$: $f(z) = z^{409} + z^{87} + 1$, $b = 1$

Standard Curve K-409. $a = 0$

$\#E(\mathbb{F}_{2^{409}}) =$ 132211193758049719790383061606554207965680936592856243856929758009152284515699676420269303383110983205638546636247092543468 4

$=$ (2)^2(330527984395124299475957654016385519914202341482140609642324395022880711289249191050673258457777458014096366590617731358671)

Weaker Curve. $\widehat{a} = 1$

$\#\widehat{E}(\mathbb{F}_{2^{409}}) =$ 132211193758049719790383061606554207965680936592856243856929760100610031978824861909806380792775130733397938173762250778234 2

$=$ (2)(5616389)(90250595219)(538258252508065812423826381099759315)(2422926717379184361670943884439581457811909443935034501042288725219735 1)

Example for $m = 571$: $f(z) = z^{571} + z^{10} + z^5 + z^2 + 1$, $b = 1$

Standard Curve K-571. $a = 0$

$\#E(\mathbb{F}_{2^{571}}) =$ 772907504603451668939070378186397468859785465941286999731447050290303828457912084907253591409082684733882685120330140584509469989626646924771872968646837001422293474110669 2

$=$ (2)^2(193226876150862917234767594546599367214946366485321749932861762572575957114478021226813397852270671183470671280082535146127367497406661731192968242161709250355573368527667 3)

Weaker Curve. $\widehat{a} = 1$

$\#\widehat{E}(\mathbb{F}_{2^{571}}) =$ 772907504603451668939070378186397468859785465941286999731447050290303828457912084907223915223686346451102761292270730286436561474790548137525290500739995298056498851818700 6

$=$ (2)(835205577201087993065806 99)(596201686362718542354710701)(776087954036971417157963313951798343506780344407592335678148510064755548342323544940279982843984107558240344658148264 97)

Table 3.2: (Contd.) Examples for NIST-recommended Koblitz curves over $\mathbb{F}_{2^m}$

Case	m	**Curve**		**Size of each prime factor of $\#E(\mathbb{F}_{2^m})$ (in bits)**
Randomly chosen curves	163	NIST B-163	E	2,163
		Weaker curve	$\widehat{E}$	$2, 5, 10, 25, 28, 95$
	233	NIST B-233	E	$2, 233$
		Weaker curve	$\widehat{E}$	$2, 3, 9, 10, 30, 70, 113$
	283	NIST B-283	E	$2, 283$
		Weaker curve	$\widehat{E}$	$2, 3, 5, 33, 66, 86, 87$
	409	NIST B-409	E	$2, 409$
		Weaker curve	$\widehat{E}$	$2, 4, 6, 10, 11, 13, 14, 69, 284$
	571	NIST B-571	E	$2, 570$
		Weaker curve	$\widehat{E}$	$2, 3, 11, 47, 80, 82, 159, 191$
Koblitz curves	163	NIST K-163	E	$2, 163$
		Weaker curve	$\widehat{E}$	$2, 10, 13, 55, 85$
	233	NIST K-233	E	$2, 232$
		Weaker curve	$\widehat{E}$	$2, 17, 27, 27, 63, 100$
	283	NIST K-283	E	$2, 281$
		Other curve	$\widehat{E}$	$2, 284$
	409	NIST K-409	E	$2, 281$
		Weaker curve	$\widehat{E}$	$2, 23, 37, 116, 234$
	571	NIST K-571	E	$2, 569$
		Weaker curve	$\widehat{E}$	$2, 87, 89, 395$

Table 3.3: Size of each prime factor of $\#E(\mathbb{F}_{2^m})$ and $\#\widehat{E}(\mathbb{F}_{2^m})$ (in bits) for the examples of Tables 3.1 and 3.2

3.2 Invalid-curve attacks on Montgomery's ladder algorithm

Consider a cryptosystem that uses a *strong* elliptic curve $E(a, b)$ defined over $\mathbb{F}_{2^m}$ with curve parameters a and b (e.g., a NIST-recommended elliptic curve), where m is an odd number. Assume that $\widehat{E}(\widehat{a}, b)$ is a weaker curve defined over $\mathbb{F}_{2^m}$ with curve parameters $\widehat{a}$ and b, such that $\mathtt{Tr}(\widehat{a}) = 1 - \mathtt{Tr}(a)$. Consider that the attacker has the computational power for computing the EC discrete logarithm using the Silver-Pohlig-Hellman algorithm in the cryptographically weaker group $\widehat{E}(\mathbb{F}_{2^m})$. Also consider that $\widehat{E}(\mathbb{F}_{2^m})$ is a cyclic group, which implies that there are $\phi(\#\widehat{E}(\mathbb{F}_{2^m}))$ points of order $\#\widehat{E}(\mathbb{F}_{2^m})$. Additionally, for the attacks presented in this section we need to obtain $\#\widehat{E}(\mathbb{F}_{2^m})$. Using Equation (3.3), this value can be obtained from $\#E(\mathbb{F}_{2^m})$ which is usually public or can be obtained with some point counting algorithms, e.g., [79] [77]. Consider that the underlying ECSM algorithm is the Montgomery ladder (Algorithm 2.6 or 2.7). Since these algorithms do not utilize the curve parameter a, depending of the input point the computation can be carried out in either $E(\mathbb{F}_{2^m})$ or $\widehat{E}(\mathbb{F}_{2^m})$. Then, the idea behind the attacks presented below is to produce an incorrect result from the computation being performed in $\widehat{E}(\mathbb{F}_{2^m})$ due to a fault.

3.2.1 Basic attack

Fault model. Let us assume that the adversary can inject a flip-fault (single or multiple bit) into the x-coordinate of the input point $P = (P_x, P_y) \in E(\mathbb{F}_{2^m})$ of a device computing the ECSM utilizing either Algorithm 2.6 or 2.7. Suppose that the resulting finite field pair after the fault injection is known and is $\widetilde{P} = (\widetilde{P}_x, P_y)$.

Consider that the result $\widetilde{Q} = k\widetilde{P} = (\widetilde{Q}_x, \widetilde{Q}_y)$ is released.

For a given $\widetilde{P} = (\widetilde{P}_x, P_y)$ we can verify if there exists a point in $\widehat{E}(\mathbb{F}_{2^m})$ with the same x-coordinate, i.e., if $\exists\ \widehat{P} \in \widehat{E}(\mathbb{F}_{2^m})$ such that $\widehat{P} = (\widetilde{P}_x, \widehat{P}_y)$ for some $\widehat{P}_y \in \mathbb{F}_{2^m}$. In fact, by Lemma 3.1 we can expect that if we flip single or multiple bits of the x-coordinate such a point exists with a probability of about 1/2. When $\widehat{P} \in \widehat{E}(\mathbb{F}_{2^m})$, in a similar way we can obtain $\widehat{Q} = (\widetilde{Q}_x, \widehat{Q}_y) \in \widehat{E}(\mathbb{F}_{2^m})$ for some $\widehat{Q}_y \in \mathbb{F}_{2^m}$.

Having $\widehat{P}, \widehat{Q} \in \widehat{E}(\mathbb{F}_{2^m})$ one can obtain $l = k$ or $\#\widehat{E}(\mathbb{F}_{2^m}) - k \bmod n$ using Algorithm 2.8, where $n = \mathtt{ord}(\widehat{P})$. This would be possible because the computation is performed in the weaker group $\widehat{E}(\mathbb{F}_{2^m})$ and not in the original group $E(\mathbb{F}_{2^m})$. One can then exhaustively search for an integer k' that satisfies (i) $l = k' \bmod n$ or $\#\widehat{E}(\mathbb{F}_{2^m}) - k' \bmod n$ and (ii) $\widetilde{Q} = k'\widetilde{P}$. Thus, the idea of the basic attack is that the adversary with only one pair $(\widehat{P}, \widehat{Q})$ and some acceptable amount of exhaustive search will be able to retrieve the secret scalar k with a probability of success ρ. Let e be a parameter such that 2^e is the maximum acceptable amount of exhaustive search space. The complete attack procedure is presented as Algorithm 3.1.

In Step 8 of Algorithm 3.1, $l = k$ or $\#\widehat{E}(\mathbb{F}_{2^m}) - k \bmod n$ is obtained. The value of l has only partial information about k. The remaining part of the scalar might be obtained using an exhaustive search. The latter involves two main steps: (i) solve a system congruences with a test candidate and the known part of the scalar (Step 11.2.1), and (ii) perform a scalar multiplication to verify if the solution of the system of congruences is the desired scalar (Step 11.2.2).

Let r be the exhaustive search space. This value depends on n and $\#\widehat{E}(\mathbb{F}_{2^m})$. In Step 11.2.1, for having a unique solution $\bmod \#\widehat{E}(\mathbb{F}_{2^m})$ it is necessary that

$$\mathtt{lcm}(n, r) = \#\widehat{E}(\mathbb{F}_{2^m}). \tag{3.5}$$

Algorithm 3.1. Basic invalid-curve attack on Montgomery's ladder ECSM algorithm

Input: E defined over $\mathbb{F}_{2^m}$, access to either Algorithm 2.6 or 2.7, the base point $P = (P_x, P_y) \in E(\mathbb{F}_{2^m})$, the order $\#\widehat{E}(\mathbb{F}_{2^m})$, a parameter for acceptable amount of exhaustive search e.
Output: Scalar k with a probability of ρ.

\# Phase 1: Collect faulty output

1. Inject a fault in $P = (P_x, P_y)$ for obtaining $\widetilde{P} = (\widetilde{P}_x, P_y)$.
2. Compute $\widetilde{Q} = k\widetilde{P} = (\widetilde{Q}_x, \widetilde{Q}_y)$ using either Algorithm 2.6 or 2.7.
3. $T \leftarrow \widetilde{Q}_x + b/\widetilde{Q}_x^2 + \widehat{a}$.
4. If $(\mathtt{Tr}(T) = 0)$ then
   4.1 $\widehat{Q}_x \leftarrow \widetilde{Q}_x$, $\widehat{Q}_y \leftarrow \widetilde{Q}_x \cdot \mathtt{Ht}(T)$;
5. Else
   5.1 Go to Step 1.

\# Phase 2: Obtain k partially using the Silver-Pohlig-Hellman algorithm

6. $\widehat{P}_x \leftarrow \widetilde{P}_x$, $\widehat{P}_y \leftarrow \widetilde{P}_x \cdot \mathtt{Ht}(\widetilde{P}_x + b/\widetilde{P}_x^2 + \widehat{a})$.
7. Obtain $n = \mathtt{ord}(\widehat{P})$.
8. Utilize Algorithm 2.8 with $(\widehat{P}, \widehat{Q}, n)$ to obtain $l \bmod n$.

\# Phase 3: Exhaustive search and verification

9. Find the smallest value of r for $\mathtt{lcm}(n, r) = \#\widehat{E}(\mathbb{F}_{2^m})$ (see Equation (3.7)).
10. If $(r = 1)$ then

10.1 Compute $R = l\widetilde{P}$ using either Algorithm 2.6 or 2.7.

10.2 If $(R = \widetilde{Q})$ then return(l); else return$(\#\widehat{E}(\mathbb{F}_{2^m}) - l)$.

11. Else if $(r \leq 2^e)$ then

11.1 $k' \leftarrow 0$.

11.2 While $(k' < r)$ do

11.2.1 Solve the system of congruences $k'' \equiv k' \pmod{r}$ and $k'' \equiv l \pmod{n}$.

11.2.2 Compute $R = k''\widetilde{P}$ using either Algorithm 2.6 or 2.7.

11.2.3 If $(R = \widetilde{Q})$ then return(k'');

11.2.4 Else if $(R = -\widetilde{Q})$ then return$(\#\widehat{E}(\mathbb{F}_{2^m}) - k'')$;

11.2.5 Else $k' \leftarrow k' + 1$.

12. Else return("failure").

For efficiency r should be selected as the minimum value that satisfies Equation (3.5). Let $\#\widehat{E}(\mathbb{F}_{2^m}) = 2^{e_0}p_1^{e_1}p_2^{e_2}\cdots p_{u-1}^{e_{u-1}}$ be the prime factorization of $\#\widehat{E}(\mathbb{F}_{2^m})$, where $e_j \geq 1$ for $j \in [0, u-1]$. Let $n = 2^{f_0}p_1^{f_1}p_2^{f_2}\cdots p_{u-1}^{f_{u-1}}$ be the prime factorization of $n = \mathtt{ord}(\widehat{P})$, where $0 \leq f_j \leq e_j$ for $j \in [0, u-1]$. Similarly, let $r = 2^{g_0}p_1^{g_1}p_2^{g_2}\cdots p_{u-1}^{g_{u-1}}$ be the prime factorization of r. Using notations similar to those utilized by Menezes et al. [66] with regard to `lcm`, we can express Equation (3.5) as

$$2^{\mathtt{max}(f_0,g_0)}p_1^{\mathtt{max}(f_1,g_1)}p_2^{\mathtt{max}(f_2,g_2)}\cdots p_{u-1}^{\mathtt{max}(f_{u-1},g_{u-1})} = 2^{e_0}p_1^{e_1}p_2^{e_2}\cdots p_{u-1}^{e_{u-1}}. \tag{3.6}$$

The exponents of the minimum value of r that satisfies Equation (3.6) are

$$g_j = \begin{cases} 0 & \text{if } e_j = f_j, \\ e_j & \text{otherwise,} \end{cases} \tag{3.7}$$

for $j \in [0, u-1]$.

Note that if $r > 2^e$, Algorithm 3.1 returns in Step 12 "failure". This means that from a specific pair $(\widetilde{P}, \widetilde{Q})$ the exhaustive search space required to obtain uniquely the value of k (i.e., r) is more than the maximum admissible exhaustive search space (i.e., 2^e). For example for a weaker group $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves, as we show below, the probability of failure is quite low even for small values of e. Moreover, in the case of not success with a particular pair $(\widetilde{P}, \widetilde{Q})$, the attacker can repeat the attack procedure until an inevitable success.

The probability of success of Algorithm 3.1 (i.e., ρ), depends on the maximum acceptable amount of exhaustive search 2^e and the order of point $\widehat{P}$. Assume that point $\widehat{P}$ is taken randomly from group $\widehat{E}(\mathbb{F}_{2^m})$. In a cyclic group, it is well known that the number of elements of order d is $\phi(d)$. Here $\#\widehat{E}(\mathbb{F}_{2^m})$ is not prime, and consequently not all the points in $\widehat{E}(\mathbb{F}_{2^m})$ have an order $\#\widehat{E}(\mathbb{F}_{2^m})$. Moreover, if $\#\widehat{E}(\mathbb{F}_{2^m})$ has several prime factors (i.e., it is expected since $\widehat{E}(\mathbb{F}_{2^m})$ is assumed to be a weaker group), the order of the points could have any combination of those prime factors or their respective prime powers. For example the number of points with the full order $\#\widehat{E}(\mathbb{F}_{2^m})$ is $\phi(\#\widehat{E}(\mathbb{F}_{2^m}))$. In contrast, there is only one point of order two which corresponds to $(0, \sqrt{b})$.

Obtaining the probability of success ρ. Let $\#\widehat{E}(\mathbb{F}_{2^m}) = 2^{n_0} p_1^{n_1} p_2^{n_2} \cdots p_{u-1}^{n_{u-1}}$ be the prime factorization of $\#\widehat{E}(\mathbb{F}_{2^m})$, where $n_j \geq 1$ for $j \in [0, u-1]$ and $p_j < p_{j+1}$ for $j \in [1, u-2]$. Assume that point $\widehat{P}$ is taken randomly from the group $\widehat{E}(\mathbb{F}_{2^m})$. Here we will obtain the probability of success ρ, first for specific values and then for an arbitrary value of e.

- *Case 1:* $e = 0$. If $e = 0$, then the attack will succeed when $\texttt{ord}(\widehat{P}) =$

$\#\widehat{E}(\mathbb{F}_{2^m})$. The number of points in $\widehat{E}(\mathbb{F}_{2^m})$ of order $\#\widehat{E}(\mathbb{F}_{2^m})$ is

$$\phi(\#\widehat{E}(\mathbb{F}_{2^m})) = 2^{n_0-1}\prod_{j=1}^{u-1} p_i^{n_i}(1-\tfrac{1}{p_i}),$$

and for this case the probability ρ is

$$\rho_{e=0} = \frac{\phi\left(\#\widehat{E}(\mathbb{F}_{2^m})\right)}{\#\widehat{E}(\mathbb{F}_{2^m})} = \frac{1}{2}\prod_{j=1}^{u-1}(1-\frac{1}{p_j}). \tag{3.8}$$

Clearly this value is bounded to 1/2. If $p_1 >> 1$, then $\rho_{e=0}$ would be close to 1/2 (e.g., all the Koblitz curves in Example 4.2).

- *Case 2:* $e = 1$. For $e = 1$, this probability can be obtained as follows

$$\rho_{e=1} = \begin{cases} \prod_{j=1}^{u-1}(1-\frac{1}{p_j}), & \text{if } n_0 = 1, \\ \frac{1}{2}\prod_{j=1}^{u-1}(1-\frac{1}{p_j}), & \text{otherwise.} \end{cases} \tag{3.9}$$

- *Case 3:* $e = 2$. For $e = 2$ we can have two cases. First, if $p_1 \neq 3$, then $\rho_{e=2}$ is

$$\rho_{e=2} = \begin{cases} \prod_{j=1}^{u-1}(1-\frac{1}{p_j}), & \text{if } n_0 = 1 \text{ or } 2, \\ \frac{1}{2}\prod_{j=1}^{u-1}(1-\frac{1}{p_j}), & \text{otherwise.} \end{cases} \tag{3.10}$$

Secondly, if $p_1 = 3$, then it is necessary to take into account points of order $\#\widehat{E}(\mathbb{F}_{2^m})/h$, with $h \in [1, 3]$. In this case $\rho_{e=2}$ is

$$\rho_{e=2} = \begin{cases} \frac{5}{6}\prod_{j=2}^{u-1}(1-\frac{1}{p_j}), & \text{if } n_0 = 1 \text{ or } 2, \text{ and } n_1 = 1, \\ \frac{2}{3}\prod_{j=2}^{u-1}(1-\frac{1}{p_j}), & \text{if } n_0 = 1 \text{ or } 2, \text{ and } n_1 \geq 2, \\ \frac{1}{6}\prod_{j=2}^{u-1}(1-\frac{1}{p_j}), & \text{if } n_0 \geq 3, \text{ and } n_1 = 1, \\ \frac{1}{3}\prod_{j=2}^{u-1}(1-\frac{1}{p_j}), & \text{otherwise.} \end{cases} \tag{3.11}$$

- *Case 4: Arbitrary e with some conditions.* Let

$$\#\widehat{E}(\mathbb{F}_{2^m}) = 2^{n_0} p_1^{n_1} p_2^{n_2} \cdots p_{t-1}^{n_{t-1}} p_t^{n_t} p_{t+1}^{n_{t+1}} \cdots p_{u-1}^{n_{u-1}}.$$

Assume that $\#\widehat{E}(\mathbb{F}_{2^m})$ splits completely in e bits such that

$$\log_2(2^{n_0} p_1^{n_1} \cdots p_{t-1}^{n_{t-1}}) \leq e \quad \text{and} \quad \log_2(p_t) > e.$$

If these conditions are satisfied, then the number of points whose order divides $p_t^{n_t} p_{t+1}^{n_{t+1}} \cdots p_{u-1}^{n_{u-1}}$ is

$$s = \sum_{i=0}^{g-1} \phi\left(2^{j_0(i)} p_1^{j_1(i)} p_2^{j_2(i)} \cdots p_{t-1}^{j_{t-1}(i)} p_t^{n_t} p_{t+1}^{n_{t+1}} \cdots p_{u-1}^{n_{u-1}}\right), \tag{3.12}$$

where

$$\begin{aligned} g &= (n_0+1)(n_1+1)\cdots(n_{t-1}+1) \\ j_0(i) &= i \bmod (n_0+1), \end{aligned}$$

$$\begin{aligned} j_1(i) &= \left\lfloor \frac{i}{n_0+1} \right\rfloor \bmod (n_1+1), \\ j_2(i) &= \left\lfloor \frac{i}{(n_0+1)(n_1+1)} \right\rfloor \bmod (n_2+1), \\ &\vdots \\ j_{t-1}(i) &= \left\lfloor \frac{i}{(n_0+1)(n_1+1)\cdots(n_{t-2}+1)} \right\rfloor \bmod (n_{t-1}+1). \end{aligned}$$

It can be shown that

$$\sum_{i=0}^{g-1} \phi\left(2^{j_0(i)} p_1^{j_1(i)} p_2^{j_2(i)} \cdots p_{t-1}^{j_{t-1}(i)}\right) = 2^{n_0} p_1^{n_1} p_2^{n_2} \cdots p_{t-1}^{n_{t-1}}$$

Since the function ϕ is *multiplicative*[1] we can reduce Equation (3.12) and obtain

$$s = 2^{n_0} p_1^{n_1} \cdots p_{t-1}^{n_{t-1}} p_t^{n_t-1}(p_t-1) p_{t+1}^{n_{t+1}-1}(p_{t+1}-1)\cdots p_{u-1}^{n_{u-1}-1}(p_{u-1}-1).$$

In this case ρ is as follows,

$$\rho = \frac{s}{\#\widehat{E}(\mathbb{F}_{2^m})} = \frac{(p_t-1)(p_{t+1}-1)\cdots(p_{u-1}-1)}{p_t p_{t+1}\cdots p_{u-1}}. \tag{3.13}$$

- *Case 5: Arbitrary e.* When we cannot split $\#\widehat{E}(\mathbb{F}_{2^m})$ in the form as in the previous case we can proceed as follows. First, search for the smallest prime factor such that $\log_2(p_i) > e$. Let t be the index of such prime factor. Let $d = p_t^{n_t} p_{t+1}^{n_t} \cdots p_{u-1}^{n_{u-1}}$. From all the possible combinations of the prime factors $p_0 p_1 \cdots p_{t-1}$ and their respective powers, we need to consider only those whose product with d have a value of r that satisfies Equation (3.5) and $r \leq e$. The

[1] If $\gcd(m,n) = 1$, then $\phi(mn) = \phi(m)\phi(n)$.

complete procedure for this case is stated in Algorithm 3.2. This algorithm also includes the computation of ρ for Cases 1-4.

Algorithm 3.2. Probability of success ρ for Algorithm 3.1

Input: The order $\#\widehat{E}(\mathbb{F}_{2^m}) = 2^{n_0} p_1^{n_1} p_2^{n_2} \cdots p_{t-1}^{n_{t-1}} p_t^{n_t} p_{t+1}^{n_t} \cdots p_{u-1}^{n_{u-1}}$, a parameter for acceptable amount of exhaustive search e, where $0 \leq e < \log_2(p_{u-1})$.
Output: Probability of success ρ.

---

1. If $(e = 0)$ then return$(\rho_{e=0})$ using Equation (3.8);
2. Else if $(e = 1)$ then return$(\rho_{e=1})$ using Equation (3.9);
3. Else if $(e = 2)$ then return$(\rho_{e=2})$ using Equation (3.10) or (3.11);
4. Else if $\#\widehat{E}(\mathbb{F}_{2^m})$ splits completely in e bits such that $\log_2(2^{n_0} p_1^{n_1} \cdots p_{t-1}^{n_{t-1}}) \leq e$ and $\log_2(p_t) > e$
   4.1 Return(ρ) using Equation (3.13);
5. Else
   5.1 Search for the smallest prime factor such that $\log_2(p_i) > e$. Set t with this index.
   5.2 $d \leftarrow p_t^{n_t} p_{t+1}^{n_{t+1}} \cdots p_{u-1}^{n_{u-1}}$.
   5.3 $\rho \leftarrow 0$.
   5.4 For $j_{t-1} = 0$ to n_{t-1} do

       For $j_{t-2} = 0$ to n_{t-2} do

       $\vdots$

       For $j_0 = 0$ to n_0 do

$h \leftarrow 2^{j_0} p_1^{j_1} \cdots p_{t-2}^{j_{t-2}} p_{t-1}^{j_{t-1}}$.
Find the smallest value of r for $\texttt{lcm}(d \cdot h, r) = \#\widehat{E}(\mathbb{F}_{2^m})$.
If $(r \leq 2^e)$ then
$\rho \leftarrow \rho + \phi(h)$.

5.5 $\rho \leftarrow \rho(p_t - 1)(p_{t+1} - 1) \cdots (p_{u-1} - 1)/(2^{n_0} p_1^{n_1} p_2^{n_2} \cdots p_{t-1}^{n_{t-1}} p_t p_{t+1} \cdots p_{u-1})$.

5.6 Return(ρ).

Probability of success ρ for $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves. Table 3.4 presents the probability of success of Algorithm 3.1 for $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves (see Example 3.2). This shows the probability of obtaining the scalar k using a single faulty point $\widetilde{P} \in \widehat{E}(\mathbb{F}_{2^m})$ and specific values of parameter e. We notice that with the minimum amount of exhaustive search (i.e., $e = 0$) the values are close to 1/2, especially for the Koblitz curve cases where the relation between the two smallest prime factors of $\#\widehat{E}(\mathbb{F}_{2^m})$ is greater (e.g., $p_1/2 \approx 10.8 \times 10^6$ for the example of the Koblitz curve over $\mathbb{F}_{2^{409}}$). Also for the Koblitz curve examples, it can be noticed that with $e = 2$ their probabilities are close to unity as shown in the fifth column of Table 3.4. In contrast, for the randomly chosen curves, similar values close to the unity are obtained with $e = 10$ as illustrated in the right-most column of this table.

Table 3.5 shows the minimum value of parameter e for obtaining a probability ρ smaller than some specific values. From this table it can be noticed that for practical situations e could be quite small for an exhaustive search (e.g., say 14) and still have a reasonably high probability of success ρ (e.g., $\rho > \frac{999}{1000}$).

Case	m	ρ				
		$e = 0$	$e = 1$	$e = 2$	$e = 5$	$e = 10$
Randomly chosen curves	163	0.48333745	0.48333745	0.96667491	0.98278616	0.99943089
	233	0.39784981	0.39784981	0.79569963	0.99462453	0.99677211
	283	0.40601504	0.40601504	0.81203008	0.94736842	0.96992481
	409	0.44966230	0.44966230	0.89932460	0.93679646	0.99732494
	571	0.42819973	0.42819973	0.85639945	0.99913270	0.99913270
Koblitz curves	163	0.49915775	0.49915775	0.99831549	0.99831549	0.99908107
	233	0.49999457	0.99998915	0.99998915	0.99998915	0.99998915
	409	0.49999991	0.99999982	0.99999982	0.99999982	0.99999982
	571	0.49999999	0.99999999	0.99999999	0.99999999	0.99999999

Table 3.4: Probability of success ρ of obtaining k with Algorithm 3.1 for $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves[3] for a given parameter e

Cost of Algorithm 3.1. Most of the computational cost of Algorithm 3.1 is involved in phases 2 and 3, i.e., obtaining k partially using the Silver-Pohlig-Hellman algorithm (Algorithm 2.8) and the exhaustive search with verification process, respectively. The cost of both phases depends on the order of $\widehat{P}$, i.e., n, and the order $\#\widehat{E}(\mathbb{F}_{2^m})$. Let us consider the cost of each phase:

- *Silver-Pohlig-Hellman's algorithm (phase 2 of Algorithm 3.1).* Step 1.3.2 of the Silver-Pohlig-Hellman algorithm (Algorithm 2.8), which is the only step in this algorithm with significant cost, needs to compute one EC discrete logarithm. This operation can be performed with a fast algorithm for ECDLP such as Pollard's rho algorithm [73] with an expected number of point operations of about $3\sqrt{p_{t-1}}$, where p_{t-1} is the largest prime divisor of n. This

[3]The case of $m = 283$ for Koblitz curves is omitted for this and any subsequent table since there does not exist a cryptographically weaker group $\widehat{E}(\mathbb{F}_{2^m})$.

Case	m	Parameter e (in bits)		
		$\rho < 1 - \frac{1}{100}$	$\rho < 1 - \frac{1}{1000}$	$\rho < 1 - \frac{1}{1 \times 10^6}$
Randomly chosen curves	163	7	10	17
	233	5	12	20
	283	11	14	14
	409	8	12	23
	571	5	5	15
Koblitz curves	163	2	10	15
	233	1	1	18
	409	1	1	1
	571	1	1	1

Table 3.5: Minimum value of parameter e for obtaining a probability ρ smaller than some given values for $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves

running time can be further reduced using a parallelized version of the Pollard's rho algorithm [86] to about $(\sqrt{\pi p_{t-1}/2})/M$ point operations, where M is the number of processors used for solving the ECDLP instance. Additionally, as shown by Gallant et al. [35] if a Koblitz curve over $\mathbb{F}_{2^m}$ is utilized, then the parallelized version of the Pollard's rho algorithm can take about $(\sqrt{\pi p_{t-1}/m})/(2M)$ point operations.

- *Exhaustive search and verification (phase 3 of Algorithm 3.1).* With $n = \texttt{ord}(\widehat{P})$ and $\#\widehat{E}(\mathbb{F}_{2^m})$, the exhaustive search space r is obtained using Equation (3.5) (see Step 9 of Algorithm 3.1). Thus, assuming $t \approx m$ the phase 3 of Algorithm 3.1 will require r scalar multiplications in the worst case which represents at most $(3mr)/2$ point operations if a binary method is utilized (e.g., Algorithm 2.1).

Example 3.3 Let us consider the cost of phases 2 and 3 of Algorithm 3.1 for $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curve K-163. For a single processor, the cost of phase 2 is of about $3\sqrt{p_4} \approx 2^{43.6}$ point operations, where p_4 is the largest prime factor of $\#\widehat{E}(\mathbb{F}_{2^m})$ (see Table 3.2). Now, assume that we have $M = 10,000$ computers for solving the instance of the ECDLP. In this case the expected number of point operations for each processor is approximately $(\sqrt{\pi p_4/163})/20000 \approx 2^{24.9}$. For the phase 3 cost, from Tables 3.1 and 3.5 we can notice that with a probability greater than $\frac{999}{1000}$ the exhaustive search space will be less than 2^{10}, which implies a number of point operations $< 3(163)(2^{10})/2 \approx 2^{17.9}$.

3.2.2 Attack with unknown faulty base finite field pair $\widetilde{P}$

Fault model. Let us assume that the adversary can inject a single bit-flip fault into the x-coordinate of the input point $P_i = (P_{i,x}, P_{i,y}) \in E(\mathbb{F}_{2^m})$ of a device computing the ECSM utilizing either Algorithm 2.6 or 2.7 for some i. Suppose that the resulting finite field pair after the fault injection $\widetilde{P}_i = (\widetilde{P}_{i,x}, P_{i,y})$ is unknown. Also, consider that the fault location is at a random position of the x-coordinate. Consider that the result $\widetilde{Q}_i = k\widetilde{P}_i = (\widetilde{Q}_{i,x}, \widetilde{Q}_{i,y})$ is realized.

Under this scenario the attacker might retrieve the secret scalar as follows. First, it is necessary to collect some faulty outputs of the form $\widetilde{Q}_i = k\widetilde{P}_i = (\widetilde{Q}_{i,x}, \widetilde{Q}_{i,y})$ for which there exists a point $\widehat{Q}_i \in \widehat{E}(\mathbb{F}_{2^m})$ such that $\widehat{Q}_i = (\widetilde{Q}_{i,x}, \widehat{Q}_{i,y})$ for some $\widehat{Q}_{i,y} \in \mathbb{F}_{2^m}$. In fact, with two different points $\widehat{Q}_i \in \widehat{E}(\mathbb{F}_{2^m})$, where $i \in \{0, 1\}$, and some acceptable amount of exhaustive search it is possible to obtain k with a high probability.

Let $\widehat{P}_i$ be a point in $\widehat{E}(\mathbb{F}_{2^m})$ with the same x-coordinate as $\widetilde{P}_i = (\widetilde{P}_{i,x}, P_{i,y})$,

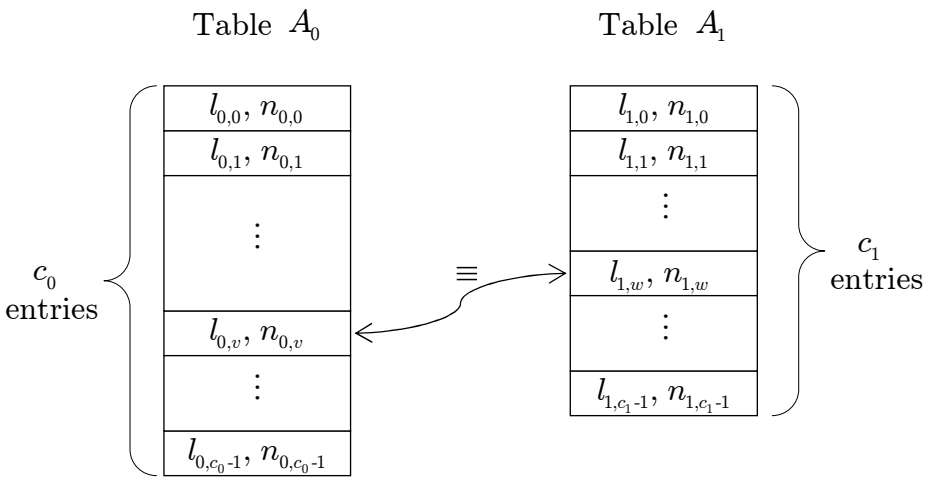

Figure 3.1: Tables A_0 and A_1 with the output of the Silver-Pohlig-Hellman algorithm for each $(R_{i,j}, \widehat{Q}_i, n_{i,j})$, where $i \in \{0,1\}$, $j \in [0, c_i - 1]$, and $n_{i,j} = \texttt{ord}(R_{i,j})$

i.e., $\widehat{P}_i = (\widetilde{P}_{i,x}, \widehat{P}_{i,y}) \in \widehat{E}(\mathbb{F}_{2^m})$ for some $\widehat{P}_{i,y} \in \mathbb{F}_{2^m}$. Since $\widetilde{P}_i$ (and consequently $\widehat{P}_i$) is unknown, we need to guess it among those finite field pairs that differ from each P_i in only one bit of their x-coordinate. Let c_i be the number of possible candidates for $\widehat{P}_i$, where $i \in \{0,1\}$. Let $R_{i,j}$ be a candidate for $\widehat{P}_i$, where $i \in \{0,1\}$ and $j \in [0, c_i - 1]$. Initially, by Lemma 3.1 we can expect that c_i is about $m/2$. However, this amount could be further reduced depending on the order of $\widehat{Q}_i$. This is possible because we known that $\texttt{ord}(\widehat{Q}_i) \leq \texttt{ord}(\widehat{P}_i)$, and more precisely $\texttt{ord}(\widehat{Q}_i) | \texttt{ord}(\widehat{P}_i)$. Let η_i be the reduction factor due to the latter condition such that $c_i \approx \eta_i \frac{m}{2}$.

After collecting the faulty outputs we can construct two tables A_i of c_i entries with the output of the Silver-Pohlig-Hellman algorithm for each $(R_{i,j}, \widehat{Q}_i, n_{i,j})$, where $i \in \{0,1\}$, $j \in [0, c_i - 1]$, and $n_{i,j} = \texttt{ord}(R_{i,j})$. These tables are illustrated in

Figure 3.1. Thus, having $l_{i,j}$ `mod` $n_{i,j}$ in each entry of Tables A_0 and A_1, we could distinguish those that are likely to be equivalent to either k or $\#\widehat{E}(\mathbb{F}_{2^m}) - k$. The idea is to search entry pairs v and w that satisfy either

$$l_{0,v} \equiv l_{1,w} \ (\texttt{mod gcd}(n_{0,v}, n_{1,w})) \quad \text{or} \tag{3.14}$$

$$l_{0,v} \equiv \#\widehat{E}(\mathbb{F}_{2^m}) - l_{1,w} \ (\texttt{mod gcd}(n_{0,v}, n_{1,w})). \tag{3.15}$$

In practical situations where $m \geq 163$ it is more likely to have a unique candidate pair that satisfies either (3.14) or (3.15). The main reason is because it is expected that $n_{i,j} >> c_i$ for $i \in \{0, 1\}$ and $j \in [0, c_i - 1]$. Nevertheless, even if there is not a unique candidate pair it is possible to verify which one is equivalent to k or $\#\widehat{E}(\mathbb{F}_{2^m}) - k$ after performing an exhaustive search similarly to the attack presented in the previous subsection. The complete attack procedure is presented in Algorithm 3.3. Let e be a parameter such that 2^e is the maximum acceptable amount of exhaustive search per candidate pair found in Step 5 of Algorithm 3.3. Also, let us define σ as the probability of success for retrieving the scalar k using Algorithm 3.3.

Algorithm 3.3. Invalid-curve attack with unknown faulty base point $\widetilde{P}$

Input: E defined over $\mathbb{F}_{2^m}$, access to either Algorithm 2.6 or 2.7, base point $P_i = (P_{i,x}, P_{i,y}) \in E(\mathbb{F}_{2^m})$ with $i \in \{0, 1\}$, the order $\#\widehat{E}(\mathbb{F}_{2^m})$, a parameter for acceptable amount of exhaustive search e.
Output: Scalar k with a probability of σ

Phase 1: Collect faulty outputs

1. $i \leftarrow 0$.

2. While $(i < 2)$ do

   2.1 Inject a fault in $P_i = (P_{i,x}, P_{i,y})$ for obtaining $\widetilde{P}_i = (\widetilde{P}_{i,x}, P_{i,y})$.

   2.2 Compute $\widetilde{Q}_i = k\widetilde{P}_i = (\widetilde{Q}_{i,x}, \widetilde{Q}_{i,y})$ using either Algorithm 2.6 or 2.7.

   2.3 $T_1 \leftarrow \widetilde{Q}_{i,x} + b/\widetilde{Q}_{i,x}^2 + \widehat{a}$.

   2.4 If $(\mathtt{Tr}(T_1) = 0)$ then

      2.4.1 $\widehat{Q}_{i,x} \leftarrow \widetilde{Q}_{i,x}$, $\widehat{Q}_{i,y} \leftarrow \widetilde{Q}_{i,x} \cdot \mathtt{Ht}(T_1)$, $i \leftarrow i + 1$.

Phase 2: Construct tables

3. For $i = 0$ to 1 do

4. $T_2 \leftarrow 1$.

   4.1 For $j = 0$ to $m - 1$ do

      4.1.1 $R_x \leftarrow P_{i,x} + T_2$.

      4.1.2 $T_3 \leftarrow R_x + b/{R_x}^2 + \widehat{a}$.

      4.1.3 If $(\mathtt{Tr}(T_3) = 0)$ then

         (a) $R_y \leftarrow R_x \cdot \mathtt{Ht}(T_3)$.

         (b) Obtain $n = \mathtt{ord}(R)$.

         (c) If $(\mathtt{ord}(\widehat{Q}_i)|n)$ then

            (i) Utilize Algorithm 2.8 with $(R, \widehat{Q}_i, n)$ to obtain $l \bmod n$.

            (ii) Store (l, n) in Table A_i.

      4.1.4 $T_2 = T_2 \ll 1$.

Phase 3: Searching for candidate pairs

5. For some entries v and w in tables Tables A_0 and A_1, respectively, search for candidate pairs that satisfy $l_v \equiv l_w \pmod{\gcd(n_v, n_w)}$ or $l_v \equiv \#\widehat{E}(\mathbb{F}_{2^m}) - l_w \pmod{\gcd(n_v, n_w)}$.

6. For the candidate pairs where $l_v \equiv \#\widehat{E}(\mathbb{F}_{2^m}) - l_w \pmod{\gcd(n_v, n_w)}$ set $l_w \leftarrow \#\widehat{E}(\mathbb{F}_{2^m}) - l_w \pmod{n_w}$ in Table A_1.

Phase 4: Exhaustive search and verification

7. For each candidate pair do

   7.1 Solve the system of congruences $l \equiv l_v \pmod{n_v}$ and $l \equiv l_w \pmod{n_w}$.

   7.2 $n \leftarrow \mathtt{lcm}(n_v, n_w)$.

   7.3 Find the smallest value of r for $\mathtt{lcm}(n, r) = \#\widehat{E}(\mathbb{F}_{2^m})$.

   7.4 If $(r = 1)$ then

      7.4.1 Compute $R = l\widetilde{P}$ using either Algorithm 2.6 or 2.7.

      7.4.2 If $(R = \widetilde{Q})$ then return(l);

      7.4.3 Else if $(R = -\widetilde{Q})$ then return$(\#\widehat{E}(\mathbb{F}_{2^m}) - l'')$.

   7.5 Else if $(r \leq 2^e)$ then

      7.5.1 $k' \leftarrow 0$.

      7.5.2 While $(k' < r)$ do

         (a) Solve the system of congruences $k'' \equiv k' \pmod{r}$ and $k'' \equiv l \pmod{n}$.

         (b) Compute $R = k''\widetilde{P}$ using either Algorithm 2.6 or 2.7.

         (c) If $(R = \widetilde{Q})$ then return(k'');

         (d) Else if $(R = -\widetilde{Q})$ then return$(\#\widehat{E}(\mathbb{F}_{2^m}) - k'')$;

         (e) Else $k' \leftarrow k' + 1$.

8. Return("failure").

Number of entries of Tables A_0 and A_1. Let $\#\widehat{E}(\mathbb{F}_{2^m}) = 2^{e_0}p_1^{e_1}p_2^{e_2}\cdots p_{u-1}^{e_{u-1}}$ be the prime factorization of $\#\widehat{E}(\mathbb{F}_{2^m})$. As stated before, the number of entries of Table A_i, c_i, depends on the reduction factor η_i. The latter in turn depends on the order of $\widehat{Q}_i$ and the order of the candidate points for $\widehat{P}_i$, $R_{i,j}$, where $i \in \{0,1\}$ and $j \in [0, c_i - 1]$. Assuming that the points $R_{i,j}$ are taken randomly from the group $\widehat{E}(\mathbb{F}_{2^m})$, it can be shown that η_i depending on $\texttt{ord}(\widehat{Q}_i)$ has the following bounds

$$\eta_{\max} \leq \eta_i \leq 1,$$

where $\eta_{\max} = \frac{1}{2}\prod_{j=1}^{u-1}(1 - \frac{1}{p_j})$. The lower bound of the above expression correspond for the case when $\texttt{ord}(\widehat{Q}_i) = \#\widehat{E}(\mathbb{F}_{2^m})$. In this case the reduction factor is maximum (i.e., $\eta_{\max}$), and consequently the number of entries of Table A_i is minimum (i.e., $c_{\min} \approx \frac{\eta_{\max} m}{2}$). On the other hand, theoretically the upper bound of η_i holds only when $\texttt{ord}(\widehat{Q}_i)$ is the point of order two $(0, \sqrt{b})$. However, for the cases where $p_1 >> 2$ (e.g., $\widehat{E}(\mathbb{F}_{2^m})$ for the Koblitz curves of Table 3.2) if $\texttt{ord}(\widehat{Q}_i) = \#\widehat{E}(\mathbb{F}_{2^m})/2^{e_0}$, then the reduction factor is close to unity. For these cases the number of entries of Table A_i is maximum (i.e., $c_{\max} \approx \frac{m}{2}$). In Table 3.6 the values of $\eta_{\max}$, $c_{\min}$, and $c_{\max}$ are given for each $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves. Also, this table shows the average cases for η_i and c_i (i.e., $\overline{\eta}$ and $\overline{c}$, respectively).

Algorithm 3.3 needs to compute in total $c_0 + c_1$ EC discrete logarithms using the Silver-Pohlig-Hellman algorithm. This number is fixed since the search for candidate pairs and the exhaustive search phases are performed after the tables's construction. If we merge these three phases, a speedup on average can be achieved. Let us describe two approaches one could take to combine these phases:

1. We can first completely construct Table A_0. Then, each time an entry of Table A_1 is obtained we can verify whether this entry satisfies the congruence

Case	m	$\eta_{\max}$	$\overline{\eta}$	$\frac{\eta_{\max} m}{2} \approx c_{\min}$	$\frac{m}{2} \approx c_{\max}$	$\frac{\overline{\eta} m}{2} \approx \overline{c}$
Randomly chosen curves	163	0.483	0.665	39.4	81.5	54.2
	233	0.398	0.574	46.3	116.5	66.9
	283	0.406	0.573	57.5	141.5	81.1
	409	0.450	0.623	92.0	204.5	127.3
	571	0.428	0.603	122.2	285.5	172.1
Koblitz curves	163	0.499	0.686	40.7	81.5	55.9
	233	0.499	0.749	58.2	116.5	87.4
	409	0.499	0.749	102.2	204.5	153.4
	571	0.499	0.749	142.7	285.5	214.1

Table 3.6: Minimum, maximum and average number of entries of Tables A_i for $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves

in (3.14) or (3.15) with any entry of A_0. For each candidate pair found (if any) we proceed with the exhaustive search and verification process. If the verification fails, then we continue to obtain the next entry of Table A_1 and repeat the process until the scalar is obtained. Even when using this approach the number of EC discrete logarithms in the worst case is the same as that using Algorithm 3.3 (i.e., $c_0 + c_1$), on average it is roughly $c_0 + \frac{1}{2}c_1$.

2. Another approach is to construct Tables A_0 and A_1 in alternate way. Each time an entry in A_i is obtained, we can search Table $A_{\bar{i}}$ for candidate pairs that satisfy either Congruence (3.14) or (3.15) for $i \in \{0, 1\}$. For each candidate pair found (if any) we proceed with the exhaustive search and verification process. This process is repeated until a candidate pair passes the verification process, i.e., the scalar is found. Let Tables A_0 and A_1 be of the same size, i.e., $c_0 = c_1$. For this case the average number of EC discrete logarithms

is $\approx \frac{4}{3}c_0$. In Appendix A we show how the latter value is obtained. This appendix also includes the case where $c_0 \neq c_1$.

Obtaining the probability of success σ. The probability of success σ of Algorithm 3.3 depends on parameter e and the order of both $\widehat{P}_0$ and $\widehat{P}_1$. Consider that the latter two points are taken randomly from the group $\widehat{E}(\mathbb{F}_{2^m})$. For each trio $(\widehat{P}_i, \widehat{Q}_i, n_i)$, the Silver-Pohlig-Hellman algorithm provides $l_i \bmod n_i$, where $i \in \{0, 1\}$ and $n_i = \mathtt{ord}(\widehat{P}_i)$. Utilizing these values, a system of congruences is solved and a solution $\bmod n$ is obtained, where $n = \mathtt{lcm}(n_0, n_1)$ (see Step 7.2). This "combination" of modulus n_i might reduce the exhaustive search space in comparison with the individual case of n_0 or n_1. This observation permits us to obtain a relation between the probabilities of success ρ and σ for Algorithms 3.1 and 3.3, respectively. In this case ρ is the probability that from an individual pair (l_i, n_i), $i = 0$ or 1, we could obtain the scalar using exhaustive search for a given value of e. Then we can express σ as follows:

$$\sigma = 2\rho - \rho^2 + \lambda. \tag{3.16}$$

The first two terms represent the probability that for a given e we could obtain the scalar from at least one of the two pairs. The third term, λ, is the probability that the "combination" does succeed in obtaining the scalar with exhaustive search when neither pair individually does so for a given value of e. Equation (3.16) gives an explicit lower bound for σ, i.e., $\sigma \geq 2\rho - \rho^2$. In fact, for the cases of $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves we notice that $\sigma \approx 2\rho - \rho^2$ for $e \geq 2$.

For obtaining a more precise value of σ one can check, from all the possible order values of two points (i.e., $\widehat{P}_0$ and $\widehat{P}_1$), which ones provide sufficient scalar information for obtaining the rest using exhaustive search for a given parameter e.

Additionally we need to consider the probability of occurrence of every point order combination. The complete procedure is put together in Algorithm 3.4.

Algorithm 3.4. Probability of success σ for Algorithm 3.3

Input: The order $\#\widehat{E}(\mathbb{F}_{2^m}) = 2^{n_0} p_1^{n_1} \cdots p_{u-1}^{n_{u-1}}$, a parameter for acceptable amount of exhaustive search e, where $e \geq 0$.

Output: Probability of success σ.

---

1. $\sigma = 0$
2. For $J_{u-1} = 0$ to n_{u-1} do

   For $J_{u-2} = 0$ to n_{u-2} do

   $\vdots$

   For $J_0 = 0$ to n_0 do

   $D \leftarrow 2^{J_0} p_1^{J_1} \cdots p_{u-1}^{J_{u-1}}$

   $N \leftarrow \phi(D)$

   For $j_{u-1} = 0$ to n_{u-1} do

   For $j_{u-2} = 0$ to n_{u-2} do

   $\vdots$

   For $j_0 = 0$ to n_0 do

   $d \leftarrow 2^{j_0} p_1^{j_1} \cdots p_{u-1}^{j_{u-1}}$

   $n \leftarrow \mathtt{lcm}(D, d)$

   Find the smallest value of r for $\mathtt{lcm}(n, r) = \#\widehat{E}(\mathbb{F}_{2^m})$.

   If $(r \leq 2^e)$ then

   $\sigma \leftarrow \sigma + N \cdot \phi(d)$.
3. $\sigma = \sigma / (\#\widehat{E}(\mathbb{F}_{2^m}))^2$
4. Return(σ).

Case	m	σ				
		$e = 0$	$e = 1$	$e = 2$	$e = 5$	$e = 10$
Randomly chosen curves	163	0.74921865	0.74921865	0.99895820	0.99973864	0.99999970
	233	0.71998855	0.71998855	0.95998473	0.99998410	0.99999555
	283	0.73265871	0.73265871	0.97687829	0.99722992	0.99926508
	409	0.74515657	0.74515657	0.99354209	0.99797754	0.99999814
	571	0.73469332	0.73469332	0.97959110	0.99999925	0.99999925
Koblitz curves	163	0.74999822	0.74999822	0.99999763	0.99999763	0.99999939
	233	0.74999999	0.99999999	0.99999999	0.99999999	0.99999999
	409	0.74999999	0.99999999	0.99999999	0.99999999	0.99999999
	571	0.74999999	0.99999999	0.99999999	0.99999999	0.99999999

Table 3.7: Probability of success σ of obtaining k with Algorithm 3.3 for $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves for a given parameter e

Probability of success σ for $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves. Table 3.7 presents the probability of success of Algorithm 3.3 for $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves. This shows the probability of obtaining the scalar k for specific values of parameter e. These values were obtained using Algorithm 3.4. We notice that the probability of success is better in comparison with the basic attack. In fact, for $e \geq 2$ the relation between the probability of success of both attacks is $\sigma \approx 2\rho - \rho^2$. In Table 3.8, we list the minimum value of parameter e for obtaining a probability σ smaller than some specific values. This table shows that even with small values of e (e.g., say 14) the probability of success is quite high (e.g., $\sigma > \frac{999{,}999}{1{,}000{,}000}$).

Cost of Algorithm 3.3. The most significant computational cost of Algorithm 3.3 is involved in phases 2 and 4, i.e., construction of tables and the exhaustive

Case	m	Parameter e (in bits)		
		$\sigma < 1 - \frac{1}{100}$	$\sigma < 1 - \frac{1}{1000}$	$\sigma < 1 - \frac{1}{1 \times 10^6}$
Randomly chosen curves	163	2	5	10
	233	5	5	12
	283	3	9	14
	409	2	6	12
	571	3	5	5
Koblitz curves	163	2	2	10
	233	1	1	1
	409	1	1	1
	571	1	1	1

Table 3.8: Minimum value of parameter e for obtaining a probability σ smaller than some given values for $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves

search with verification process, respectively. Let us consider the cost of each phase:

- *Construction of tables (phase 2 of Algorithm 3.3).* Comparing with the basic attack presented in the previous subsection (Algorithm 3.1), Algorithm 3.3 needs to perform $c_0 + c_1$ instances of the Silver-Pohlig-Hellman algorithm (Algorithm 2.8) instead of one, where c_i is the size of Table A_i for $i \in \{0, 1\}$. Similar to the cost of phase 2 of Algorithm 3.1, the cost to construct the tables with a single processor is about $3(c_0 + c_1)\sqrt{p_{t-1}}$ point operations, where p_{t-1} is the largest prime divisor of $\#\widehat{E}(\mathbb{F}_{2^m})$. If M processors are used, then about $(c_0 + c_1)\sqrt{\pi p_{t-1}/2}/M$ point operations are required. If a Koblitz curve over $\mathbb{F}_{2^m}$ is utilized, then this cost can be reduced to about $(c_0 + c_1)(\sqrt{\pi p_{t-1}/m})/(2M)$ point operations. These costs clearly depends directly on values of c_i which depends on the order of $\widehat{Q}_i$ and the order of the

candidate points for $\widehat{P}_i$. As discussed earlier, the bounds for c_i are approximately $\frac{\eta_{\max} m}{2} \leq c_i \leq \frac{m}{2}$, where $\eta_{\max}$ is the maximum reduction factor which depends on $\#\widehat{E}(\mathbb{F}_{2^m})$.

- *Exhaustive search and verification (phase 4 of Algorithm 3.3).* In phase 3 of Algorithm 3.3 using Tables A_0 and A_1, a search for candidate pairs that satisfy either (3.14) or (3.15) is performed. As discussed earlier, for today's applications where $m \geq 163$ it is expected to have a unique candidate pair. In this way, in phase 4 an exhaustive search is performed in order to obtain the full value of the scalar. Here, the exhaustive search space r is obtained in Steps 7.2 and 7.3. Thus, assuming $t \approx m$ the phase 4 of Algorithm 3.3 will require r scalar multiplications in the worst case which represents at most $(3mr)/2$ point operations if a binary method is utilized (e.g., Algorithm 2.1).

Example 3.4 Let us consider the cost of phases 2 and 4 of Algorithm 3.3 for $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curve K-163. Let us use the minimum and maximum values of c_i form Table 3.6 to give an interval for each cost. For a single processor, the cost of phase 2 is approximately in the interval $[6c_{\min}\sqrt{p_4}, 6c_{\max}\sqrt{p_4}] \approx [2^{49.9}, 2^{50.9}]$ point operations, where p_4 is the largest prime factor of $\#\widehat{E}(\mathbb{F}_{2^m})$ (see Table 3.2). Now, assume that we have $M = 10,000$ computers for solving the instances of the ECDLP. In this case the expected number of point operations for each processor is approximately in the interval $[\frac{c_{\min}(\sqrt{\pi p_4/163})}{10000}, \frac{c_{\max}(\sqrt{\pi p_4/163})}{10000}] \approx [2^{31.2}, 2^{32.2}]$. For the phase 4 cost, from Tables 3.1 and 3.5 we can notice that with a probability greater than $\frac{999}{1000}$ the exhaustive search space will be $r \leq 4$. Here the cost of phase 4 is negligible.

3.3 Countermeasures

The attacks presented in the previous section only need one or two faulty outputs to break the given instance of ECSM with a high probability of success. Hence, this may constitute a threat to cryptosystems using the Montgomery ladder ECSM for elliptic curves over the binary field. Therefore, some countermeasures are needed. In the following, we will describe possible protections against the attacks presented in this chapter.

Group formulas change. A possible countermeasure is to use alternative group formulas that include both elliptic curve parameters a and b. However, such formulas are likely to require more computations and hence cause a degradation in terms of performance. Additionally, if this approach is the only protection used, no errors due to faults are detected and this might constitute a risk for other attacks such as the DFA attack presented by Biehl et al. [9].

Curve selection. The attacks presented in this chapter assume that $\widehat{E}(\mathbb{F}_{2^m})$ is a cryptographically weaker group where the ECDLP could be solved in a reasonable period of time for a given $E(\mathbb{F}_{2^m})$. However, this assumption is not true if both $\#E(\mathbb{F}_{2^m})$ and $\#\widehat{E}(\mathbb{F}_{2^m})$ are almost prime. From the NIST-recommended curves, the only curve that satisfies this condition is referred to as K-283. Although, this curve selection criteria is an effective countermeasure against the fault-based attacks presented in this chapter, it might be too restrictive from the practical point of view. Moreover, the following two countermeasures represent a possible solution without limiting the use of particular group $E(\mathbb{F}_{2^m})$ even when the order of $\widehat{E}(\mathbb{F}_{2^m})$ is not an almost prime number.

Point verification (PV). It is important to verify that the input point is in $E(\mathbb{F}_{2^m})$. In the case that this checking could be bypassed, it is more important to verify whether or not the output is on the original elliptic curve. This countermeasure not only prevents from the attacks presented in this chapter, but also others such as those described by Biehl et al. [9], Ciet and Joye [21], and Antipa et al. [6]. It is important to note that this verification needs to be implemented in a secure environment. Otherwise the attacker might bypass this protection and carry out an invalid-curve attack such as one of those described earlier in this chapter.

Coherency check (CC). In addition to PV that could be applied to any ECSM algorithm, the Montgomery ladder ECSM algorithm permits us to have another way to detect errors in scalar multiplication using coherency check (CC). We can use the fact that the temporary pair (Q_0, Q_1) is of the form $(l \cdot P, (l+1)P)$ for some integer l at any value of i during the loop of Algorithms 2.5-2.7. Since the difference between Q_1 and Q_0 should be P at any iteration, one can check this during and after the ECSM operation. Note that if the attacker is able to modify the input point P in the way described in Algorithms 3.1 and 3.3, the operation $Q_1 - Q_0$ needs to be implemented using group formulas that include both curve parameters, a and b, or at least parameter a for avoiding that this checking operation is performed in $\widehat{E}(\mathbb{F}_{2^m})$. This approach for error detection will be presented in more detail in Chapter 5.

3.4 Conclusion

In this chapter we have presented two invalid-curve attacks that apply to the Montgomery ladder ECSM algorithms proposed by López and Dahab [58]. These attacks exploit the fact that parameter a is not used in the group formulas for these particular algorithms. In this way, if $\widehat{E}(\mathbb{F}_{2^m})$ is a weaker group with the same parameters than the original group $E(\mathbb{F}_{2^m})$ except for parameter a and we are able to inject a fault in the input point as described in Algorithms 3.1 and 3.3, then we would retrieve the scalar k with a high probability of success. For the purpose of the NIST-recommended curves, we have shown that there exists a weaker group for nine of the ten cases that include the randomly chosen and Koblitz curves. The only exception is the curve K-283 for which $\#E(\mathbb{F}_{2^m})$ and $\#\widehat{E}(\mathbb{F}_{2^m})$ are almost prime. Also, we have obtained the theoretical probability of success for each of the presented attacks. Additionally, we have determined numerical values of the probabilities of success for $\widehat{E}(\mathbb{F}_{2^m})$ from the NIST-recommended curves. And finally, we have presented some countermeasures to prevent the attack described in this chapter.

Chapter 4

Robust ECSM Using Repeated and Parallel Computations

In this chapter we present some structures that permit detection of errors in ECSM without modifying the curve parameters. These are based on re-computation and parallel computation. We use a number of encoding techniques that rely on the properties of elliptic curves and provide a high probability of detection of errors caused by faults that occurred naturally or injected deliberately by an attacker.

In addition, we consider fault-tolerant ECSM. While error detection is a sufficient countermeasure for preventing fault-based attacks, fault-tolerant characteristic enables a system to perform its normal operation in spite of faults. For certain fault models, we propose structures that can perform correct ECSM operations in the presence of faults that may occur in a limited number of ECSM modules, primarily due to natural causes such as abnormal temperature, electromagnetic interference, or power supply changes. An attacker who is not able to inject faults at

precise time and locations (i.e., "less sophisticated" attacker) is assumed to have an effect similar to that of natural causes. On the other hand, a "sophisticated" attacker is able to inject faults at precise locations in arbitrary number of ECSM modules. For provide resistance against a sophisticated attacker, the encoding techniques used with the inputs can essentially prevent such structures from outputting incorrect results due to faults. Most of the work presented in this chapter has been presented by Domínguez and Hasan [26] [27].

The organization of the remainder of this chapter is given as follows: In Section 4.1, we present encoding schemes for error detection for ECSM and probability of undetected errors. In Section 4.2, we give fault-tolerant ECSM structures. We present overhead costs and experimental results for the probability of undetected errors in Sections 4.3 and 4.4, respectively. Finally, we make some concluding remarks in Section 4.5.

4.1 Encoding/decoding and error detection for ECSM

In this section, we propose error-detecting structures for ECSM. Here, we consider a high-level design, where the ECSM module is the main block implemented in hardware to accelerate some ECC applications and may become faulty either by natural causes or by deliberate attacks from an adversary. Other modules used in our structures are much less complex[1] than the ECSM module and are assumed to be implemented in a secure environment – either in software or hardware.

[1]In Section 4.3, area and time complexities of these modules are given for $\mathbb{F}_{2^{163}}$.

For practicality of the above-mentioned assumption, one can consider a scenario where ECC is to be implemented in an embedded system that stores the secret key in a tamper-resistant memory and allows the key to be available to ECC operations via a protected bus. If the memory and the bus for the secret key are not secure enough to begin with, then these two would likely become an adversary's first points of attacks for easily extracting the key. The embedded system is likely to have a general-purpose processor or microcontroller, which avoids being tied up or attempts to improve system performance by off-loading the time consuming scalar multiplication operations to the ECSM module. The latter acts much like a co-processor and may have been acquired from a commercial vendor as an ASIC or even as an IP core for other implementation choices, such as field programmable gate arrays. In our work, the adversary is assumed to be able to inject faults only into the ECSM module, where the sensitive information (i.e., secret key) is utilized for the cryptographic computation.

The party responsible for the implementation of ECC may have no access to or lack proper knowledge of the internal circuit design of the ECSM module, and hence, deploys the module without any modification. The ECSM module does not have built-in encoder/decoder, comparator/voter or PV units. These can be implemented using dedicated hardware or the general purpose processor that the embedded system has in it. However, they require considerably less computation than a scalar multiplier does and, hence, can be implemented in a fault-tolerant manner, say by applying the triple modular redundancy (TMR) technique, without requiring excessive resources in terms of silicon area and/or computation time.

Throughout the rest of the document, we use the following assumptions:

- The input point P is verified by the cryptosystem which uses the ECSM module to be on the valid elliptic curve before each ECSM computation. This validation is especially important for preventing the attacks described in Chapter 3 and those proposed by Biehl et al. [9] and Antipa et al. [6].

- The order of a selected point P is a sufficiently large prime. This check guarantees that P is not in a small subgroup of $E(\mathbb{F}_q)$ of order dividing the cofactor h [40].

- An appropriate elliptic curve has been selected (e.g., using the guidelines of a recognized standard such as FIPS 186-2 [32]).

- The input and output of the ECSM are given in projective coordinates.

4.1.1 Encoding/decoding for ECSM

The encoding/decoding process in Figures 2.2 and 2.3 plays an important role in the detection of errors caused by faults in the compute (i.e., ECSM) module. For example, without the encoding/decoding the re-computation based scheme in Figure 2.2 would fail to detect errors produced in two cases: (i) the errors are produced by permanent errors and (ii) the same transient fault is present for both 'runs.' In both cases, the erroneous results at times t_0 and t_1 will be the same and the comparator would not detect such errors. Similarly, for the parallel computation based scheme in Figure 2.3, if the two ECSM modules have the same permanent and/or transient faults, in the absence of the encoding/decoding the two modules generate erroneous but the same results. Again, such errors are not detected by the comparator.

Below, we present encoding/decoding schemes suitable for ECSM operations. These schemes are primarily based on properties of elliptic curves. For a given input pair k and P, a 'good' encoding scheme will produce input representations that are different for the lower and the upper data paths (see Figures 2.2 and 2.3), and hence, faults – whether or not identical on the two data paths – are likely to affect the two ECSM operations differently. Consequently, the two values to be compared by the comparator are also likely to be different.

Encoding for input point P

Taking advantage of the simplicity of negating an EC point, it is possible to use the negative of a point as the encoded input. For a point P given in the affine coordinate system, it is well known that the point negation is simply

$$-P = -(x, y) = \begin{cases} (x, x + y) & x, y \in \mathbb{F}_{2^m}, \\ (x, -y) & x, y \in \mathbb{F}_p. \end{cases} \tag{4.1}$$

For projective coordinates, namely the López and Dahab system for curve E over $\mathbb{F}_{2^m}$ and the Jacobian system for E over $\mathbb{F}_p$, the point negation is

$$-P = -(X, Y, Z) = \begin{cases} (X, XZ + Y, Z) & X, Y, Z \in \mathbb{F}_{2^m}, \\ (X, -Y, Z) & X, Y, Z \in \mathbb{F}_p. \end{cases} \tag{4.2}$$

In Figure 2.2, if the input is point P, then the encoder performs the point negation in accordance with Equation (4.1) or (4.2). For the encoded input, the output of

the ECSM module is $k(-P) = -kP$. Hence, the decoder in Figure 2.2 also performs a point negation to generate the expected output kP.

Although, the encoding (and decoding) using point negation is simple – for projective coordinates, one multiplication and one addition for E over $\mathbb{F}_{2^m}$, and only one addition for E over $\mathbb{F}_p$ – it is important to note that such an encoding scheme changes only the y-coordinate of the input point, while the x- and, if applicable, the z-coordinate remain unchanged. As a result, even in the presence of some faults in the ECSM module, it is possible that the comparator in Figure 2.2 gets two equal but incorrect points at its input and generates an 'ok' signal. This is illustrated in Appendix B.

To change all the coordinate values of the input point, we can use a property of projective coordinate systems, which consists of having multiple representations for a given point. This principle is known as point randomization [23] and is applicable to all the projective coordinate systems [46]. For the López and Dahab and the Jacobian projective systems, Equations (2.3) and (2.4) become

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4 \tag{4.3}$$

and

$$Y^2 = X^3 + aXZ^4 + bZ^6, \tag{4.4}$$

respectively. It is easy to verify that trios $(\gamma X, \gamma^2 Y, \gamma Z)$, where $\gamma \in \mathbb{F}^*_{2^m}$, satisfy Equation (4.3) and have the same affine representation as that of (X, Y, Z). Thus, these trios can give different projective representations of a single point on the curve defined by Equation (2.3). Similarly trios $(\gamma^2 X, \gamma^3 Y, \gamma Z)$, where $\gamma \in \mathbb{F}^*_p$,

have different projective representations of a single point in the curve defined by Equation (2.4).

Based on the above discussion, a single projective representation (X, Y, Z) can be encoded, in some random way, to one of the $q-1$ possible projective representations $(\gamma^c X, \gamma^d Y, \gamma Z)$, where for the López and Dahab system, $c = 1$, $d = 2$, and $\gamma \in \mathbb{F}_{2^m}^*$; and for the Jacobian system $c = 2$, $d = 3$ and $\gamma \in \mathbb{F}_p^*$. Since (X, Y, Z) and $(\gamma^c X, \gamma^d Y, \gamma Z)$ correspond to the same point, the two ECSM operations, i.e., $k(X, Y, Z)$ and $k(\gamma^c X, \gamma^d Y, \gamma Z)$, result in the same point on the curve, i.e.,

$$k(X, Y, Z) \sim k(\gamma^c X, \gamma^d Y, \gamma Z). \tag{4.5}$$

One implication of (4.5) is that if the encoder in Figure 2.2 or 2.3 is only for the mapping

$$(X, Y, Z) \mapsto (\gamma^c X, \gamma^d Y, \gamma Z), \tag{4.6}$$

then the decoder is not needed.

Encoding for scalar k

It is well known that the order of any point P divides the order of the group $\#E(\mathbb{F}_q)$, i.e., $\#E(\mathbb{F}_q)P = \mathcal{O}$. Let $k' = \#E(\mathbb{F}_q) - k$. Then

$$k'P = (\#E(\mathbb{F}_q) - k)P \equiv -kP.$$

Thus, k can be simply encoded to k', which can be viewed as some kind of scalar negation operation, and the corresponding decoding process involves a point nega-

tion to convert $-kP$ to kP. However, this decoding process can be omitted if the corresponding input point P is also negated in the encoding process (i.e., $k'(-P) = kP$).

For the scalar k, more complex encoding schemes are possible. When k is fixed, in order to resist differential power analysis attacks, it has been suggested [23] to randomize k by the following mapping:

$$k \mapsto k'' = k + j\#E(\mathbb{F}_q), \tag{4.7}$$

where j is a random integer of at least 20 bits long[2]. Such randomization can also be used as an encoding scheme, since

$$k''P = (k + j\#E(\mathbb{F}_q))P \equiv kP.$$

Such an encoding requires an integer multiplication; however, no computations are needed for decoding. We assume that the bus of the scalar input is wide/flexible enough to carry the encoded scalar $k + j\#E(\mathbb{F}_q)$, where k is the secret key.

Among other possible encoding schemes, the binary unsigned representation of k can be converted into one of the many binary signed representations of k. This conversion/encoding can be done in a random way. For an m-bit integer k, there are $O(3^{\lfloor \frac{m}{2} \rfloor})$ different binary signed digit representations on average [29]. This however requires the ECSM module to support the double-and-add/sub algorithm and is not considered here.

[2]For today's applications, a 20-bit random integer j is recommended by Coron in order to resist the DPA attack [23].

4.1.2 Error-detecting structures and probability of undetected error

As stated before, for the purpose of encoding, one can map input pair k and $P = (X, Y, Z)$ to $k + j\#E(\mathbb{F}_q)$ and $(\gamma^c X, \gamma^d Y, \gamma Z)$, respectively. These encoding schemes produce different representations of inputs that are equivalent to the original input and do not require the decoding process. In addition, as shown in Section 4.4 with experimental results, these two encoding schemes lead to a lower probability of undetected errors when compared with other encoding schemes discussed earlier, namely $P \mapsto -P$ alone or combined with $k \mapsto k' = \#E(\mathbb{F}_q) - k$. As a result, in this work, we use these two encoding schemes for error detection in ECSM. In particular, for re-computation based error detection, at t_0 we compute the ECSM with (k, j_0, P, γ_0) as inputs, and then at t_1 compute another ECSM with (k, j_1, P, γ_1), where j_0 and j_1 are two random integers of appropriate length (say 20 bits for a 160 bits long k), and γ_0 and γ_1 are two random non-zero elements of the underlying finite field.

In the comparator unit, the outputs of the two ECSM operations that are in projective representation need to be compared. For this matter, below we apply an idea originally presented by Meloni [63] in a different context, namely to speed up ECC scalar multiplication. Assume that $Q_0 = (X_0, Y_0, Z_0)$ and $Q_1 = (X_1, Y_1, Z_1)$ are the two input points of the comparator. Then, similar to (4.5), we can transform each Q_0 and Q_1 with $\gamma = Z_1$ and Z_0, respectively, i.e.,

$$Q_0 \sim (X_0 Z_1^c, Y_0 Z_1^d, Z_0 Z_1),$$

$$Q_1 \sim (X_1 Z_0^c, Y_1 Z_0^d, Z_0 Z_1),$$

where for the López and Dahab system $c = 1$ and $d = 2$, and for the Jacobian system $c = 2$ and $d = 3$. If Q_0 and Q_1 map to the same affine representation, then the new X- and Y-coordinates of Q_0 and Q_1 must be equal since the new Z-coordinates are the same. Thus, the cost of comparison of these two points is four finite field multiplications and two squarings for the López and Dahab system, and six finite field multiplications and two squarings for the Jacobian system. In addition, two $\lceil \log_2 q \rceil$-bit comparators are needed. The main advantage of this comparison scheme is that an explicit transformation from projective to affine coordinates is not needed resulting in the elimination of expensive field inversion operations.

Finally, if the compared points are the same, one of them is produced as the final output. Otherwise, no final output is given. This scheme is shown in Figure 4.1, and we refer to it as full re-computation based scheme or RC for short.

Because of random γ_i and j_i for $i = 0$ and 1, one can assume that in the presence of faults – whether identical or not – the ECSM module's output Q_i is a random trio (X_i, Y_i, Z_i) of finite field elements. If the fault makes the ECSM module produce an incorrect result, then $Q_i \neq kP$. Since the number of elements in the finite field is q and Q_i has $q - 1$ different projective representations, the probability that $Q_0 = Q_1$ and $Q_i \neq kP$ (i.e., undetected error) can be expressed as follows:

$$\Pr(\text{undetected error})_{\text{RC}} \approx \frac{q-1}{q^3} \approx \frac{1}{q^2}. \tag{4.8}$$

For many of today's security applications , where $q \approx 2^{160}$, the above probability of undetected error is quite small. The counterpart of RC that uses parallel com-

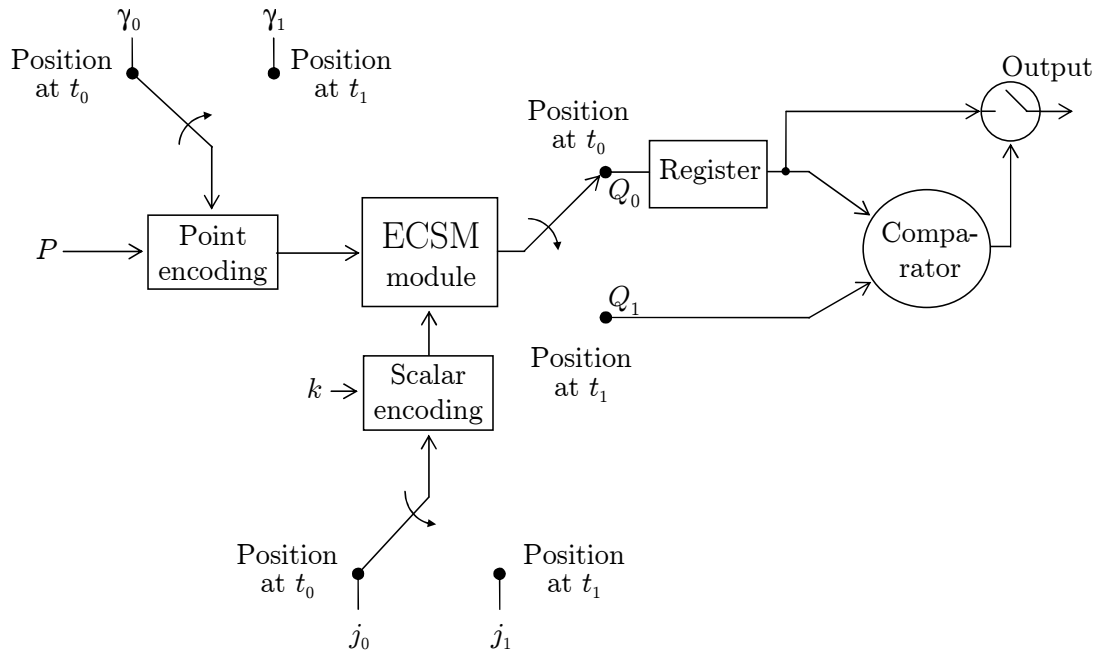

Figure 4.1: ECSM using full re-computation with point and scalar randomization (RC)

putation is shown in Figure 4.2. This parallel computation based scheme (or PC for short) can detect any error confined in one module. Additionally, if both ECSM modules have errors, since they use different input representations, the probability of having equal erroneous outputs from these modules is low as given in Equation (4.8).

4.1.3 Error detection using partial re-computation

The main penalty of using full re-computation (i.e., using RC) is that it doubles the running time of ECSM. In applications where the ECSM module is subject to only permanent faults injected by an attacker or caused naturally, the running time

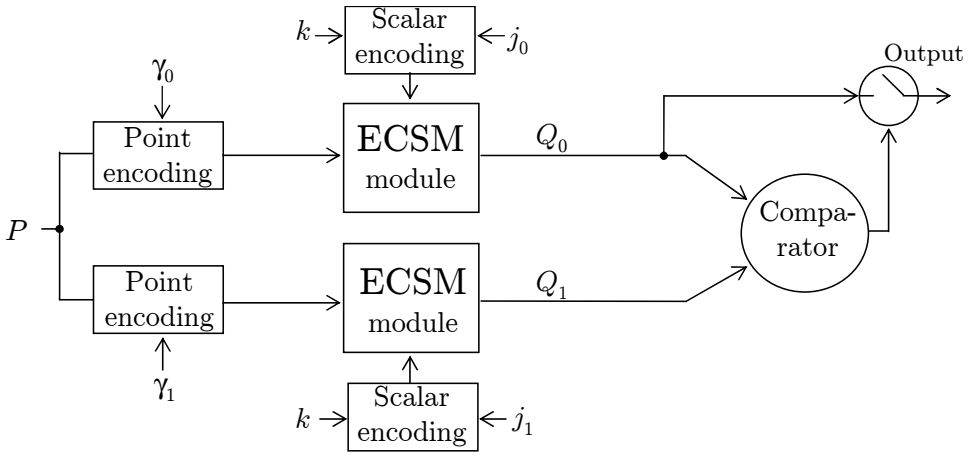

Figure 4.2: Parallel computation based ECSM with point and scalar randomization (PC)

can be reduced by performing a partial re-computation (RC_partial) as discussed below.

Under RC_partial, input point P is encoded twice using point randomization described in (4.6) so that it has different representations at t_0 and t_1. Input scalar k is encoded to k'' using (4.7) for ECSM at t_0, but no encoding is done for t_1 and only a few bits (say, least significant l bits, where $0 \leq l \leq m$) of the encoded k are used. Let these l bits correspond to integer $k_s = \sum_{i=0}^{l-1} 2^i k_i''$, where $k_i'' \in \{0, 1\}$.

Based on the above discussion, at time t_0, an ECSM is computed with (k, j_0, P, γ_0). Let us denote the output of this computation as Q_0. As part of the computation, assuming a left-to-right double-and-add algorithm, $k_s P$ is expected to be generated after l iterations of the algorithm. This value (i.e., point $k_s P$) or an erroneous version of it – in case there are errors due to faults – is saved for a later comparison. Let us denote the saved value as $Q_{0,\text{ partial}}$.

At time t_1, an ECSM is computed with $(k_s, 1, P, \gamma_1)$. Since k_s is l bits long, this

computation requires $l-1$ point doublings and $H(k_s)-1$ point additions, where $H(k_s)$ denotes the Hamming weight of k_s. Let us define the output of this ECSM as Q_1. If the affine representations of Q_1 and $Q_{0,\text{ partial}}$ are the same, then Q_0 computed at t_0 is produced as the final result.

The probability of undetected errors in the RC_partial scheme depends on the value of l. If l takes the maximum value i.e., $l = m$, then RC_partial is the same as RC discussed earlier and doubles the running time, but the probability of undetected errors is very low as given in Equation (4.8). On the other hand, if l is minimum i.e., 0, then there is no re-computation at all and no errors will be detected unless other techniques such as PV are used. For practical purposes, l may be taken as a small fraction, say 10%, of the number of bits in k and still achieve a low probability of undetected error since the point encoding technique described in (4.6) produces two random projective representations of the point at t_0 and t_1.

4.1.4 Error detection under faults injected by a sophisticated attacker

In this subsection, we discuss scenarios where a sophisticated adversary could mount an attack using the principles of today's best known fault attacks for ECC. Here, two attacks are considered. First, like the scenario given by Biehl et al. [9], the attacker is able to flip a single bit in a register that holds an intermediate point during the ECSM operation. Second, the attacker exploits the possibility of changing the sign of an intermediate point in order to mount the SCF attack as described by Blömer et al. [14]. For details on these attacks, the reader is referred to [9] and [14].

1) ECC differential fault attack: Suppose that the attacker can inject a fault at a random state in a register that holds partial results during the ECSM operation as described by Biehl et al. [9]. In such a case, the point operations result in normal additions before the fault and *pseudo-additions* [9] after the fault. The latter, in turn, represents an operation that leaves the original group structure. Thus, after the fault a random finite field trio is obtained at the ECSM output and the PV process can detect such errors with a probability of

$$\Pr(Q_i \in E(\mathbb{F}_q)) = \frac{(\#E(\mathbb{F}_q) - 1)(q-1) + 1}{\text{Number of finite field element trios}} \approx \frac{q^2}{q^3} = \frac{1}{q}. \tag{4.9}$$

Therefore, for large q, PV after the ECSM (Figure 2.1) might represent a countermeasure against this attack. However, that is not the case for all ECC fault attacks as shown in the following scenario.

2) SCF attack: Consider a cryptosystem based on an elliptic curve over prime field where a sign change in a point implies only a change in the Y-coordinate. Assume that the attacker is able to change this sign in an intermediate point during the ECSM operation as described by Blömer et al. [14]. In such a case, the erroneous results, which are valid points on the original elliptic curve, would be undetectable by the PV process. Hence, the probability of undetected error under this attack for the scheme of Figure 2.1 would be equal to unity. Now, let us consider the RC scheme under this attack where the adversary would need to inject an SCF into both ECSM runs. Here, the original assumption that a random finite field trio is generated after the fault is bypassed. The reason is that the output is now restricted only to the set of points on the curve. Due to random input point and

scalar, if we consider that the output is now a random point on the curve, then the probability of undetected error is about $1/q$. The same probability applies for the PC scheme if the attacker is able to inject an SCF to each ECSM module. Both schemes, RC and PC, can be utilized as a countermeasure to the SCF attack. The extra cost for these schemes in terms of time and area will be presented in Section 4.3.

4.2 Fault-tolerant structures for ECSM

From the cryptographic point of view, while error detection is a sufficient countermeasure for preventing fault-based attacks, fault-tolerant characteristic enables a system to perform its normal operation in spite of faults. This will result in more reliable systems where faults may occur due to deliberate attacks or due to natural causes such as abnormal temperature, electromagnetic interference, or power supply changes.

In this section, we present methods for fault tolerance for ECSM. Here, we assume static redundancy only. This means that the system can tolerate faults using masking; in fact, such faults are bypassed without any reparability or reconfiguration capabilities. First, a classical example of hardware redundancy, triple modular redundancy (TMR), is considered for ECSM. Then, by taking advantage of the simplicity of PV operation, we present a double modular redundancy (DMR) based fault-tolerant scheme, namely DMR_PV. Finally, by combining parallel computation with re-computation, we present another ECSM fault-tolerant structure that is as robust as TMR, and achieves an area efficiency close to DMR_PV. For each of the fault-tolerant schemes, we also give its reliability, which is defined as

its characteristic expressed by the probability that it will perform its function [85]. Finally, we provide a reliability comparison among these schemes.

4.2.1 TMR based fault-tolerant ECSM

Traditional TMR utilizes three elements performing the same operation [87]. In the context of the work presented here, these elements would correspond to ECSM modules as shown in Figure 4.3. In this TMR scheme, as long as two or all three ECSM modules yield correct results, the majority voter produces a correct final output. When two or more ECSM modules become faulty by natural causes or by some simple malicious action by a less sophisticated attacker, then the faulty modules are likely to produce different but incorrect results, and the majority voter will produce no final output. However, when a sophisticated attacker can deliberately inject faults that may cause two or more ECSM modules to generate the same but incorrect result, then the majority voter will produce an erroneous output and the TMR scheme will fail.

In an attempt to reduce the chance of TMR producing an erroneous result, we proceed as follows: Scalar k and point P, which are inputs to the ECSM modules, are encoded using the randomization techniques discussed in Section 4.1, as illustrated in Figure 4.4. The ECSM outputs are connected to a secure majority voter implemented in either software or hardware.

Like the comparator unit, the majority voter needs to process their inputs in projective coordinates. Assume that $Q_0 = (X_0, Y_0, Z_0)$, $Q_1 = (X_1, Y_1, Z_1)$, and $Q_2 = (X_2, Y_2, Z_2)$ are the three input points of the majority voter. Then, similar to (4.5), we can transform each Q_0, Q_1, and Q_2 with $\gamma = Z_1Z_2$, Z_0Z_2, and Z_0Z_1,

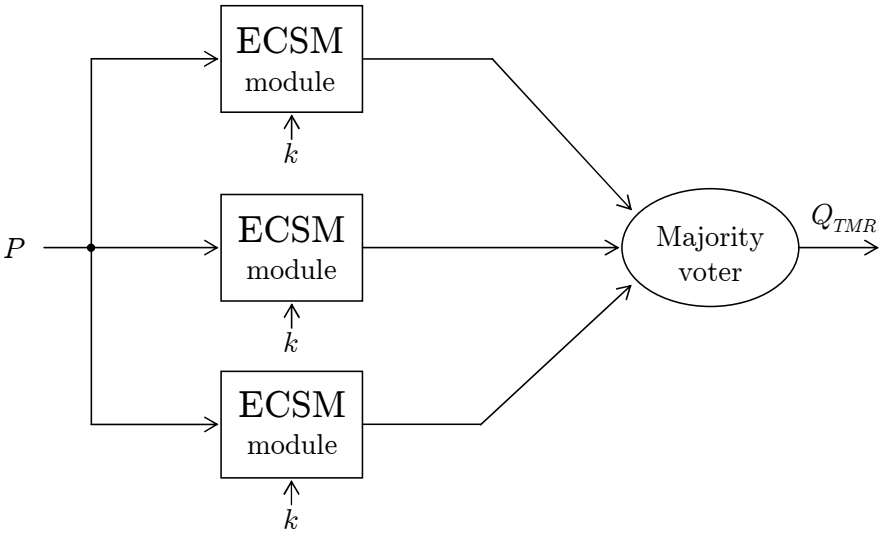

Figure 4.3: Traditional TMR based ECSM

respectively, i.e.,

$$Q_0 \sim (X_0(Z_1Z_2)^c, Y_0(Z_1Z_2)^d, Z_0Z_1Z_2),$$
$$Q_1 \sim (X_1(Z_0Z_2)^c, Y_1(Z_0Z_2)^d, Z_0Z_1Z_2),$$
$$Q_2 \sim (X_2(Z_0Z_1)^c, Y_2(Z_0Z_1)^d, Z_0Z_1Z_2),$$

where c and d are as defined earlier. With this new transformed points, a normal voting process can be performed that will produce a final result Q_{TMR}, which is the majority of these points. If there is no majority, then no final output is generated. Clearly, if errors occur in only one of the ECSM modules, then their effects can be masked and a correct final output can be obtained.

Assume that an attacker can inject the same fault, transient or permanent, in two or all three modules in an attempt to make the faulty modules produce the same but incorrect result. In such circumstances, the point and scalar encodings

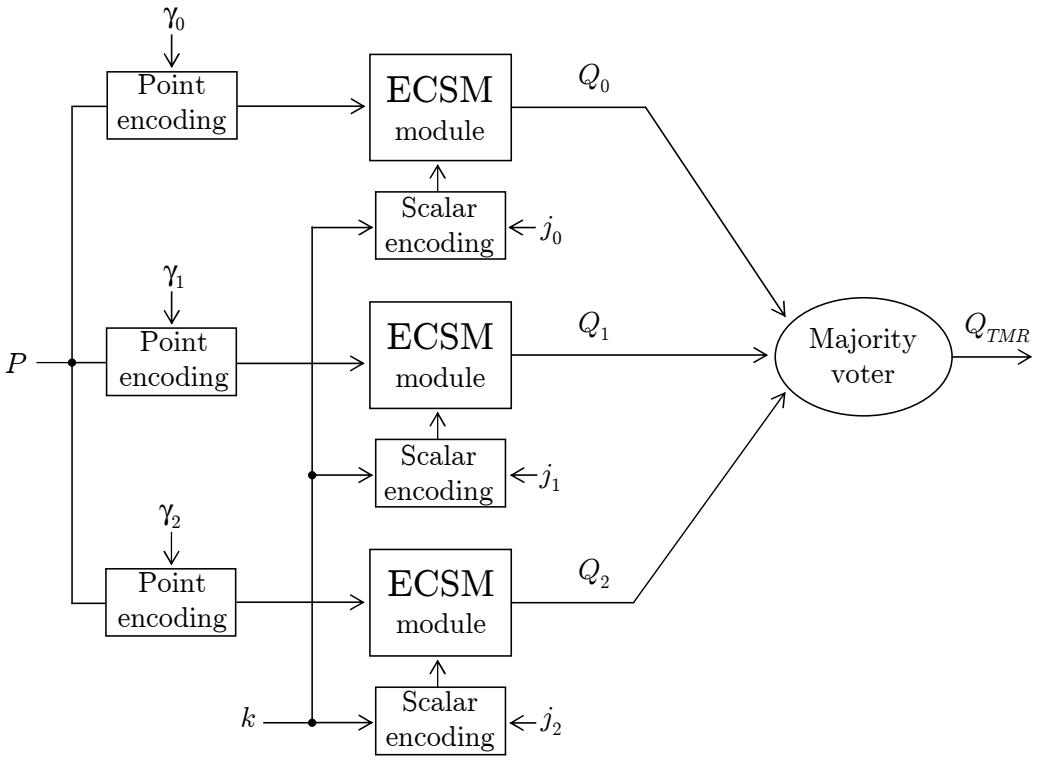

Figure 4.4: TMR based ECSM

help to keep the risk of giving an incorrect result to a low level. This is explained as follows: Because of the encoding of inputs, in the presence of faults, each ECSM module is expected to produce a random projective representation, which maps to one of the q^2 affine coordinates representations. An erroneous final result is produced by the majority voter if two or all three outputs represent the same but incorrect points. Thus, the probability that the final result has an error is

$$\begin{aligned}\Pr(Q_{TMR} \neq kP) &\approx \binom{3}{2} \times \frac{q^2(q^2-1)}{q^2 \times q^2 \times q^2} + \frac{q^2}{q^2 \times q^2 \times q^2} \\ &= \frac{3}{q^2} + \frac{1}{q^4} \approx \frac{3}{q^2} \quad \text{(for large } q\text{)}. \end{aligned} \tag{4.10}$$

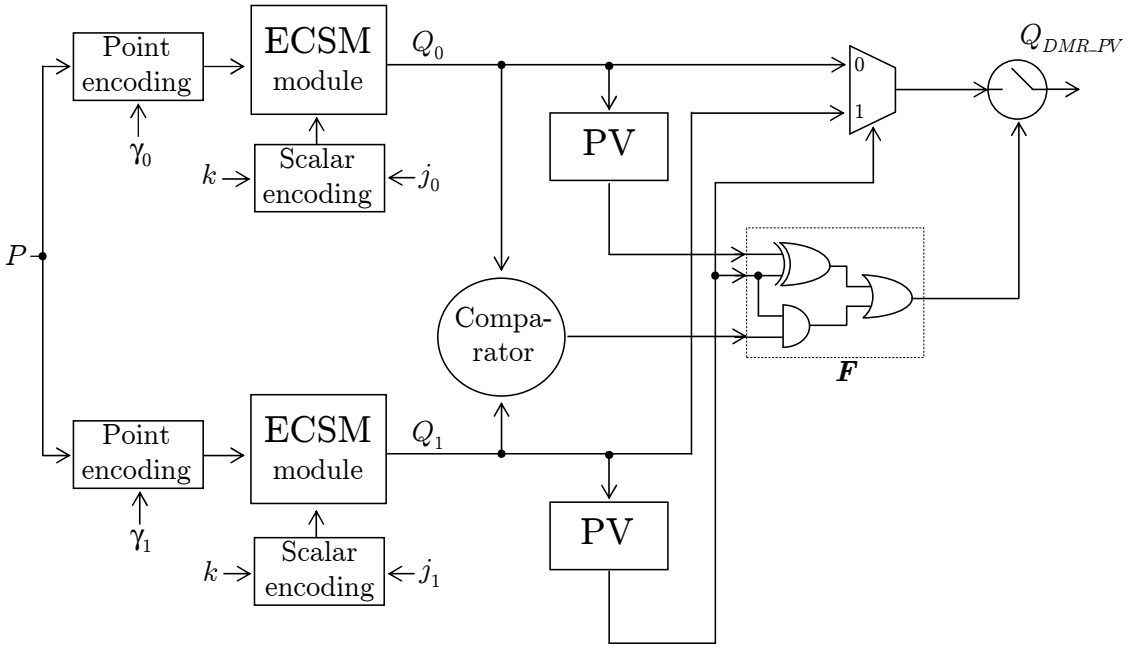

Figure 4.5: DMR_PV fault-tolerant ECSM

We note that TMR can be viewed as a special case of an N-modular redundant system with N modules, where faults confined in up to $\lfloor \frac{N-1}{2} \rfloor$ modules can be masked.

4.2.2 DMR_PV fault-tolerant ECSM

In applications where it is important to reduce silicon area, perhaps at the expense of increases in the probability of an incorrect result, instead of using the TMR based scheme discussed earlier, one can use the dual modular redundant (DMR) system combined with PV, namely DMR_PV, as shown in Figure 4.5. As it can be clearly seen, compared to TMR, DMR_PV uses one less ECSM module. An important aspect is that this scheme is only appropriate for elliptic curves over the

binary finite field $\mathbb{F}_{2^m}$. For elliptic curves over the prime field $\mathbb{F}_p$, the PV module is not sufficient for detecting errors produced by the SCF attack. However, for the case of elliptic curves over $\mathbb{F}_{2^m}$, where the SCF attack does not apply, we can use the PV module to validate the ECSM operations.

As shown in Figure 4.5, the two ECSM modules operate in parallel and their outputs are verified by PV modules. The F block is used to stop the system output if $Q_0, Q_1 \notin E(\mathbb{F}_{2^m})$; or when $Q_0 \neq Q_1$, and $Q_0, Q_1 \in E(\mathbb{F}_{2^m})$. The final output of the DMR_PV based system is given as follows:

$$Q_{DMR\_PV} = \begin{cases} Q_1 \text{ (or } Q_0) & \text{if PV}(Q_0 \text{ or } Q_1) = \text{ok and } Q_0 = Q_1, \\ Q_i & \text{if } Q_i \in E(\mathbb{F}_{2^m}) \text{ and } Q_{\bar{i}} \notin E(\mathbb{F}_{2^m}), \\ \text{no output} & \text{otherwise,} \end{cases}$$

where $\bar{i} = 1 - i$.

The DMR_PV scheme can clearly tolerate any faults confined in only one of the two modules and can produce the correct final output. Additionally, the scheme can detect some situations where errors exist in the outputs of both modules and hence helps to avoid producing erroneous results at the final output. However, there are two cases of reasonably low probability when this scheme fails and gives an incorrect result: first, if $Q_0 = Q_1$ and $Q_0, Q_1 \in E(\mathbb{F}_{2^m}) \setminus kP$; secondly, when $Q_0 \neq Q_1$, $Q_i \in E(\mathbb{F}_{2^m}) \setminus kP$, and $Q_{\bar{i}} \notin E(\mathbb{F}_{2^m})$ for $i = 0$ or 1. Then, the probability of giving an incorrect result when both ECSM modules are in error is

$$\Pr(Q_{DMR\_PV} \neq kP) \approx \frac{1}{q^3} + 2\left(1 - \frac{1}{q^2}\right)\left(\frac{1}{q}\right)\left(1 - \frac{1}{q}\right) \approx \frac{2}{q} \quad \text{(for large } q\text{)}. \tag{4.11}$$

4.2.3 Parallel and re-computation based fault-tolerant ECSM

The above DMR_PV scheme reduces the silicon area requirement but increases the probability of an incorrect result. It is possible to have a fault-tolerant ECSM system that is as area efficient as DMR_PV and has the same low probability of incorrect result as TMR. To this end, one can use a parallel and re-computation[3] (PRC) based scheme shown in Figure 4.6. In PRC, at time t_0, with input $(k, j_{0,t_0}, P, \gamma_{0,t_0})$ and $(k, j_{1,t_0}, P, \gamma_{1,t_0})$ the two ECSM modules produce Q_{0,t_0} and Q_{1,t_0}, respectively. These two points are then compared using the technique on page 94. If the points are the same, then one of them is produced as the final output Q_{PRC}. Otherwise, the ECSM modules perform re-computations with $(k, j_{0,t_1}, P, \gamma_{0,t_1})$ and $(k, j_{1,t_1}, P, \gamma_{1,t_1})$ and produce Q_{0,t_1} and Q_{1,t_1}. If errors are confined in only one of the ECSM modules, then for only one of the two values of i, i.e., either $i = 0$ or $i = 1$, Q_{i,t_0} and Q_{i,t_1} are the same after their Z-coordinates are made equal and one of them can be produced as the final output Q_{PRC}.

An erroneous final Q_{PRC} may be produced in the following two cases: (i) the ECSM modules produce the same but incorrect result at t_0, and (ii) any of the two ECSM modules gives an incorrect but same result at both t_0 and t_1. For the PRC operation described above, the probability that an erroneous final result is produced is

$$\Pr(Q_{PRC} \neq kP) \approx \frac{1}{q^2} + 2\left(1 - \frac{1}{q^2}\right)\left(\frac{1}{q^2}\right) \approx \frac{3}{q^2} \quad \text{(for large } q\text{)} \tag{4.12}$$

[3]The combination of time and hardware redundancy for fault-tolerant system design has been considered in other contexts, e.g., Lima et al. [55] have used this combination for having fault-tolerance on FPGAs.

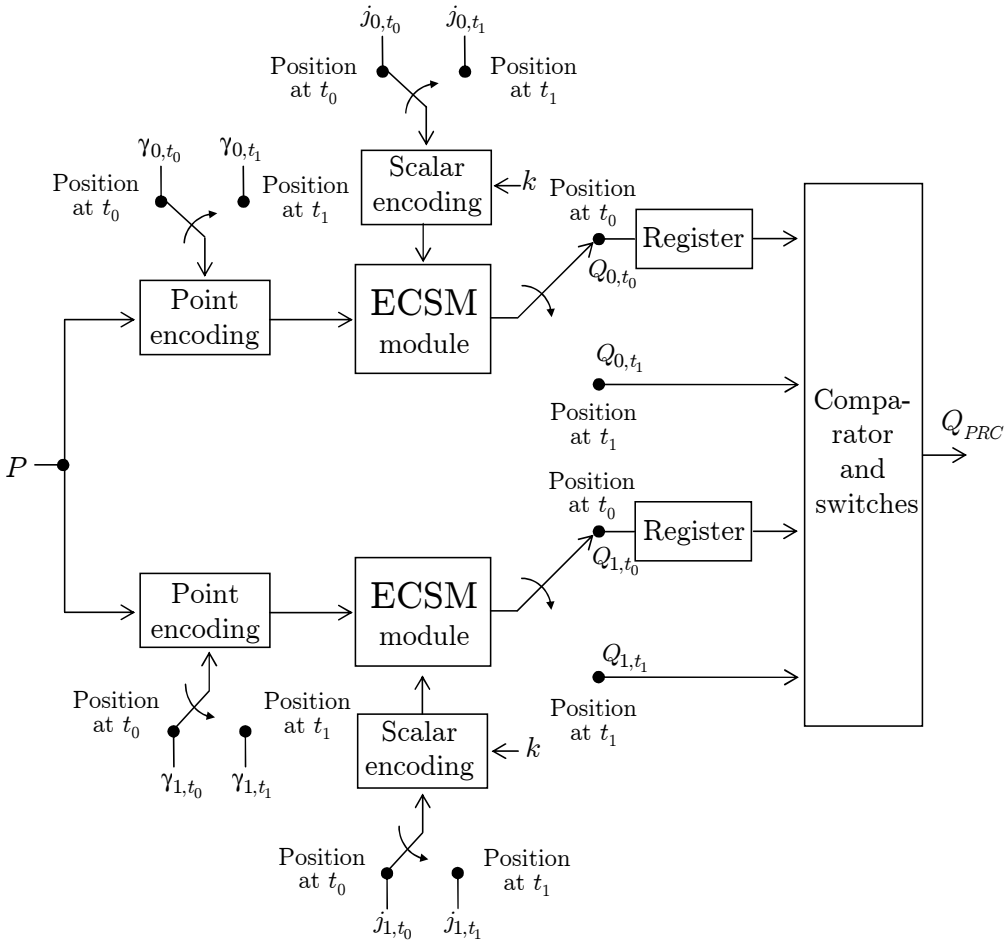

Figure 4.6: Parallel and re-computation (PRC) based fault-tolerant ECSM

which matches (4.10) for TMR.

We note that the re-computation in the PRC based scheme is performed when one or both ECSM modules generate incorrect results and hence doubles the running time under such situations. If both ECSM modules are properly working, then the running time of PRC is the same as that of TMR and DMR_PV.

4.2.4 Effect on reliability

The reliability comparison between several fault-tolerant systems has been considered in the literature [83] [54]. In the context of this work, note that compared to the ECSM module, each of the other modules in Figures 4.4, 4.5, and 4.6, namely the majority voter, the point and the scalar randomization modules, PV modules, comparators, registers, and multiplexors would require a lot less hardware in practice. Thus, these small components can be implemented such that each of those will have a reliability factor that is close to unity. Then, for the case of TMR based ECSM, i.e., a correct result is obtained when at least two out of three ECSM modules are performing their functions, the reliability is

$$R_{TMR} \approx r_{ECSM}^3 + 3r_{ECSM}^2(1 - r_{ECSM}) = 3r_{ECSM}^2 - 2r_{ECSM}^3, \tag{4.13}$$

where r_{ECSM} is the reliability factor of a single ECSM module. On the other hand, for both DMR_PV and PRC based ECSM schemes their reliability $R_{DMR\_PV/PRC}$ is related to the probability that at least one of the two ECSM modules work without errors. The resulting expression for $R_{DMR\_PV/PRC}$ is:

$$R_{DMR\_PV/PRC} \approx r_{ECSM}^2 + 2r_{ECSM}(1 - r_{ECSM}) = 2r_{ECSM} - r_{ECSM}^2. \tag{4.14}$$

If we subtract Equation (4.13) from (4.14), we obtain

$$\begin{aligned} & R_{DMR\_PV/PRC} - R_{TMR} \\ = \; & 2r_{ECSM} - r_{ECSM}^2 - (3r_{ECSM}^2 - 2r_{ECSM}^3) \\ = \; & 2r_{ECSM}(1 - r_{ECSM})^2. \end{aligned}$$

Since $0 < r_{ECSM} < 1$, $R_{DMR\_PV/PRC} - R_{TMR}$ is always positive, i.e., we can state that the reliabilities of DMR_PV and PRC based schemes are greater than that of the TMR based system (see Figure 4.7). The improved reliability of DMR_PV and PRC schemes is primarily due to their ability to mask out errors confined in one module with fewer number of ECSM modules compared to the TMR. The use of the reduced number of ECSM modules is possible due to the PV modules in DMR_PV and the re-computation in PRC.

4.3 Overhead cost

The error-detecting and fault-tolerant ECSM structures presented in Sections 4.1 and 4.2 require a number of extra modules, namely PV module, point randomization module, scalar randomization module, register, comparator, majority voter, and multiplexor. In this section, we give costs of these components based on hardware implementation using FPGAs. We also give the time impact on the ECSM operation, in terms of number of clock cycles, due to the inclusion of these extra components. We also note that these components must be implemented in a secure environment so that they are not vulnerable against faults.

For ECSM operation, we use the performance results given by Lutz [59], where

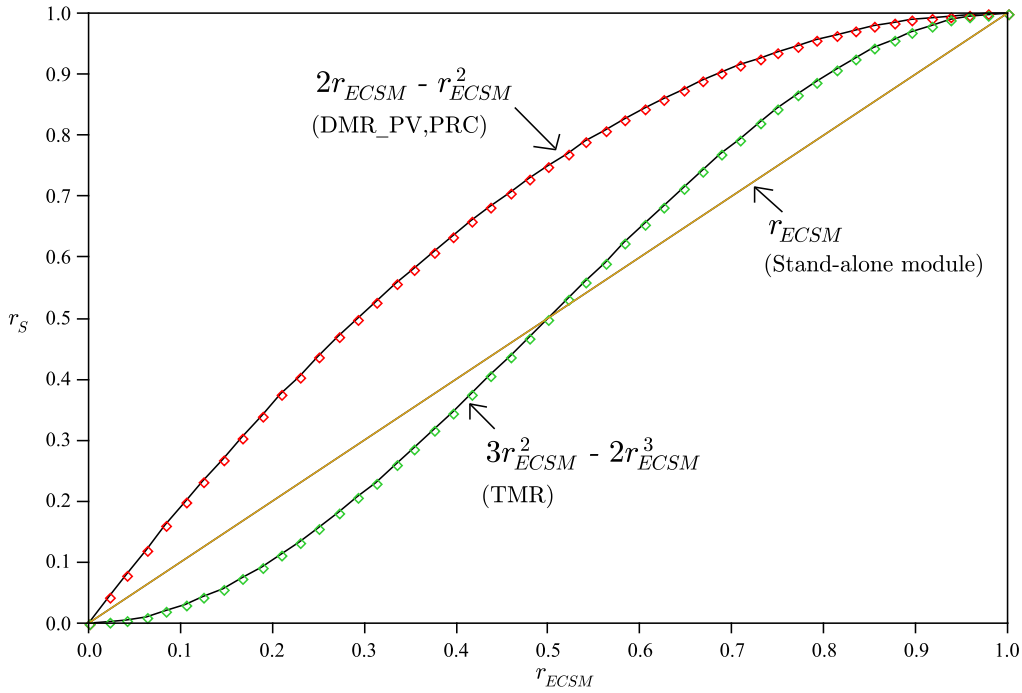

Figure 4.7: Reliability comparison among TMR, DMR_PV and PRC schemes

a NIST-recommended elliptic curve over $\mathbb{F}_{2^{163}}$ has been used. The performance results are based on a Xilinx Virtex 2000E FPGA implementation, and the ECSM operation uses a finite field multiplier, a squaring unit, and an adder. The area and timing results of these arithmetic units are:

Multiplier:	2364	slices and	4 cycles
Squaring unit:	165	slices and	1 cycle
Adder:	94	slices and	0 cycles.

The entire module, which performs ECSM, with input and output points in projective coordinates, requires 5009 slices and 17160 clock cycles for each scalar multiplication.

For implementing the extra modules (i.e., PV module, point and scalar random-

Element	Slices	Cycles
ECSM module	5009	17160
PV module	760	824
Point randomization module	528	653
Scalar randomization module[1]	815	18
Register	245	1
Comparator	549	656
Majority voter	824	1476
Multiplexor	245	0

[1]Size of $j = 20$ bits.

Table 4.1: Cost and performance for the ECSM and other extra modules used for error-detecting and fault-tolerant structures

ization modules, register, comparator, majority voter, and multiplexor), we have used the same FPGA to be consistent with the ECSM module implemented by Lutz [59]. For some modules, namely PV, point randomization, comparator, and majority voter, in order to optimize the area requirement we have used a low speed multiplier which occupies 286 slices and 163 cycles for each finite field multiplication. The results of performance and cost for these extra modules are given in Table 4.1.

Incorporating the extra modules with the ECSM modules as shown in the error-detecting structures of Section 4.1, we obtain the time and space complexity results given in Table 4.2. The corresponding results for the fault-tolerant structures of Section 4.2 are shown in Table 4.3.

The clock cycle data in Tables 4.2 and 4.3 correspond to the time needed to produce a final ECSM output given input pair k and P. In applications where many such ECSM operations are to be performed, a more important measure of

ECSM with	Figure No.	Slices	Latency (Cycles)	Throughput ratios[1]
no error detection	NA	5009	17160	1
PV	2.1	5769	17984	1
RC	4.1	7146	35795	$\frac{1}{2}$
RC_partial[2]	NA	7146	20319	$\frac{1}{1+l/m}$
PC	4.2	11910	19452	1

[1]With respect to the ECSM with no error detection.

[2]Size of $l = 16$ bits.

Table 4.2: Performance and cost for error-checking systems over $\mathbb{F}_{2^{163}}$

Scheme	Figure No.	Slices	Latency (Cycles)	Throughput ratios[1]
TMR based fault-tolerant ECSM	4.4	17194	21090	1
DMR_PV ECSM	4.5	12916	19452	1
PRC ECSM	4.6	13136	19452[2]	1^2 or $\frac{1}{2}$

[1]With respect to the ECSM with no error detection.

[2]If no error is detected at t_0, otherwise the running time is doubled/throughput is halved.

Table 4.3: Performance and cost for fault-tolerant systems over $\mathbb{F}_{2^{163}}$

performance is the throughput, which can be defined as the number of ECSM outputs per unit of time. In the rightmost column of Table 4.2, we give ratios of the throughput with no error detection to that with various error detection schemes, such as PC, RC, and RC_partial. Similarly, in Table 4.3, we give such ratios for fault-tolerant schemes. In determining the ratios, we assume that the operations of an ECSM module and comparator/PV can be overlapped. The ratio results show a number of trade-offs one can consider between area and throughputs.

For the FPGA implementation described here, the clock frequency is 66 MHz.

At this speed, the ECSM with the PV module, as shown in the second row of Table 4.2, will require $17984/(66 \times 10^6) \approx 272.48$ μsec. On the other hand, if all modules are implemented in software, then using the results reported by Hankerson et al. [39] for finite field arithmetic units, an ECSM unit with the PV module will require approximately 5.38 msec. These results may vary depending on the target platform and the level of optimization used; nevertheless, they show the speed advantage of using hardware over software.

4.4 Experimental results for undetected errors with a small prototype

As discussed earlier, when an error occurs due to a fault in the ECSM module, it is possible that the error remains undetected by the schemes presented in Section 4.1. Whether or not a fault causes an undetected error depends on a number of factors including the input pair k and P, the implementation, and the fault itself. The latter, in turn, could be characterized by some parameters that include location, type (e.g., stuck-at, bit flip, or bit set or reset), number of bits affected (e.g., single or multiple), and duration (e.g., permanent or transient). In this section, we present experimental results of undetected error probabilities based on an *exhaustive* generation of faults for a small prototype ECSM module. The prototype is modeled using VHDL with a Xilinx Spartan 3 1000 FPGA as the hardware target.

An exhaustive generation of faults at finite field level is chosen in order to obtain average numbers that do not depend only on a specific fault location. For the experiments, we also included all the possible input pairs (k, P) to avoid the de-

pendability of the results for a specific input pair. Clearly, these exhaustive fashion experiments limit the maximum feasible finite field size that the system can handle. However, the main factor for this limitation is the number of ECSM operations needed for finding errors in structures where the probabilities of undetected errors are expected to be very low (e.g., $1/q$, $1/q^2$).

In order to have experiments with the above description in a reasonable amount of time, we have selected the prototype to be based on the finite field $\mathbb{F}_{2^{11}}$. For this small prototype, the simplest experiment that involves only the PV module, the number of ECSM operations is approximately 0.5 billion[4] requiring roughly 2.5 hours to complete in the actual hardware. Other experiments involve much more operations, for example in each parallel computing scheme the number of ECSM operations is approximately 35 billions[5] and about seven days to complete. Although this ECSM prototype is small for a real application, it allows us to perform experiments in an exhaustive way, which in turn permits us to illustrate the error detection coverage for the schemes presented earlier.

4.4.1 Parameters, fault model, and process

For efficient implementation, the irreducible binary trinomial $z^{11} + z^2 + 1$ of degree 11 has been selected to construct the finite field $\mathbb{F}_{2^{11}}$. The elliptic curve used is E: $y^2 + xy = x^3 + x^2 + 1$, which has an order of $1982 = 2 \times 991$, where 991 is prime. The double-and-add algorithm has been used for obtaining the ECSM with the López and Dahab projective coordinates. For finite field operations, the methods illustrated in Table 4.4 have been utilized. In total, 134 gates have been used for

[4] 1980(input points)×134(fault locations)×989(input scalars)×2(fault types).
[5] 1980(input points)×134^2(fault locations)×989(input scalars) for each fault type.

implementing the finite field operations (117, 6 and 11 for finite field multiplication, squaring and addition, respectively) over $\mathbb{F}_{2^{11}}$. The fault model used is a permanent stuck-at-0 or stuck-at-1 fault in these gates, and only one gate can be faulty at a time.

Faults have been injected in a similar way as described by Zarandi et al. [95]. The idea is to add a multiplexor to each gate to be tested, such that we can select if the gate will work normally or with a stuck-at-0 or stuck-at-1 fault at its output (see Figure 4.8). Presence or absence of a fault is controlled by the *fault injection signal* (FIS). The *fault selector signal* (FSS) chooses a stuck-at-0 or stuck-at-1 fault for the experiment.

Finite field operation	**Method**	**Number of gates**
Multiplication	Look-up table-based group level Multiplication [41] [1]	117 [2]
Squaring	Modified look-up table-based group level Multiplication [59] [1]	6 [3]
Inversion	Itoh & Tsujii [44]. With 4 multiplications and 10 squarings.	0
Addition	Bit-wise XOR	11 [3]

[1]Group size = 6. [2]81 2-input AND gates, 20 2-input XOR gates, 11 5-input XOR gates, and 5 2-input OR gates. [3]2-input XOR gates.

Table 4.4: Methods utilized for finite field operations over $\mathbb{F}_{2^{11}}$

The elliptic curve utilized has 1980 points on it that are of order either 1982 or 991. In our experiments, these 1980 points[6] are input for each value of scalar k in the range of 2 to 990. For each fixed set of input point and scalar, a fault-free

[6]The remaining two points include the point at infinite and `(0x00,0x01)`. The latter has the order of two. Both of these points are not considered suitable for cryptographic applications.

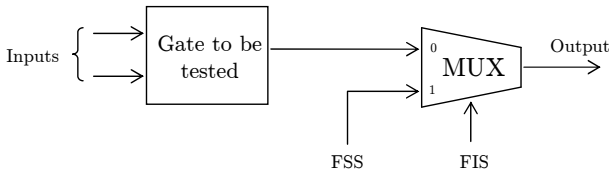

Figure 4.8: Stuck-at-0 or 1 fault injection method

computation is performed to obtain kP. Then a stuck-at-0 or stuck-at-1 fault is injected for each individual gate used for the finite field operations and the ECSM operation is repeated. The result is then compared with the fault-free case to determine if the fault has produced an erroneous result.

For the case of PV, the experiment consists of obtaining, after a single fault (stuck-at-0 or 1), the probability that an erroneous result is in $E(\mathbb{F}_{2^{11}})$. For the re-computation based schemes, after the injected fault, two scalar multiplications with their respective encoding process are performed (at t_0 and at t_1) and then the results are compared. Additionally, in order to show the importance of the choice of the encoding/decoding processes, to each re-computation scheme presented in Section 4.1 (i.e., RC and RC_partial), we include other types of encoding/decoding processes in the experiment.

Similar to the re-computation case, an experiment is performed for the PC scheme presented in Section 4.1. In this scheme, there are two ECSM modules. In one set of experiments, we inject faults in only one of the two modules at a time. In another set of experiments we consider the case where each of the two ECSM modules has a single fault at the same time (at the same or different location).

4.4.2 Results obtained

Encoding schemes

As we stated in Section 4.1, the encoding/decoding process plays an important role for error detection capability for schemes shown in Figures 2.2 and 2.3. To see their relative effectiveness, we ran experiments for different encoding techniques for ECSM re-computation based schemes. Table 4.5 shows their probabilities of undetected errors. The results of the table imply that the combination of the encoding schemes (4.6) and (4.7) i.e., $(k, P) \mapsto (k'', P')$ yields the least probability of undetected errors, and this is why we have earlier presented our theoretical results using this encoding technique. In the following discussions, we present experimental results using this encoding for the proposed error detection schemes.

Encoding	**Pr(undetected error)**	
	Stuck-at-0 fault	**Stuck-at-1 fault**
$(k, -P)$ [1]	$\frac{1}{117}$	$\frac{1}{12{,}692}$
$(k', -P)$ [2]	$\frac{1}{128}$	$\frac{1}{1{,}395{,}753}$
(k, P') [3]	$\frac{1}{421{,}190}$	$\frac{1}{2{,}029{,}025}$
(k'', P') [3, 4]	$\frac{1}{1{,}097{,}914}$	$\frac{1}{7{,}497{,}185}$

[1] Point negation is needed for decoding. [2] $k' = \#E(\mathbb{F}_{2^{11}}) - k$.
[3] $P' = (\gamma X, \gamma^2 Y, \gamma Z)$, $\gamma \in \mathbb{F}^*_{2^m}$. [4] $k'' = k + j\#E(\mathbb{F}_{2^{11}})$.

Table 4.5: Probabilities of undetected errors for re-computation based schemes using different encoding in the experiment over $\mathbb{F}_{2^{11}}$

Error detection schemes

For each error-detecting scheme presented in Section 4.1, the probability of undetected error was obtained. Table 4.6 shows these probabilities for the stuck-at-0 and stuck-at-1 fault models. The PV alone gives the worst error detection coverage for these schemes (i.e., $\frac{1}{217}$ for the stuck-at-0 fault model).

Our experiment for scheme RC shows values that are close of the value from Equation (4.8) if an average of the stuck-at-0 and stuck-at-1 results is considered (i.e., $q^2 = 4,194,304$ vs. 4,297,550). For the PC scheme, if the faults are confined in only one of the two ECSM modules, then there are no undetected errors as shown in the fourth row of Table 4.6. The results for the case where both ECSM modules had a fault each are shown in the last row.

Scheme	**Pr(undetected error)**		**Pr($Q = \mathcal{O}, P$|undetected error)**	
	Stuck-at-0 fault	**Stuck-at-1 fault**	**Stuck-at-0 fault**	**Stuck-at-1 fault**
ECSM with PV	$\frac{1}{217}$	$\frac{1}{1,547}$	$\frac{536,935}{603,404}$	$\frac{2,030}{7,711}$
ECSM with PV[1]	$\frac{1}{1,974}$	$\frac{1}{2,100}$	NA	NA
RC	$\frac{1}{1,097,914}$	$\frac{1}{7,497,185}$	$\frac{236}{239}$	$\frac{197}{1,400}$
PC[2]	0	0	NA	NA
RC_partial[3]	$\frac{1}{19,653}$	$\frac{1}{397,980}$	$\frac{28}{81}$	$\frac{20}{81}$
PC[4]	$\frac{1}{1,678,848}$	$\frac{1}{9,806,261}$	$\frac{20,705}{20,944}$	$\frac{370}{3,447}$

[1]Modifying the PV module to exclude $\mathcal{O}$ and P as valid outputs. [2]Injecting faults only to one ECSM module. [3]$l = 3$. [4]Injecting faults to both ECSM modules.

Table 4.6: Probabilities of undetected errors for our experiment over $\mathbb{F}_{2^{11}}$

4.4.3 Comments

We have noticed that for both types of faults and especially for the stuck-at-0 fault model, the probability of obtaining two special points, namely $\mathcal{O}$ and P is relatively high. The following two scenarios contribute to these higher probabilities. First, if the projective Z-coordinate Q_Z of the ECSM result Q is zero, then irrespective of the values of X- and Y-coordinates the result is the $\mathcal{O}$ point. Second, consider the case where the Z-coordinate of the variable representing Q is zero in the last iteration of the loop in the double-and-add algorithm for ECSM. In such a case, assuming a left-to-right version of the algorithm, the final result will be $\mathcal{O}$ or P depending of the value of the least significant bit of k.

With the above observations, it is useful that the PV module does not consider $\mathcal{O}$ or P as a valid output of ECSM. In fact, from the cryptographic point of view, if P and k are selected as a non trivial value (i.e., $k \neq \{0, 1, \texttt{ord}(P)\}$), a valid result will not be either $\mathcal{O}$ or P. If we assume that the PV module is modified such that $\mathcal{O}$ and P are not considered as a valid ECSM output, the resulting probabilities are very close to the value obtained using Equation (4.9), i.e., $\frac{1982}{2^{11}2^{11}} \approx \frac{1}{2^{116}}$ as shown in the second row of Table 4.6.

Another interesting observation of this experiment is that it is more likely to obtain a result equal to $\mathcal{O}$ with faults that are stuck-at-0 than those with stuck-at-1. For our experiments, stuck-at-0 faults tend to reduce the Hamming weight of the Z-coordinate of the output as shown in Table 4.7. In contrast, on average stuck-at-1 faults produce results with more binary 1s. For this reason, the stuck-at-0 faults have more cases with $Q_Z = 0$, and consequently, the resulting point on the curve corresponds to $Q = \mathcal{O}$. This fact implies that the probabilities of undetected

Case	Average($H(\widetilde{Q}_Z)$)
Fault free	5.5
Stuck-at-0	5.159
Stuck-at-1	5.534

Table 4.7: Average Hamming weight of the Z-coordinate of the result for our experiment over $\mathbb{F}_{2^{11}}$

error for stuck-at-1 cases are less than their stuck-at-0 counterparts (e.g., $\frac{1}{7,497,185}$ vs. $\frac{1}{1,097,914}$ for RC).

As stated earlier, the error detection capability depends on various factors including the actual inputs, the faults and implementation architecture. For our experiments, the architecture might have considerable impact on the numerical results, since the finite field arithmetic unit was invoked several times during each ECSM operation. In addition, the fault type (stuck-at or bit-flip), fault location, etc. might have affected our experiments.

4.5 Conclusion

In this chapter, we have presented error-detecting and fault-tolerant structures for ECSM. For the purpose of error detection, the concepts of re-computation and parallel computation have been used. In order to have a higher probability of error detection during the ECSM operation, we have presented encoding/decoding schemes suitable for ECSM computation. Schemes are based on the concepts of scalar and point randomization. These schemes provide resistance to attacks where fault-induced operations do not leave the original elliptic curve (e.g., the SCF at-

tack). By generating single stuck-at faults exhaustively for a small ECSM prototype we have given experimental results that show the probabilities of undetected errors for the proposed error-detecting schemes.

Traditionally, the concept of having only two modules working in parallel has been associated with error-detecting systems only. However, for ECSM, we have shown that with only two ECSM modules along with either PV (i.e., the proposed DMR_PV based scheme) or re-computation (i.e., PRC based scheme), it is possible not only to detect but also correct errors due to faults. These fault-tolerant schemes are more efficient, i.e., it uses one less ECSM module than the TMR based scheme.

Chapter 5

Algorithm-level Error Detection for ECSM

In Chapter 4 we have presented error detection and fault tolerance in ECSM at the module level. That is, we have added external elements to the ECSM module in order to detect errors and/or tolerate faults, where the underlying scalar multiplication algorithm is not modified. In contrast, this chapter presents error detection at the algorithm level. Here, we add protections inside the ECSM algorithm in order to detect errors caused either by natural causes or deliberately by faults injected by an attacker. 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 in this chapter are algorithm specific. On the contrary, PV can be applied to any ECSM method.

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

- Any variable utilized in the ECSM can be a target of natural faults or faults injected by an attacker. For simplicity in the analysis we assume that variables such as the loop counter i and 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 5.1 and 5.2 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 5.3 we relax this assumption considering the *double*-fault attack proposed by Yen et al. [94] and refined by Kim and Quisquater [48] for RSA cryptosystems.

- In this chapter 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 $(V_0, V_1, \ldots V_{j-1})$ which is composed of the variables utilized in the ECSM algorithm. As a consequence of faults produced naturally or injected by an attacker the vector might be changed to $(\widetilde{V}_0, \widetilde{V}_1, \ldots \widetilde{V}_{j-1})$. 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 $(V_0, V_1, \ldots V_{j-1})$. 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 chapter we use the following definition [56]:

Definition 5.1 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 [90].

The organization of the remainder of this chapter is as follows. In Section 5.1, we consider error detection in the Montgomery ladder ECSM. Section 5.2 presents error detection in the double-and-add-always method. In Section 5.3, we give a countermeasure that can be used against the double-fault attack. Finally, we make some concluding remarks in Section 5.4.

5.1 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 [58]. First, we consider the case where a PV process is placed at the end of the ECSM. Then, we use CC among the involved variables

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 2.5), 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 being $n \pm 1$ and n, and $n/2$ if n is even.

5.1.1 PV process at the end of the ECSM

The Montgomery ladder algorithm in affine coordinates proposed by López and Dahab [58] (Algorithm 2.6) is shown in Algorithm 5.1 with a PV process at the end of the ECSM. This algorithm restricts the occurrence of the exceptional cases described above. In Step 3.1, as defined by Equation (2.11), the y-coordinate of the output (i.e., Q_{0_y}) is computed using the following function:

$$g(Q_{0_x}, Q_{1_x}, x, y) = \frac{(Q_{0_x} + x)\left[(Q_{0_x} + x)(Q_{1_x} + x) + x^2 + y\right]}{x} + y. \qquad (5.1)$$

For Algorithm 5.1 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 5.1. 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 5.1 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 $(\widetilde{Q}_{0_x}, \widetilde{Q}_{0_y})$ in the governing elliptic curve equation (Equation (2.3)) to obtain:

$$\widetilde{Q}_{0_y}^2 + \widetilde{Q}_{0_x}\widetilde{Q}_{0_y} = \widetilde{Q}_{0_x}^3 + a\widetilde{Q}_{0_x}^2 + b. \tag{5.2}$$

Using Equation (5.1), 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 (5.2) 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

$$w_1 = \widetilde{Q}_{0_x}^2 + \frac{\widetilde{y}^2}{\widetilde{x}^2} + \widetilde{x}^2 + \frac{\widetilde{y}}{\widetilde{x}} + \widetilde{x} + a + \frac{b}{\widetilde{Q}_{0_x}^2}. \tag{5.3}$$

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

$$\widetilde{Q}_{1_{x(a)}} = \frac{\widetilde{x}\widetilde{Q}_{0_x}}{\widetilde{x}^2 + \widetilde{Q}_{0_x}^2}\mathtt{Ht}(w_1), \tag{5.4}$$

$$\widetilde{Q}_{1_{x(b)}} = \frac{\widetilde{x}\widetilde{Q}_{0_x}}{\widetilde{x}^2 + \widetilde{Q}_{0_x}^2}(\mathtt{Ht}(w_1) + 1). \tag{5.5}$$

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 (5.4) or (5.5), then $\mathrm{PV}(\widetilde{Q}_0) = 1$. In this scenario PV fails to detect such errors.

Lemma 5.1 For an arbitrary vector $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$, the number of combinations where $\mathrm{PV}(\widetilde{Q}_0) = 1$ in Step 3.2 of Algorithm 5.1 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 5.1 is

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

Using Lemma 5.1 we can obtain the code rate c_R for this case as

$$c_R = \frac{\log_2(2^m(\#E(\mathbb{F}_{2^m}) - 2)(2^m - 2))}{4m} \approx \frac{3}{4}. \tag{5.6}$$

In Step 3.1 of Algorithm 5.1 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 difference is lost, then presumably the corresponding $\widetilde{Q}_0$ will become a finite field pair that does not rely on $E(\mathbb{F}_{2^m})$. Then PV process at the end of ECSM will detect such errors. This is not the case for the combinations obtained in Lemma 5.1.

5.1.2 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 correspond-

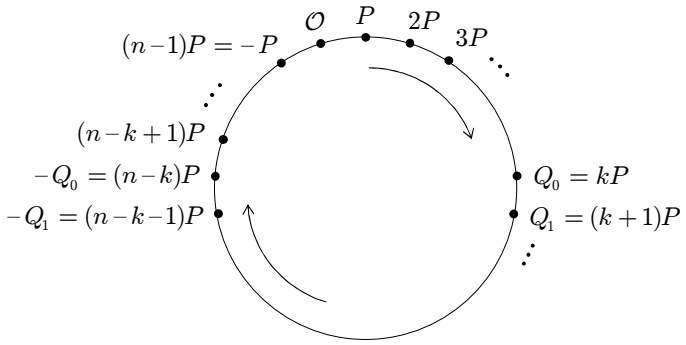

Figure 5.1: Multiples of point P of order n

ing 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 5.1). Adding also $-\widehat{Q}_0 \uplus P$ 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 5.2 implements this function. Additionally, if the CC passes, then the corresponding Q_{0_y} is also returned. This algorithm utilizes the group formulas given on pages 17-18.

Algorithm 5.2. 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 5.3. Now let us examine the cases where Algorithm 5.3 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 5.1, 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 $\varepsilon_P = 0$. By Theorem 2.3, $\widehat{Q}_0 = (\widetilde{Q}_{0_x}, \widehat{Q}_{0_y}) \in E(\mathbb{F}_{2^m})$ exists if and only if $\mathtt{Tr}(w_2) = 0$, where

$$w_2 = \widetilde{Q}_{0_x} + \frac{b}{\widetilde{Q}_{0_x}^2} + a. \tag{5.7}$$

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 5.2, 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:

$$\widetilde{Q}_{1_{x(a)}} = \frac{\widetilde{x}\widetilde{Q}_{0_x}}{\widetilde{x}^2 + \widetilde{Q}_{0_x}^2}\left(\mathtt{Ht}(w_2) + \widetilde{x} + \frac{\widetilde{y}}{\widetilde{x}} + \widetilde{Q}_{0_x} + \frac{\varepsilon_P}{\widetilde{x}\widetilde{Q}_{0_x}}\right), \tag{5.8}$$

$$\widetilde{Q}_{1_{x(b)}} = \frac{\widetilde{x}\widetilde{Q}_{0_x}}{\widetilde{x}^2 + \widetilde{Q}_{0_x}^2}\left(\mathtt{Ht}(w_2) + \widetilde{x} + \frac{\widetilde{y}}{\widetilde{x}} + \widetilde{Q}_{0_x} + \frac{\varepsilon_P}{\widetilde{x}\widetilde{Q}_{0_x}} + 1\right). \tag{5.9}$$

Algorithm 5.3. 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 $k = (k_{t-1} \cdots k_1 k_0)_2$, 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. Use Algorithm 5.2 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 5.3 fails to detect errors is the same as Algorithm 5.1 obtained in Lemma 5.1. Consequently, c_R for Algorithm 5.3 is also $\approx 3/4$. In the next subsection we compare the error detection coverage of these two algorithms.

5.1.3 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 5.2 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 5.1 as function of r, s, u, and v.

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

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

Proof For $\text{PV}(\widetilde{Q}_0)$, to obtain solutions to $\widetilde{Q}_{1_x}$ using Equations (5.4) and (5.5) we should have $\texttt{Tr}(w_1) = 0$, where w_1 is defined by Equation (5.3). Similarly for $\text{CC1}(Q_{0_x}, Q_{1_x}, x, y)$, $\widehat{Q}_0 \in E(\mathbb{F}_{2^m})$ exists if and only if $\texttt{Tr}(w_2) = 0$, where w_2 is defined by Equation (5.7). From Equation (5.3) we can obtain

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

which corresponds to $\texttt{Tr}(w_2)$, i.e., $\texttt{Tr}(w_1) = \texttt{Tr}(w_2)$. Then (i) and (ii) are true since $\texttt{Tr}(w_1)$ (and $\texttt{Tr}(w_2)$) 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 5.2, 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 5.3 Let l, r, s, u, and v be defined as in Lemma 5.2.

(i) There exists a vector (r, s, u, v) for which $\text{PV}((r, l)) = 1$ and $\text{CC1}(r, s, u, v) = 1$, i.e., cases where both Algorithms 5.1 and 5.3 fail in detecting the same vector $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$ (the overlapping area in Figure 5.2).

(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}.

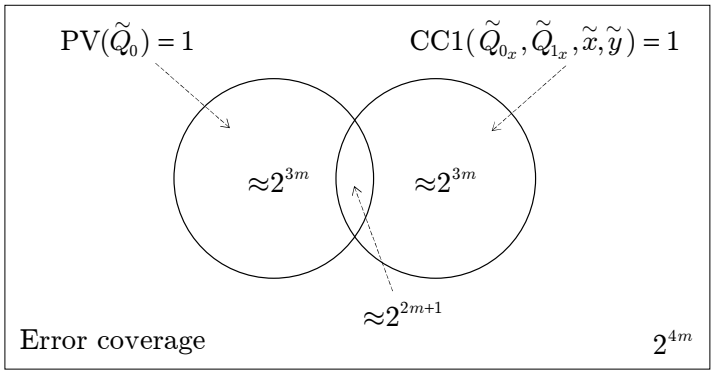

Figure 5.2: Error detection coverage for arbitrary $(\widetilde{Q}_{0_x}, \widetilde{Q}_{1_x}, \widetilde{x}, \widetilde{y})$

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 (5.4-5) and (5.8-9) 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 (5.4) and (5.8), and Equations (5.4) and (5.9) we obtain

$$\widetilde{x}\widetilde{Q}_{0_x}\mathtt{Tr}\left(\widetilde{Q}_{0_x} + \frac{\widetilde{y}}{\widetilde{x}} + \widetilde{x}\right) = \varepsilon_P, \tag{5.10}$$

$$\widetilde{x}\widetilde{Q}_{0_x}\left[\mathtt{Tr}\left(\widetilde{Q}_{0_x} + \frac{\widetilde{y}}{\widetilde{x}} + \widetilde{x}\right) + 1\right] = \varepsilon_P, \tag{5.11}$$

respectively. Depending on the value of $\mathtt{Tr}(\widetilde{Q}_{0_x} + \frac{\widetilde{y}}{\widetilde{x}} + \widetilde{x})$, Equations (5.10) and (5.11) are reduced, one to

$$\varepsilon_P = 0, \tag{5.12}$$

and the other to

$$\varepsilon_P = \widetilde{x}\widetilde{Q}_{0_x}, \tag{5.13}$$

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 (5.4) or (5.5), and either Equation (5.12) or (5.13), then PV($\widetilde{Q}_0$) and 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:

$$\widetilde{y}^2 + \widetilde{x}\widetilde{y} + \widetilde{x}^3 + a\widetilde{x}^2 + b + \widetilde{x}\widetilde{Q}_{0_x} = 0. \tag{5.14}$$

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

$$\mathtt{Tr}\left(\widetilde{x} + a + \frac{b}{\widetilde{x}^2}\right) = \mathtt{Tr}\left(\frac{\widetilde{Q}_{0_x}}{\widetilde{x}}\right). \tag{5.15}$$

  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 (5.15) 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 (2.3)). 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 (5.15) 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 (5.14). Then the number of combinations for which both Algorithms 5.1 and 5.3 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., $\approx 2^{2m+1}$). □

From Lemma 5.3 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

$$c_R \approx \frac{2m+1}{4m} = \frac{1}{2} + \frac{1}{4m}. \tag{5.16}$$

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.

5.1.4 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. [9],

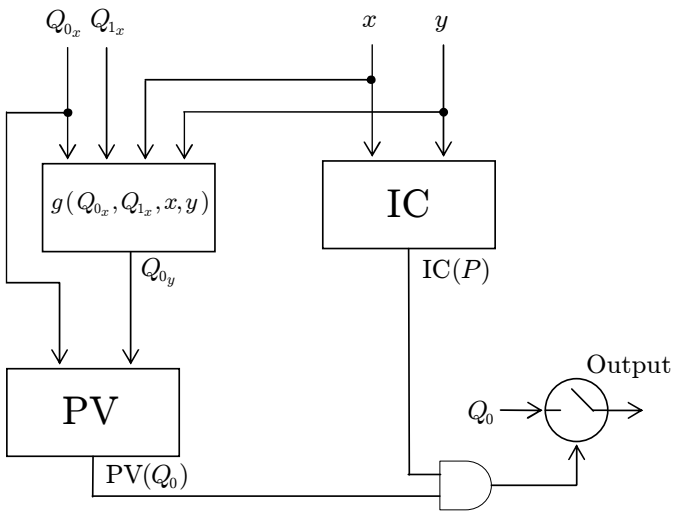

Figure 5.3: Error detection utilizing PV and IC processes

Ciet and Joye [21], Antipa et al. [6], Blömer et al. [14], and Domínguez and Hasan [27]. The second approach consists of checking the coherency between the involved variables which is an extension of the work presented by Giraud [37] 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 5.3. 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 5.3

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 (5.2) we get the following quadratic expression for $\widetilde{Q}_{1_x}$

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

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

$$w_3 = \widetilde{Q}_{0_x}^2 + x^2 + \frac{b}{x^2} + \frac{b}{\widetilde{Q}_{0_x}^2}. \tag{5.17}$$

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

$$\widetilde{Q}_{1_{x(a)}} = \frac{x\widetilde{Q}_{0_x}}{x^2 + \widetilde{Q}_{0_x}^2}\mathtt{Ht}(w_3), \tag{5.18}$$

$$\widetilde{Q}_{1_{x(b)}} = \frac{x\widetilde{Q}_{0_x}}{x^2 + \widetilde{Q}_{0_x}^2}\left(\mathtt{Ht}(w_3) + 1\right). \tag{5.19}$$

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 (5.18) and (5.19) can be rewritten as follows

$$\widetilde{Q}_{1_{x(a)}} = \frac{x\widetilde{Q}_{0_x}}{x^2 + \widetilde{Q}_{0_x}^2}\left[\mathtt{Ht}(w_3') + \frac{y}{x} + x + \mathtt{Tr}(x)\right], \tag{5.20}$$

$$\widetilde{Q}_{1_{x(b)}} = \frac{x\widetilde{Q}_{0_x}}{x^2 + \widetilde{Q}_{0_x}^2}\left[\mathtt{Ht}(w_3') + \frac{y}{x} + x + \mathtt{Tr}(x) + 1\right]. \tag{5.21}$$

where $w_3' = \widetilde{Q}_{0_x}^2 + \frac{b}{\widetilde{Q}_{0_x}^2} + a$. Here for each possible value of $\widetilde{Q}_{0_x}$ (i.e., $(\#E(\mathbb{F}_{2^m})/2 - 2)$)

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:

$$c_R = \frac{\log_2(\#E(\mathbb{F}_{2^m}) - 4)}{4m} \approx \frac{1}{4}, \tag{5.22}$$

which represents the best 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 $\mathbb{F}_p$).

5.1.5 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 involved variables. Now, let us consider the basic Montgomery ladder ECSM (Algorithm 2.5). As discussed in Subsection 2.3.1 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 point 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

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

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

$$c_R \approx \frac{m}{6m} = \frac{1}{6}. \tag{5.23}$$

Even when this approach theoretically has an improved error detection capability, it has an important drawback in terms of performance as illustrated in Table 5.1. This table compares the operation counts for ECSM among the Montgomery ladder algorithms proposed by López and Dahab [58] (Algorithms 2.6 and 2.7), the Montgomery ladder with PV at the end, the basic Montgomery ladder (Algorithm 2.5), and the basic Montgomery ladder (Algorithm 2.5) with $\mathrm{PV}(Q_0)$ and $\mathrm{CC2}(Q_0, Q_1, P)$. This table shows that the basic Montgomery ladder with $\mathrm{PV}(Q_0)$ and $\mathrm{CC2}(Q_0, Q_1, P)$ 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 $\mathrm{PV}(Q_0)$ and $\mathrm{CC2}(Q_0, Q_1, P)$ requires about the triple more finite field multiplications than the Montgomery ladder with PV at the end, i.e., $(18t - 6)M$ vs. $(6t + 6)M$.

5.1.6 Security 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

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 2.6. [2]With error detection scheme of Figure 5.3. [3]Using Algorithm 2.5.
[4]With PV(Q_0) and CC2(Q_0, Q_1, P). [5]Using Algorithm 2.7.

Table 5.1: Operation counts for computing the ECSM utilizing error detection at the algorithm level for Montgomery's ladder method

attack proposed by Blömer et al. [14] 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 [52] and simple power analysis [53].

For invalid-curve attacks, including those discussed in Chapter 3, verifying if 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 during 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, "a security sys-

tem is only as strong as its weakest link. It doesn't matter how strong the other parts are" [31]. 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 5.3 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").

5.2 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 [23] 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 [91] and Yen et al. [92] have shown that adding dummy instructions makes possible an SE attack.

Here we first present a left-to-right version of the double-and-add-always that is resistant to attacks such as SE and DFA attacks. As we discussed in Chapter 2, for applications utilizing projective coordinates the left-to-right version permits the use of mixed coordinates. Thereafter, we present an extension of the right-to-left version presented in the context of RSA cryptosystems by Boscher et al. [16] (Algorithm 2.10). 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 [14], utilizing the Montgomery ladder algorithm without the y-coordinates, and those that use scalar randomization [27]. For this section, let us consider elliptic curves defined over either $\mathbb{F}_{2^m}$ or $\mathbb{F}_p$, i.e., $\mathbb{F}_q$.

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

Compared to Algorithm 2.1, Algorithm 2.3 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 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 5.4. Here, Q_0 follows the same sequence of operations as the original double and add method (Algorithm 2.1), 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 5.4. 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(Q_1, k, P) = 1)) then
   3.1 Return(Q_0);
4. Else return("Error detected").

In addition to the SE attack protections, Algorithm 5.4 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 (5.23) $c_R \approx \frac{1}{6}$. Even when this code rate might be quite good for some cases (e.g., random errors), for curves defined over $\mathbb{F}_p$ Algorithm 5.4 is insecure against the SCF attack.

In a similar way as described by Blömer et al. [14], 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 5.4 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.

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

Similar to the left-to-right version, Algorithm 2.4 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 to ECC the idea from Boscher et al. [16], 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 5.5. After the main loop the expected value of the following points is:

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

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

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

In addition to CC, Algorithm 5.5 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 5.5. 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 $((\text{PV}(Q_0) = 1)$ and $(\text{PV}(Q_1) = 1)$ and $(\text{CC4}(Q_0, Q_1, Q_2, P) = 1)$ and $(\text{IC}(P) = 1))$ then
   3.1 Return(Q_0);
4. Else return("Error detected").

Security Analysis on Algorithm 5.5

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 5.5.

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 the main loop we will have $\widetilde{Q}_1 \neq \overline{k}P$. Here Algorithm 5.5 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., $\mathrm{CC4}(Q_0, \widetilde{Q}_1, Q_2, P) = 1$) if, and only if,

$$\widetilde{Q}_1 = (2^t - k - 1 + jn)P, \tag{5.24}$$

where j is an integer and $n = \mathtt{ord}(P)$. 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 (5.24) since he/she does not know k. As a result, Algorithm 5.5 can be considered as SE resistant.

SCF Attack: Let us assume that the attacker can inject one SCF into an intermediate point of Algorithm 5.5 at some random and unknown loop iteration $i \in [0, t-1]$. This fault model is similar to the assumed by Blömer et al. [14]. 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. From Equation (2.6), we can obtain the output of Algorithm 5.5 $Q_0 = kP$ as

$$\begin{aligned} Q_0 &= k_{t-1}2^{t-1}P \uplus k_{t-2}2^{t-2}P \cdots \uplus k_{i+1}2^{i+1}P \uplus \underbrace{k_i 2^i P \cdots \uplus k_1 2P \uplus k_0 P}_{Q_{0,i}}, \\ &= \sum_{j=i+1}^{t-1} k_j 2^j P \uplus Q_{0,i}, \end{aligned} \tag{5.25}$$

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 (5.25) 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 5.5 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 (5.25) 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. [14], Algorithm 5.5 is SCF attack resistant. This is an interesting result because this algorithm does not use a combined curve [14] or randomization as RC and PC schemes presented in Chapter 4. This protection is based on CC among the involved variables and it resists both the SE and the SCF attacks.

5.2.3 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 5.2 we present

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 2.3. [2]Using Algorithm 2.4. [3]Using Algorithm 5.4 to obtain kP and Algorithm 2.3 for $H(\overline{k})P$. [4]Using Algorithm 5.5. [5]$t' = t + \lceil \log_2(t-1) \rceil$.

Table 5.2: Operation counts for the double-and-add-always ECSM method for curves defined over $\mathbb{F}_{2^m}$

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 2.3 and 2.4 which do not include any error-detecting processes. In contrast, the last four rows shows their counterparts that includes PV and CC (Algorithms 5.4 and 5.5). Table 5.3 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 5.4 and 5.5, 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 5.4 and 5.5, respectively. Additionally, we can notice a speed advantage

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 2.3. [2]Using mixed coordinates for point addition. [3]Using Algorithm 2.4. [4]Using Algorithm 5.4 to obtain kP and Algorithm 2.3 for $H(\overline{k})P$, assuming $H(\overline{k}) = \lceil t/2 \rceil = 82$. [5]Using Algorithm 5.5.

Table 5.3: Number of finite field operations for the double-and-add-always ECSM method for $t = m = 163$ for curves over $\mathbb{F}_{2^{163}}$

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 2.3 and 2.4, respectively; and $1I + 2056M + 1200S$ vs. $1I + 2637M + 1319S$ for Algorithms 5.4 and 5.5, respectively. This is mainly for the use of mixed coordinates.

Similarly, Table 5.4 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 5.5 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 5.4 and 5.5, 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 5.4 and 5.5, 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%[1]. For the case of Algorithm 5.5 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 5.5 utilizing projective coordinates is about 27.4%[1]. This contrasts with the $30 - 40\%$ reported by Blömer et al. [14] or about the 100% for RC or PC schemes.

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 2.3. [2]Using Algorithm 2.4. [3]Using Algorithm 5.4 to obtain kP and Algorithm 2.3 for $H(\overline{k})P$. [4]Using Algorithm 5.5. [5]$t' = t + \lceil \log_2(t-1) \rceil$.

Table 5.4: Operation counts for the double-and-add-always ECSM method for curves defined over $\mathbb{F}_p$

[1]We assume that $I = 80M$ and $S = 0.85M$ as [40].

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 2.3. [2]Using mixed coordinates for point addition. [3]Using Algorithm 2.4. [4]Using Algorithm 5.4 to obtain kP and Algorithm 2.3 for $H(\overline{k})P$, assuming $H(\overline{k}) = \lceil t/2 \rceil = 96$. [5]Using Algorithm 5.5.

Table 5.5: Number of finite field operations for the double-and-add-always ECSM method for $t = 192$ for curves over $\mathbb{F}_{p_{192}}$

5.3 Double-fault attack resistant ECSM

Yen et al. [94] 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 [37]. In such a case this attack might not be possible. However, Kim and Quisquater [48] showed that general propose microcontrollers can be target of a 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 protec-

tions should be added. In this section we present algorithm level countermeasures against this type of attacks for ECSM.

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 ($\mathrm{PV}(Q) = 1$) then").

Let us base our countermeasure on Montgomery's ladder ECSM for non-supersingular elliptic curves over the binary finite field proposed by López and Dahab [58] (Algorithm 2.6). However, these concepts can generally be extended to the corresponding Montgomery's ladder ECSM for prime fields presented by Brier and Joye [17]. 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 [23] for making DFA impractical. However, for the affine system we can use the concept of point "blinding" presented by Coron [23]. The idea is to add a random point R to the initial values of Q_0 and Q_1. Let r be defined as $r = \log_P R$, i.e., $R = rP$. Using Equation (2.7), 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 (2.8) 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:

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

Then, to obtain $Q = kP$ we need to compute $Q = Q_0 - 2^{t-1}R$. This is depicted in Figure 5.4. The complete procedure that implements this version of Montgomery's ladder is presented as Algorithm 5.6. 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. [9] 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 5.6 in comparison with Algorithm 2.6 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. $(2t+1)I + (2t+1)M + (2t+1)S$.

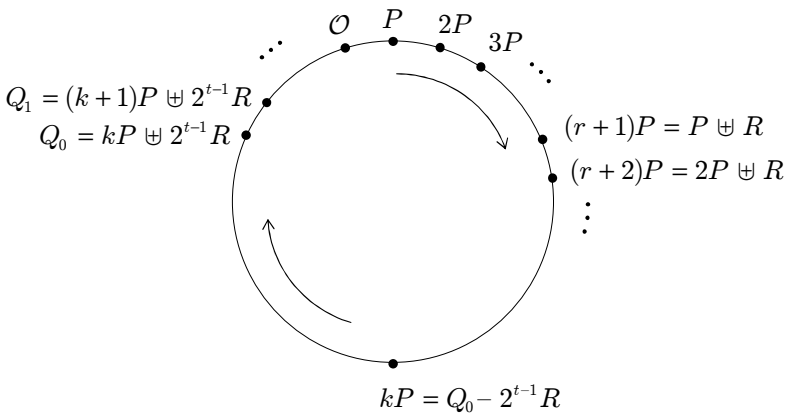

Figure 5.4: Blinding point P for the Montgomery ladder ECSM

Algorithm 5.6. 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").

5.4 Conclusion

In this chapter 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 [23] have been shown to be susceptible to the SE attack [91] [92]. In this chapter, 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. [14]. This result is interesting since this algorithm do not use the combined curve [14], 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. [14], 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.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, several aspects of fault-based attacks and countermeasures for ECC have been studied. We have presented a new fault-based attack against a popular algorithm, namely the Montgomery ladder ECSM for curves over the binary field. In addition, we have presented error-detecting schemes for ECSM at both module and algorithm levels.

In Chapter 3, we have introduced an invalid-curve attack on the Montgomery ladder ECSM algorithm proposed by López and Dahab. This attack is based on the fact that curve parameter a is not utilized for point operation formulas in this ECSM method. From an original group $E(\mathbb{F}_{2^m})$, we have assumed that there exists a weaker group with the same parameters except for a. We have shown that this assumption is true for nine of the ten NIST-recommended elliptic curves over the binary field. Under a specific fault model, we have presented two versions of this

attack with their respective probabilities of success. This attack underlines the importance of verifying the correctness of the ECSM operation.

Error detection is an essential task for protecting against fault-based attacks. Applying some properties of elliptic curves, it is possible to detect errors during the ECSM operation utilizing different techniques. The most obvious is to verify if the output relies on the original elliptic curve. However, it has been shown that in some cases this protection is not sufficient. Other techniques utilized to detect errors during the ECSM computation have been presented in Chapters 4 and 5.

In Chapter 4, we have used the concepts of re-computation and parallel computation to achieve error detection and fault tolerance on ECSM. By means of this we have introduced encoding/decoding schemes suitable for ECSM which are based on scalar and point randomization. The proposed structures permit us to detect errors with high probability. Additionally they resist attacks such as the SCF attack without modifying the curve parameters. This might be important for crypto-processors that have fixed parameters (e.g., those recommended by NIST). Also, utilizing a small ECSM prototype, we have presented experimental results for the probability of undetected errors for the error-detecting structures presented. We have also shown that, with two ECSM modules working in parallel, it is possible to have fault-tolerant schemes that will not only detect but also correct errors. This contrasts with the three modules needed for the TMR based systems. Using an FPGA as hardware target with an NIST-recommended elliptic curve over $\mathbb{F}_{2^{163}}$, we have shown that the savings in terms of area with respect to TMR based ECSM is about 24.8% for DMR_PV and 23.6% for PRC.

In Chapter 5, we have presented error detection at the algorithm level for ECSM. We have utilized the concepts of PV, CC, and IC. The idea of CC has been carried

out by algorithm specific functions CC1-CC4. For the Montgomery ladder ECSM algorithm for elliptic curves over the binary field, we have demonstrated that PV has the same error detection coverage as CC1 if an IC of the input point is performed. We have shown that using PV and IC it is possible to have improved error detection without degrading the performance. This contrasts with the notable error detection coverage of the basic Montgomery ladder ECSM algorithm where the penalty in performance might be unacceptable. Utilizing CC as error-detecting technique, we have proposed two versions of the double-and-add-always ECSM method (Algorithms 5.4 and 5.5). We have shown that both algorithms resist the SE attack. In addition, we have proved that the right-to-left version (Algorithm 5.5) resists the SCF attack. This is the first countermeasure against the SCF attack reported in the literature that does not use a combined curve [14] or randomization as RC and PC schemes presented in Chapter 4. For today's applications, where $t \approx 192$, we have shown that our countermeasure has a overhead in terms of finite field operations of about 0.8% when using the affine coordinate system and about 27.4% for applications utilizing the projective coordinate system. These values contrast with the overhead utilizing a combined curve (i.e., $30 - 40\%$ [14]) or re-computation or parallel computation (i.e., about 100% for RC and PC schemes).

6.2 Future work

In Chapter 4, we have presented experimental results for probabilities of undetected errors with a small prototype. The fault model utilized was based on single-bit stuck-at faults into the gates used for finite field operations. It would be interesting to obtain results for other fault models such as flip-bit faults in registers holding

partial results of the ECSM and/or multiple-bit faults. Additionally, it would also be interesting to obtain experimental results for the ECSM with protections at the algorithm level described in Chapter 5. Also we are interested to have some experimental results for a real size system, e.g., $m = 163$, used in practical applications.

A basic assumption for the error-detecting and fault-tolerant structures for ECSM presented in Chapter 4 is that all modules except the ECSM module are implemented in a secure environment. It would be interesting to design those extra modules using error detection and/or fault tolerant techniques and show that this assumption can be carried out in practice.

In Chapter 5 we have considered algorithms that do not use precomputation. It is well known that if P is fixed the ECSM can be speeded up by precomputing some data that depends exclusively on P. Some ECSM methods that use this principle are: fixed-base windowing, fixed-base comb, and interleaving. It would be interesting to study protections for fault attacks on algorithms that use precomputation.

With some results of Chapter 5, we have noted the advantages of including countermeasures against fault-attacks at the algorithm level. It would be interesting to extend these concepts to fault-tolerance in ECSM.

Finally, it would also be interesting to extend some of the concepts of the research work presented in this thesis to hyperelliptic curves.

Appendix A

Average Number of EC Discrete Logarithms for Algorithm 3.3

In this appendix we include the computations of the average number of EC discrete logarithms for Algorithm 3.3 using the second improved approach described on page 75. As assumed in Subsection 3.2.2, the fault location is at a random position of the x-coordinate of the base point P. This assumption implies that the value of $k \bmod n_i$ is at a random position in Tables A_i, where $n_i = \mathtt{ord}(\widehat{P}_i)$ and $i \in \{0, 1\}$. Let us define the random variable w as the number of entries needed for having $k \bmod n_i$ in both tables. The order of the possible values of w is shown in Figure A.1 for the case $c_0 < c_1$.

Case: $c_0 = c_1$. In this case the accumulative probability distribution $F(w)$ for some given values is as follows:

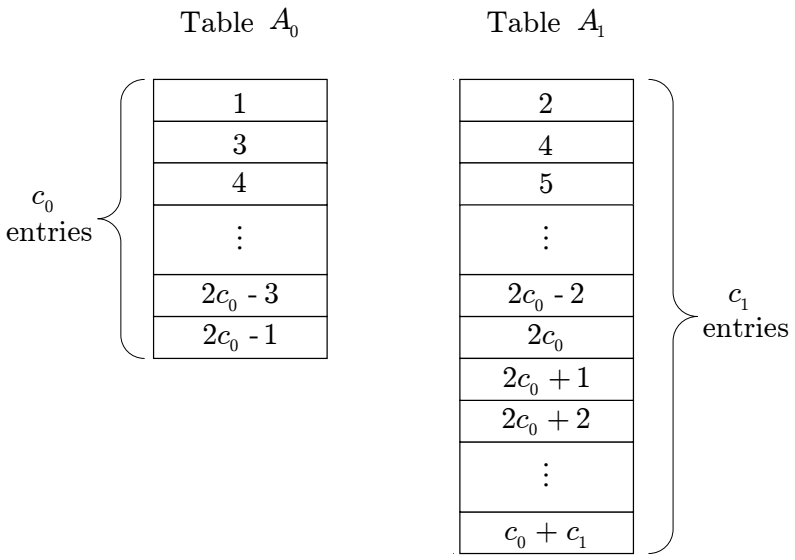

Figure A.1: Values of the random variable w according to entries of Tables A_0 and A_1 considering $c_0 < c_1$

$$
\begin{array}{ll}
F(1) = 0 & F(2) = 1/c_0^2 \\
F(3) = 2/c_0^2 & F(4) = 4/c_0^2 \\
F(5) = 6/c_0^2 & F(6) = 9/c_0^2 \\
F(7) = 12/c_0^2 & F(8) = 16/c_0^2 \\
\vdots & \vdots \\
F(2c_0 - 1) = (c_0 - 1)/c_0 & F(2c_0) = 1
\end{array}
$$

We can write $F(w)$ as

$$
F(w) = \begin{cases} (w^2 - 1)/(4c_0^2) & w \text{ odd, and } 1 \leq w \leq 2c_0 - 1, \\ w^2/(4c_0^2) & w \text{ even, and } 2 \leq w \leq 2c_0. \end{cases}
$$

Then the probability distribution $f(w) = F(w) - F(w-1)$ is

$$f(w) = \begin{cases} (w-1)/(2c_0^2) & w \text{ odd, and } 1 \leq w \leq 2c_0 - 1, \\ w/(2c_0^2) & w \text{ even, and } 2 \leq w \leq 2c_0. \end{cases}$$

The mean μ can be expressed by

$$\mu = \sum_{w=1}^{2c_0} w f(w).$$

After performing a change of variables (i.e., $y = \frac{w-1}{2}$ and $y = \frac{w}{2}$ for the odd and even number cases, respectively) μ can be re-written as

$$\mu = \sum_{y=0}^{c-1} \frac{y(2y+1)}{c_0^2} + \sum_{y=1}^{c_0} \frac{2y^2}{c_0^2} = \frac{8c_0^2 + 3c_0 + 1}{6c_0} \approx \frac{4}{3}c_0 \quad (\text{for } c_0 >> 1). \tag{A.1}$$

Case: $c_0 < c_1$. Similar to the previous case we can write $F(w)$ for some given values as follows

$$\begin{aligned} &F(1) = 0 && F(2) = 1/(c_0 c_1) \\ &F(3) = 2/(c_0 c_1) && F(4) = 4/(c_0 c_1) \\ &F(5) = 6/(c_0 c_1) && F(6) = 9/(c_0 c_1) \\ &\vdots && \vdots \\ &F(2c_0 - 1) = (c_0 - 1)/c_1 && F(2c_0) = c_0/c_1 \end{aligned}$$

$$\begin{array}{ll} F(2c_0+1) = (c_0+1)/c_1 & F(2c_0+2) = (c_0+2)/c_1 \\ \vdots & \vdots \\ F(c_0+c_1-1) = (c_1-1)/c_1 & F(c_0+c_1) = 1 \end{array}$$

We express $F(w)$ as

$$F(x) = \begin{cases} (x^2-1)/(4c_0c_1) & x \text{ odd, and } 1 \le x \le 2c_0-1, \\ x^2/(4c_0c_1) & x \text{ even, and } 2 \le x \le 2c_0, \\ (x-c_0)/c_1 & 2c_0+1 \le x \le c_0+c_1. \end{cases}$$

For this case the probability distribution $f(x)$ is

$$f(x) = \begin{cases} (x-1)/(2c_0c_1) & x \text{ odd, and } 1 \le x \le 2c_0-1, \\ x/(2c_0c_1) & x \text{ even, and } 2 \le x \le 2c_0, \\ 1/c_1 & 2c_0+1 \le x \le c_0+c_1. \end{cases}$$

We can obtain the mean μ as follows

$$\mu = \sum_{x=1}^{c_0+c_1} xf(x)$$

After performing a change of variables (i.e., $y = \frac{x-1}{2}$ and $y = \frac{x}{2}$ for the odd and even number cases, respectively, where $1 \le x \le 2c_0$) μ can be expressed as

$$\mu = \sum_{y=0}^{c_0-1} \frac{y(2y+1)}{c_0 c_1} + \sum_{y=1}^{c_0} \frac{2y^2}{c_0 c_1} + \sum_{x=2c_0+1}^{c_0+c_1} \frac{x}{c_1},$$

$$\mu = \frac{3c_1^2 - c_0^2 + 6c_0c_1 + 3c_1 + 1}{6c_1}. \tag{A.2}$$

Case: $c_0 > c_1$. This case is vary similar to the previous case. In fact, from Equation (A.2) we can perform the changes of variables $c_0 \leftarrow c_1$ and $c_1 \leftarrow c_0$ to obtain the mean for this case:

$$\mu = \frac{3c_0^2 - c_1^2 + 6c_0c_1 + 3c_0 + 1}{6c_0}. \tag{A.3}$$

Appendix B

Example of Undetected Errors with Point Negation Encoding/Decoding

Suppose we compute the ECSM using re-computation (Figure 2.2) and point negation as encoding and decoding process using the affine coordinate system. Let the elliptic curve utilized be $E(a, b)$ and be defined over $\mathbb{F}_{2^m}$. Assume that this scheme is implemented in hardware with one fault in the ECSM module. For simplicity assume that we have a stuck-at-1 fault, and it is located at the gate that gets bit i of the y-coordinate y_{p_i} as shown in Figure B.1. Let the elliptic curve points that are input to the ECSM module at time t_0 and t_1 be $P = (x_p, y_p)$ and $-P = (x_{\overline{p}}, y_{\overline{p}})$, respectively, where $x_{\overline{p}} = x_p$ and $y_{\overline{p}} = x_p + y_p$. The input y-coordinates passing

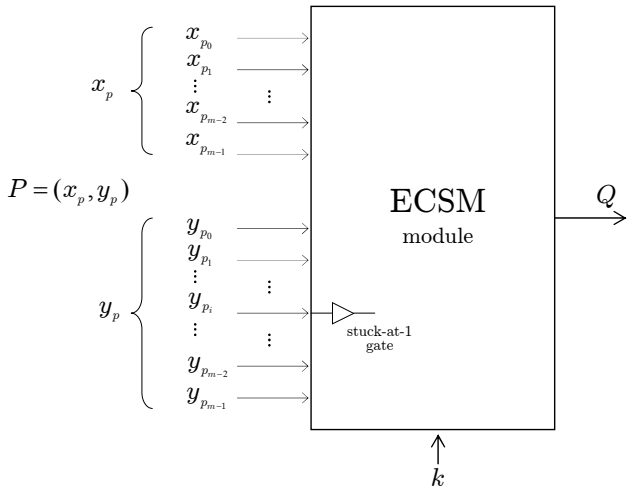

Figure B.1: Stuck-at-1 fault at the gate that gets bit i of the y-coordinate

through the faulty gate are as follows:

$$\widetilde{y_p} = y_p \mid (00\ldots010...00),$$

$$\widetilde{y_{\overline{p}}} = (x_p + y_p) \mid (00\ldots010...00),$$

where the vector $(00\ldots010...00)$ has 0s in all its bits with the exception of bit position i, and the operator $\mid$ is a bit-wise OR.

If $y_{p_i} = 1$ and $x_{p_i} + y_{p_i} = 1$, this fault does not produce any error in either of the two ECSM runs (see Table B.1). This is because the i-th bit of both y-coordinates is 1 and the fault does not change this bit's value. Any other combination of y_{p_i} and $x_{p_i} + y_{p_i}$ gives a different value as shown in Table B.1. For the second (respectively third) case we can see that an original input point, P (respectively $-P$), is present at time t_0 (respectively t_1). For these two cases the results will be different. This is because it is necessary to have complementary points at both input modules in

y_{p_i}	$x_{p_i} + y_{p_i}$	**ECSM input at t_0 (after the fault)**
1	1 ($x_{p_i} = 0$)	$(x_p, \widetilde{y}_p)$, where $\widetilde{y}_p = \ y_p$ (i.e., P)
1	0 ($x_{p_i} = 1$)	$(x_p, \widetilde{y}_p)$, where $\widetilde{y}_p = \ y_p$ (i.e., P)
0	1 ($x_{p_i} = 1$)	$(x_p, \widetilde{y}_p)$, where $\widetilde{y}_p \neq \ y_p$ (i.e., $\neq \pm P$)
0	0 ($x_{p_i} = 0$)	$(x_p, \widetilde{y}_p)$, where $\widetilde{y}_p \neq \ y_p$ (i.e., P_1)
y_{p_i}	$x_{p_i} + y_{p_i}$	**ECSM input at t_1 (after the fault)**
1	1 ($x_{p_i} = 0$)	$(x_{\overline{p}}, \widetilde{y}_{\overline{p}})$, where $\widetilde{y}_{\overline{p}} = \ y_{\overline{p}}$ (i.e., $-P$)
1	0 ($x_{p_i} = 1$)	$(x_{\overline{p}}, \widetilde{y}_{\overline{p}})$, where $\widetilde{y}_{\overline{p}} \neq \ y_{\overline{p}}$ (i.e., $\neq \pm P$)
0	1 ($x_{p_i} = 1$)	$(x_{\overline{p}}, \widetilde{y}_{\overline{p}})$, where $\widetilde{y}_{\overline{p}} = \ y_{\overline{p}}$ (i.e., $-P$)
0	0 ($x_{p_i} = 0$)	$(x_{\overline{p}}, \widetilde{y}_{\overline{p}})$, where $\widetilde{y}_{\overline{p}} \neq \ y_{\overline{p}}$ (i.e., P_2)

Table B.1: Possible alteration of the input coordinates

order to have the two outputs to match. For the fourth case, the input points are different from P and $-P$, say P_1 and P_2. The resultant finite field element pairs are the negative of each other, say $P_1 = -P_2$. Furthermore, they are not part of the original elliptic curve $E(a, b)$, but they are part of another elliptic curve, say $\widetilde{E}(a, \widetilde{b})$. Because the point addition and doubling might not use the parameter b, the two ECSM runs will be performed over $\widetilde{E}(\mathbb{F}_{2^m})$. When the computations are complete, the two results will be the same but incorrect. In this particular case the system will fail by accepting an erroneous result. Similar analysis and results can be obtained for a stuck-at-0 fault at the same bit position of the y-coordinate (i.e., y_{p_i}), or if the fault is located at the gate that gets the i-th bit of the x-coordinate (i.e., x_{p_i}).

Bibliography

[1] "The elliptic curve cryptosystem for smart cards," Certicom Corp., 1998. [Online]. Available: http://www.certicom.com/whitepapers 13

[2] 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

[3] 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. 3

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

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

[6] 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. 35, 36, 37, 38, 41, 83, 88, 136

[7] 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: Cryptographic Hardware and Embedded Systems*, ser. LNCS 2523. Springer-Verlag, 2002, pp. 260–275. 45

[8] E. R. Berlekamp, H. Rumsey, and G. Solomon, "On the solution of algebraic equations over finite fields," *Information and Control*, vol. 10, no. 6, pp. 553–564, 1967. 14

[9] 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. 4, 33, 35, 36, 37, 38, 41, 49, 82, 83, 88, 97, 98, 135, 155

[10] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *CRYPTO 1997: Advances in Cryptology*, ser. LNCS 1294. Springer-Verlag, 1997, pp. 513–525. 3

[11] I. F. Blake, G. Seroussi, and N. P. Smart, *Elliptic Curves in Cryptography*. Cambridge University Press, 1999. 9, 13, 19

[12] 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. 45

[13] 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. 45

[14] ——, "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. 4, 37, 38, 41, 97, 98, 136, 140, 142, 144, 147, 149, 152, 157, 158, 161

[15] 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, 4, 33, 36, 45

[16] 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. 45, 46, 47, 141, 144

[17] 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. 25, 38, 154

[18] C.-L. Chen, "Formulas for the solutions of quadratic equations over $GF(2^m)$," *IEEE Transactions on Information Theory*, vol. 28, no. 5, pp. 792–794, 1982. 14

[19] D. V. Chudnovsky and G. V. Chudnovsky, "Sequences of numbers generated by

addition in formal groups and new primality and factorization tests," *Advances in Applied Mathematics*, vol. 7, no. 4, pp. 385–434, 1986. 21

[20] 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. 45

[21] ——, "Elliptic curve cryptosystems in the presence of permanent and transient faults," *Designs, Codes and Cryptography*, vol. 36, no. 1, pp. 33–43, 2005. 34, 36, 37, 38, 41, 49, 83, 136

[22] H. Cohen and G. Frey, Eds., *Handbook of elliptic and hyperelliptic curve cryptography*. CRC Press, 2005. 9, 21, 23, 42

[23] 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. 5, 24, 90, 92, 141, 154, 157

[24] R. E. Crandall, "Method and apparatus for public key exchange in a cryptographic system," United States Patent 5,159,632, October 1992. 25

[25] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976. 1, 31

[26] A. Domínguez-Oviedo and M. A. Hasan, "Error-detecting and fault-tolerant structures for ECC"," CACR Technical Reports CORR 2005-10, University of Waterloo, Tech. Rep., 2005. 86

[27] ——, "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. 41, 86, 136, 142

[28] N. Ebeid, "Key randomization countermeasures to power analysis attacks on elliptic curve cryptosystems," Ph.D. dissertation, University of Waterloo, 2007. 40

[29] N. Ebeid and M. A. Hasan, "On randomizing private keys to counteract DPA attacks," in *SAC 2003: Selected Areas in Cryptography*, ser. LNCS 3006. Springer-Verlag, 2003, pp. 58–722. 92

[30] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985. 31

[31] N. Ferguson and B. Schneier, *Practical Cryptography*. Wiley, 2003. 141

[32] FIPS 186-2 Digital Signature Standard (DSS), *Federal Information Processing Standards Publication 186-2*. National Institute for Standards and Technology, 2000. 13, 88

[33] FIPS 186 Digital Signature Standard (DSS), *Federal Information Processing Standards Publication 186*. National Institute for Standards and Technology, 1994. 1, 31

[34] G. Frey, "Applications of arithmetical geometry to cryptographic construc-

tions," in *Proceedings of the Fifth International Conference on Finite Fields and Applications*. Springer-Verlag, 2001, pp. 128–161. 31

[35] R. Gallant, R. Lambert, and S. Vanstone, "Improving the parallelized Pollard lambda search on anomalous binary curves," *Mathematics of Computation*, vol. 69, no. 232, pp. 1699–1705, 2000. 69

[36] P. Gaudry, F. Hess, and N. P. Smart, "Constructive and destructive facets of Weil descent on elliptic curves." *Journal of Cryptology*, vol. 15, no. 1, pp. 19–46, 2002. 31

[37] 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. 45, 47, 136, 153

[38] S. W. Golomb and G. Gong, *Signal Design for Good Correlation: For Wireless Communication, Cryptography, and Radar*. Cambridge University Press, 2005. 12

[39] D. Hankerson, J. López, and A. Menezes, "Software implementation of elliptic curve cryptography over binary fields," in *CHES 2000: Cryptographic Hardware and Embedded Systems*, ser. LNCS 1965. Springer-Verlag, 2000, pp. 1–24. 112

[40] D. Hankerson, A. Menezes, and S. A. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2003. 9, 20, 21, 31, 33, 88, 152

[41] M. A. Hasan, "Look-up table-based large finite field multiplication in memory

constrained cryptosystems," *IEEE Transactions Computers*, vol. 49, no. 7, pp. 749–758, 2000. 114

[42] IEEE P1363, *IEEE Standard Specifications for Public Key Cryptography.* IEEE, 2000. 1, 19, 20, 29, 36

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

[44] T. Itoh and S. Tsujii, "Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$," *Electronics Letters*, vol. 24, no. 6, pp. 334–335, 1988. 114

[45] B. W. Johnson, "Fault-tolerant microprocessor-based systems," *IEEE Micro*, vol. 4, no. 6, pp. 6–21, 1984. 44

[46] M. Joye and C. Tymen, "Protections against differential analysis for elliptic curve cryptography — an algebraic approach," in *CHES 2001: Cryptographic Hardware and Embedded Systems*, ser. LNCS 2162. Springer-Verlag, 2001, pp. 377–390. 90

[47] 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. 25, 45

[48] 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 Comput-*

ing Systems, International Workshop, ser. LNCS 4462. Springer-Verlag, 2007, pp. 215–228. 3, 45, 122, 153

[49] E. W. Knudsen, "Elliptic scalar multiplication using point halving," in *ASIACRYPT 1999: Theory and Applications of Cryptology and Information Security*, ser. LNCS 1716. Springer-Verlag, 1999, pp. 135–149. 14

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

[51] ——, *A Course in Number Theory and Cryptography*. Springer-Verlag, 1994. 9

[52] 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, 27, 140

[53] 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, 27, 140

[54] P. K. Lala, *Self-Checking and Fault-Tolerant Digital Design*. Morgan Kaufmann Publishers, 2001. 107

[55] F. Lima, L. Carro, and R. A. da Luz Reis, "Designing fault tolerant systems into SRAM-based FPGAs," in *DAC*. ACM, 2003, pp. 650–655. 105

[56] S. Lin and D. J. Costello, *Error Control Coding*. Prentice-Hall, 2004. 123

[57] J. López and R. Dahab, "Improved algorithms for elliptic curve arithmetic in $GF(2^n)$," in *SAC 1998: Selected Areas in Cryptography*, ser. LNCS 1556. Springer-Verlag, 1998, pp. 201–212. 21

[58] ——, "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. 6, 25, 27, 29, 49, 84, 123, 124, 139, 154

[59] J. Lutz, "High performance elliptic curve cryptographic co-processor," Master's thesis, University of Waterloo, 2003, part of this work appears in [60]. 108, 110, 114

[60] J. Lutz and M. A. Hasan, "High performance FPGA based elliptic curve cryptographic co-processor," in *ITCC 2004: Information Technology Coding and Computing*, vol. 2. IEEE Computer Society, 2004, pp. 486–492. 181

[61] M. Maurer, A. Menezes, and E. Teske, "Analysis of the GHS Weil descent attack on the ECDLP over characteristic two finite fields of composite degree," *LMS Journal of Computation and Mathematics*, vol. 5, pp. 127–174, 2002. 31

[62] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*. Kluwer Academic Publishers, 1987. 9

[63] N. Meloni, "Fast and secure elliptic curve scalar multiplication over prime fields using special addition chains," Cryptology ePrint Archive, Report 2006/216, 2006. 93

[64] A. Menezes, *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1994. 9, 20, 29

[65] A. Menezes, T. Okamoto, and S. A. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field," *IEEE Transactions on Information Theory*, vol. 39, no. 5, pp. 1639–1646, 1993. 31

[66] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 2001. 11, 61

[67] 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

[68] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Mathematics of Computation*, vol. 48, pp. 243–264, 1987. 25

[69] 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. 25

[70] M. Otto, "Fault attacks and countermeasures," Ph.D. dissertation, University of Paderborn, 2004. 38

[71] J. H. Patel and L. Y. Fung, "Concurrent error detection in ALUs by recomputing with shifted operands," *IEEE Transactions on Computers*, vol. 31, no. 7, pp. 589–595, 1982. 44

[72] S. Pohlig and M. Hellman, "An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance," *IEEE Transactions on Information Theory*, vol. 24, pp. 106–110, 1978. 31

[73] J. M. Pollard, "Monte Carlo methods for index computation (mod p)," *Mathematics of Computation*, vol. 32, pp. 918–924, 1978. 31, 68

[74] D. A. Raynolds and G. Metze, "Fault detection capabilities of alternating logic," *IEEE Transactions on Computers*, vol. 27, no. 12, pp. 1093–1098, 1978. 44

[75] H.-G. Rück, "A note on elliptic curves over finite fields," *Mathematics of Computation*, vol. 49, no. 179, pp. 301–304, 1987. 19

[76] T. Satoh and K. Araki, "Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves," *Commentarii Mathematici Universitatis Sancti Pauli*, vol. 47, pp. 81–92, 1998. 31

[77] T. Satoh, B. Skjernaa, and Y. Taguchi, "Fast computation of canonical lifts of elliptic curves and its application to point counting," *Finite Fields and Their Applications*, vol. 9, pp. 89–101, 2003. 19, 58

[78] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley, 1995. 13

[79] R. Schoof, "Elliptic curves over finite fields and the computation of square roots mod p," *Mathematics of Computation*, vol. 44, no. 170, pp. 483–494, 1985. 58

[80] I. A. Semaev, "Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p," *Mathematics of Computation*, vol. 67, pp. 353–356, 1998. 31

[81] A. Shamir, "Method and apparatus for protecting public key schemes from timing and fault attacks," United States Patent 5,991,415, November 1999. 45

[82] D. Shanks, "Class number, a theory of factorization, and genera," in *Proceedings of the Symposium in Pure Mathematics*, vol. 20. American Mathematical Society, 1971, pp. 415–440. 31

[83] M. L. Shooman, *Reliability of Computer Systems and Networks*. Wiley, 2002. 107

[84] S. Skiribogatov and R. Anderson, "Optical fault induction attacks," in *CHES 2002: Cryptographic Hardware and Embedded Systems*, ser. LNCS 2523. Springer-Verlag, 2002, pp. 2–12. 3

[85] U.S. Department of Defense, *The Reliability Design Handbook*. The Reliability Analysis Center, Grifiss Air Force Base, New York, 1976. 100

[86] P. C. van Oorschot and M. J. Wiener, "Parallel collision search with cryptanalytic applications," *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, vol. 12, no. 1, pp. 1–28, 1999. [Online]. Available: citeseer.ist.psu.edu/vanoorschot96parallel.html 69

[87] J. von Neumann, *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*. Princeton University Press, 1956. 100

[88] D. Wagner, "Cryptanalysis of a probable secure CRT-RSA algorithm," in *ACM Conference on Computer and Communications Security*. ACM, 2004, pp. 82–91. 45

[89] L. C. Washington, *Elliptic Curves, Number Theory and Cryptography*. CRC Press, 2003. 19

[90] R. B. Wells, *Applied Coding and Information Theory for Engineers*. Prentice-Hall, 1999. 123

[91] 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. 4, 24, 39, 141, 157

[92] 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. 4, 39, 141, 157

[93] ——, "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. 185

[94] ——, "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 [93]. 45, 122, 153

[95] H. R. Zarandi, S. G. Miremadi, and A. Ejlali, "Dependability analysis using a fault injection tool based on synthesizability of HDL models," in *DFT 2003: Defect and Fault Tolerance in VLSI Systems*. IEEE Computer Society, 2003, pp. 485–492. 114