On Error Detection and Recovery in Elliptic Curve Cryptosystems

by

Abdulaziz Mohammad Alkhoraidly

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

Waterloo, Ontario, Canada, 2011

Abstract

Fault analysis attacks represent a serious threat to a wide range of cryptosystems including those based on elliptic curves. With the variety and demonstrated practicality of these attacks, it is essential for cryptographic implementations to handle different types of errors properly and securely. In this work, we address some aspects of error detection and recovery in elliptic curve cryptosystems. In particular, we discuss the problem of wasteful computations performed between the occurrence of an error and its detection and propose solutions based on frequent validation to reduce that waste. We begin by presenting ways to select the validation frequency in order to minimize various performance criteria including the average and worst-case costs and the reliability threshold. We also provide solutions to reduce the sensitivity of the validation frequency to variations in the statistical error model and its parameters. Then, we present and discuss adaptive error recovery and illustrate its advantages in terms of low sensitivity to the error model and reduced variance of the resulting overhead especially in the presence of burst errors. Moreover, we use statistical inference to evaluate and fine-tune the selection of the adaptive policy. We also address the issue of validation testing cost and present a collection of coherency-based, cost-effective tests. We evaluate variations of these tests in terms of cost and error detection effectiveness and provide infective and reduced-cost, repeated-validation variants. Moreover, we use coherency-based tests to construct a combined-curve countermeasure that avoids the weaknesses of earlier related proposals and provides a flexible trade-off between cost and effectiveness.

Acknowledgements

All praise be to Allah, the Creator and Sustainer of the worlds.

I would like to convey my great appreciation to my supervisor, Professor M. Anwar Hasan, for his guidance, support and patience throughout the time I spent in Waterloo. I would also like to thank Professor Douglas Stinson, Professor Guang Gong and Professor Pin-Han Ho for serving in the examination committee and for their interesting feedback. Moreover, I would like to thank Professor Vassil Dimitrov from the University of Calgary for taking the time to review this work as an external examiner and for his insightful comments.

I would also like to express my gratitude to all of my friends and colleagues in Waterloo. In particular, I would like to thank my friend Saad Alaboodi for many hours of fruitful discussions and for putting up with my stubbornness. I would also like to thank Agustín Domínguez-Oviedo for his insights that provided the basis for my work.

I owe an eternal debt of gratitude to my family for all the unconditional love, support and encouragement they provide me. My parents have been my lifelong mentors and role models, and there is still a lot that they can teach me. I pray that they live long and happy lives. My wife and best friend, Malak, has always been by my side despite the long distance and my slightly bumpy ride. Thank you, my love.

Last but not least, I would like to acknowledge the support of King Fahd University of Petroleum and Minerals, through the Saudi Arabian Cultural Bureau in Canada, that made this work possible.

To my parents and my wife

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In their 1976 paper titled “New Directions in Cryptography” [21], Whitfield Diffie and Martin Hellman introduced a method, based on the hard problem of finding discrete logarithms in a prime field, by which parties that have no shared secrets can establish a shared secret over an insecure channel, effectively inventing the notion of public-key cryptography. Since then, many public-key cryptosystems have been invented, but few have stood the tests of time and eager cryptanalysts. Among those are the Rivest-Shamir-Adleman (RSA) cryptosystem [61] and the Rabin cryptosystem [59], both based on the difficulty of factoring integers with large prime factors.

Elliptic curve public-key cryptography was introduced independently by Miller [51] and Koblitz [43]. The group of points on an elliptic curve defined over a finite field has many interesting properties that make it suitable for cryptographic applications. Most importantly, the discrete logarithm problem in this group appears to be significantly harder than problems like integer factorization and finding discrete logarithms in finite fields. As such, a comparable security level can be achieved using much smaller system parameters, which leads to faster computations and less storage requirements. This gives Elliptic Curve Cryptography (ECC) an advantage in terms of efficiency for both software and hardware implementation. The most important, as well as the most computationally intensive, operation in ECC is called the Elliptic Curve Scalar Multiplication (ECSM).

ECSM is conceptually similar to the modular exponentiation used in RSA.

Despite their theoretical security, elliptic curve cryptosystems are vulnerable to a variety of Side-Channel Attacks (SCAs) that target the implementation of the cryptosystem rather than its mathematical weaknesses and exploit the information leakage during the proper or improper use of the cryptosystem. SCAs are generally passive, i.e., the attacker observes a working cryptosystem without influencing its operations. Examples of side-channel attacks include timing attacks originally presented in [44] and power analysis attacks introduced in [45].

Fault Analysis Attacks (FAAs), on the other hand, are active attacks that utilize faults to influence the operation of the system. Faults can occur in a device either naturally or as a result of a deliberate action, and can be caused by one of many reasons. For instance, they can be caused by variations in standard operation conditions like supply voltage, clock frequency or operating temperature [7]. FAAs range in complexity and effectiveness from the simple to the highly sophisticated. In essence, they seek to expose the secret information partially or fully using invalid outputs that result from natural or deliberate faults. Most fault attacks on ECC attempt to move the computation from the secure curve to another, probably weaker, curve. This can be achieved by injecting faults in the curve parameters, base point, or during the scalar multiplication. Generally, these attacks have to be repeated many times to reduce the uncertainty in the guesses made by the attacker, and in most cases some exhaustive search is required. Examples of this class of fault attacks include those presented in [10, 5, 15, 27, 22].

On the other hand, the sign change attack presented in [11] targets the sign of an intermediate point in the scalar multiplication, and results in a faulty output point that still belongs to the original curve. Moreover, some fault attacks like the safe-error attack presented in [71] exploit a countermeasure against simple timing analysis. This attack targets algorithms where dummy intermediate variables are used to thwart timing and simple power analysis attacks, and where the output values are checked for errors. A fault is injected during an iteration of the algorithm and then it is observed whether the resulting error is detected or not. This reveals whether the operation that was targeted was a dummy operation, and consequently it exposes the corresponding bit of the key. Other attacks exploit the validation tests [73]. Since a comparison in a decisional validation test uses the status register, and particularly the zero flag, a fault injected in that register can probably cause the test to malfunction and allow an invalid output to pass.

A variety of countermeasures have been proposed to handle errors caused by faults. Some of these countermeasures attempt to prevent the injection of faults, e.g., metal shields and tamper-resistant chips [7]. Others attempt to detect the occurrence of errors in order to discard the faulty data, e.g., point validation [10], hardware and time redundancy along with randomization [24], and coherency checking [25]. Some countermeasures work by masking the faulty results and allowing valid results to pass, like infective computation [73]. It is also possible to extend a detection technique to allow for error recovery, e.g., dual modular redundancy with point validation [24].

1.1 Motivation and Scope

When error detection is incorporated in an ECSM implementation using validation tests, the test is commonly placed at the end of the scalar multiplication just before the final result is returned. This satisfies the purpose of error detection, but it can be easily seen that whenever an error occurs, all computations performed after the occurrence of the error and until its detection are wasteful as they are processing faulty data. Since the detection of an error will result in the rejection of the computation's outcome, it is sensible to attempt to detect an error as soon after its occurrence as possible.

A similar issue is encountered when error recovery is implemented using time redundancy, i.e., recomputation. As suggested in [23], the result of the scalar multiplication is validated at the end of the computation, and if an error is detected the whole computation is repeated. Clearly, full recomputation requires repeating all the valid operations that were performed before the error's occurrence, which can again be seen as wasteful.

For both of these observations, the intuitive solution is early detection of errors, which implies validating the computation's intermediate results. However, while ECSM algorithms generally allow the validation of internal state, it is not clear how often this validation should be performed. Moreover, it is not clear that the potential reduction in wasteful computations will offset the incurred overhead of repeated validation testing.

In order to investigate this issue, we have constructed cost models that can be used to select a validation frequency that minimizes the wasteful computations and the overhead of repeated testing for both error detection and recovery. However, it became apparent that the outcomes of these models are sensitive to the changes in the statistical error model and its parameters. Moreover, when the overhead of frequent validation is examined, it turns out that testing overhead constitutes a nontrivial portion of the total overhead.

In this work, we attempt to address both of these problems. We investigate approaches to reduce or eliminate the dependency of the validation frequency on the details of the statistical error model. Furthermore, we study ways to reduce the overhead of testing and provide a flexible trade-off between the cost of a validation test and its effectiveness. We limit our investigation to fault attacks and countermeasures that work at the point arithmetic level, which provides some degree of platform independence. While the ideas proposed and discussed in this work will probably be useful in implementing fault-tolerant arithmetic at the field and machine levels, we consider the attacks and countermeasures at these levels to be out of scope of our work.

1.2 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 provides the required background for this work. It starts with an introduction to elliptic curves and their use in cryptography. Then, fault analysis attacks that target elliptic curves are classified and reviewed. It follows by a discussion of related countermeasures and error handling techniques.

In Chapter 3, we address the use of fixed-block repeated validation in error detection and recovery. Cost models that describe the overhead associated with repeated validation are constructed for both error detection and recovery, and used to find optimal block sizes in terms of the expected cost. Moreover, the issue of block size sensitivity to the error model is addressed. In the case of error detection, we propose minimizing the block size for the worst-case rather than the average-case cost and show that this gives a block size that is independent of the statistical error model. In the case of error recovery, we propose two solutions, namely, minimizing a combination of the expected overhead and its standard deviation and minimizing the reliability threshold. Both of these solutions reduce the variations in the block size caused by variations in the parameters of the error model, and in effect, relax the need to know the error model accurately in advance.

In Chapter 4, we discuss the idea of using an adaptive, rather than fixed, block size. We show that the use of a simple adaptive policy that gives preference to smaller block sizes allows adaptive error recovery to approach the performance of the optimized fixed-block error recovery without being tied to the error model. The two approaches are compared under different scenarios and the advantages of the adaptive approach are confirmed with simulation. To evaluate the use of the aforementioned adaptive policy, we model the block size update as a statistical inference problem, and use a Bayesian inference procedure to learn the unknown parameters from the observed, incomplete data. Using this approach under two different statistical error models with unknown parameters, we show that an adaptive policy that gives preference to smaller blocks agrees with the inference outcome under both models. Moreover, this approach provides a way to fine-tune the level of sensitivity of the adaptive policy to more recent observations.

Chapter 5 addresses a different aspect of the problem of efficient error detection and recovery. It presents a collection of cost-effective, coherency-based tests that provide a trade-off between the cost of testing and the effectiveness of error detection. It begins by discussing coherency-based validation testing and then presents some alternative coherency-based tests. Each of these tests is evaluated in terms of cost and error detection effectiveness when used with a varying number of point validations. Moreover, it illustrates how the use of these tests in the context of repeated validation can help reduce their cost by eliminating operations that are common with the underlying scalar multiplication. Infective variants of these tests are discussed as well. Furthermore, the use of coherency-based validation tests in a combined-curve countermeasure is discussed. The resulting countermeasure avoids many of the shortcomings of earlier, related countermeasures.

Chapter 6 concludes this work by providing a summary of the results in the preceding chapters and discussing some related open problems.

1.3 Summary of Contributions

In what follows, we outline the contributions of this work:

- We present detailed cost models for error detection and recovery using fixed-block repeated testing, and illustrate the use of these models to find block sizes that are optimized for low cost, high reliability or insensitivity to the parameters of the error model
- We propose adaptive error recovery and illustrate its advantages relative to the fixed-block approach under a variety of error models. We also use statistical inference to evaluate adaptive policies and set their parameters to achieve the desired level of sensitivity.
- We present and evaluate a collection of cost-effective, coherency-based validation tests, and discuss their infective and reduced-cost variants. We also introduce a countermeasure that employs coherency-based validation in the setting of combined curves to provide a highly flexible trade-off between its cost and detection effectiveness.

Chapter 2

Background

This chapter provides the foundation on which the following chapters are built. This includes elliptic curves and their use in cryptography, fault attacks on elliptic curve cryptosystems, and the various known countermeasures against these attacks. Section 2.1 starts by giving a background on elliptic curves and their use in cryptography. It covers the elliptic curve group, point arithmetic and the elliptic curve scalar multiplication. It also briefly discusses the security of elliptic curve cryptosystems. Then, Section 2.2 reviews fault analysis attacks targeting elliptic curve cryptosystems. After a brief overview of fault injection methods, it discusses general techniques to detect and handle faults. Then, a review of known fault attacks is presented. Attacks are classified into invalid-curve attacks, the sign change attack, and attacks that target dummy or validation operations. For each attack, a description of the attack is presented as well as an outline of its countermeasures. Section 2.3 summarizes the known countermeasures for fault attacks on elliptic curve cryptosystems. Countermeasures are classified into prevention, detection, masking and recovery techniques. For each of these classes, examples are given and discussed. The material in this chapter has appeared in [2].

2.1 Elliptic Curves

This section aims to provide a brief overview of the mathematical concepts often referred to throughout this document. Reference books in abstract algebra, like [54], and in cryptography, like [32] and [16], can be consulted for a more extensive treatment of these concepts.

2.1.1 Preliminaries

A *group* $(G, \oplus)$, written additively, is a set G associated with a binary operation $\oplus$ defined on G such that it is closed, associative, has an identity element, and such that each element of G has an inverse. Moreover, the group is called commutative or abelian if $\oplus$ is commutative. For a finite group, the number of elements is referred to as the *group order* or *cardinality*. On the other hand, the *order of an element* a in a finite group G, denoted by $\operatorname{ord}_G(a)$, is the smallest integer c such that

$$ca = \underbrace{a \oplus a \oplus \cdots \oplus a}_{c \text{ times}} = 0$$

where 0 commonly represents the identity element of $\oplus$. The order of a group is divisible by the order of any of its elements.

A *ring* $(R, \oplus, \otimes)$ is a set R associated with two binary operations, $\oplus$ and $\otimes$, defined on R such that $(R, \oplus)$ is a commutative group and $\otimes$ is closed, associative and distributive over $\oplus$. Furthermore, the ring is commutative if $\otimes$ is commutative. In a ring R, if the operation $\otimes$ has an identity element the ring is called a *ring with identity*. The operation $\otimes$ is usually written multiplicatively and its identity, if one exists, is denoted by 1. An example of a commutative ring with identity is the set $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ under addition and multiplication modulo n.

In a ring with identity R, if the nonzero elements form a commutative group under $\otimes$, then R is a *field*. In other words, a field can be seen as a commutative group with respect to two binary operations such that one operation is distributive over the other. While the set of rational numbers $\mathbb{Q}$ is an example of a field, $\mathbb{Z}$ is an example of a ring that is not a field since the only elements that have multiplicative inverses are 1 and -1. A field that has a finite number of elements is called a *finite field*, and can be either a prime field or an extension field. As the name indicates, a *prime field* $\mathbb{F}_p$ has a prime cardinality p. An *extension field*, on the other hand, has a cardinality of p^d, where p is prime and $d > 1$ is an integer. Such a field is created by extending the prime field $\mathbb{F}_p$ where d denotes the extension *degree*. The *characteristic* of both $\mathbb{F}_p$ and $\mathbb{F}_{p^d}$, denoted by $\mathrm{char}(\mathbb{F}_p)$, is p. An interesting fact is that all finite fields of the same cardinality are *isomorphic*, i.e., have the same structure even if they are represented differently. In other words, these fields can be made identical by renaming their elements.

2.1.2 The Elliptic Curve Group

An elliptic curve E over a field $\mathbb{F}$, whose algebraic closure is denoted by $\overline{\mathbb{F}}$, is the set of points (x, y), $x, y \in \overline{\mathbb{F}}$ that satisfy the Weierstrass equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.1}$$

where the coefficients $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}$ and such that the curve is nonsingular, i.e., the partial derivatives do not vanish simultaneously at any point. The set of points $(x, y) \in \mathbb{F} \times \mathbb{F}$ that satisfy the curve equation, along with the point at infinity $\mathcal{O}$, is denoted by $E(\mathbb{F})$.

The full Weierstrass equation can be simplified depending on the characteristics of the underlying field, $\mathbb{F}$. For prime fields, and when $\mathrm{char}(\mathbb{F}) \neq 2, 3$, (2.1) can be simplified to

$$E : y^2 = x^3 + ax + b \tag{2.2}$$

where $a, b \in \mathbb{F}$. In a binary field, i.e., when $\mathrm{char}(\mathbb{F}) = 2$, and assuming that $a_1 \neq 0$ and that E is non-supersingular, (2.1) can be simplified to

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

where $a, b \in \mathbb{F}$.

The *quadratic twist* of an elliptic curve $E(\mathbb{F}_q) : y^2 = x^3 + ax + b$ is a curve of the form

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

where $\alpha \in \mathbb{F}_q$ is a quadratic nonresidue, i.e., an element that has no square root in $\mathbb{F}_q$. The number of points on an elliptic curve $E(\mathbb{F}_q)$ is denoted $\#E(\mathbb{F}_q)$, and its value is bounded by the Hasse theorem [66].

Theorem 1 (Hasse Theorem) *Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$. Then, $\#E(\mathbb{F}_q) = q + 1 - t$, where $|t| \leq 2\sqrt{q}$. The interval $[q + 1 - 2\sqrt{q}, q + 1 + 2\sqrt{q}]$ is called the Hasse interval while t is called the trace of E over $\mathbb{F}_q$.*

The points on an elliptic curve, together with the point at infinity $\mathcal{O}$, form an abelian group under the operation of *point addition*, which has an intuitive geometric interpretation illustrated in Figure 2.1. When $\mathrm{char}(\mathbb{F}) \neq 2, 3$ and $P \neq -Q$, we can find $R = (x_3, y_3) = P + Q$ as follows.

$$\begin{aligned} \mathfrak{m} &= \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{if } P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1}, & \text{if } P = Q \end{cases} \\ x_3 &= \mathfrak{m}^2 - 2x_1 \\ y_3 &= \mathfrak{m}(x_1 - x_3) - y_1 \end{aligned} \tag{2.5}$$

Each point operation on an affine elliptic curve requires a field inversion, which is significantly more expensive in general than a field multiplication. To address this problem, various projective coordinates were introduced, e.g., [17], and are often employed to reduce the number of field inversions at the cost of more field multiplications and storage space. Different projective coordinates have different costs in terms of field operations. Moreover, it has been shown that adding points represented in different projective coordinates can be more efficient than addition of points in the same coordinate system [16].

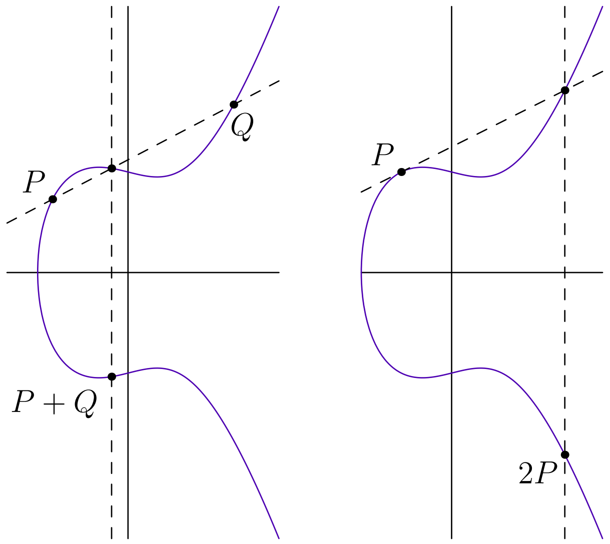

Figure 2.1: Geometric interpretation of point addition and doubling.

The security of ECC is based mainly on the hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP), which can be defined as finding the scalar k given $P, kP \in E$. Since all the known solutions for this problem in general elliptic curves have an exponential complexity, parameters should be of appropriate sizes to make the instance intractable. Moreover, the ECDLP is easier, i.e., there exist sub-exponential solutions, for some special elliptic curves. As such, these curves should be avoided in ECDLP-cased cryptography.

2.1.3 Elliptic Curve Scalar Multiplication

Scalar multiplication, the operation of repetitively adding a point to itself on an elliptic curve, is the most important primitive in ECC and its execution time dominates the time required to perform ECC operations. This operation is analogous in many ways to the modular exponentiation operation employed in RSA and similar systems.

A naive way to perform a scalar multiplication is repeated addition, which requires a number of group operations that is proportional to the group size. A much better solution is the double-and-add algorithm, analogous to the square-and-multiply algorithm

Algorithm 1 Binary double-and-add scalar multiplication

```
Input:  $P \in E$, $k = \sum_{i=0}^{n-1} k_i 2^i$
Output:  $kP$
1: $Q \leftarrow \mathcal{O}$
2: for $i = n-1$ downto $0$ do
3:     $Q \leftarrow 2Q$
4:     if $k_i = 1$ then
5:         $Q \leftarrow Q + P$
6:     end if
7: end for
8: return $Q$
```

for modular exponentiation. Algorithm 1 illustrates a basic variant of the double-and-add method. This approach requires a number of group operations that is proportional to the logarithm of the group size.

It can be seen from Algorithm 1 that a doubling operation is required in every iteration, while an addition operation is required only when the corresponding bit of the scalar k is 1, which is true in half of the iterations on average. It is possible to reduce the cost of the scalar multiplication by reducing the number of point additions, which can be achieved by recoding the scalar k. For example, the Non-adjacent Form (NAF) representation, with the property that no two nonzero digits are adjacent, can be used to reduce the number of 1's in the scalar. The simplest instance is the 2-NAF, which uses the digits 0, 1, and -1 and ensures that no two consecutive digits are nonzero. The use of 2-NAF representation reduces the number of additions from $n/2$ to $n/3$ on average [53].

Double-and-Add-Always Variant. It can be easily seen that the iterations of Algorithm 1 take different times for different values of k_i, which can be exploited by simple power or timing analysis attacks to determine the secret scalar k. To solve this problem, the computations performed should not depend on the secret data and branching should

Algorithm 2 Double-and-add-always scalar multiplication [18]

```
Input: $P \in E$, $k = \sum_{i=0}^{n-1} k_i 2^i$
Output: $kP$
1: $Q[0] \leftarrow P$
2: for $i = n - 2$ downto $0$ do
3:     $Q[0] \leftarrow 2Q[0]$
4:     $Q[1] \leftarrow Q[0] + P$
5:     $Q[0] \leftarrow Q[k_i]$
6: end for
7: return $Q[0]$
```

be avoided. This results in regular iterations as illustrated in Algorithm 2 presented in [18]. More recently, other regular scalar multiplication algorithms have been introduced, e.g., in [38].

Montgomery Ladder. It has been shown that, for some elliptic curves, the y-coordinate is not essential in the point addition and doubling operations [52], so the x-coordinate of the resulting point, $P + Q$, can be computed from the x-coordinates of the points P, Q, and $Q - P$. In particular, let $P = (x_1, y_1)$, $Q = (x_2, y_2)$ and $Q - P = (x_3, y_3)$ be points on the elliptic curve $E : y^2 = x^3 + ax + b$, then the x-coordinates of $P + Q = (x_4, y_4)$ and $2Q = (x_5, y_5)$ can be computed by the following equations.

$$\begin{aligned} x_4 &= \frac{2\left(x_1 + x_2\right)\left(x_1 x_2 + a\right) + 4b}{\left(x_1 + x_2\right)^2} - x_3 \\ x_5 &= \frac{\left(x_1^2 - a\right)^2 - 8bx_1}{4\left(x_1^3 + ax_1 + b\right)} \end{aligned} \tag{2.6}$$

It follows that the result of kP can be found by calculating a sequence of pairs (Q, H), which has the property that $H - Q = P$. Algorithm 3 demonstrates the steps of a Montgomery ladder for scalar multiplication. It can be noted from Algorithm 3 that the

Algorithm 3 Montgomery ladder scalar multiplication [52]

```
Input:  $P \in E$, $k = 2^{n-1} + \sum_{i=0}^{n-2} k_i 2^i$
Output:  The $x$-coordinate of $kP$
1: $Q[0] \leftarrow P$, $Q[1] \leftarrow 2P$
2: for $i = n-2$ downto $0$ do
3:     $Q[\overline{k_i}] \leftarrow Q[0] + Q[1]$
4:     $Q[k_i] \leftarrow 2Q[k_i]$
5: end for
6: return $Q[0]$
```

Montgomery ladder is a variant of the double-and-add-always method, and hence can be used as a countermeasure against simple timing attacks [37, 41]. Moreover, in [49], and more recently in [69], it was shown that this algorithm is particularly efficient for curves defined over binary fields.

2.2 Fault Analysis Attacks on Elliptic Curve Cryptosystems

2.2.1 Faults in Digital Systems

In general, it is important for a digital system to be fault-free and consistently give the correct results. Faults can occur for a variety of natural and artificial reasons, and various methods have been proposed to counter their effects on the performance and reliability of the system. Fault detection and tolerance are even more important for cryptosystems given the existence of fault attacks that can exploit faults to discover secret information and threaten the security of the whole system.

Faults can occur in a device either naturally or as a result of a deliberate action, and can be caused by one of many reasons. In general, variations in standard operation conditions can be used effectively to inject faults in a system. For example, the variation in the supply voltage or the clock frequency can disrupt the execution and cause the processor to skip instructions or disrupt input/output operations. Moreover, exposing the device to temperatures outside its operational range can cause random modifications of the memory and inconsistencies in memory access. It is also possible to inject faults more accurately using photoelectric effects that are inherent in all electric circuits. The exposure to photons induces currents in the circuit that can disrupt normal operation. In effect, targeting and timing can be made more precise using lasers in fault injection. Faults can be injected in packaged circuits without removing the packaging using X-rays and ion beams [7].

Faults in electronic circuits can either be permanent or transient. Permanent faults are caused by intentional or unintentional defects in the chip. As the name indicates, they have a permanent effect on the behavior of the circuit. On the other hand, a transient fault does not cause a permanent change in the behavior of the circuit. Such faults are caused by local ionization that induces a current which can be misinterpreted by the circuit as an internal signal.

Several solutions have been devised to avoid or detect faults, or prevent the attempt to inject them. Other solutions help to recover from the occurrence of faults and produce a correct output in spite of their existence. Some of these methods are implemented in hardware while some can be implemented in software. The main countermeasure against faults and errors is the use of redundancy in the design, which makes it possible to detect erroneous results and behaviors. It also may permit the recovery from faults. A common form of redundancy is *hardware redundancy*, which entails replicating some part of the hardware to prevent the existence of a single point of failure. Another form of redundancy is *time redundancy*, which amounts to repeating the computation or a part of it to confirm the earlier results and detect transient faults. A third form of redundancy is *information redundancy*, which is commonly employed in data communication through error detecting and correcting codes. The principle behind information redundancy is the use of more bits to represent the data than is actually necessary. This way, some of the representable bit patterns do not correspond to valid data and can be used to detect and correct errors. It is also possible to combine two or more types of redundancy in a single scheme to get the advantages of different types of redundancy. A summary of the known countermeasures against fault attacks in elliptic curve cryptosystems is the subject of Section 2.3.

2.2.2 Invalid-curve Fault Attacks

The choice of the elliptic curve and the underlying field is an important one as it significantly influences the security of the system. The general aim of invalid-curve fault attacks is to move the computation from a secure curve to a weaker one, enabling the attacker to use known mathematical attacks against the faulty outputs. This can be achieved by targeting either the system parameters or the running computation. In both cases, there are known countermeasures that can be used to detect the attack and prevent the faulty output.

2.2.2.1 Targeting the Base Point

The base point is one of the key parameters in a scalar multiplication operation. It is also relatively easy to target as it is commonly presented to the system as an input. Various known fault attacks target the base point. Some of which assume that the attacker knows the faulty value of the base point while others relax this assumption.

Known or Chosen Faulty Base Point. In the attacks introduced by [10], the representation of a point P on a strong elliptic curve E is modified as a result of a fault to move the computation to a different, often weaker, curve E'. The resulting faulty output values can be used to deduce partial information about the secret key. Usually, the attack has to be performed repeatedly since in most cases the guessed values are not unique.

Attack Description. Let E be a strong elliptic curve defined over a finite field $\mathbb{F}$ as

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

and let P and $Q = kP$ be two points on E. To be able to mount this attack, suppose that the device does not check whether P and Q are actually on E.

According to the ANSI X9.63 and IEEE 1363 standards, a_6 is not used in the addition operation. It follows that for a point $P' = (x', y')$ with $x', y' \in \mathbb{F}$ and $P' \notin E$, the calculation of $Q' = kP'$ occurs over the curve $E'(a_1, a_2, a_3, a_4, a_6')$ where

$$a_6' = y'^2 + a_1x'y' + a_3y' - x'^3 - a_2x'^2 - a_4x'$$

instead of the original curve E. If P' is chosen in such a way that E' is a cryptographically weak curve, i.e., P' has a relatively small order over E', then the value of k modulo $\mathrm{ord}(P')$ can be found by solving a DLP over the subgroup generated by P'. The relatively small size of this group enables the use of the known general DLP computation algorithms like Shank's Baby-step-giant-step algorithm [65], or Pollard's rho and Pollard's lambda algorithms [58]. It is also possible to select P' such that it has a smooth order, i.e., composite with small factors, over E'. In this case, Pohlig-Hellman algorithm [57] can be used to break the bigger DLP into smaller problems in the subgroups of prime order, which can be solved independently.

This process can be repeated until sufficient residues of k are collected, and then Chinese remaindering is used to recover k. This attack runs in polynomial time, and has been extended to standardized EC-based cryptographic protocols in [5].

Unknown Faulty Base Point. It is also possible to exploit a fault that results in an unknown faulty base point. Biehl et al. [10] discussed this as an extension to their attack presented earlier where they assume that the faulty base point has an error in one unknown bit position. Ciet and Joye [15] extended this attack to any number of bit errors in the base point.

Attack using a Faulty Base Point with One Bit Error [10]. Suppose that the device checks whether the base point P lies on E before starting the computation, and assume a fault can be injected in P in an unknown position right between the test and the computation. Again, the modified point P' will lie on a curve E' with different, yet unknown, a_6' value. Using the incorrect output $Q' = kP'$, the value of a_6' can be computed. For each of the possible values of P' that lie on E', we find k' such that $Q' = k'P'$ by solving a DLP on E'. Then, we proceed to compute k as in the more basic attack discussed earlier.

This attack applies also to the Elliptic Curve Digital Signature Algorithm (ECDSA). Since the faulty base point can not be given as an input as it is fixed by the protocol, the attacker injects a fault in the base point P to replace it by P' at the beginning of the scalar multiplication. Then, the algorithm computes $r' = (kP')_x \bmod p$, where $(\cdot)_x$ is the x-coordinate of the point and p is the prime order of P. It then computes $s' = k^{-1}(H(m) + dr') \bmod p$ where $1 < k < p - 1$ is a random number, $H(m)$ is the hash of the message m, and d is a the secret key. The resulting signature is the pair (r', s'). The attacker can then find the faulty curve E', on which P' lies, and use r' to find a small set of candidates for $(kP')_x$. Then, the corresponding y-coordinates are found using the equation of E'. Assuming the DLP on E' is weak, and given the candidates for kP', the attacker solves the DLP on E' to find candidate values for k, and then uses those to compute candidates for the secret key as $d = r'^{-1}(s'k - H(m)) \bmod p$.

Attack using a Faulty Base Point with Multiple Bit Error [15]. Let E be an elliptic curve over $\mathbb{F}$ and let $P = (x_P, y)$ be the base point. Faults can be assumed to occur in either or both of the coordinates of P. For example, assume that the value stored for the x-coordinate, x', is corrupted in some unknown bit positions and let $P' = (x', y)$. Then, $Q' = kP'$ can be computed and will lie as before on the curve E', which shares all the parameters of E except a_6. The corresponding value for E', a_6', can be computed from the coordinates of Q' as

$$a_6' = y_{Q'}^2 + a_1 x_{Q'} y_{Q'} + a_3 y_{Q'} - x_{Q'}^3 - a_2 x_{Q'}^2 - a_4 x_{Q'}$$

Then, we know that x' is a root in $\mathbb{F}$ of

$$x^3 + a_2 x^2 + (a_4 - a_1 y)x + (a_6' - y^2 - a_3 y)$$

since $P' \in E'$.

Assuming that r, the order of P' in E', is small, the root of the above polynomial with the least Hamming distance from the original x_P can be used as a candidate for x'. Assuming that the DLP on E' is weak, the Pohlig-Hellman reduction can be used to obtain a candidate for k.

Note that this attack applies in a similar fashion when errors are injected in the y-coordinate instead. However, when both coordinates are faulty, it becomes harder to recover the faulty point P' as the attacker only knows that it lies on E'. One way to recover candidate values of P' is to perform the scalar multiplication repeatedly using modified copies of the scalar as illustrated in more details in [15].

Attacks on the Montgomery Ladder. Fouque et al. [27] have proposed a fault attack on the Montgomery ladder algorithm over prime fields. Their work uses the fact that the y-coordinate is not used in the scalar multiplication, and hence, the faulty computation can leave the original curve and move to its quadratic twist. More specifically, since the y-coordinate is not used in the Montgomery ladder algorithm [41] the computation applies also to the set of points (x, y) with $x \in \mathbb{F}_p$ and $y \in \mathbb{F}_{p^2}$ that form the twist of the original elliptic curve E, denoted by $\widetilde{E}$.

The order of an elliptic curve and of its twist are roughly the same. In practice, an elliptic curve is chosen to have a group with an almost prime order, but the group order of the twist is not necessarily prime. In fact, among the five NIST-recommended curves over prime fields only the curve denoted by P-384 has a twist with a prime order. For the remaining curves, the group orders of their twists are composite and are easier to attack.

In [27], two attacks have been presented. The basic attack is when the adversary is able to choose the input point P and the implementation does not use point validation at the end of the scalar multiplication. The second attack assumes that the attacker can not select P and that the implementation validates the resulting point at the end of the scalar multiplication. In this case, the attacker needs to inject two faults. First, the attacker injects a fault into the x-coordinate of the input point P. Since half of the x's correspond to points that lie on the original curve while the other half lies on the twist, the resulting value will correspond to point on the twist $\widetilde{E}$ with probability $\frac{1}{2}$. Second, at the end of the computation just before the point validation the attacker needs to inject a fault into the x-coordinate of the result. Note that due to the missing y-coordinate, point validation will accept any x for which $x^3 + ax + b$ is a square, i.e., with probability $\frac{1}{2}$. In effect, the point validation can be bypassed and the faulty output can be obtained by the attacker with probability $\frac{1}{4}$.

A similar attack that targets the Montgomery ladder over a binary field was proposed independently in [22]. This attack takes advantage of the fact that the parameter a in (2.3) is not used during the scalar multiplication, and the authors adopt the same single-bit flip fault model proposed in [13] which has been shown to be practical [67]. It can be shown that for a fixed value of the curve parameter b there are only two isomorphic classes of curves, one for each value of $\mathtt{Tr}(a)$, where $\mathtt{Tr}(\cdot)$ is the trace function. It follows that it is possible to define two elliptic curves, E_0 and E_1, one for each of these isomorphic classes:

$$\begin{aligned} E_0 : y^2 + xy &= x^3 + b \\ E_1 : y^2 + xy &= x^3 + x^2 + b. \end{aligned} \tag{2.7}$$

It is known that the orders of E_0 and E_1 are roughly the same. It is not necessary, however, that both curves are strong (in the cryptographic sense). In fact, among the ten NIST-recommended binary curves, there is only one for which the orders of both E_0 and E_1 are almost prime, namely, the curve K-283.

The key idea behind this attack is to produce an invalid result from the computation being performed on the weaker curve of the pair E_0 and E_1. It is assumed that the degree of extension is odd, i.e., $\mathtt{Tr}(1) = 1$ in the field. The attack starts by injecting a fault into the x-coordinate of the input point $P = (x_P, y_P) \in E(\mathbb{F}_{2^m})$ of a device computing the scalar multiplication. If the resulting finite field pair after the fault injection is known and the result $\widetilde{Q} = k\widetilde{P} = (\widetilde{x}_Q, \widetilde{y}_Q)$ is released it is possible to obtain the full scalar with a high probability of success. Moreover, a variant of this attack has been presented where the faulty finite field pair $\widetilde{P}$ is unknown and two computations with the same scalar are obtained. In such a case, the scalar can also be obtained.

Countermeasures. It is commonly advised to check the correctness of the input points, i.e., whether they belong to the original curve. One way to achieve that is to compute and verify the value of a_6 using the coordinates of P. It is advisable to avoid NIST curves that have cryptographically-weak twists. It is also important to ensure that the output points lie on the original curve. Another countermeasure, which has been reported in [22] and extends the approach presented in [29] for RSA, uses the invariant in the Montgomery algorithm (Algorithm 3), i.e., $Q[1] - Q[0] = P$ to validate the variables before results are returned.

2.2.2.2 Targeting the System Parameters

It is also possible to target the parameters of the system, e.g., the field representation or the parameters of the curve equation. This will generally lead to the computation being performed on a weaker curve or a weaker curve-field combination.

Faulty Field Representation. Faults can be injected in the field parameters, either in storage or in transit. This attack, introduced in [15], exploits this fact. Let E be a curve defined over a prime field $\mathbb{F}_p$ of characteristics other than 2 and 3. Assume that a bit error is injected in p to give the almost similar value p' and that all field operations will then be performed modulo p' instead. In particular, the values of P, Q, a and b will be represented modulo p' as P', Q', a' and b', respectively.

Since Q' satisfies the equation of E', it follows that

$$b' \equiv {y'}_Q^2 - {x'}_Q^3 - a' \equiv b \equiv y^2 - x^3 - a \mod p'$$

Hence, $p'|D$ where $D = \left|{y'}_Q^2 - {x'}_Q^3 - a' - (y^2 - x^3 - a)\right|$ and p' can be revealed through factoring D as the product of factors that has the shortest Hamming distance from p.

Using these factors, the value of k can be computed by solving a set of small DLPs and then Chinese remaindering.

Moreover, p' can be found more efficiently when p is a (generalized) Mersenne prime. *Generalized Mersenne primes* are primes of the form

$$p = 2^{\omega_0} + \sum_{i=1}^{B} \pm 2^{\omega_i} \tag{2.8}$$

where B is typically small [68]. These primes are commonly used to enable highly efficient field reduction and to significantly reduce storage requirements for p. For example, when $B \leq 4$, as is the case for all NIST prime curves [26], p can be written as

$$p = 2^{\omega_0} + \sum_{i=1}^{3} \sigma_i 2^{\omega_i} + \sigma_4 \tag{2.9}$$

where $\sigma_i \in \{-1, 0, 1\}$ and $\sigma_4 \neq 0$. Then, p can be stored efficiently by storing the values of the ω_i's and σ_i's. Since an error injected in p will affect any of these quantities but will not affect the form given by (2.9), the number of possible values of p' that can result from factoring D will significantly decrease as only candidates that satisfy (2.9) need to be considered.

The same principle applies to binary fields [15], where pentanomials (and trinomials if any exists) are used as they enable efficient reduction and reduced storage requirements. It is known that, for all $d \leq 1000$, there exists an irreducible pentanomial that can be used to represent the elements of $\mathbb{F}_{2^d}$ [1]. Since a pentanomial can be stored efficiently by only storing its exponents, an error will modify one or more of the exponents but will not affect its form. It follows that all candidates for the faulty reduction polynomial that do not have the same form can be discarded, which significantly reduces the search space.

Faulty Curve Parameters. Errors injected in one or more of the curve parameters can be exploited in a similar way to enable a more efficient attack [15]. In general, the parameter a_6 in (2.1) is not targeted since it does not affect the final result as it is not used during the point addition operation. Assuming that only one of the other parameters is modified by an error, the valid value of the point P and the faulty resulting point $\widetilde{Q}$ can be used to solve for the value of the randomly modified parameter and the associated $\widetilde{a_6}$. In effect, E' is known, and if $\mathrm{ord}(P)$ on E' is small or smooth, the DLP can be solved on E' to get k modulo $\mathrm{ord}(P)$.

Countermeasures. Checksums can be used to avoid faults in system parameters. For each of the system parameters, the checksum should be computed and checked after reading it from the memory and before outputting the resultant point.

2.2.2.3 Targeting Intermediate Variables

In addition to the attacks discussed earlier, it is also possible to exploit random faults that occur during the scalar multiplication. In particular, faults injected in intermediate variables lead to faulty outputs that can be used, along with the correct result, to guess parts of the scalar. The following attack, introduced in [10], illustrates this.

Exploiting Random Faults during the Computation. Assume that E is defined over a field $\mathbb{F}_q$, and that the binary algorithm is used to perform the scalar multiplication, as in Algorithm 1. Also, assume that the attacker can repeatedly input a point P and induce a fault during a specific iteration of computation, and that the correct result $Q = kP$ is known. Let Q_i denote the values of Q at the i^{th} iteration. During the computation, a fault is injected in a random iteration j to modify Q_j to Q'_j and get Q' as an output. Then, the values of Q, Q' and j can be used to determine the intermediate value Q'_j, which can be used to guess the higher $n - j$ bits of k. The process can then be repeated going downwards through the bits of k.

It has been shown in [10] that finding a secret key of length n bits using this attack requires $O(n \log n)$ faulty scalar multiplications and $O(n \log n)$ bit operations for subsequent calculations. It is also possible to decrease the accuracy of random fault injection, i.e., inject faults in blocks of at most m consecutive iterations rather than in a specific iteration. The choice of the block size m presents a trade-off between the number of register faults required and the time needed to analyze the faulty results.

Countermeasures. As seen earlier, point validation is essential when fault attacks are considered. In particular, output points should be validated and any point that does not belong to the original curve should be discarded.

2.2.3 Sign Change Fault Attack

Earlier fault attacks on ECC worked by inducing faults in a way that would move the computation to a different, probably weaker, elliptic curve. This can be achieved by either modifying the base point, an intermediate point, or a parameter of the curve. However, this can be easily countered by verifying the correctness of the parameters and that the resulting point belongs to the original curve.

The attack described in [11] does not move the computation to a different curve, but rather results in a faulty point on the original curve. By collecting enough of these faulty results, the secret can be recovered in expected polynomial time. It follows that point validation is not an effective countermeasure against this attack. Actually, it may help the attacker to remove useless faulty points, i.e., points that fall off the curve, and hence it makes a less precise attack more effective.

Attack Description. As the name indicates, this attack involves changing a sign of an intermediate variable during the scalar multiplication, which can be achieved by replacing an intermediate point with its inverse or by changing a digit of the NAF-encoded scalar from 1 to -1. Applying this attack repeatedly enables the attacker to recover the scalar k starting from the least significant bit. A simpler variant of the attack, where it is assumed that the attacker can target a specific iteration, is described here. However, the attack can be extended to allow for uncertainty in fault injection [11].

Suppose that basic double-and-add algorithm (Algorithm 1) is used to compute $Q = kP$, and that the attacker has the correct value for Q. It is assumed that the attacker can inject a temporary sign change fault in the doubling step $Q \leftarrow 2Q$ so it effectively becomes $Q \leftarrow -2Q$. The attack starts by targeting the last iteration of the algorithm where $i = 0$. This results in the following faulty output

$$\widetilde{Q} = -\sum_{i=1}^{n-1} 2^i k_i P + k_0 P = -Q + 2k_0 P$$

which lies on the original curve and can not be detected by point validation. Moreover, depending on whether k_0 is equal to 0 or 1, $\widetilde{Q}$ will be equal to either $-Q$ or $-Q + 2P$, respectively, which easily reveals k_0 to the attacker. In general, if a sign change fault is injected in iteration i, the resulting faulty point can be written as

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

where only one of the $i + 1$ least significant bits of k is unknown, namely, k_i. As a result, $\widetilde{Q}$ will take one of two values.

$$\widetilde{Q} = \begin{cases} -Q + 2\sum_{j=0}^{i-1} k_j 2^j P & k_i = 0 \\ -Q + 2P + 2\sum_{j=0}^{i-1} k_j 2^j P & k_i = 1 \end{cases}$$

The attack can be generalized to cases where the attacker may not know the precise iteration in which the change happened. The idea is to recover the bits of k in blocks of $1 \leq r \leq m$ bits, where m is chosen to control a trade-off between the required exhaustive search and the number of faulty results required to give a success probability of at least $\frac{1}{2}$.

Countermeasures. The sign change fault attack is not applicable to Montgomery's scalar multiplication algorithm [52] since it does not use the y-coordinate, which prevents the attacker from changing the sign of intermediate points. Moreover, randomizing the scalar, e.g., by splitting, is effective since the same fault at the same stage of the algorithm would generate different result every time.

Another countermeasure to the sign change attack has been proposed in [11], and extends a countermeasure presented in [64] for RSA. The idea is to employ another curve E_t defined over a smaller field $\mathbb{F}_t$ for some prime t. Moreover, a base point P_t is chosen to have a large order within E_t. A combined curve E_{pt} and a new base point P_{pt} can be constructed using the Chinese Remainder Theorem, and then the scalar multiplication is performed once on each of E_{pt} and E_t. If the results do not match modulo t, they are rejected.

A similar countermeasure has been proposed in [6]. Most notably, in this countermeasure, the integer t, curve E_t and point P_t are selected randomly and t is not necessarily prime. However, it has been shown in [39] that this setup allows a non-negligible proportion of faults to pass undetected, and that a non-random setup like the one proposed in [11] is significantly more secure.

A countermeasure based on coherency checking is described [23]. This countermeasure extends the coherency-based countermeasure presented in [29] for RSA signature algorithms. As illustrated in Algorithm 4, this countermeasure is based on the right-to-left double-and-add-always scalar multiplication algorithm. It includes within it point validation of resulting points, and since it is a double-and-add-always algorithm, it is inherently resistant to simple power and timing analysis attacks. It is worthy to mention, however, that only single-fault attacks are considered in this algorithm. By tracing the iterations in this algorithm, we notice that in an error-free run, Q_1 is expected to have the value kP, while Q_0 will have the value $\overline{k}P$, where $\overline{k} = 2^n - 1 - k$. Since Q_2 has the value 2^nP, it follows that $Q_0 + Q_1 + P = Q_2$. As shown in [23], a sign change fault in any of the intermediate values can be detected by checking this invariant. This countermeasure incurs significantly less overhead than the ones based on combined curves.

Algorithm 4 Right-to-left double-and-add-always scalar multiplication algorithm with point validation and coherency checking. [23]

```
Input:  $P \in E$, $k = \sum_{i=0}^{n-1} k_i 2^i$
Output: $kP$
 1: $Q_0 \leftarrow \mathcal{O}$, $Q_1 \leftarrow \mathcal{O}$, $Q_2 \leftarrow P$
 2: for $i = 0$ to $n - 1$ do
 3:     $Q_{k_i} \leftarrow Q_{k_i} + Q_2$
 4:     $Q_2 \leftarrow 2Q_2$
 5: end for
 6: if $Q_0 \in E$ and $Q_1 \in E$ and $Q_2 = Q_0 + Q_1 + P$ then
 7:     return $Q_1$
 8: else
 9:     return $\mathcal{O}$
10: end if
```

2.2.4 Fault Attacks on Dummy and Validation Operations

2.2.4.1 Safe-error Fault Attacks

A common countermeasure against simple power and timing analysis attacks is the use of dummy operations and variables, e.g., a double-and-add-always algorithm like Algorithm 2. When a fault is injected in a dummy variable or during a dummy operation, it results in a *safe error*, i.e., an error that does not affect the final result. A safe-error fault attack, as presented in [71] and [72], works by injecting a fault resulting in a potential safe error during an iteration of the scalar multiplication and observing the resulting output. A correct output confirms that the targeted operation or variable was dummy, while a faulty output indicates the opposite. In both cases, the corresponding bit of the scalar is exposed. Note that input or output validation does not help in countering this attack.

Countermeasures. To counter a safe-error fault attack, it is essential to eliminate the potential of safe-errors. In particular, it should be possible to detect errors resulting from any injected fault even when they do not affect the final result. As an example, Algorithm 4, which employs coherency checking, allows for the detection of safe-error attacks by validating the dummy intermediate variable.

2.2.4.2 Double-fault Attacks

Most countermeasures for fault attacks, e.g., point validation, integrity checking and coherency checking, are based on invariants that are assumed to hold in an error-free computation. As such, they are commonly implemented as logical tests that indicate whether the final output is faulty or not. It is possible, however, for an attacker to influence the result of such a test by injecting a fault in the testing hardware or the status register [73]. This leads to these validation tests becoming a single point of failure.

Countermeasures. This can be avoided using *infective computation*, presented originally in [73] and adapted for elliptic curves in [23]. In essence, the validation test is replaced by a computation that allows the correct result to pass through unchanged and masks any faulty results randomly. For example, instead of testing whether $a = b$ and returning a if the equality holds, the algorithm returns $(a - b)r + a$ for a random r, so the equality test becomes implicit and is no longer dependent on the value of a single bit. This way, the attacker can not use the faulty result since it is no longer correlated with the secret information.

2.3 Summary of Countermeasures for Fault Attacks

As discussed earlier, fault attacks on elliptic curve cryptosystems can be divided into various classes, and each class has its corresponding countermeasures. Generally, these countermeasures work by either preventing the injection of faults, detecting the resulting errors, or masking the faulty result randomly. It is also possible to extend some of these techniques to recover from detected errors. While some of these techniques are only applicable to ECC or to specific classes of faults, others apply more generally. In this section, we briefly review the known countermeasures for fault attacks and comment on their effectiveness and limitations.

Prevention. The occurrence of some types of faults can be prevented through physical means like *sensors* and *metal shields* [7]. Moreover, sign change fault attacks can be prevented using *Montgomery ladder* [52] since it does not use the y-coordinate, and hence does not allow sign change.

Detection. As for detection, *checksums* can be used to detect errors in system parameters or its memory [7]. Moreover, *point validation* can be used to detect invalid-curve errors as the representation of an elliptic curve point has some inherent information redundancy [10]. It is also possible to use *time* and/or *hardware redundancy*, accompanied by comparison, to detect faulty results [24]. *Randomization* can also be used in a variety of ways while encoding the scalar, base point or curve parameters. Combined with hardware or time redundancy, it prevents similar errors from generating similar faulty results thus aiding in the detection of errors [24]. Most notably, some algorithms involve redundant computations that allow for detecting errors by *checking the coherency* of the results. For example, in Algorithm 4, intermediate variables satisfy a set of invariants that can be used to check for coherency. These invariants allow for detection of a wide range of errors including those resulting from the classes of FAAs discussed earlier [23]. Another approach that uses redundant computations is the *combined curve* approach used in [11] and [6].

Masking. It is also possible to use some techniques that mask the faulty results. For example, *randomization* is effective in masking some types of errors, particularly those resulting from sign change faults [11]. Moreover, since a validation test is a logical test, its outcome can be manipulated by flipping a single bit, effectively becoming a single point of failure. *Infective computation*, as presented in [73], circumvents this problem by masking faulty results randomly.

Recovery. Error detection can be combined with time or hardware redundancy to allow for error recovery. One such approach is the use of *N-modular redundancy*, e.g., *Triple Modular Redundancy* (TMR). Triple modular redundancy with a majority vote works well when faults are limited to one block, and can detect errors when two or more blocks produce different faulty results. However, an attacker can inject the same fault in two or more modules and force the structure to output a faulty result. This issue has been addressed in [24] using randomized input encoding, which results in randomized computations and similar faults causing different faulty results.

Another method for combining hardware redundancy with information redundancy is *Dual Modular Redundancy with Point Validation* (DMR-PV) [24]. This is a simple replication with comparison scheme combined with point validation. The inputs to each of the modules are randomly encoded and the results of each are tested by a point validation module. This structure can recover from both natural errors and errors caused by an invalid-curve fault attack limited to one module, and can detect them when they occur in both modules. However, this structure can not always detect errors caused by sign change faults since an attacker can bypass the validation tests by injecting a sign change fault in one of the modules and a random fault in the other.

It is also possible to combine time and hardware redundancy as in the *Parallel and Recomputation* (PRC) scheme [24]. This scheme is a combination of multiple types of redundancy, namely hardware redundancy, time redundancy and randomized input encoding. Two modules are used and their inputs are encoded, then their results are compared. If the results are not equal, the computation is performed again with different input encoding for both modules and the new results are compared with the old ones to find the correct result. This structure can recover from all errors limited to one of the modules and detect them when they occur in both modules.

2.4 Conclusion

The use of elliptic curves in cryptography has gained a wide acceptance since it was first proposed and has become a core component of many industry standards. This can be attributed to the rich mathematical structure of elliptic curves that enables novel applications and implementations, and to their apparent resistance against general mathematical attacks.

However, elliptic curve cryptosystems are vulnerable, like other cryptosystems, to side-channel attacks, which target the implementation weaknesses rather than the mathematical structure. Similarly, they are vulnerable to the various classes of fault analysis attacks, which include invalid-curve attacks, sign change attacks and attacks on validation and dummy operations. The feasibility and effectiveness of a specific fault attack depends on the properties of the implementation and the level of access the attacker has. Some fault attacks are generic with relatively relaxed assumptions and simple procedures, while others assume a sophisticated attacker and expensive apparatus.

It is essential for designers to consider and evaluate the risk of fault attacks against their implementations and implement the necessary countermeasures to defeat these and other side-channel attacks. It is also important to consider the interactions between countermeasures and attacks since a countermeasure of one attack can enable another one.

Chapter 3

Error Detection and Recovery by Frequent Validation

3.1 Introduction

In general, all computations performed between the occurrence of an error and its detection are corrupted and their results will be discarded. This is a significant loss of time and power especially for an operation like ECSM. For example, if a fault occurs near the middle of a scalar multiplication over the above mentioned EC defined over a 256-bit prime field, then processing time and power corresponding to about 1700 field multiplication and squaring operations will be lost, while an exhaustive error-detection test costs about 54 field operations [23]. This can be a significant concern for certain applications.

In this chapter, we propose and analyze new designs for fault-tolerant scalar multiplication structures. The aim here is to address the problem of random transient faults and design more efficient and reliable elliptic curve scalar multiplication structures. In essence, we use known validation techniques to detect errors early and reduce the wasted computation between the error occurrence and its discovery. Also, partial recomputation is used to allow error recovery without requiring a large amount of redundant computations.

In our analysis, we show that our designs can be optimized when the error rate is known and show that it can be significantly more efficient and reliable compared to known designs especially when high error rates are prevalent. We also discuss the applicability of these designs to cases where errors can not be conveniently modeled.

This chapter is organized as follows. In Section 3.2, we give a brief overview of the assumptions that guide this work and the threat models considered. Moreover, we discuss the statistical modeling of errors and review two of the most commonly used distributions. Section 3.3 covers the use of frequent validation for error detection. We give an analysis of the expected overhead and use it to select an optimal block size. Moreover, we offer an alternative choice for the block size based on worst-case analysis and evaluate both options. In Section 3.4, the use of frequent validation and partial recomputation for error recovery is addressed. We analyze the expected overhead and show that it can be used to select an optimal block size. Also, we propose a way to select the block size based on given reliability requirements and compare the two choices. Section 3.5 discusses the practical application and security implications of our approach. Most of the material in this chapter has appeared in [3].

3.2 Assumptions and Error Models

In what follows, we describe the assumptions and threat model adopted in this work. Moreover, we discuss the statistical modeling of errors and review the definitions and properties of two statistical distributions commonly used in modeling the occurrence of errors.

Throughout this chapter and the later chapters, it is assumed that the attacker can not force a specific bit to a specific value, i.e., inject a fault in a way that is precise both in terms of location/timing and resulting value. This is the most intrusive of fault attack models and can only be countered with partial success using randomization [40]. In other words, in all the attacks considered here, the attacker can not predict with certainty the effect of the injected faults. This is generally true for the fault attacks discussed in the previous chapter. It is also assumed that the implementation of the validation test is error-free. This can be achieved using modular redundancy or other hardware measures discussed earlier. It is possible to relax this assumption when infective validation tests are used as discussed in Chapter 5.

In order to estimate the probabilistic costs of testing and recomputation, a statistical model that describe the occurrence of errors is generally required. The selection of the error model and its parameters is influenced by various factors including the types of attacks considered and the implementation details. When modeling the behavior of a complex system under attack, a practical and effective approach is to focus on the effects of the attacks on the system rather than the details of the attacks themselves [55]. In other words, the attacks are modeled from the point of view of the system rather than the point of view of the attacker. Moreover, the language and notation of reliability theory can be effectively used to capture the operational security of a system even when the details of a potential attack are unknown [48]. This way, even an attack that is deterministic from the point of view of the attacker can be modeled probabilistically from the point of view of the system.

Various statistical models can be used to estimate the occurrence times of errors. Here we review two of the commonly used models in the area of reliability and fault tolerance. One of the most commonly used statistical distribution in this area is the *exponential distribution*, which describes the time until the occurrence of the next event when the occurrence rate of events is constant. For example, it is widely used to describe the occurrence of errors or failures when the constant error-rate assumption applies [46]. For a continuous random variable $X \in [0, \infty)$ that is exponentially-distributed, the probability density function is of the form

$$f(x; \lambda) = \lambda e^{-\lambda x} \tag{3.1}$$

where λ is the rate parameter. Its cumulative distribution function is of the form

$$F(x; \lambda) = \Pr[X \leq x] = 1 - e^{-\lambda x} \tag{3.2}$$

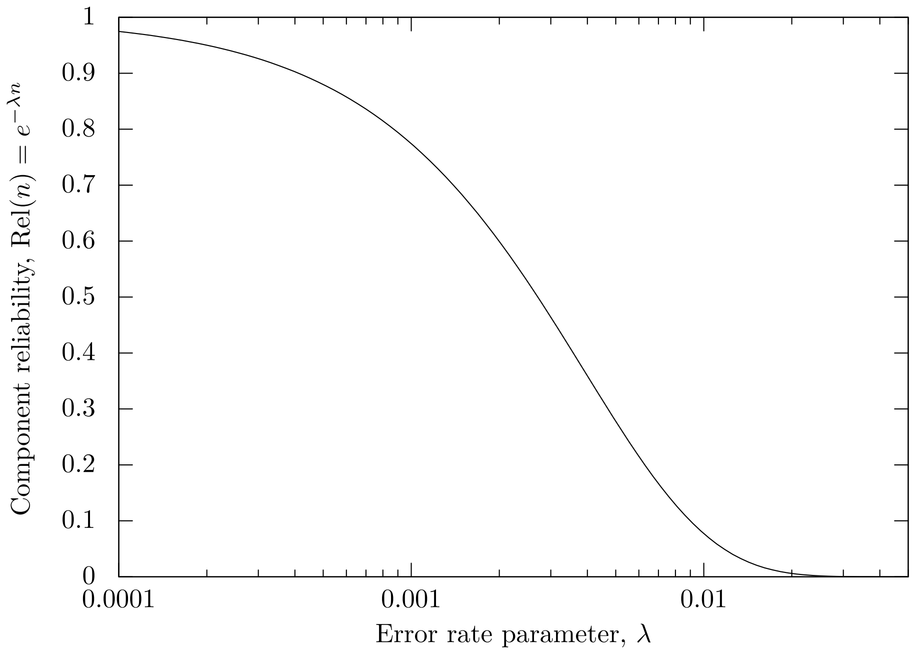

Figure 3.1: Reliability vs. error rate for exponentially-distributed errors when $n = 256$ iterations

Moreover, the complement of $F(x)$ is called the reliability and is denoted by $\mathrm{Rel}(x)$. It describes the probability that an event will only occur beyond time x. For an exponentially-distributed variable X, the expected value is $\mathrm{E}[X] = \frac{1}{\lambda}$. The exponential distribution is *memoryless*, meaning that when looking forward, the next occurrence always has the same distribution as the first occurrence. More formally,

$$\Pr[X > x + s | X > x] = \Pr[X > s] \tag{3.3}$$

To find the practical range of values for λ, we find the reliability of the scalar multiplication implementation for various values of λ. Figure 3.1 illustrates the reliability associated with a range of error rates for $n = 256$ iterations, and indicates that the practical region of error rate is $\lambda < 0.05$.

The *Pareto distribution*, on the other hand, was first introduced by the Italian economist Vilfredo Pareto to model the allocation of wealth among individuals, and was later used to model a wide range of natural, social and economical phenomena. For example, it was shown to capture the characteristics of network traffic [56, 19] and to model hardware errors [63] better than other commonly used distributions like the exponential and the Weibull distributions. The Pareto distribution has a probability density function of the form

$$f(x; x_0, \gamma) = \begin{cases} \dfrac{\gamma x_0^\gamma}{x^{\gamma+1}} & \text{if } x \geq x_0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

where x_0 and γ denote the scale and shape parameters, respectively. Note that x_0 represents the minimum value that can be taken by x. Its cumulative distribution function is

$$F(x; x_0, \gamma) = \Pr[X \leq x] = \begin{cases} 1 - \left(\dfrac{x_0}{x}\right)^\gamma & \text{if } x \geq x_0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

The expected value of a Pareto distribution is $\mathrm{E}[X] = \frac{\gamma x_0}{\gamma - 1}$ for $\gamma > 1$.

The Pareto distribution is one of the simplest examples of a *heavy-tailed* distribution. Heavy-tailed distributions are characterized by the asymptotic shape of the distribution. More formally, a distribution X is heavy-tailed if

$$\Pr[X > x] \sim x^{-\gamma} \text{ as } x \to \infty, \gamma > 0 \tag{3.6}$$

When $\gamma \leq 2$, the distribution has an infinite variance while $\gamma \leq 1$ implies an infinite mean. It follows that for a heavy-tailed distribution, extreme values have a non-negligible probability. In contrast, for thin-tailed distributions like the exponential or normal distributions, the probability of extreme values goes to zero exponentially fast. The Pareto distribution is also characterized by *self-similarity*, i.e., any part of the distribution curve is similar to the whole curve under appropriate scaling. The combination of these two properties, namely, the heavy-tail and self-similarity, allows the Pareto distribution to capture long-range correlations [19].

In our use of the Pareto distribution, we set $x_0 = 1$ since an error can only be detected after the first iteration. In order to determine the practical range of values for γ, we find

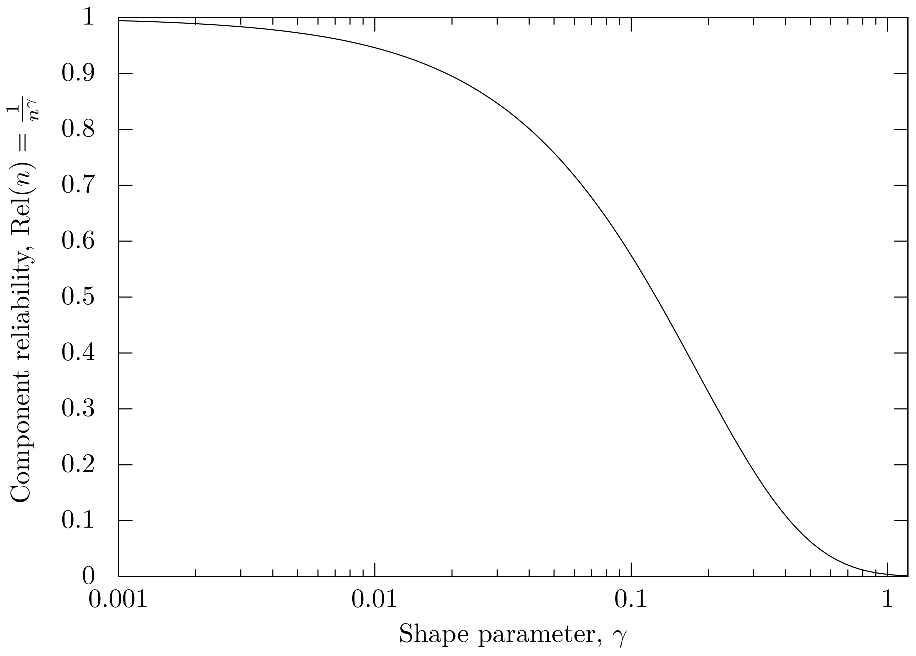

Figure 3.2: Reliability vs. γ for Pareto-distributed errors when $n = 256$ iterations

the reliability of the scalar multiplication implementation for different values of γ. Figure 3.2 illustrates the reliability associated with a range of values of γ for $n = 256$ iterations, and indicates that the practical region of error rate is $\gamma < 1$.

A more extensive treatment of the Pareto distribution and heavy-tailed distributions in general can be found in references like [50, 60, 33].

3.3 Frequent Validation for Error Detection

As discussed previously, error detection and recovery for elliptic curve scalar multiplication operation have been achieved by testing the final result for for validity and by randomizing the encoding of inputs, essentially treating the scalar multiplication operation as a black-box. While that approach has its merits, particularly in that it minimizes the modifications to existing scalar multiplication structures and the number of tests required, it has the disadvantage of allowing an error to propagate and corrupt all consequent iterations. In this section, we propose the use of frequent validation as a more efficient way to achieve error detection in elliptic curve scalar multiplication.

3.3.1 Approach

Error detection is generally achieved using a test at the end of the computation which is designed to detect a one or more types of errors, and these tests are usually efficient. However, when an error occurs, all computations performed after the occurrence of the error will be affected, and performing them becomes wasteful. This is significant in terms of time and power consumption especially for restricted environments and mobile devices. So, it is important to design error detection structures in a way that is not only efficient, but also limits the wasted computations after the occurrence of an error.

Our approach to reduce this problem is the use of frequent validation. We use the fact that the state of the algorithm, i.e., intermediate results, can be tested for validity after each iteration and that the validation tests are relatively efficient. The aim is to prevent error propagation using multiple validation tests of intermediate results as illustrated in Figure 3.3 rather than a single test at the end. While this approach will increase the cost of testing, this overhead can be offset by the saving in iterations lost as a result of error propagation.

This leads to the question of how frequent should the intermediate state be tested for validity. The two extremes are testing once at the end of the computation and testing after every iteration. While the former saves on the testing cost, it does not solve the problem of wasteful computation when an error occurs. On the other hand, the latter eliminates the loss but incurs a large overhead for testing. Clearly, there is a point between these extremes that provides a better trade-off, and there are various criteria by which this trade-off can be evaluated. One way to find better trade-off points is to make some assumptions on the statistical properties of errors.

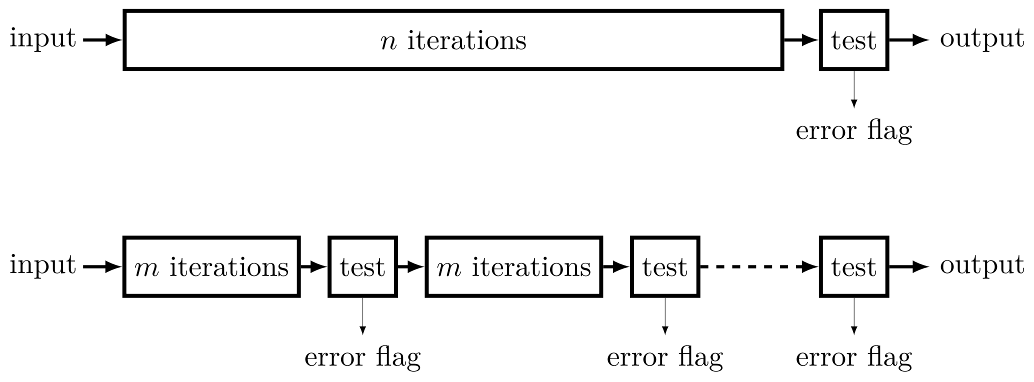

Figure 3.3: Error detection by frequent validation

3.3.2 A Cost Model for Frequent Validation

In this section, we analyze the cost of the increased testing and the gain of preventing the execution of faulty computation following an error. The aim is to find values for the block size, denoted by m, that provide reasonable trade-offs between cost and saving. We use two criteria to determine these trade-off points, namely, minimizing the expected overhead and minimizing the worst-case overhead.

Cost measures. To describe the cost of computations and testing, we use the following notation. Let c_b be the average cost of an iteration of the base algorithm, which is assumed to be a bare-bone scalar multiplication algorithm with no added redundancy to enable error detection. For example, for the conventional double-and-add algorithm, $c_b = \frac{3}{2}$ point operations on average, while for 2-NAF double-and-add algorithm, $c_b = \frac{4}{3}$ on average. Let c be the cost of an iteration of an error-detecting algorithm, e.g., Algorithm 4 which has $c = 2$ point operations. Let c_v be the cost of the validation test used to detect the occurrence of errors in the state of the algorithm. In the examples that follow, these costs are presented in terms of point operations. However, this is not a limitation since any other suitable computational cost unit, e.g., field or bit operations, can be used.

3.3.2.1 Minimizing the Expected Overhead

Let n denote the total number of iterations in the scalar multiplication algorithm, and let X be a random variable that describes the time (in iterations) until the next error occurs. In the following analysis, no assumptions are made about X except that it is the same for all iterations.

The probability of m consecutive iterations without an error (i.e., an error-free block) is given by $\Pr[X \geq m]$, while the probability of an error or more in a block is the complement $\Pr[X \leq m]$. Let Y denote a random variable that describes the number of blocks computed when a faulty block is observed. Given the assumption that blocks are identical and statistically independent, since a block is only started after the previous one is tested, Y follows a geometric distribution with mean $\mathrm{E}[Y] = 1/p_y$, where p_y is the probability that a block will have one or more errors, i.e., $p_y = \Pr[X \leq m]$. However, since the number of blocks is finite, i.e., $Y \in \{1, \ldots, n/m\}$, the expected value of Y has to be modified to reflect its finite range as follows.

$$\mathrm{E}[Y|Y \leq \frac{n}{m}] = \frac{\sum_{i=1}^{\frac{n}{m}} i \Pr[Y = i]}{\Pr[Y \leq \frac{n}{m}]} \tag{3.7}$$

We can now estimate the overhead associated with the frequent validation approach to error detection. By checking early for an error, our aim is to reduce the computation performed after the error while limiting the testing overhead. There are two cases to be considered: (i) no error or an error after the $\frac{n}{m}$ blocks, or (ii) an error within the $\frac{n}{m}$ blocks. In each case we will estimate the cost of testing and the wasteful computation caused by the occurrence of an error.

Case 1. In the first case, namely no error or an error after $\frac{n}{m}$ blocks, the number of tests would be equal to the number of blocks, $\frac{n}{m}$, so their cost would be $c_v \frac{n}{m}$. Moreover, the overhead required to allow for error detection will be $n(c - c_b)$. There would be no wasteful faulty computations since there was no error in the $\frac{n}{m}$ blocks. The probability of this case is $\Pr[Y > \frac{n}{m}]$. The contribution of this case to the total cost would then be $\Pr[Y > \frac{n}{m}]\left(c_v\frac{n}{m} + n(c - c_b)\right)$.

Case 2. In the second case, namely that an error occurred within the $\frac{n}{m}$ blocks, we observe that a test is needed for each block until one is found faulty. So, the expected number of tests is equal to the expected number of blocks needed to observe the first error, $\mathrm{E}[Y|Y \leq \frac{n}{m}]$. The expected overhead of the tests is then $\mathrm{E}[Y|Y \leq \frac{n}{m}]c_v$. Moreover, for each of these blocks, the extra computations required for error detection is $m(c - c_b)$, so this part of the overhead is $\mathrm{E}[Y|Y \leq \frac{n}{m}]m(c - c_b)$. To estimate the wasteful computations caused by the late detection of an error, we consider all iterations after the occurrence of an error until it is detected at the end of a block. Within a block, the number of iterations until an error occurs is described by the random variable X defined earlier. Since a block has a finite number of iterations, the expected value of X given that an error has occurred is

$$\mathrm{E}[X|X \leq m] = \frac{\int_0^m xf(x)dx}{\Pr[X \leq m]}. \tag{3.8}$$

The wasted iterations would be all iterations after an error occurred, i.e., $m - \mathrm{E}[X|X \leq m]$, and their cost is $c_b(m - \mathrm{E}[X|X \leq m])$. Hence, the total cost for this case would be $\mathrm{E}[Y|Y \leq \frac{n}{m}](c_v + m(c - c_b)) + c_b(m - \mathrm{E}[X|X \leq m])$. The probability of this case is $\Pr[Y \leq \frac{n}{m}]$ and its total contribution is

$$\Pr[Y \leq \frac{n}{m}]\left(\mathrm{E}[Y|Y \leq \frac{n}{m}](c_v + m(c - c_b)) + c_b(m - \mathrm{E}[X|X \leq m])\right)$$

Combining case 1 and case 2. It follows that the expected total cost of error detection with frequent validation can be expressed as a function D of n, m, X, c, c_b and c_v as follows.

$$\begin{aligned} D(n, m, X, c, c_b, c_v) = &\Pr[Y > \frac{n}{m}]\left(c_v\frac{n}{m} + n(c - c_b)\right) \\ &+ \Pr[Y \leq \frac{n}{m}]\left(\mathrm{E}[Y|Y \leq \frac{n}{m}](c_v + m(c - c_b)) + c_b(m - \mathrm{E}[X|X \leq m])\right) \end{aligned} \tag{3.9}$$

Given this expression, it is possible to select a block size, denoted by m_{opt}, in a way that minimizes the expected overhead, as will be illustrated in the example in Section 3.3.3.

3.3.2.2 Cost Expressions for Specific Error Models

In this section, we substitute distribution-specific expressions in (3.9) to get cost expressions for specific error models. In particular, we consider exponentially-distributed errors as well as errors that follow the Pareto distribution.

Errors follow an exponential distribution. In this case, X is fully determined by the rate parameter λ. In particular, $\Pr[X \leq m] = 1 - e^{-\lambda m}$ which implies that

$$\begin{aligned} \mathrm{E}[X|X \leq m] = \frac{\int_0^m xf(x)dx}{\Pr[X \leq m]} &= \frac{1 - e^{-\lambda m}(1 + m\lambda)}{\lambda(1 - e^{-\lambda m})} \\ &= \frac{1}{\lambda} - \frac{m}{e^{\lambda m} - 1} \\ &= \mathrm{E}[X] - \frac{m}{e^{\lambda m} - 1} \end{aligned} \tag{3.10}$$

As for Y, it can be shown using the properties of the geometric distribution that $\Pr[Y = y] = e^{-\lambda m(y-1)}(1 - e^{-\lambda m})$. This implies that $\mathrm{E}[Y] = \frac{1}{1-e^{-\lambda m}}$ and that $\Pr[Y > \frac{n}{m}] = e^{-\lambda n}$. We can then find $\mathrm{E}[Y|Y \leq \frac{n}{m}]$ as follows.

$$\begin{aligned} \mathrm{E}[Y|Y \leq \frac{n}{m}] &= \frac{\sum_{y=1}^{\frac{n}{m}} y \Pr[Y = y]}{\Pr[Y \leq \frac{n}{m}]} \\ &= \frac{e^{(m+n)\lambda}m + n - e^{m\lambda}(m+n)}{(e^{m\lambda} - 1)(e^{n\lambda} - 1)m} \\ &= \frac{1}{1 - e^{-\lambda m}} - \frac{n}{m(e^{\lambda n} - 1)} \\ &= \mathrm{E}[Y] - \frac{n}{m(e^{\lambda n} - 1)} \end{aligned} \tag{3.11}$$

This is all that is needed to rewrite (3.9) as a function of λ instead of X and Y.

Errors follow a Pareto distribution. In this case, X is determined by two parameters, namely, x_0 and γ, the scale and shape parameters, respectively. We set $x_0 = 1$ since this the minimum value of X in our scenario, i.e., an error in the first iteration. It follows that $\Pr[X \leq m] = 1 - \frac{1}{m^\gamma}$ which implies that

$$\begin{aligned}
\mathrm{E}[X|X \leq m] = \frac{\int_1^m xf(x)dx}{\Pr[X \leq m]} &= \frac{\gamma(m^\gamma - m)}{(\gamma - 1)(m^\gamma - 1)} \\
&= \frac{\gamma}{\gamma - 1}\left(1 - \frac{m-1}{m^\gamma - 1}\right) \\
&= \mathrm{E}[X]\left(1 - \frac{m-1}{m^\gamma - 1}\right) \qquad (3.12)
\end{aligned}$$

Unlike $\mathrm{E}[X]$, this value is defined for $\gamma < 1$, while for $\gamma = 1$, its limit can be found as follows.

$$\lim_{\gamma \to 1} \mathrm{E}[X|X \leq m] = \frac{m \ln m}{m - 1} \qquad (3.13)$$

With respect to Y, we can see that $\Pr[Y = y] = \frac{1}{m^{\gamma(y-1)}}(1 - \frac{1}{m^\gamma})$ and that $\mathrm{E}[Y] = \frac{m^\gamma}{m^\gamma - 1}$. Also, it can be shown using the properties of the geometric distribution that $\Pr[Y > \frac{n}{m}] = \frac{1}{m^{\gamma n/m}}$. As for the conditional expectation for Y, we can write

$$\begin{aligned}
\mathrm{E}[Y|Y \leq \frac{n}{m}] &= \frac{\sum_{y=1}^{\frac{n}{m}} y \Pr[Y = y]}{\Pr[Y \leq \frac{n}{m}]} \\
&= 1 + \frac{1}{m^\gamma - 1} - \frac{n}{m(m^{\gamma n/m} - 1)} \\
&= \mathrm{E}[Y] - \frac{n}{m(m^{\gamma n/m} - 1)} \qquad (3.14)
\end{aligned}$$

This is all that is needed to rewrite (3.9) as a function of γ instead of X and Y.

3.3.2.3 Minimizing the Worst-case Overhead

By examining the overhead of detection expressed in (3.9), it is clear that the value of m that minimizes the overhead depends on the error model, and particularly on the distribution of X. In some cases, it is preferable to find a sub-optimal m that limits the overhead while being independent of X. This can be achieved by optimizing the worst-case, rather than average, overhead of detection. In this case, the worst-case scenario is the scenario with both the highest number of validation tests and the highest loss due to faulty computations, i.e., when a fault occurs in the first iteration of the last block. This translates to $\Pr[Y \leq \frac{n}{m}] = 1$, $Y = \frac{n}{m}$ and $X = 1$. Then, by substitution in (3.9), the cost expression of the worst-case overhead can be written as follows.

$$D_{\mathrm{W}}(n, m, c, c_b, c_v) = c_v \frac{n}{m} + n(c - c_b) + c_b(m - 1) \tag{3.15}$$

This expression can be minimized to find a sub-optimal block size, denoted by $m_{\text{w-opt}}$, that is independent of the statistical error model.

3.3.2.4 Parallel Validation Tests

In the analysis above, it has been assumed that the validation tests are performed sequentially relative to the main computation. However, these tests can be performed partially or completely in parallel relative to the main computation. In the case of full parallelism, the time overhead resulting from frequent validation tests is eliminated at the cost of an increase in the hardware requirements. On the other hand, partial parallelism can be achieved using the idle time of the modules used to implement the scalar multiplication, e.g., the field multiplier or squarer.

To model the total cost with full or partial parallel detection, we use a parameter ρ to measure the amount of overlap between the scalar multiplication and the validation tests. When $\rho = 0$, there is complete overlap, i.e., full parallelism, while when $\rho = 1$, the operations are completely sequential. It is assumed throughout that the cost of a validation test is not greater than the cost of a block of iterations, which is generally true. If this is not the case, the following analysis can be modified accordingly.

In the case of fully or partially parallel validation, the validation test is performed in parallel with the main computation. If the test determines that the result is valid, the computation is not interrupted. On the other hand, if the test detects an error in the result, the computation of the current block is interrupted. It follows that for all non-faulty blocks, the cost of validation test is ρc_v, while for faulty blocks, the cost is still c_v. Hence, (3.9) can be modified to model fully and partially parallel tests by modifying the cost of the tests associated with non-faulty blocks, except for the last block.

$$\begin{aligned}D_{\mathrm{P}}(n, m, X, c, c_b, c_v, \rho) = &\Pr[Y > \frac{n}{m}] \Big(\rho c_v \frac{n}{m} + n(c - c_b)\Big) \\ &+ \Pr[Y \leq \frac{n}{m}] \Big(\mathrm{E}[Y | Y \leq \frac{n}{m}](\rho c_v + m(c - c_b)) \\ &\qquad + c_b(m - \mathrm{E}[X | X \leq m])\Big) \\ &+ c_v(1 - \rho) \end{aligned} \tag{3.16}$$

3.3.3 A Numerical Example and Evaluation

In this section, we discuss an example of error detection using frequent validation in elliptic curve scalar multiplication. For the underlying curve, we consider elliptic curves defined over a prime field of size $n = 256$. Also, we will use the conventional double-and-add algorithm as the reference algorithm for scalar multiplication. This approach is applicable to different curves and scalar multiplication algorithms. Moreover, it is assumed in this example that errors are exponentially-distributed with parameter λ.

3.3.3.1 Error Detection Example

In this example, we analyze the relative advantage of a scalar multiplication unit that performs error detection with frequent validation. We base this design on Algorithm 4, proposed in [23] and discussed in Chapter 2. This algorithm can detect errors resulting from invalid-curve, sign change and safe-error attacks using a combination of point validation and consistency checking. It also has the added benefit of resisting timing and simple power-analysis attacks.

Algorithm 4 achieves its goal of detecting errors with a simple and efficient test. However, if we consider the fault model presented in Section 3.2, the expected number of iterations needed to observe an error, $\mathrm{E}[X | X \leq n]$, can be considerably less than n. As

Algorithm 5 Scalar multiplication with frequent validation

Input: $P \in E(\mathbb{F})$, $k = (k_{n-1}, k_{n-2}, k_{n-3}, \ldots, k_0)$, $m \leq n$

Output: kP

```
1: $Q_0 \leftarrow \mathcal{O}$, $Q_1 \leftarrow \mathcal{O}$, $Q_2 \leftarrow P$,
2: for $i = 0$ to $n - 1$ do
3:     $Q_{k_i} \leftarrow Q_{k_i} + Q_2$
4:     $Q_2 \leftarrow 2Q_2$
5:     if $i + 1 \bmod m = 0$ or $i = n - 1$ then
6:         if $Q_0 \notin E(\mathbb{F})$ or $Q_1 \notin E(\mathbb{F})$ or $Q_2 \neq Q_0 + Q_1 + P$ then
7:             return $\mathcal{O}$
8:         end if
9:     end if
10: end for
11: return $Q_1$
```

an example, when $n = 256$ and with an error rate $\lambda = 0.01$, we have $\mathrm{E}[X|X \leq n] \approx 79$, while for $\lambda = 0.001$, we have $\mathrm{E}[X|X \leq n] \approx 122$. This illustrates that if an error can be detected early, time and power can be conserved instead of being used to compute a faulty result.

A suitable solution is to check the validity of the intermediate values of Q_0, Q_1 and Q_2 in blocks of size m iterations, where m is chosen to minimize the cost function given in (3.9). Algorithm 5 illustrates this approach. Note that in case an error is detected, this algorithm returns the point at infinity as an error signal. Other options are to raise an error flag or an exception condition.

We also note the following, keeping in mind that these values can be tuned for a more accurate model:

- We assume without loss of generality that a point addition and a point doubling have the same cost, which is a common assumption and is particularly true in the case of curves in the Edwards form [8].

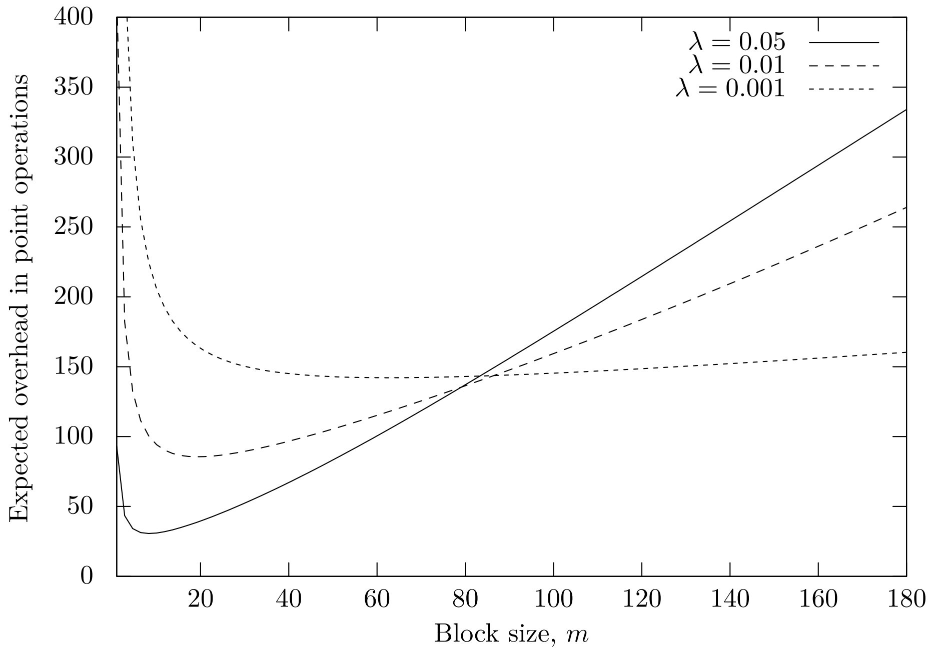

Figure 3.4: Expected overhead of frequent validation in the error detection example

- Each iteration of Algorithm 4 costs 2 point operations, i.e., $c = 2$, while for the base scalar multiplication, $c_b = 3/2$.
- The validation test in Algorithm 4 requires two point validations and two point additions. As such, the overall cost of testing is approximately four point operations, i.e., $c_v = 4$.

Given these facts and assumptions, we can substitute for n, c, c_b and c_v in (3.9) to get the following cost function.

$$D_{\text{Ex1}}(m, \lambda) = \frac{e^{-n\lambda}\left(e^{n\lambda} - 1\right)\left(3 + e^{m\lambda}(4(2+m)\lambda - 3)\right)}{2\lambda\left(e^{m\lambda} - 1\right)} \tag{3.17}$$

which can be used to find an optimal block size, denoted by m_{opt}, for different error rates. Figure 3.4 illustrates the expected cost associated with different block sizes at three different error rates.

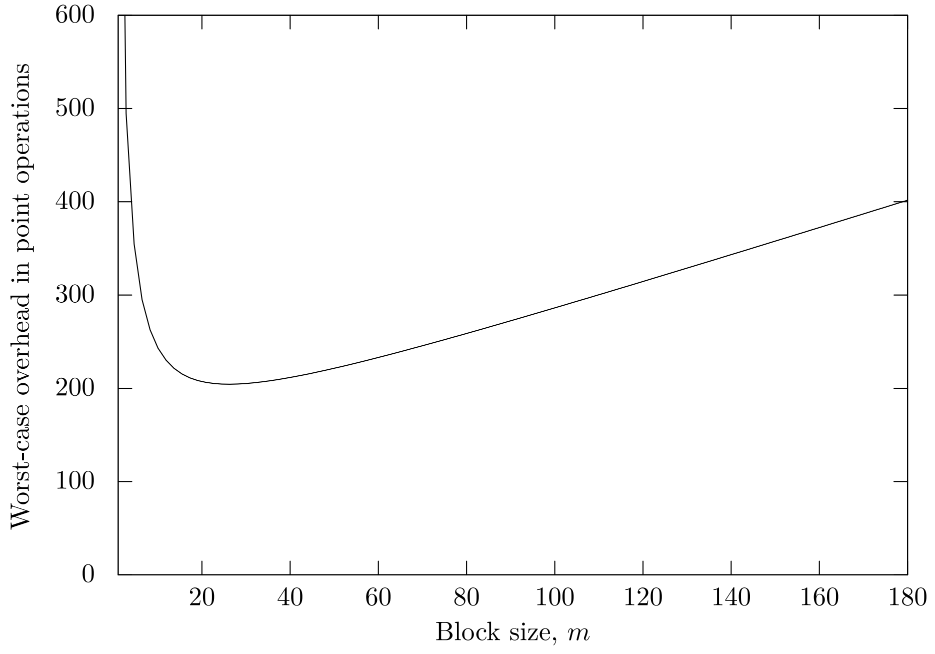

Figure 3.5: Worst-case overhead of frequent validation in the error detection example

Also, we can substitute in (3.15) to get a function that computes the worst-case overhead in point operations as a function of m as follows.

$$D_{\text{W-Ex1}}(m) = \frac{3(m-1)+n}{2} + \frac{4n}{m} \tag{3.18}$$

Figure 3.5 illustrates how this function behaves for a range of values of m. The block size that minimizes this expression, denoted by $m_{\text{w-opt}}$ can also be found.

Table 3.1 gives a summary of the expected and worst-case overhead and saving associated with different values of λ and three values of m, namely, $m = n = 256$, m_{opt} that minimizes expected overhead, and $m_{\text{w-opt}}$ that minimizes worst-case overhead.

Figure 3.6 illustrates the three choices of block size m in a range of error rates. Clearly, the maximum block size, i.e., $m = n$, and $m_{\text{w-opt}}$ are independent of λ, while m_{opt} decreases with increasing error rate as expected.

We also compare the expected overhead of each of the choices of m in Figure 3.7, which shows that the expected overhead when $m = n$ grows very quickly as the error rate

Table 3.1: Expected and worst-case cost of frequent validation for the error detection example in point operations

λ		m	Exp. overhead	Exp. saving	Worst-case overhead	Worst-case saving
0.05	$m = n$	256	486	-	514.5	-
	m_{opt}	8	30.7	93.7%	266.5	48.2%
	$m_{\text{w-opt}}$	26	47	90.3%	204.9	60.2%
0.01	$m = n$	256	377	-	514.5	-
	m_{opt}	19	85.5	77.3%	209	59.4%
	$m_{\text{w-opt}}$	26	87.3	76.8%	204.9	60.2%
0.001	$m = n$	256	177.2	-	514.5	-
	m_{opt}	63	142.1	19.8%	237.3	53.9%
	$m_{\text{w-opt}}$	26	154	13.1%	204.9	60.2%

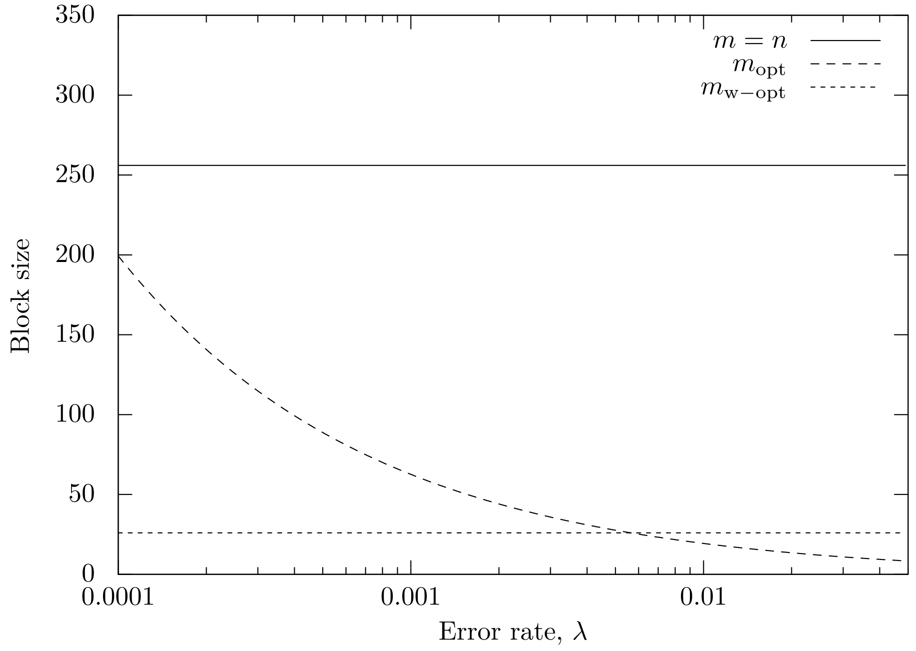

Figure 3.6: Comparing three block sizes in a range of error rates

increases. Moreover, an interesting observation is that the expected overhead associated with $m_{\text{w-opt}}$ stays close to the minimum overhead over a range of values of λ. Another way to compare the three choices of m is the worst-case overhead, as illustrated in Figure 3.8.

3.3.3.2 Evaluation

It can be seen from this example that the use of frequent validation for error detection has a clear advantage over the straight-forward solution. In particular, the early detection reduces the loss caused by errors significantly, and in turn, preserves time and reduces power consumption.

We also have shown two ways to chose the block size, namely optimizing for expected overhead and optimizing for worst-case overhead. While the former is the natural choice to minimize the overhead over a range of error rates, we have shown that the latter has the advantages of being independent of the statistical model. It also gives an expected

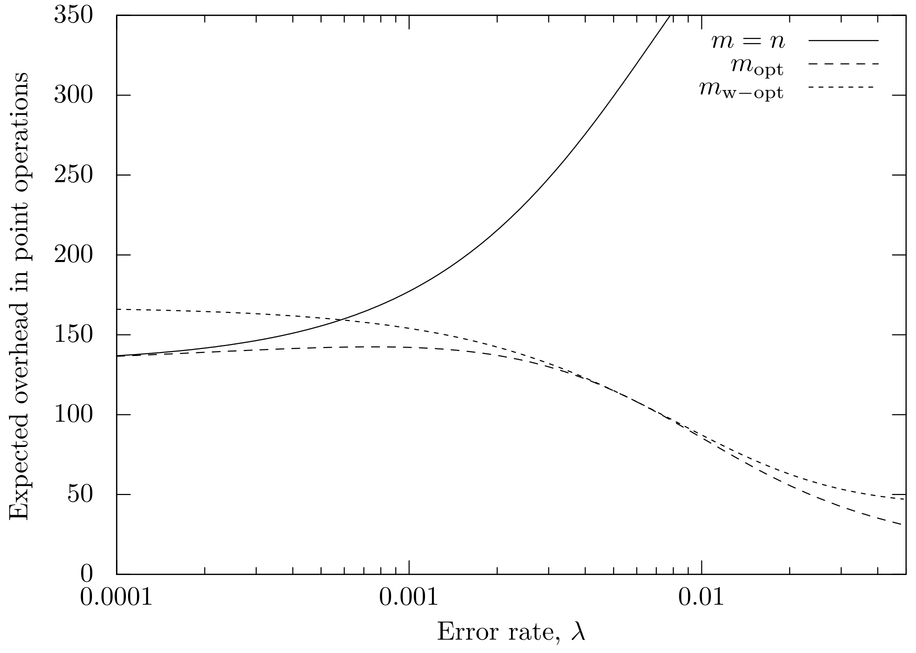

Figure 3.7: Expected overhead of three different values of m in a range of error rates

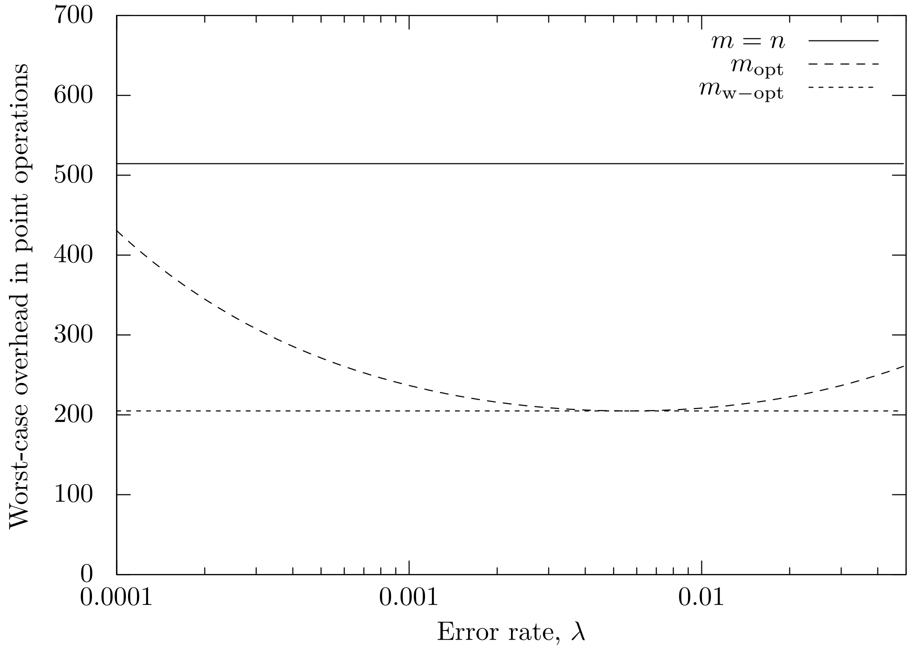

Figure 3.8: Worst-case overhead of three different values of m in a range of error rates

overhead that is close to the optimal over a wide range of values of λ.

3.4 Frequent Validation with Partial Recomputation for Error Recovery

Error recovery is generally achieved through various forms of redundancy. In particular, time redundancy can be effective against transient faults, while hardware redundancy is necessary to counter permanent faults. In a recomputation-based error recovery scheme, the computation is performed and the results are checked for validity. If the test fails, the computation is repeated and the results are tested again. In this section, we present our approach to error recovery which combines recomputation with frequent validation.

3.4.1 Approach

While an error recovery approach based on full recomputation is effective against errors caused transient faults, its time overhead is potentially high. It can be readily observed that the reason behind this large potential overhead is the unnecessary repetition of valid computations. We propose the use of frequent validation and partial recomputation as a low-overhead form of time redundancy to achieve efficient error recovery in elliptic curve scalar multiplication implementations. In particular, validation tests are employed at specific intervals to detect errors early and recompute only the faulty parts without the need to repeat parts of the computation that proceeded correctly, as illustrated in Figure 3.9.

Similar to the earlier discussion on frequent validation in error detection, two extreme cases can be identified when setting the frequency of the validation tests; the intermediate results can be validated once at the end of the computation or after every iteration. This range provides a trade-off between the deterministic cost of testing and the probabilistic cost of recomputation. Better trade-off points can be found by making some assumptions

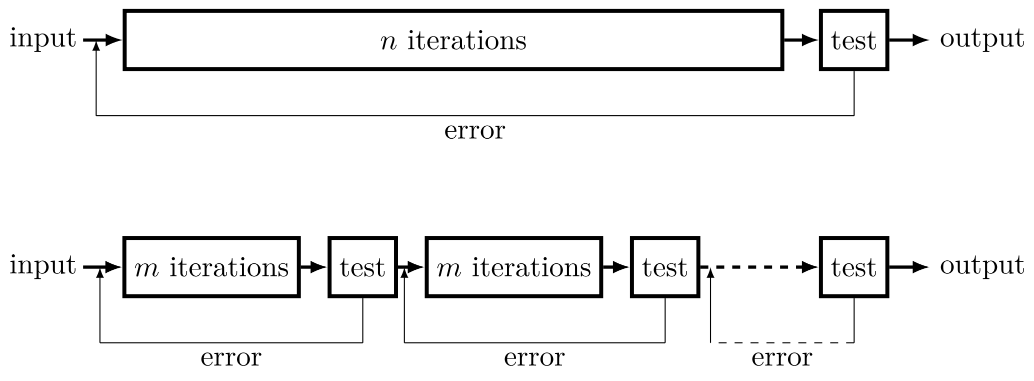

Figure 3.9: Error recovery by frequent validation

on the statistical properties of errors and setting an evaluation criteria. In this work, we use two evaluation criteria, namely, cost and reliability as will be explained later.

3.4.2 A Cost Model for Frequent Validation and Partial Recomputation

In our approach, when an error is detected, only the faulty block is recomputed and the results are tested again. In this section we analyze the cost of the testing and the time overhead of recomputation in this approach. In this context, the overhead includes all operations that are not part of the base scalar multiplication, i.e., validation tests, recomputation and extra point operations. This estimate of the overhead is then used to select values of the block size based on two criteria, namely, minimizing the expected overhead and minimizing the reliability threshold, i.e., the expected overhead associated with a given reliability requirement.

3.4.2.1 Minimizing the Expected Overhead

To construct a model for the expected overhead, we use the conventions introduced in Section 3.2. In particular, we recall that the probability of an error-free block of m iterations is $\Pr[X \geq m]$ where X is the distribution of the occurrence time of the first error. For a correct final result, all $\frac{n}{m}$ blocks should be error-free. Moreover, any faulty block have to be recomputed, so the total number of blocks computed will depend on the number of faulty blocks detected, but the number of error-free blocks will always be $\frac{n}{m}$.

Let Z be a random variable that describes the number of faulty blocks encountered until $\frac{n}{m}$ error-free blocks are found. It follows that Z has a *negative binomial* distribution. Recall that a negative binomial distribution with parameters r and p is the probability distribution of the number of failed trials before observing the r-th success in a Bernoulli process with probability of success p for each trial. It has the expected value of $r(1/p-1)$. In the case of Z, $r = \frac{n}{m}$ and $p_z = \Pr[X \geq m]$. The expected value of Z is

$$\mathrm{E}[Z] = \frac{n}{m}\left(\frac{1}{\Pr[X \geq m]} - 1\right) \tag{3.19}$$

The total time overhead of this approach can be divided into three parts: (*i*) the overhead required to allow for error detection in error-free blocks, (*ii*) the overhead of block recomputations for faulty blocks, and (*iii*) the overhead of testing for all blocks.

The first part applies to all error-free blocks. For each iteration, an overhead of $c - c_b$ is required to allow for error detection. It follows that for a block of m iterations, the overhead is $m(c - c_b)$. Since this part of the overhead applies only to the n/m error-free blocks, the total contribution of this part is $n(c - c_b)$.

For the second part, it is clear that only faulty blocks need to be recomputed. The expected number of faulty blocks is $\mathrm{E}[Z]$, and each of these blocks has m iterations. Each of these iterations costs c point operations, which covers both the basic cost of the underlying algorithm, c_b, and the overhead required for error detection, $c - c_b$. It follows that the overhead of recomputations is $cm\mathrm{E}[Z]$.

For the third part of the time overhead, it can be seen that a validation test is required for every block, whether faulty or not. The expected total number of blocks is $\mathrm{E}[Z] + n/m$, so the time overhead of testing is $c_v(\mathrm{E}[Z] + n/m)$.

The total overhead can then be expressed as a function R of n, m, X, c, c_b and c_v as follows.

$$R(n, m, X, c, c_b, c_v) = n(c - c_b) + cm\mathrm{E}[Z] + c_v\left(\mathrm{E}[Z] + \frac{n}{m}\right) \tag{3.20}$$

This function can be minimized to find an optimal value of m with the least expected overhead.

3.4.2.2 Cost Expressions for Specific Error Models

In this section, we assume that the statistical distribution of X is known, and proceed to find the optimal value of m according to the cost expression in (3.20).

Errors follow an exponential distribution. When X is exponentially-distributed with parameter λ, we can write $\mathrm{E}[Z]$ as follows.

$$\mathrm{E}[Z] = \frac{n}{m}\left(\frac{1}{\Pr[X \geq m]} - 1\right) = \frac{n}{m}\left(\frac{1}{e^{-\lambda m}} - 1\right) \tag{3.21}$$

Then, we can substitute for $\mathrm{E}[Z]$ in (3.20) as follows.

$$R(n, m, \lambda, c, c_b, c_v) = n(c - c_b) + cn\left(\frac{1}{e^{-\lambda m}} - 1\right) + \frac{c_v n}{me^{-\lambda m}} \tag{3.22}$$

To find the value of m that minimizes this expression, we take its derivative and equate it to zero then solve for m. This results in the following expression for m.

$$m_{\text{opt}} = \frac{2\sqrt{c_v}}{\lambda\sqrt{c_v} + \sqrt{\lambda(4c + \lambda c_v)}} \tag{3.23}$$

Errors follow a Pareto distribution. When X follows a Pareto distribution, it is determined by two parameters. Like before, we set $x_0 = 1$. Since $\Pr[X \geq m] = \frac{1}{m^\gamma}$, we can write $\mathrm{E}[Z]$ as follows.

$$\mathrm{E}[Z] = \frac{n}{m}\left(\frac{1}{\Pr[X \geq m]} - 1\right) = \frac{n}{m}\left(m^\gamma - 1\right) \tag{3.24}$$

Then, we can substitute for $\mathrm{E}[Z]$ in (3.20) as follows.

$$R(n, m, \gamma, c, c_b, c_v) = n(c - c_b) + cn\left(m^\gamma - 1\right) + c_v n m^{\gamma - 1} \tag{3.25}$$

This expression is minimized by the following value of m.

$$m_{\text{opt}} = \frac{c_v(1-\gamma)}{c\gamma} \tag{3.26}$$

Note that this expression can be used only when $\gamma < 1$. Otherwise, the block size should be the smallest possible, namely, $m = 1$.

3.4.2.3 Minimizing a Combination of the Expected Overhead and its Standard Deviation

While minimizing the expected value of the overhead assures that the resulting block size has the minimum expected overhead, it gives no guarantees about the variability of the overhead. In fact, as we will see later in a numerical example, minimizing the expected overhead alone can lead to a significant variability in the result, i.e., while the mean is minimal, some values that are relatively far from the mean have a non-negligible probability. This problem can be addressed by optimizing for a combination of the mean and the standard deviation of the overhead.

We start by finding the variance of the cost expression in (3.20).

$$\begin{aligned}\text{Var}\left[n(c-c_b) + cmZ + c_v\left(Z + \frac{n}{m}\right)\right] &= \text{Var}\left[cmZ + c_vZ\right] \\ &= (c^2m^2 + c_v^2)\text{Var}\left[Z\right] \end{aligned} \tag{3.27}$$

since the variance of a constant is zero and the variance of aX where a is a constant is $a^2\text{Var}[X]$. The variance of Z can be found as follows.

$$\text{Var}[Z] = \frac{r(1-p)}{p^2} = \frac{n(1-\Pr[X \geq m])}{m(\Pr[X \geq m])^2} \tag{3.28}$$

The standard deviation of the overhead is the square root of its variance. We then modify (3.20) to include the standard deviation as follows.

$$R(n, m, X, c, c_b, c_v) = n(c-c_b) + cm\text{E}[Z] + c_v\left(\text{E}[Z] + \frac{n}{m}\right) + \delta\sqrt{(c^2m^2 + c_v^2)\text{Var}[Z]} \tag{3.29}$$

where δ is a parameter that determines the number of standard deviation included in the cost expression. For example, $\delta = 0$ indicates optimizing for the mean alone while $\delta = 3$ indicates the minimization of the mean plus three standard deviation. To guide our selection of δ, we recall the one-sided Chebyshev's inequality [31]. For any random variable W with mean μ and standard deviation σ, the following holds.

$$\Pr[W \geq \mu + \delta\sigma] \leq \frac{1}{1+\delta^2} \tag{3.30}$$

for any real number $\delta > 0$. This implies that for $\delta = 3$, for example, less than 10% of the distributions mass will fall beyond $\mu + 3\sigma$. In other words, minimizing the expression in (3.29) implies minimizing a higher quantile of the overhead distribution rather than its mean.

3.4.2.4 Minimizing the Reliability Threshold

The reliability of a component is conventionally defined as $\text{Rel}(t) = \Pr[T > t]$, where T is the random variable representing the lifetime of the component and t is the time for which reliability is computed, which is usually taken to be the whole time of the computation [46]. In other words, reliability is the probability that the component will go through the computation without a failure. For example, the reliability of a TMR structure can be computed as $\text{Rel}_{\text{TMR}} = 3\text{Rel}^2_{\text{ECSM}} - 2\text{Rel}^3_{\text{ECSM}}$, where Rel_{ECSM} is the reliability of a stand-alone ECSM unit, while for both DMR-PV and PRC schemes discussed in Section 2.3, $\text{Rel}_{\text{DMR-PV}} = \text{Rel}_{\text{PRC}} = 2\text{Rel}_{\text{ECSM}} - \text{Rel}^2_{\text{ECSM}}$ [24].

This conventional expression for reliability assumes that a computation will always terminate within a fixed, known time frame unless a failure occurs. However, this assumption does not hold in our case since the time requirements of our approach vary with the number of errors observed. As such, we use a more general definition of reliability to reflect the adaptive nature of the computation's time requirements.

In this context, we define reliability as the probability that the computation will be completed within a given time. It follows that the more time is allowed for the computation, the higher the reliability of the structure since more time will be allowed for recomputations. This presents a trade-off between the component reliability and the allowed time overhead. However, it is essential to keep in mind that this definition applies only to transient faults.

Setting a time threshold for the computation is useful in two ways. First, it allows for an estimate for the reliability of the design to be computed, and second, it prevents the system from going into an infinite loop in case of a permanent fault. We compute reliability for our structure as the cumulative distribution function (CDF) of Z. The CDF of a given value gives the probability that the random variable will fall on or below that value, i.e., $\text{CDF}(z) = \Pr[Z \leq z]$, and can be computed for a random variable that has a negative binomial distribution with parameter r and a trial success probability p as $I_p(r, k+1)$, where $I_p(x, y)$ is the regularized incomplete beta function defined as

$$I_p(x,y) = \sum_{j=x}^{x+y-1} \frac{(x+y-1)!}{j!(x+y-1-j)!} p^j (1-p)^{x+y-1-j} \tag{3.31}$$

This allows for m to be selected in a way that minimizes the overhead at a given reliability level rather than the expected overhead. This can be achieved by setting a certain, typically high, value for the required reliability. Then, for a range of values of m, one is chosen such that it achieves the required reliability with the least overhead. This overhead can then be used as a time threshold for the computation.

3.4.2.5 Parallel Validation Tests

The validation tests can be performed in sequence with the main computation as assumed above. Another possibility is to perform them partially or fully in parallel for a significant reduction in the time overhead. In particular, partial parallelism can be achieved without a significant increase in the hardware requirements using the idle cycles of the same subunits used for the scalar multiplication. Similar to the case of error detection with frequent validation, a parameter ρ is used to model the amount of overlap between the main computation and the validation tests. When $\rho = 0$, there is complete overlap, i.e., full parallelism, while when $\rho = 1$, the operations are completely disjoint.

In the case of full or partial parallel validation, the validation test is performed in parallel with the main computation. If the test determines that the result is valid, the computation is not interrupted. On the other hand, if the test detects an error in the result, the computation of the current block is interrupted and the previous block is recomputed. It follows that for all non-faulty blocks, the cost of validation test is ρc_v, while for faulty blocks, the cost is still c_v.

Hence, (3.20) can be modified to model full and partial parallel tests by modifying the cost of the tests associated with non-faulty blocks, except for the last block.

$$R_{\mathrm{P}}(n, m, X, c, c_b, c_v, \rho) = n(c - c_b) + cm\mathrm{E}[Z] + c_v\left(\mathrm{E}[Z] + \rho\left(\frac{n}{m} - 1\right) + 1\right) \tag{3.32}$$

3.4.3 A Numerical Example and Evaluation

In our approach, error recovery is achieved by employing an efficient error detection scheme and performing frequent validation tests. When an error is detected in a block, the block is recomputed and checked again. For a block size equal to n, this means repeating the whole scalar multiplication. However, if smaller blocks are used, less recomputation is required.

3.4.3.1 Error Recovery Example

This example is analogous to the error detection example in Section 3.3.3. Like before, we adopt Algorithm 4, which is able to detect errors resulting from invalid-curve, sign change and safe-error attacks and modify it to incorporate frequent validation and partial recomputation as shown in Algorithm 6.

Assuming that errors follow an exponential distribution, we can substitute in (3.20) using the same parameters used in the example in Section 3.3.3, namely $n = 256$, $c = 2$,

Algorithm 6 Scalar multiplication with frequent validation and partial recomputation

```
Input:  $P \in E(\mathbb{F})$, $k = (k_{n-1}, k_{n-2}, k_{n-3}, \ldots, k_0)$, , block size $m$
Output:  $kP$
 1: $Q_0 \leftarrow \mathcal{O}$, $Q_1 \leftarrow \mathcal{O}$, $Q_2 \leftarrow P$
 2: $j \leftarrow 0$, $H_0 \leftarrow Q_0$, $H_1 \leftarrow Q_1$, $H_2 \leftarrow Q_2$
 3: for $i = 0$ to $n - 1$ do
 4:     $Q_{k_i} \leftarrow Q_{k_i} + Q_2$
 5:     $Q_2 \leftarrow 2Q_2$
 6:     if $i \bmod m = 0$ or $i = n - 1$ then        ▷ perform the check for blocks of size $m$
 7:         if $Q_0 \in E(\mathbb{F})$ and $Q_1 \in E(\mathbb{F})$ and $Q_2 = Q_0 + Q_1 + P$ then
 8:             $j \leftarrow i$, $H_0 \leftarrow Q_0$, $H_1 \leftarrow Q_1$, $H_2 \leftarrow Q_2$        ▷ store the current state
 9:         else
10:             $i \leftarrow j$, $Q_0 \leftarrow H_0$, $Q_1 \leftarrow H_1$, $Q_2 \leftarrow H_2$   ▷ restore the previous correct state
11:         end if
12:     end if
13: end for
14: return $Q_1$
```

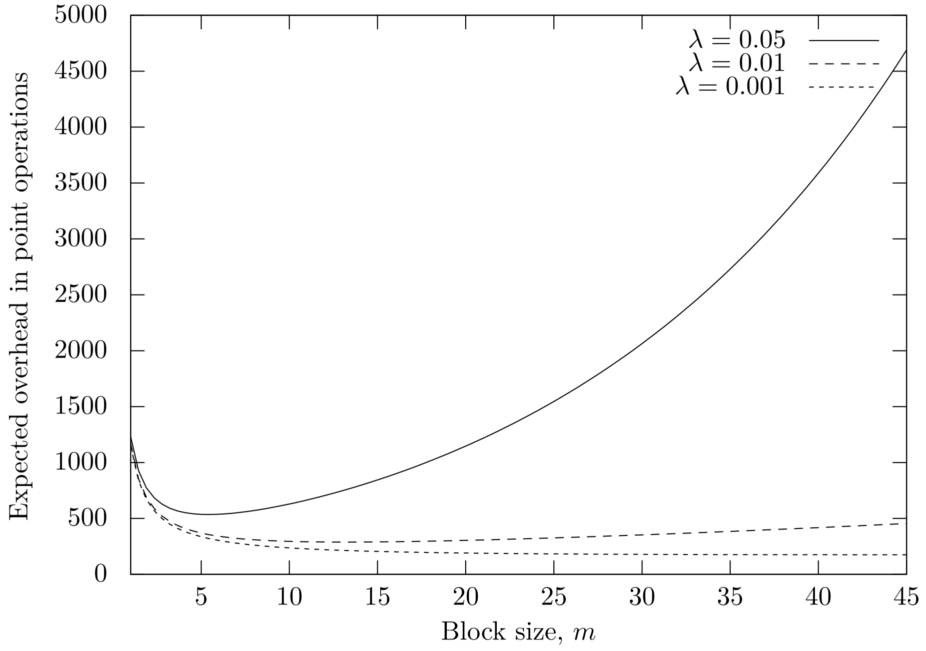

Figure 3.10: Expected overhead of frequent validation in the error recovery example

$c_b = 3/2$ and $c_v = 4$, to get the following overhead function

$$R_{\mathrm{Ex2}}(m, \lambda) = \frac{n\left(4e^{m\lambda}(2+m) - 3m\right)}{2m} \tag{3.33}$$

which can be used to find a block size, denoted by m_{opt}, that minimizes the expected overhead. Figure 3.10 illustrates the overhead associated with different error rates as a function of the block size. Moreover, the value of m_{opt} can be given explicitly as a function of λ as follows.

$$m_{\mathrm{opt}} = \sqrt{\frac{2}{\lambda} + 1} - 1 \tag{3.34}$$

For every value of λ, we can find a value of m, denoted by $m_{\text{r-opt}}$, that minimizes the overhead required for a given reliability requirement. In this example, we assume a required reliability of 99.99%. Figure 3.11 shows the values of m_{opt} and $m_{\text{r-opt}}$ for a range of values of λ. Table 3.2 gives the values of m_{opt} and $m_{\text{r-opt}}$ for some values of λ, and gives the associated overhead and reliability in every case. It also compares the

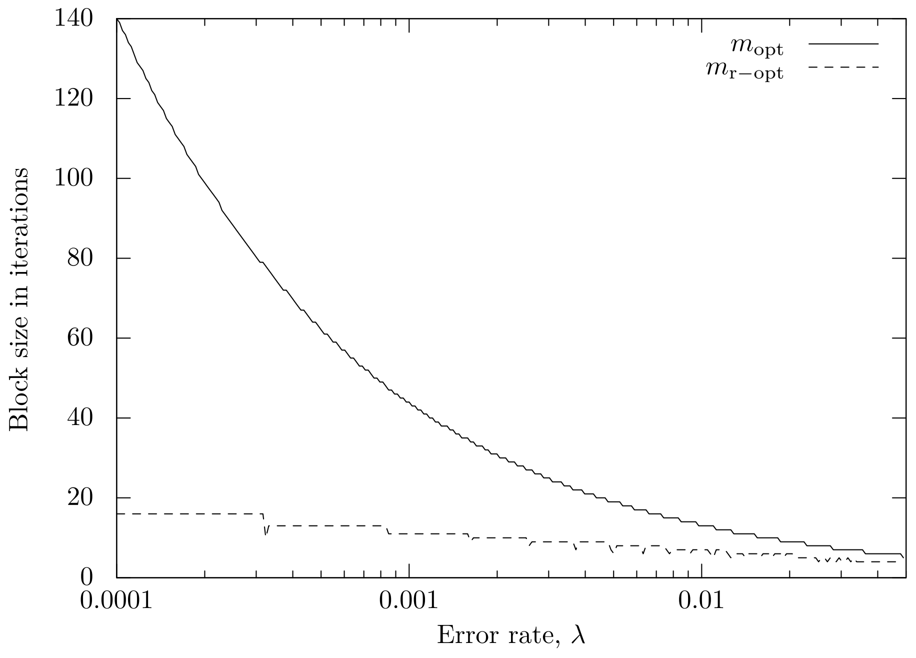

Figure 3.11: Overhead-optimized and reliability-optimized values of m in the error recovery example

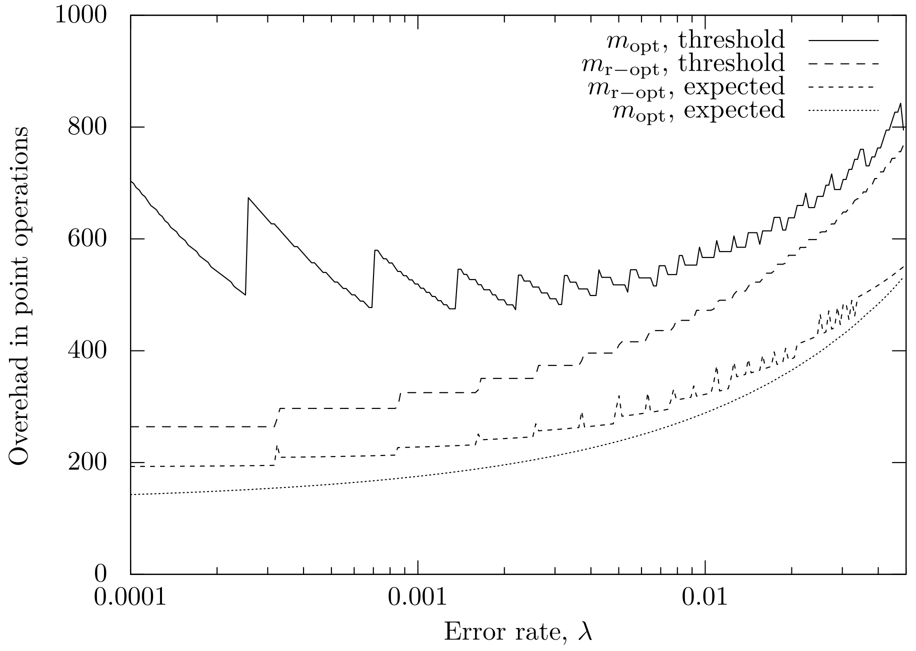

Figure 3.12: Expected overhead and reliability threshold for m_{opt} and $m_{\text{r-opt}}$ in the error recovery example

achieved reliability with corresponding values for TMR and DMR-PV/PRC structures. The overhead in Table 3.2 is given relative to a basic double-and-add scalar multiplication algorithm so all extra computations, tests and recomputations are taken into account while estimating the overhead.

Figure 3.12 shows the expected overhead and the 99.99% reliability threshold for both m_{opt} and $m_{\text{r-opt}}$ over a range of values of λ. An interesting observation in Figure 3.12 is the shape of the gap between the expected overhead of m_{opt} and its reliability threshold. This gap starts considerably large at low values of λ and shrinks as λ increases. The reason for this behavior is that the choice of m_{opt} is based only on the expected overhead, so the value of m_{opt} is quite large for lower values of λ, as illustrated in Figure 3.11. Large values of m_{opt} significantly impact the required reliability threshold since a block recomputation is more expensive for larger blocks.

Table 3.2: Expected overhead and reliability of frequent validation with partial recomputation for the error recovery example in point operations

λ	Rel_{ECSM}	Rel_{TMR}	$\text{Rel}_{\text{DMR-PV,PRC}}$		m	Exp. overhead	Exp. Rel	99.99% Rel threshold
0.05	~0%	~0%	~0%	m_{opt}	5	536 (140%)	57.0%	809 (210%)
				$m_{\text{r-opt}}$	3	607 (158%)	57.1%	779 (202%)
0.01	7.7%	1.7%	14.9%	m_{opt}	13	289 (75%)	65.4%	567 (148%)
				$m_{\text{r-opt}}$	7	322 (84%)	65.6%	472 (123%)
0.001	77.4%	87%	95%	m_{opt}	44	175 (46%)	86.0%	519 (135%)
				$m_{\text{r-opt}}$	11	228 (59%)	86.1%	325 (85%)

On the other hand, the overhead associated with $m_{\text{r-opt}}$ behaves differently. While the expected overhead is higher than that of m_{opt}, the reliability threshold is much lower, especially for lower values of λ. This property, in addition to less variability in the block size as λ varies, makes $m_{\text{r-opt}}$ a more practical choice.

3.4.3.2 Evaluation and Comparison

In this section, we evaluate the results of the preceding example and compare them to the structures reviewed in Section 2.3.

Frequent validation with partial recomputation can be described as an adaptive time redundancy solution, where only faulty blocks are recomputed. The effect of the limited redundancy is clear in the low expected overhead reported in the error recovery example, even for relatively high error rates. The allowed redundancy can be adjusted to satisfy certain reliability requirements. Also, the block size can be chosen in a way that reduces the reliability threshold with a limited effect on expected overhead. As illustrated in Figure 3.12, which shows the time threshold associated with a reliability value of 99.99%, the required threshold is relatively low even at high error rates. This is mainly because of the smaller block sizes.

We now compare known solutions to examine the effects of frequent validation with partial recomputation on the performance and reliability of fault-tolerant structures. The solutions considered include triple-modular redundancy (TMR), double-modular redundancy with point validation (DMR-PV) and the parallel and recomputation scheme (PRC), all of which have been reviewed in Section 2.3. From the error recovery example, we consider $m_{\text{r-opt}}$, which is chosen to minimize the threshold required for a reliability of 99.99%. All of these solutions are evaluated relative to a stand-alone scalar multiplication with no error detection or recovery capabilities.

Reliability. We begin by comparing the structure reliability of each of the methods relative to the reliability of a scalar multiplication unit. As Figure 3.13 illustrates, the

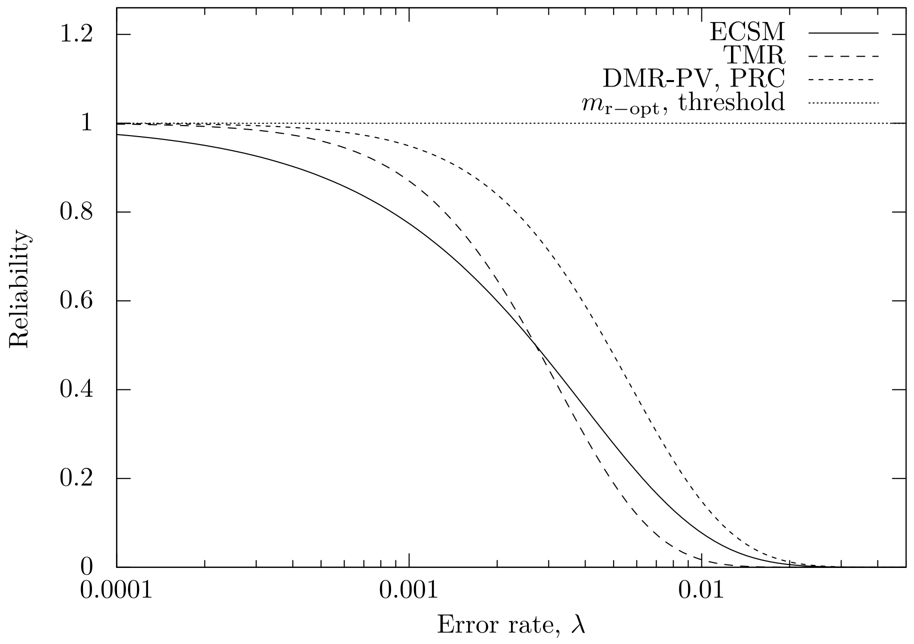

Figure 3.13: Reliability of various schemes in a range of values of λ

reliability of conventional redundancy schemes falls rapidly with the reliability of the scalar multiplication unit. However, the reliability of the frequent validation with partial recomputation scheme does not fall as quickly. In fact, it can be maintained as high as required. This is caused mainly by the use of smaller block sizes at higher error rates and by the limited recomputation of only the faulty blocks. Note that this reliability estimate is limited to transient faults since recomputation can not recover from permanent faults.

Time overhead. We also compare the expected time overhead of various solutions. TMR and DMR-PV are not included in this comparison because they are hardware redundancy schemes and their time overhead is negligible. As Figure 3.14 illustrates, the expected time overhead of all schemes, except for recomputation-based scheme (RC), grows with the error rate. However, our structure is the only one that maintains a high reliability of 99.99% over the range of values of λ, which justifies the required time overhead.

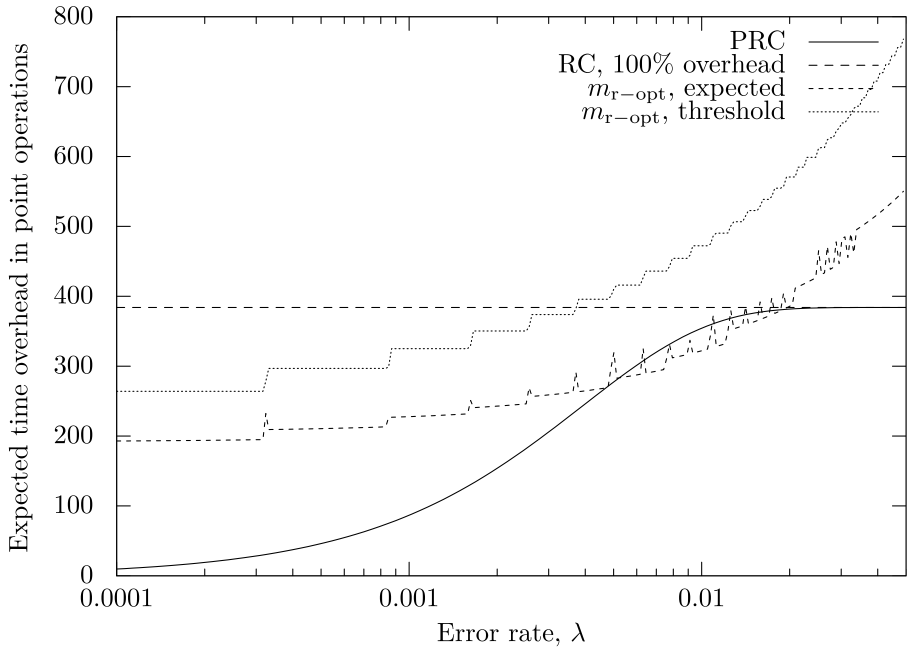

Figure 3.14: Expected time overhead of various schemes in point operations in a range of values of λ

Hardware overhead. Schemes that depend on hardware redundancy, like TMR, DMR-PV and PRC, have the advantage of being tolerant to permanent faults. However, this comes at the cost of doubling the hardware requirements in the case of DMR-PV and PRC, and tripling it for TMR. On the other hand, a scheme based on frequent validation with partial recomputation have no significant hardware overhead, and are only able to detect, but not recover from, permanent faults. For this reason, these schemes are optimal for environments where space is scarce and high reliability is required.

3.5 Practical Aspects and Security Implications

3.5.1 Using Frequent Validation in Practice

Frequent validation provides a trade-off between the cost of testing and the loss or overhead incurred when faults occur. This trade-off is controlled by the selection of the block size. We have shown earlier that the block size can be selected optimally when some assumptions are made about the statistical properties of errors. For instance, when it is assumed that errors follow an exponential distribution, the error rate λ becomes the controlling parameter. This model works well when the value of λ is known with reasonable accuracy, e.g., as the historical rate of hardware faults. However, in many real-life scenarios, it might not be possible to give an accurate estimate of λ, or it might even be argued that the random fault model used earlier does not apply.

Recall that, for both error detection and error recovery, we discussed two criteria to select the block size given λ, namely, minimizing the expected overhead and minimizing the worst-case overhead, which includes minimizing the reliability threshold in the case of error recovery. For both criteria, we have found the optimal block size over a wide range of error rates that represent a range of component reliabilities from the most to the least reliable. For the first criteria, the variation in the value of the optimal block size over λ in both error detection and recovery is significant, while for the second criteria, the range of block sizes is much more limited. In other words, selecting the block size according to the second criteria, i.e., minimizing the worst-case overhead, significantly reduces the block size sensitivity to the changes in λ, and completely eliminates it in the case of error detection. This fact enables the use of frequent validation in cases where the statistical distribution of errors is not fully known, e.g., λ can not be accurately determined. For error detection, the optimal block size based on worst-case overhead is independent of λ, and hence is independent of the statistical error model used. For error recovery, the range of optimal block sizes corresponding to a wide variation in component reliability is limited, and hence a near-optimal selection can be made within this range based on the designer's perception of the potential of faults.

3.5.2 Security Implications of Frequent Validation

When a technique that resists a side-channel attack is employed, it is essential to verify that it does not expose the design to other attacks. In this section, we argue that the use of frequent validation for error detection and recovery does not enable other side-channel attacks.

3.5.2.1 Fault Analysis Attacks

When frequent validation is used in error detection or recovery, the validation test adopted must be able to detect errors resulting from all anticipated types of faults. In our earlier discussion, we used a validation test based on consistency checking that is known to detect errors resulting from random faults as well as faults injected by less-sophisticated attackers, which covers invalid-curve, sign change and safe-error attacks [23].

However, it has been assumed in our earlier discussion that the validation circuit or routine is error-free. If the validation circuit/routine is vulnerable to errors, infective computation can be used. In particular, all intermediate tests remain as before, while the last one is modified to use infective computation. This way, even if the attacker can inject faults during the computation and manipulate all the intermediate tests, the last test will mask the faulty result with random data and defeat the attack. Another option in the case of hardware implementations is to replicate the testing circuit to reduce or avoid testing errors.

3.5.2.2 Timing and Simple Power Analysis Attacks

In general, timing and simple power analysis attacks exploit the simple correlation of the computation time or power consumption with the value of the secret information. As such, countermeasures for this class of attacks attempt to eliminate this correlation such that simple analysis does not reveal any secret information. For example, iterations in a double-and-add-always algorithm have the same length regardless of values of corresponding bits of the key.

Assuming that the underlying algorithm is immune to timing attacks, the use of frequent validation introduces variability to the time requirements of the computation since the time taken to abort the computation (in the case of error detection) or finish it (in the cast of error recovery) is variable. However, this variability is independent of the secret information, and is only correlated with the occurrence times of faults. Even when a fault occurs, all iterations still take the same time and look similar to an attacker. This means that no information about the secret key can be learned through timing the computation.

3.5.2.3 Differential Power Analysis Attacks

Differential power analysis attacks are countered generally by randomizing an aspect of the computation, e.g., the base point, the exponent, intermediate points or the underlying curve. The designs presented above can be adapted to use any of the known randomization schemes. Moreover, randomization can be applied to the computation as a whole or to separate blocks. The latter can prevent an attacker from collecting a number of samples of a single block by repeatedly injecting faults and causing recomputations of that block. In this case, the cost of randomization can be included in the cost of validation testing.

3.6 Conclusion

In this chapter, we have proposed the use of frequent validation in error detection and recovery for transient faults for elliptic curve scalar multiplication systems. Earlier proposals dealt with the scalar multiplication as a black box and considered only testing the inputs/outputs for error detection and time or space redundancy for fault tolerance. In our approach, we divide the scalar multiplication into equal blocks of iterations and use efficient error detection schemes to detect errors early and reduce the loss caused by faults. Moreover, we use the same error detection schemes with partial recomputation to achieve efficient error recovery without requiring complete time or hardware redundancy.

In both applications, i.e., error detection and recovery, we show how the expected overhead can be used to select an optimal block size. We also show how a sub-optimal block size can be selected in a way that gives some advantages over the optimal choice, especially when the statistical properties of errors are unknown. The analysis and examples given illustrate that the use of frequent validation is considerably more efficient and reliable than known error detection and recovery schemes especially when errors are frequent or in the case of a fault attack. Thus, for many scenarios, the use of frequent validation in error detection and recovery can be a promising approach.

Chapter 4

Adaptive Error Recovery

4.1 Introduction

As discussed earlier, when a fault occurs, all computations performed between its occurrence and its detection are corrupted and their results will have to be discarded. This is a significant loss in time and power especially for an operation like ECSM, which has hundreds of iterations each of which involves a relatively large number of finite field operations on operands that are hundreds of bits in width.

This issue has been addressed in Chapter 3 where the use of frequent validation for error detection and recovery was proposed. In essence, the scalar multiplication operation is partitioned into equal blocks of iterations and the output of each block is validated before starting the next one. If an error is detected in the output of a block, that block is recomputed. Chapter 3 shows that error recovery based on frequent validation is more reliable and has less overhead compared to conventional error detection and recovery designs especially when errors are frequent.

However, a practical challenge to the approach proposed in Chapter 3 is that the selection of the overhead-optimal (or reliability-optimal) block size requires the knowing the parameters of the statistical error model. While this is acceptable in a theoretical setting, a more practical approach is to reduce or eliminate the need for an accurate error model. In this work, we propose an extension to the approach presented in Chapter 3 by which a statistical model for errors is no longer needed to achieve efficient error recovery. Instead of fixing the block size to an optimal value set in advance, we allow the block size to vary adaptively in a given range based on whether an error is detected or not. We show that adaptive error recovery can approach the efficiency and reliability of optimized fixed-block error recovery.

The remainder chapter is organized as follows. Section 4.2 discusses adaptive error recovery and describes an analytical model that can be used to estimate the expected overhead of this approach under the assumption of a constant error rate. These estimates are confirmed with simulation. In Section 4.3, simulation is used to explore the performance of adaptive error recovery under a more general error model with a variable error rate. Then, in Section 4.4, we show how Bayesian inference can be used to select a suitable adaptive policy under a given error model with unknown parameters. Finally, Section 4.5 discusses the effects of adaptive error recovery on the security of the ECSM design, and how some of the known security issues can be addressed. Part of the material in this chapter has appeared in [4].

4.2 Adaptive Error Recovery under a Constant Error Rate

In this section, we describe adaptive error recovery and present an analytical model that can be used to estimate its expected overhead. Then, a numerical example is presented and used, along with simulation, to validate the analytical results. While the following analysis assumes exponentially-distributed errors with fixed λ, it can be easily adapted to other error models with constant parameters.

4.2.1 Adaptive Error Recovery

While fixed-block error recovery has some advantages over comparable approaches to fault tolerance, it has the disadvantage of requiring the optimal (or reliability-optimal) block size to be determined prior to the computation based on the knowledge of error model parameters. As an alternative, we propose the use of adaptive error recovery, which employs a similar idea but where the block size is varied in response to the perceived error rate, resulting in less validation tests when errors are rare and more tests when errors are more frequent. While this approach can be less efficient than the fixed-block approach for a specific, known error rate, it will require no prior knowledge of the error rate and probably perform better in error rates different from the one for which an optimal fixed block size was selected.

Specifically, adaptive error recovery differs from fixed-block error recovery in the following aspects:

1. Rather than being constant, the block size, denoted by m, varies within a given range, $[m_l, m_h]$.
2. The variation of m is controlled by an adaptive policy based on whether or not an error is detected.

As a result, the number of iterations completed successfully does not progress in regular increments like before, but changes by an amount and probability that both depend on the current m. Algorithm 7 illustrates the adaptive error recovery approach. Note the use of two functions, *mUp()* and *mDown()*, that define the adaptive policy based on the occurrence or absence of errors. Although these functions can be chosen arbitrarily, it is generally more efficient to limit them to functions that are easily provided by a hardware register, e.g., an increment or a shift, rather than more complex functions.

Algorithm 7 Scalar multiplication with adaptive error recovery

Input: $P \in E$, $l = (l_{n-1}, l_{n-2}, l_{n-3}, \ldots, l_0)$, $[m_l, m_h]$ block size range, m_{init} block size initial value

Output: lP

```
 1: $Q_0 \leftarrow \mathcal{O}$, $Q_1 \leftarrow \mathcal{O}$, $Q_2 \leftarrow P$
 2: $j \leftarrow 0$, $H_0 \leftarrow Q_0$, $H_1 \leftarrow Q_1$, $H_2 \leftarrow Q_2$
 3: $m_c \leftarrow m_{\text{init}}$
 4: for $i = 0$ to $n - 1$ do
 5:     $Q_{l_i} \leftarrow Q_{l_i} + Q_2$
 6:     $Q_2 \leftarrow 2Q_2$
 7:     if $i - j = m_c$ or $i = n - 1$ then                    ▷ test at end of block
 8:         if $Q_0 \in E$ and $Q_1 \in E$ and $Q_2 = Q_0 + Q_1 + P$ then    ▷ error-free
 9:             $j \leftarrow i$, $H_0 \leftarrow Q_0$, $H_1 \leftarrow Q_1$, $H_2 \leftarrow Q_2$
10:             $m_c \leftarrow mUp(m_c)$        ▷ increase block size with an upper limit $m_h$
11:         else                                                 ▷ error detected
12:             $i \leftarrow j$, $Q_0 \leftarrow H_0$, $Q_1 \leftarrow H_1$, $Q_2 \leftarrow H_2$
13:             $m_c \leftarrow mDown(m_c)$      ▷ decrease block size with a lower limit $m_l$
14:         end if
15:     end if
16: end for
17: return $Q_1$
```

4.2.2 Modeling Adaptive Error Recovery

In this section, we describe a model to calculate the time overhead associated with adaptive error recovery. This model is similar in essence to the one presented in Chapter 3, but is superior in that it can be used to estimate the overhead of both fixed-block and adaptive error recovery.

4.2.2.1 A Statistical Cost Model

Recall the notation presented in the previous chapter. The random variable X represents the time until the occurrence of the next errors, while n is the total number of iterations. The cost estimates are given in terms of the following cost measures:

- c_b: cost of an iteration in a bare-bone ECSM algorithm.
- c: cost of an iteration in an error recovery algorithm.
- c_v: cost of a validation test.

These cost measures are represented using the same unit. In what follows, point operations are used to measure the cost, but other units like field or bit operations can be used.

The overhead of error recovery using frequent validation and partial recomputation is defined as the increase in time required relative to a bare-bone scalar multiplication algorithm. In order to estimate this overhead, it is divided into three parts:

1. Iteration redundancy overhead: This part captures the cost of extra computations added in each of the original n iterations to allow for validation tests. It can be found by $n(c - c_b)$.
2. Extra iterations overhead: This part captures the cost of extra iterations, beyond the original n, that are required for recomputation in case of an error occurrence. It can be computed as $(\tilde{n} - n)c$, where $\tilde{n}$ denotes the total number of iterations including recomputations.

3. Testing overhead: Each block requires a validation test. This part captures that cost and can be computed as kc_v, where k denotes the total number of blocks.

Summing up these parts, we get the following function that describes the total overhead in point operations.

$$\begin{aligned} f_o(\tilde{n}, k) &= n(c - c_b) + (\tilde{n} - n)c + kc_v \\ &= \tilde{n}c - nc_b + kc_v \end{aligned} \tag{4.1}$$

This expression depends on two variables, $\tilde{n}$ and k. It follows by the linearity of expectations that

$$\mathrm{E}[f_o(\tilde{n}, k)] = \mathrm{E}[\tilde{N}]c - nc_b + \mathrm{E}[K]c_v \tag{4.2}$$

where $\tilde{N}$ and K denote the random variables describing $\tilde{n}$ and k, respectively.

At block boundaries, the state of the system can be completely described by the number of valid iterations performed so far, denoted by n_d, and the current block size, denoted by m_c. Let N be a random variable representing this state as follows.

$$N \in \{(n_d, m_c) : n_d, m_c \in \mathbb{Z}, n_d \in [0, n), m_c \in [m_l, m_h]\} \cup \{(n)\} \tag{4.3}$$

where (n) is the absorbing state representing the end of the computation. The total number of states is $n(m_h - m_l + 1) + 1$. Let the sequence $\{N_s\}$ be such that $N_0 = (0, m_{\text{init}})$ and

$$N_{s+1} = \begin{cases} (N_s(n_d), mDown(N_s(m_c))) & \text{if an error is detected} \\ (N_s(n_d) + N_s(m_c), mUp(N_s(m_c))) & \text{if } m_c < n - n_d \text{, error-free} \\ (n) & \text{if } m_c \geq n - n_d \text{, error-free} \end{cases} \tag{4.4}$$

where $N_s(n_d)$ and $N_s(m_c)$ are the count of valid iterations performed so far and the current block size at step s, respectively, while $mDown$ and mUp are the functions that define the adaptive policy. Since the next state depends only on the current state and whether or not the current block has an error, this sequence is a Markov chain, and it can be used to find $\mathrm{E}[\tilde{N}]$ and $\mathrm{E}[K]$.

To describe this Markov chain we use an *Augmented Transition Matrix*, denoted by $\mathcal{A}(x)$, which is defined as follows.

$$\mathcal{A}(x) = \begin{pmatrix} \mathbf{A}(x) & \mathbf{A}^0(x) \\ \mathbf{0} & 1 \end{pmatrix} \tag{4.5}$$

where $\mathbf{A}(x)$ is a square matrix with dimension $n(m_h - m_l + 1)$, and $\mathbf{A}^0(x)$ is column vector satisfying $\mathbf{A}^0(1) + \mathbf{A}(1)\mathbf{1} = \mathbf{1}$. Here, $\mathbf{0}$ and $\mathbf{1}$ represent the all-0 and all-1 vectors, respectively, and the last row and column of $\mathcal{A}(x)$ correspond to the absorbing state (n). This matrix is a more general form of the conventional transition probability matrix in the sense that each transition probability is augmented with the corresponding transition distance or cost as the exponent of the variable x. More formally, each element $a_{ij}(x)$ of $\mathbf{A}(x)$ is defined as follows.

$$\begin{aligned} a_{ij}(x) &= p_{ij}x^{d_{ij}} \\ p_{ij} &= \Pr[N_{s+1} = (n_d, m_c)_j | N_s = (n_d, m_c)_i] \\ d_{ij} &= \min((m_c)_i, n - (n_d)_i) \end{aligned} \tag{4.6}$$

A conventional transition probability matrix can be obtained as $\mathcal{A}(1)$. The transition between any two states is determined by the adaptive policy and the error probability. All non-terminal states belong to one of two classes:

1. $(n_d, m_c) : m_c < n - n_d$. These states represent all blocks except for the last one in the computation. In this case, the next state is either $(n_d + m_c, mUp(m_c))$ with probability $\Pr[X \geq m_c] = e^{-\lambda m_c}$ (error-free), or $(n_d, mDown(m_c))$ with probability $\Pr[X \leq m_c] = 1 - e^{-\lambda m_c}$ (error detected). In both cases, the power of x is the distance, which is m_c.

2. $(n_d, m_c) : m_c \geq n - n_d$. These states represent the (potentially) last block. The next state is either $(n_d, mDown(m_c))$ with probability $\Pr[X \leq (n - n_d)] = 1 - e^{-\lambda(n-n_d)}$ (error detected), or the absorbing state (n), with probability $\Pr[X \geq (n - n_d)] = e^{-\lambda(n-n_d)}$ (error-free). In both cases, the power of x is the distance, which is $n - n_d$.

A probability vector, denoted by α, can be used to represent the initial conditions, e.g., the initial block size.

4.2.2.2 Finding E$[K]$

We now show how $\mathcal{A}(x)$ can be used to find the expected value of K, the number of blocks. In general, for a transition matrix $\mathcal{T}$ of the form

$$\mathcal{T} = \begin{pmatrix} \mathbf{T} & \mathbf{T}^0 \\ \mathbf{0} & 1 \end{pmatrix} \tag{4.7}$$

where $\mathbf{T}$ is a square matrix, $\mathbf{T}^0$ is a column vector and $\mathbf{T}^0 + \mathbf{T1} = \mathbf{1}$, the number of steps required to reach the absorbing state follows a *Discrete Phase-type distribution*, denoted by DPH$(\tau, \mathbf{T})$, where τ is a probability vector representing the initial distribution. Note that when $\mathcal{T} = \mathcal{A}(1)$ and $\tau = \alpha$, this distribution applies to K. It follows from the properties of the discrete phase-type distribution that

$$\Pr[K = k] = \alpha \mathbf{A}^{k-1}(1)\mathbf{A}^0 \tag{4.8}$$

and

$$\mathrm{E}[K] = \alpha(I - \mathbf{A}(1))^{-1}\mathbf{1} \tag{4.9}$$

where I is the appropriately-sized identity matrix.

4.2.2.3 Finding E$[\tilde{N}]$

The matrix $\mathcal{A}(x)$ can also be used to find the expected total number of iterations, E$[\tilde{N}]$. Recall that for each non-zero element $a_{ij}(x)$ of $\mathcal{A}(x)$, the coefficient represents the transition probability from state i to j while the exponent represents the distance, i.e., number of iterations in the transition. So, to find the expected number of iterations between any two states i and j, we multiply the exponents by their respective coefficients and then divide by the sum of coefficients to normalize.

More formally, for a polynomial $a_{ij}(x) \neq 0$, we can find the expected number of iterations in the transition between states i and j as $\frac{a'_{ij}(1)}{a_{ij}(1)}$, where $a'_{ij}(1)$ is the first derivative of $a_{ij}(x)$ with respect to x evaluated at $x = 1$. This idea extends naturally to powers of $\mathcal{A}(x)$ to find the expected number of iterations over more than one block.

To find $\mathrm{E}[\tilde{N}]$, we find the expected number of iterations from the starting state to the absorbing state for all values of k and find their sum weighted by the probability of each respective k. More formally,

$$\begin{aligned}\mathrm{E}[\tilde{N}] &= \sum_{k \in K} \Pr[K = k] \mathrm{E}[\tilde{N} | K = k] \\ &= \sum_{k \in K} \Pr[K = k] \sum_{\tilde{n} \in \tilde{N}} \tilde{n} \Pr[\tilde{N} = \tilde{n} | K = k] \\ &= \sum_{k \in K} \Pr[K = k] \frac{(\frac{d}{dx}(\alpha \mathcal{A}^k)_{-1})(1)}{(\alpha \mathcal{A}^k)_{-1}(1)} \qquad (4.10)\end{aligned}$$

where $(\alpha \mathcal{A}^k)_{-1}$ is the last element in the row vector $\alpha \mathcal{A}^k$. It is important to note that the last element in the last row of $\mathcal{A}(x)$ corresponding to $\Pr[N_{s+1} = (n) | N_s = (n)]$ should be set to 0 to prevent the accumulation of elements in the last column of $\mathcal{A}^k(x)$ between successive values of k. This reflects the fact that no more steps are taken beyond the absorbing state. Using (4.9) and (4.10), we can find the expected value of the total overhead in (4.1).

Note on computational complexity. Finding $\mathrm{E}[\tilde{N}]$ requires raising $\mathcal{A}(x)$ to the power of all values of k, which will cause the number of nonzero elements to increase quickly. This can be mitigated by using α to reduce matrix multiplications to vector-matrix multiplications, e.g., computing $\alpha \mathcal{A}^3(x)$ as $(((\alpha \mathcal{A}(x))\mathcal{A}(x))\mathcal{A}(x))$ rather than $\alpha(\mathcal{A}^3(x))$. However, since elements are polynomials, this still allows the number of terms in each polynomial element to increase rapidly. As such, computing the powers of $\mathcal{A}(x)$ can become costly unless the number of states is limited. Here we give another representation of $\mathcal{A}(x)$ that significantly reduces the computational cost of finding $\mathrm{E}[\tilde{N}]$.

Note that in (4.10), for each k, we need to raise $\mathcal{A}(x)$ to the power of k in order to get a specific polynomial, denoted here by $a(x)$, which describes the k-steps transition from the starting state to the absorbing state. The average distance using k steps can then be found by $\frac{a'(1)}{a(1)}$. This value is multiplied by the probability of k and accumulated to get the average total number of iterations required. The key here is the computation of the value $\frac{a'(1)}{a(1)}$. Since $a(x)$ is a dot product, we can write $a(x) = \sum_i h_i(x)g_i(x)$ where $h(x)$ and $g(x)$ are row and column vectors, respectively. It follows that

$$a'(x) = \sum_i (h_i(x)g_i(x))' = \sum_i h_i'(x)g_i(x) + h_i(x)g_i'(x) = h'(x)g(x) + h(x)g'(x) \tag{4.11}$$

So, all we need to calculate $a(1)$ and $a'(1)$ is to know $h(1)$, $h'(1)$, $g(1)$ and $g'(1)$. It follows that the required powers of the polynomial matrix $\mathcal{A}(x)$ can be found using a pair of purely-numerical matrices, $\mathcal{A}(1)$ and $\mathcal{A}'(1)$. This reduces the computation cost significantly since it replaces each polynomial multiplication with three numerical multiplication, one to obtain $a(1)$ and two to obtain $a'(1)$. It also leads to a significant saving in memory requirements as it avoids the rapid growth in the size of polynomial elements.

4.2.2.4 Short- and Long-term Behavior

In the preceding analysis, it was mentioned that the probability vector α can represents the initial distribution and can be used to select the initial block size by setting all elements to 0 except for the corresponding state, $(0, m_{\text{init}})$, which is set to 1. This effectively captures the transient behavior of the system and models the scenario where the initial block size is reset to m_{init} before starting each scalar multiplication.

However, it is possible not to reset the initial block size and keep the last used block size between completed computation. This has the advantage of maintaining the acquired information about the error rate and the suitable block size. To model this behavior, we need to find the long-term distribution for the block sizes and use it as α. We find this information as follows. First, we define M, which is the random variable that describes the current block size. Then, another Markov chain, denoted by $\{M_i\}$, is constructed which only describes the variation in block size, i.e., where states represent different block sizes within the allowed range. In particular, for $M_i = m$, the next state will be either

$M_{i+1} = mUp(m)$ with probability $e^{-\lambda m}$, or $M_{i+1} = mDown(m)$ with probability $1 - e^{-\lambda m}$. Given the transition matrix of this chain, we need to find the associated stationary distribution. Such distribution, denoted by π, can be computed as the normalized left eigenvector of the transition matrix associated with the eigenvalue 1. Given π, we can set the initial state in α to reflect the stationary distribution, and the estimated overhead will then be the expected long-term overhead.

4.2.3 Numerical Examples

4.2.3.1 Fixed-block Error Recovery Example

We illustrate the use of the model discussed earlier and validate it by comparing its results to the model developed earlier in Chapter 3. To enable this comparison, we use Algorithm 6 as in Chapter 3 with similar parameters, i.e., $n = 256$, $c = 2$, $c_b = 3/2$ and $c_v = 4$. Two criteria were proposed in Chapter 3 to find an optimal block size. The more intuitive one is to select the block size, denoted by m_{opt}, to minimize the expected overhead. It is also possible to select the block size, denoted by $m_{\text{r-opt}}$, to minimize the required overhead to achieve a given reliability requirement, e.g., 99.99%. In this example, we use the optimal block sizes for various values of λ as computed in Chapter 3. Table 4.1 compares the results obtained from the model described in this section to the earlier results. It can be seen from the table that the two models produce matching results, thus confirming the validity of the preceding analysis.

We also use simulation to validate the results the preceding analysis. For fixed-block error recovery, Table 4.2 shows some of the simulation results and compares them with the results obtained through analysis.

4.2.3.2 Adaptive Error Recovery Example

We also demonstrate the results that can be achieved using adaptive error recovery. This example uses the same basic parameters used in the earlier example. Moreover, we have

Table 4.1: Expected overhead results in point operations using the earlier model in Chapter 3 and the model in Section 4.2.2.1. Values between parentheses represent the percentage overhead relative to a bare-bone ECSM implementation.

λ		m	Exp. overhead, earlier model	Exp. overhead, (4.1)
0.05	m_{opt}	5	536 (140%)	539
	$m_{\text{r-opt}}$	3	607 (158%)	609.8
0.01	m_{opt}	13	289 (75%)	289.2
	$m_{\text{r-opt}}$	7	322 (84%)	323.4
0.001	m_{opt}	44	175 (46%)	175.5
	$m_{\text{r-opt}}$	11	228 (59%)	230.6

Table 4.2: Comparison of expected overhead using the model in Section 4.2.2.1 with the simulation results. The expected overhead from the simulation is shown with a 99% confidence interval. Also, the reliability of the expected overhead and the overhead at 99% reliability are shown

λ		m	Analysis	Simulation		
			E[OH]	E[OH]	Rel at E[OH]	OH at 99% Rel.
0.05	m_{opt}	5	539	540.5 $\pm$4.8	50%	686
	$m_{\text{r-opt}}$	3	609.8	609.6 $\pm$4.3	48%	702
0.01	m_{opt}	13	289.2	290.0 $\pm$4.3	50%	448
	$m_{\text{r-opt}}$	7	323.4	322.5 $\pm$2.5	52%	402
0.001	m_{opt}	44	175.5	174.9 $\pm$3.8	78%	336
	$m_{\text{r-opt}}$	11	230.6	230.7 $\pm$1.1	77%	302

to select an adaptive policy, which amounts to setting the following parameters:

1. The block size bounds, m_l and m_h.
2. The block size update functions, $mUp()$ and $mDown()$.

As shown in Chapter 3, block sizes that are optimized for reliability are smaller than ones optimized for efficiency. Moreover, the former have an expected overhead that is close to the latter, and the former is less sensitive to the variation in the error rate. As such, it is sensible to give preference to smaller blocks as opposed to bigger blocks when setting the adaptive policy parameters. It is sensible as well to increase the block size (in the absence of error) slower than it is decreased (when an error is detected). This is because smaller blocks generally provide a better trade-off between efficiency and reliability. Based on this argument, a good choice for the block size modification functions can be $mUp(m) = m + 1$ and $mDown(m) = \lfloor m/2 \rfloor$, as they are both easy to implement and satisfy the grow-slow/shrink-fast requirement. We look more closely at the selection of the adaptive policy in Section 4.4.

As for the range of block size values, we can select a range that includes the optimal block sizes for a range of values of λ, e.g., $[5, 40]$. In general, higher values of m should be avoided as they lead to less frequent validation tests, and potentially more expensive recomputations when errors occur.

Now that we have some values for these parameters, we can examine the results over a range of values for the initial block size, m_{init}, and the error rate, λ. Table 4.3 shows some analytical results for this example while Table 4.4 shows some simulation results. Figures 4.1, 4.2 and 4.3 show the expected overhead and the 99% reliability threshold for the short-term adaptive alternative compared to the fixed-block alternative for three values of λ over a range of initial block sizes. These tables and figures show that adaptive error recovery is generally more efficient than the fixed-block alternative except when the fixed block size happens to be near the optimal block size for a certain λ. Even in this case, the overhead of the adaptive alternative is not much higher. Moreover, while the 99%

Table 4.3: The expected overhead for adaptive error recovery for three values of λ. The expected overhead and expected block size are shown for three values of m_{init} in three error recovery scenarios, namely, fixed-block, and short- and long-term adaptive.

λ	m_{init}	Fixed-block	Adaptive, short-term		Adaptive, long-term	
		E[OH]	E[OH]	E[M]	E[OH]	E[M]
0.05	5	539.0	568.4	6.81	570.8	6.90
	15	845.8	587.3	7.23		
	40	3419.0	676.5	8.15		
0.01	5	372.6	304.3	10.13	302.5	11.83
	15	293.5	301.2	12.45		
	40	408.0	338.2	17.69		
0.001	5	339.6	215.6	13.00	181.3	28.33
	15	208.7	194.4	18.58		
	40	177.1	178.8	33.44		

reliability threshold in the adaptive case varies with the initial block size, its variation is limited and indicates a low sensitivity to the choice of the initial block size.

On the other hand, Figure 4.4 compares the expected overhead of long-term adaptive error recovery to the overhead of the optimal fixed block size, i.e., one selected with the knowledge of λ, and shows that adaptive error recovery closely matches the overhead of optimized fixed-block error recovery in the long term. This indicates that, while the selection of m_{init} affects the expected overhead in the short term, the behavior in the long term will approach that of the best choice of m_{init} regardless of the actual initial block size. These results lead to the conclusion that the use of adaptive error recovery alleviates both the need to know the error rate in advance and the need to select a good value for m_{init}.

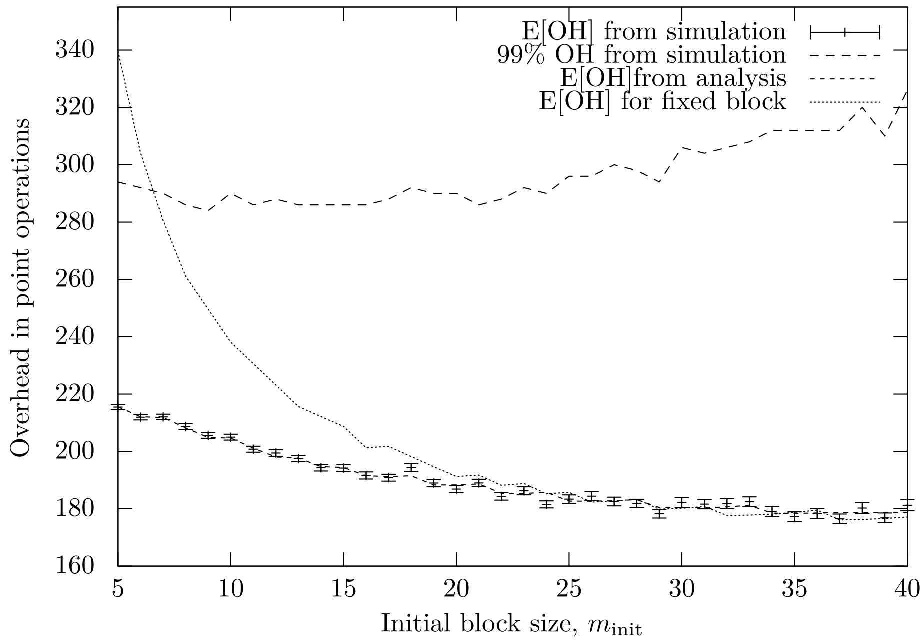

Figure 4.1: The expected overhead and 99% reliability threshold of adaptive error recovery compared to the fixed-block case for $\lambda = 0.001$. Simulation results are shown within 99% confidence interval.

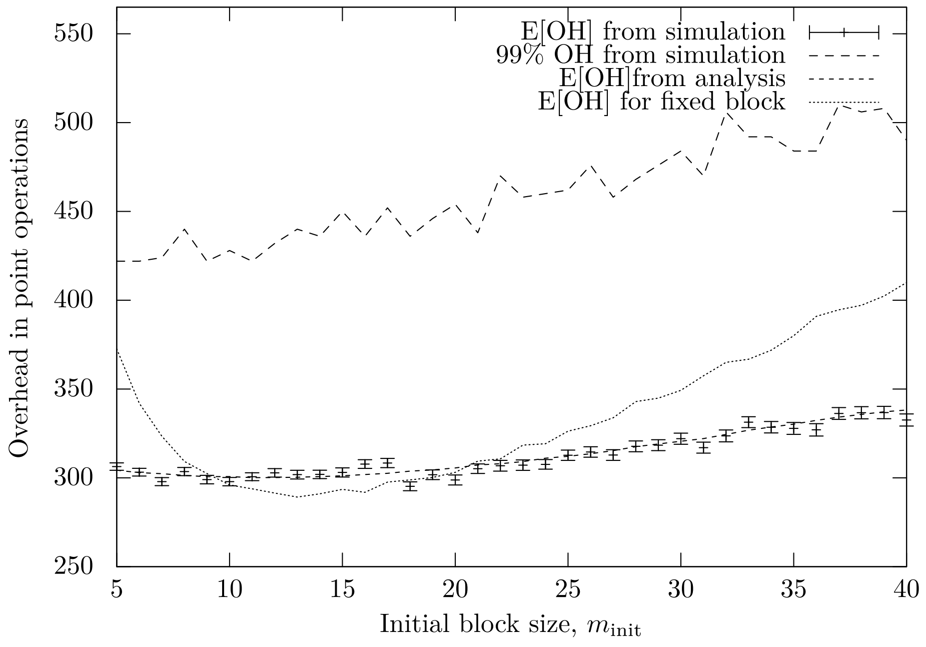

Figure 4.2: The expected overhead and 99% reliability threshold of adaptive error recovery compared to the fixed-block case for $\lambda = 0.01$.

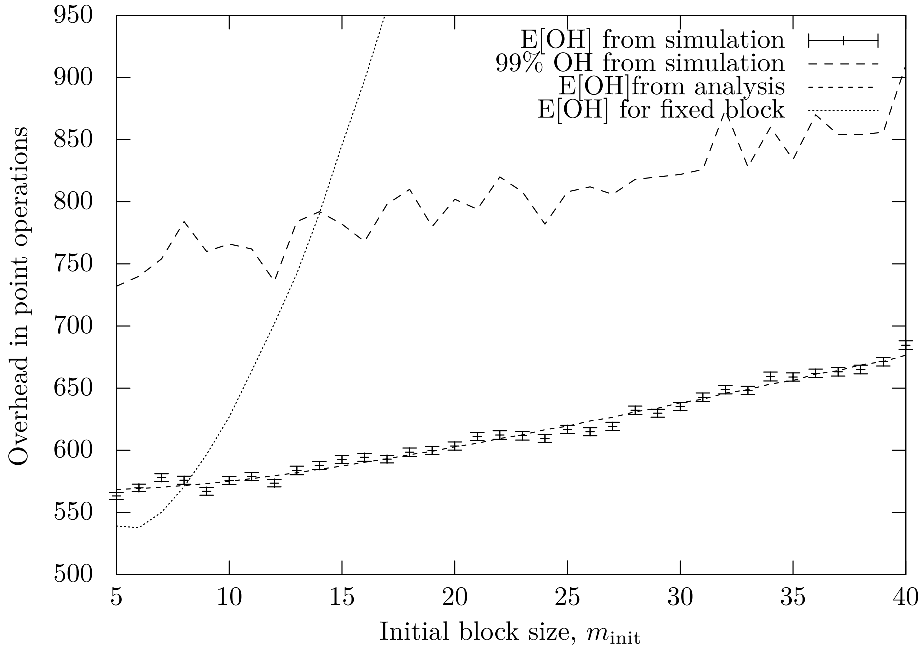

Figure 4.3: The expected overhead and 99% reliability threshold of adaptive error recovery compared to the fixed-block case for $\lambda = 0.05$.

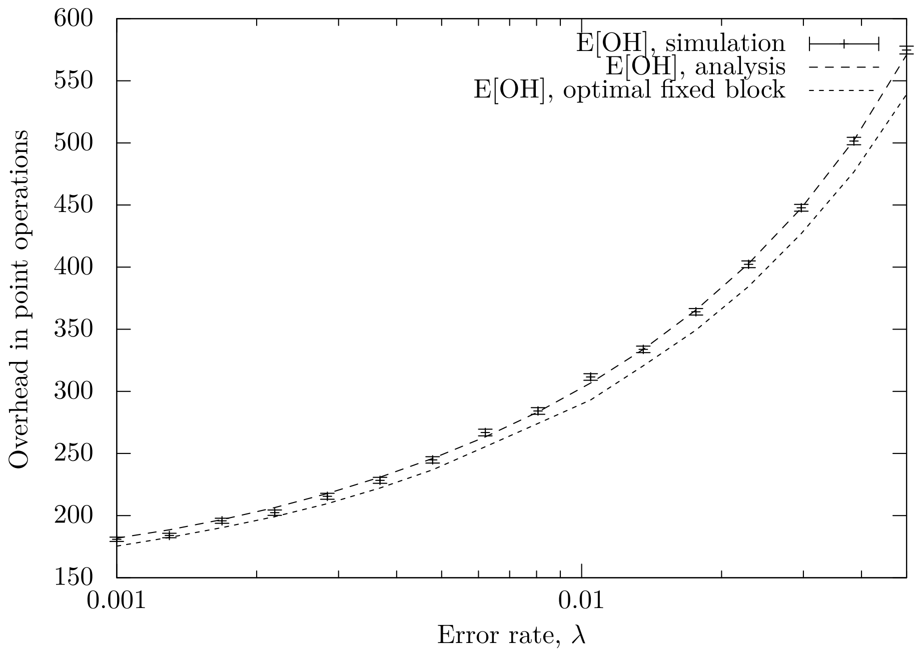

Figure 4.4: The expected overhead for long-term adaptive error recovery compared to the overhead of the optimal block size for a range of values of λ.

4.3 Error Recovery under a Variable Error Rate

In the previous section, the overhead associated with fixed-block and adaptive error recovery has been estimated using an analytical model and simulation under the assumption of a constant error rate. Both fixed-block and adaptive error recovery can perform relatively well under a constant error rate. In particular for the fixed-block alternative, the knowledge of the error rate is needed to optimize the block size for a particular error rate. It can be argued, however, that the constant error rate assumption does not capture the complexity of error occurrences in reality where the error rate is mostly either unknown or variable. In this section, we use simulation to explore the performance of both fixed-block and adaptive error recovery under randomly varying error rate.

4.3.1 A More General Error Model

For a more realistic error model, it is essential to address the possibility of burst errors, i.e., sudden increases in the perceived error rate that lead to the concentration of errors in a limited time period. This can be achieved by extending the error model discussed earlier using a variable error rate. In particular, a range of error rates, $[\lambda_{\text{low}}, \lambda_{\text{high}}]$, is selected and λ is picked randomly at block boundaries following a log-uniform distribution with parameters $\ln \lambda_{\text{low}}$ and $\ln \lambda_{\text{high}}$, i.e., $\ln \lambda \sim U(\ln \lambda_{\text{low}}, \ln \lambda_{\text{high}})$. The expected value of λ will then be $\frac{\lambda_{\text{high}} - \lambda_{\text{low}}}{\ln \lambda_{\text{high}} - \ln \lambda_{\text{low}}}$. This distribution has the property that most of its mass is closer to the lower end, but that occasionally results can come from the higher end. In effect, this represents the scenario where errors are generally rare with bursts of higher error rate.

4.3.2 A Numerical Example

We give an example to demonstrate the effectiveness of both fixed-block and adaptive error recovery under a variable error rate. We simulate both under the extended error model described earlier and gather information like the average overhead, the average block size and the overhead threshold that provides 99% reliability. The parameters used here are the same as earlier examples. For the range of λ, we use the conservative and relatively wide range $[0.001, 0.05]$. Error rates higher than 0.05 are excluded as they amount to 0 reliability of the original design, while error rates lower than 0.001 are low and can skew the simulation results to be more optimistic.

Table 4.5 shows the simulation results. As expected, adaptive error recovery is generally less sensitive to the value of m_{init}. Moreover, it can be more efficient and offer less expensive reliability than the fixed-block alternative. The only exception is when the fixed m_{init} is selected to be close to the unknown, optimal block size, which naturally results in less expected overhead for the optimized fixed-block. However, even then, adaptive blocks respond better to the variability of λ and give smaller reliability threshold.

Figure 4.5 shows the overhead for both short- and long-term adaptive error recovery compared to the fixed-block approach under a variable error rate. With the exception block sizes that happen to be close to optimal, adaptive blocks have less expected overhead than fixed ones. Figure 4.6 shows the 99% reliability overhead for short- and long-term adaptive error recovery compared to fixed blocks. Similar to the preceding figure, for all block sizes except for a small near-optimal range of fixed block sizes, adaptive error recovery offers the same reliability with significantly smaller expected overhead.

4.4 Adaptive Policy Selection

Earlier, we have observed that the use of smaller blocks results in higher reliability and reduced recomputations especially when errors are frequent. This led us to propose an adaptive policy that decreases the block size quickly when an error is detected and increases it slowly otherwise. In particular, we used the policy $mDown(m) = \lfloor m/2 \rfloor$ and $mUp(m) = m + 1$ in our analysis as a specific realization that is easy to implement in a hardware register. The numerical example given earlier shows that this simple adaptive policy is effective in approximating the performance of an optimized fixed-block

Table 4.4: Simulation results for fixed-block and both short- and long-term adaptive error recovery under a constant error rate.

λ	m_{init}	Fixed-block		Adaptive, short-term				Adaptive, long-term			
		E[OH]	99% Rel OH	E[OH]	E[OH] Rel	99% Rel OH	E[M]	E[OH]	E[OH] Rel	99% Rel OH	E[M]
	5	539	700	568	51%	752	6.8				
0.05	15	835	1458	586	51%	776	7.2	572	51%	756	7
	40	3333	7608	676	50%	860	8.1				
	5	372	434	304	52%	424	10.1				
0.01	15	293	438	298	50%	436	12.5	303	51%	448	12.7
	40	405	912	331	50%	506	18.1				
	5	339	364	215	77%	292	13				
0.001	15	207	268	195	77%	288	18.6	180	77%	316	30.6
	40	176	324	179	77%	316	33.5				

Table 4.5: The simulation results for both fixed-block and adaptive error recovery starting from the same value of m_{init} under a variable λ in the range $[0.001, 0.05]$. In all runs, $\mathrm{E}[\lambda] = 0.0125$.

m_{init}	Fixed			Adaptive, short-term				Adaptive, long-term			
	E[OH]	Rel E[OH]	99% Rel OH	E[OH]	Rel E[OH]	99% Rel OH	E[M]	E[OH]	Rel E[OH]	99% Rel OH	E[M]
5	380	61%	448	319	48%	448	9.8	317	46%	464	11.9
15	307	61%	506	314	46%	462	11.9	315	47%	468	12
40	411	62%	912	348	45%	536	17.0	317	46%	466	11.9

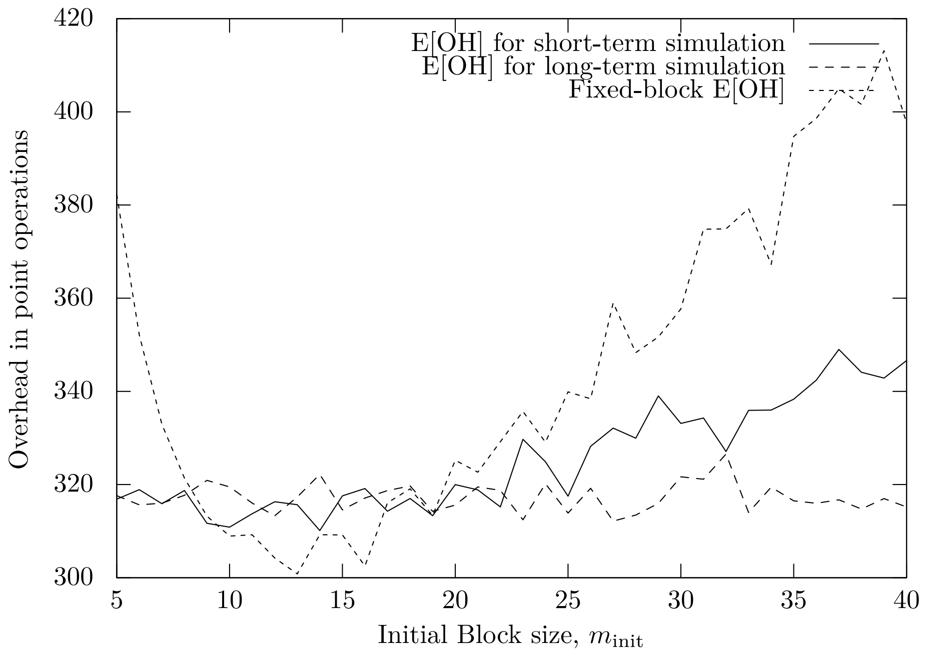

Figure 4.5: The expected overhead for adaptive compared to fixed-block for both short- and long-term under a variable error rate

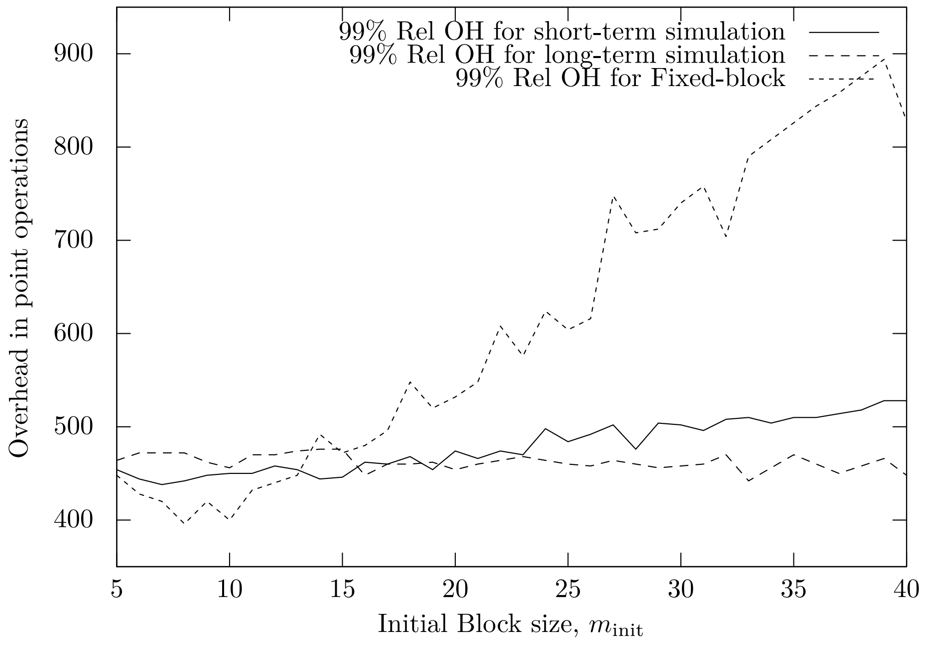

Figure 4.6: The 99% reliability overhead for adaptive compared to fixed-block for both short- and long-term under a variable error rate

design while resulting in a smaller overhead variance and avoiding the need to know the parameters of the error model in advance.

In this section, we take a closer look at the selection of the adaptive policy with the aim of evaluating the policy proposed earlier. We achieve this by modeling the process of updating the block size as a statistical inference procedure in which the distribution family followed by the data is known but the parameter that identifies the specific distribution is unknown. The goal of the inference procedure is to infer or learn the value of the parameter using the observed data.

4.4.1 Bayesian Inference in a Nutshell

This section gives a brief introduction of Bayesian inference. A reference on the subject of Bayesian statistics like [36] or [12] can be consulted for further details.

In the Bayesian approach to inference, newly acquired data or evidence is used to update a prior distribution over the hypotheses space resulting in a posterior distribution that captures the new evidence. This approach relies on the use of Bayes' rule, which can be stated as follows.

$$\Pr[A|B] = \frac{\Pr[B|A]\Pr[A]}{\Pr[B]} \tag{4.12}$$

for any random variables A and B.

Under the assumption that a random variable X follows a known distribution with unknown parameter θ, the inference starts by encoding the initial belief or information about θ in a probability distribution over the domain of θ, denoted $\Pr[\theta]$. Then, when a sample x is observed, the belief in θ is updated as follows.

$$\Pr[\theta|x] = \frac{\Pr[\theta]\Pr[x|\theta]}{\Pr[x]} \tag{4.13}$$

where $\Pr[x|\theta]$ is the likelihood of observing x under θ, $\Pr[x]$ is the probability of observing x, and $\Pr[\theta|x]$ is the posterior distribution of θ given that x was observed.

Generally, any distribution can be used to describe $\Pr[\theta]$. However, for some statistical distributions of X, the prior distribution can be selected in a way that makes the prior and posterior distributions belong to the same family, which is analytically and computationally convenient. Such distributions are called conjugate priors, and their parameters are called the hyperparameters. When conjugate priors are employed, observed data is used to update the hyperparameters of the prior distribution to generate the hyperparameters of the posterior. This works because the prior and the posterior belong to the same family.

For example, for a random variable X that follows an exponential distribution, θ represents the error rate, λ. The conjugate prior of the exponential distribution is the gamma distribution, which has two parameters, α and β. So, the prior distribution, $\Pr[\lambda]$, follows a $\text{Gamma}(\alpha, \beta)$ distribution for some initial α and β. Assuming that a sample x was observed, it can be shown that the posterior, $\Pr[\lambda|x]$, follows a $\text{Gamma}(\alpha + 1, \beta + x)$ distribution. This procedure can be repeated when new samples are observed by using the posterior as the new prior. The gamma distribution is also the conjugate prior for the Pareto distribution, and specifically for the parameter γ. The posterior, however, has a different form, i.e., $\Pr[\gamma|x] \sim \text{Gamma}(\alpha + 1, \beta + \ln x)$.

The posterior distribution can be used to establish confidence intervals for the value of the parameter. However, when a point estimate of the unknown parameter is required, it is common to select the mode of the posterior, i.e., the value with the highest probability as the parameter's point estimate. This estimate is commonly called the Maximum A posteriori Probability (MAP) estimate. MAP estimates are similar to the well-known maximum likelihood estimates with a key difference, namely, that the MAP estimate takes the prior distribution into account.

4.4.2 Modeling the Adaptive Policy using Bayesian Inference

As mentioned earlier, we model the block size updates using Bayesian inference. It is assumed that the error statistical model is known and has parameter θ, and that we have an initial block size, m_{init}. In what follows, we discuss the general procedure, which is illustrated in Figure 4.7. Then, we give specific steps for errors following the exponential and Pareto distributions.

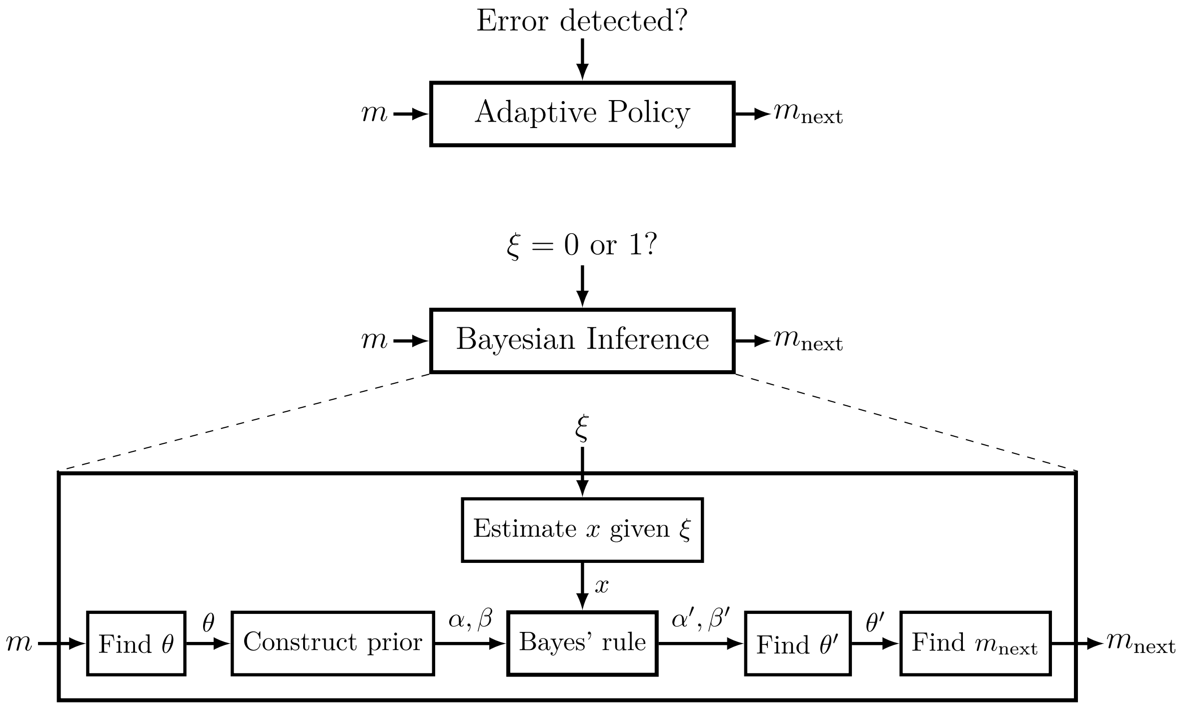

Figure 4.7: The use of Bayesian inference to model the adaptive policy.

4.4.2.1 General Procedure

1. Setup: We construct a prior distribution for the parameter θ.
    (a) We have expressions for the optimal block size given the parameter. We use these expressions to find the value of θ for which m_{init} is optimal and call it θ_{init}.
    (b) Then, we select the hyperparameters such that θ_{init} is the mode of the prior. The hyperparameter α acts as a counter of the number of samples observed plus one so we set $\alpha = 2$ since we have one observation that is the first block size. The hyperparameter β can then be found directly using θ_{init} and the mode expression.
2. After each block: We use the observed result of the validation test to update the hyperparameters and obtain the posterior distribution. Denote the result of the validation test by $\xi \in \{0, 1\}$, 0 for error-free and 1 when an error is detected.
    (a) Since the exact occurance time of an error is not known, we update the hyperparameters using $\mathrm{E}[X|\xi]$ in place of the unknown x.
    (b) The posterior is now known using the updated hyperparameters. We take its mode as the new θ, and use the optimal block size expressions to find the new m.

Issues. There are two issues that need to be addressed here. The first is the use of $\mathrm{E}[X|\xi]$ in place of the missing observation x, while the second is the approach used to accumulate the observations.

The use of $\mathrm{E}[X|\xi]$ in place of x effectively implies the inclusion of all values of x weighted by their probability given the observed value of ξ. This is commonly done when inference is performed using incomplete or censored data, e.g., in the expectation-maximization algorithm [20]. Given that $\xi = 0$ and 1 implies no error and error detected, respectively, it follows that $\mathrm{E}[X|\xi = 0] = \mathrm{E}[X|X \geq m]$ and $\mathrm{E}[X|\xi = 1] = \mathrm{E}[X|X \leq m]$.

The second issue is the procedure for accumulating the observations. There are at least four ways to accumulate the observed data when modifying the hyperparameters.

1. Accumulate all data with equal weights: This is a commonly-used approach in Bayesian learning, The advantage of this approach for learning is that the effect of new observations on the hyperparameters gets smaller and smaller as more data is gathered, i.e., the variance of the posterior distribution decreases as more data is collected. However, in our case, it is preferable that the posterior can adapt to recent observations since they are relatively more informative than older observations.

2. Keep only the latest observation: On the other end of the spectrum, this approach eliminates all past observations and uses only the most recent. This approach has the opposite problem, namely, the behavior becomes volatile and outliers have an exaggerated effect since all past observations are forgotten.

3. Keep a moving window of observations: This approach partially fixes the problems with the previous two. However, the problem described in the second approach above still occurs when the oldest observation in the window and the most recent one have a big difference, which can lead to sudden jumps.

4. Use an exponentially-smoothed accumulation: In this approach, the new hyperparameter is a linear combination of its previous value the most recent observation. This has the advantages of the window approach while avoiding its problem. The preference of older vs. newer observations can be controlled by the weight used in the linear combination, which is aptly called the smoothing coefficient. This is the approach that is most suitable in our case.

4.4.2.2 Exponentially-distributed Errors

In this section, we tailor the general procedure given above to exponentially-distributed errors. The parameter we need to infer or learn is the error rate λ, and as mentioned earlier, the conjugate prior of the exponential distribution is the gamma distribution.

The procedure for the exponential distribution is as follows. Initially, we only know the initial block size m_{init}.

1. Setup: Construct the prior distribution.

   (a) Find λ_{init} from m_{init}: Since $m_{\text{opt}} = \sqrt{\frac{2}{\lambda} + 1} - 1$, we can write this expression in terms of m to get $\lambda_{\text{init}} = \frac{2}{(m_{\text{init}}+1)^2 - 1}$.

   (b) Let $\alpha_{\text{init}} = 2$. As explained earlier, this is because we only have one data point so far.

   (c) Since the mode of Gamma(α, β) is $\frac{\alpha-1}{\beta}$, let $\beta_{\text{init}} = \frac{\alpha_{\text{init}}-1}{\lambda_{\text{init}}}$.

   (d) The prior distribution is then Gamma$(\alpha_{\text{init}}, \beta_{\text{init}})$.

2. At the end of each block: Update the hyperparameters using ξ to obtain the posterior.

   (a) $\beta \leftarrow w\mathrm{E}[X|\xi] + (1-w)\beta$ where

$$\mathrm{E}[X|\xi] = \begin{cases} \mathrm{E}[X|X \geq m] = \mathrm{E}[X] + m & \text{for } \xi = 0 \\ \mathrm{E}[X|X \leq m] = \mathrm{E}[X] - \frac{m}{e^{\lambda m}-1} & \text{for } \xi = 1 \end{cases} \tag{4.14}$$

   As an aside, note that $\mathrm{E}[X|X \geq m]$ is a shifted version of $\mathrm{E}[X]$, which is a consequence of memorylessness.

   (b) Let λ be the mode of the posterior. $\lambda = \frac{\alpha-1}{\beta}$.

   (c) Let m be the optimal block size for λ. $m = \left\lfloor \sqrt{\frac{2}{\lambda} + 1} - 1 \right\rfloor$.

Note that α is not updated after each block. The reason is that observations are combined with weights that add to one. In effect, β contains only one observation that is a weighted average of all the past observations. Sine α tracks the number of observations, it stays unchanged.

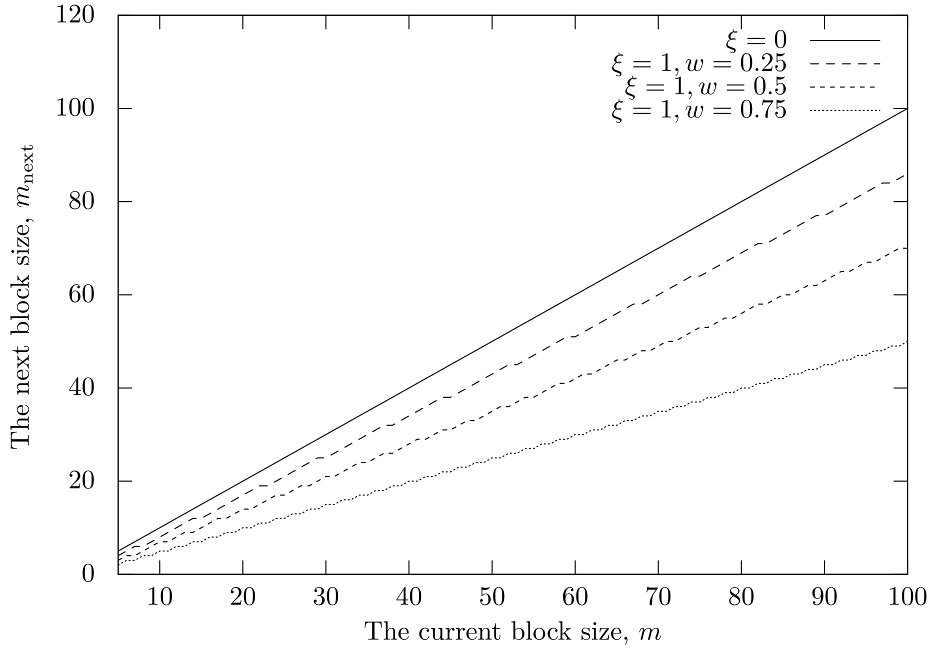

Figure 4.8: The next block size as a function of the current block size for different values of ξ and w assuming exponentially-distributed errors.

Resulting block sizes. When this procedure is applied to the current m, we get the following expressions for the next block size m_{next} as a function of current block size m and the smoothing coefficient w.

$$m_{\text{next}} = \begin{cases} \left\lfloor \sqrt{(1+m)^2 + 2mw} \right\rfloor - 1 & \text{for } \xi = 0 \\ \left\lfloor \sqrt{1 + m(2+m+w) - mw \coth\left(\frac{1}{2+m}\right)} \right\rfloor - 1 & \text{for } \xi = 1 \end{cases} \tag{4.15}$$

where coth is the hyperbolic cotangent function. Figure 4.8 illustrates these expressions by plotting the m_{next} versus the current m for three values of w, namely, 0.25, 0.5 and 0.75. Moreover, Table 4.6 shows the slopes of the lines that fit the curves in Figure 4.8.

4.4.2.3 Pareto-distributed Errors

Now we tailor the general approach to Pareto-distributed errors. The parameter we need to estimate is γ, the shape parameter. Recall that we set x_0, the scale parameter, to one

Table 4.6: Slopes of the lines that fit the curves in Figure 4.8

Value of ξ	Value of w	Resulting slope, $\frac{m_{\text{next}}}{m}$
0 (no error)	0.25–0.75	1
1 (error)	0.25	0.87
	0.5	0.71
	0.75	0.5

since it is the minimum value taken by X. The conjugate prior for the Pareto distribution is again the gamma distribution.

The procedure for the Pareto distribution is as follows. Initially, we only know the initial block size m_{init}.

1. Setup: Construct the prior distribution.
   (a) Find γ_{init} from m_{init}: Since $m_{\text{opt}} = \frac{2(1-\gamma)}{\gamma}$, we can find γ in terms of m using $\gamma_{\text{init}} = \frac{2}{m_{\text{init}}+2}$.
   (b) Let $\alpha_{\text{init}} = 2$.
   (c) Since the mode of Gamma(α, β) is $\frac{\alpha-1}{\beta}$, let $\beta_{\text{init}} = \frac{\alpha_{\text{init}}-1}{\gamma_{\text{init}}}$.
   (d) The prior distribution is then Gamma$(\alpha_{\text{init}}, \beta_{\text{init}})$.

2. At the end of each block: Update the hyperparameters using ξ to obtain the posterior.
   (a) $\beta \leftarrow w \ln(\mathrm{E}[X|\xi]) + (1-w)\beta$ where

$$\mathrm{E}[X|\xi] = \begin{cases} \mathrm{E}[X|X \geq m] = \frac{m\gamma}{\gamma-1} = m\mathrm{E}[X] & \text{for } \xi = 0 \\ \mathrm{E}[X|X \leq m] = \mathrm{E}[X]\left(1 - \frac{m-1}{m^{\gamma}-1}\right) & \text{for } \xi = 1 \end{cases} \tag{4.16}$$

   Note that $\mathrm{E}[X|X \geq m]$ is a scaled version of $\mathrm{E}[X]$, which is a consequence of self-similarity.

Important observation. Recall that we assume $\gamma \leq 1$ since a higher γ implies a very low reliability. Since $\mathrm{E}[X|X \leq m]$ is an expectation over a bounded interval $[1, m]$, it exists for $\gamma < 1$ unlike $\mathrm{E}[X]$. When $\gamma = 1$, we can take its limit instead. $\lim_{\gamma \to 1} \mathrm{E}[X|X \leq m] = \frac{m \ln m}{m-1}$.

However, this will not work for $\mathrm{E}[X|X \geq m]$. Since $\mathrm{E}[X]$ does not exist for $\gamma \leq 1$, $\mathrm{E}[X|X \geq m] = m\mathrm{E}[X]$ will not exist as well. We are not aware of any consistent solution for this problem, and hence, we replace $\mathrm{E}[X|X \geq m]$ with e^{β} which implies no change to β when no error is detected. It follows that the block size will not increase when no error is detected. This has no effect, however, on the other part of the adaptive policy, i.e., when an error occurs.

(b) Let γ be the mode of the posterior. $\gamma = \frac{\alpha-1}{\beta}$.

(c) Let m be the optimal block size for γ. $m = \max\left(1, \left\lfloor \frac{2(1-\gamma)}{\gamma} \right\rfloor\right)$.

Resulting block sizes. When this procedure is applied to the current m, we get the following expressions for the next block size m_{next} as a function of current block size m and the smoothing coefficient w.

$$m_{\text{next}} = \begin{cases} m & \text{for } \xi = 0 \\ \left\lfloor m - (2+m)w + 2w \ln\left(\frac{2\left(m - m^{\frac{2}{2+m}}\right)}{m\left(m^{\frac{2}{2+m}} - 1\right)}\right) \right\rfloor & \text{for } \xi = 1 \end{cases} \tag{4.17}$$

Note that due to the nonexistence of $\mathrm{E}[X|\xi = 0]$ when $\gamma \leq 1$, the block size is not modified when $\xi = 0$. Figure 4.9 illustrates these expressions by plotting the new m against the current m for three values of w, namely, 0.25, 0.5 and 0.75. Moreover, Table 4.7 shows the slopes of the lines that fit the curves in Figure 4.9.

4.4.2.4 Evaluation

We have used Bayesian inference under two different statistical error models to determine the next block size as a function of the current block size. Whether an error was detected

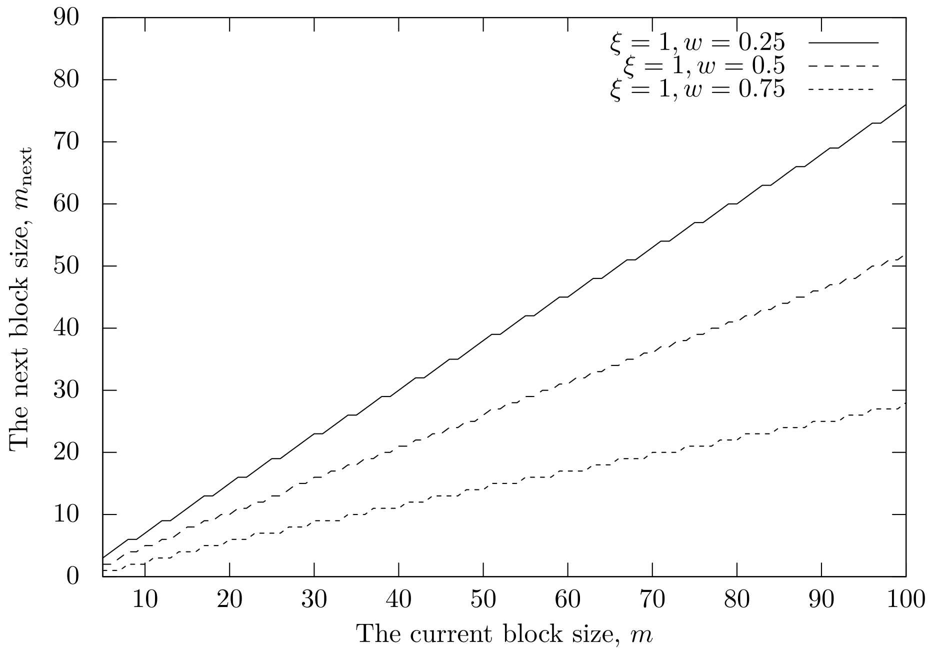

Figure 4.9: The next block size as a function of the current block size for different values of ξ and w assuming Pareto-distributed errors.

Table 4.7: Slopes of the lines that fit the curves in Figure 4.9

Value of ξ	Value of w	Resulting slope, $\frac{m_{\text{next}}}{m}$
1 (error)	0.25	0.76
	0.5	0.52
	0.75	0.28

or not, the behavior of the inference procedure matches that of the general adaptive policy discussed earlier, i.e., the grow-slow/shrink-fast policy. This indicates that the general adaptive policy can track the estimated value of the unknown parameter given only the observed occurance or absence of errors. Moreover, for some value of w under both models, the behavior matches exactly the $mDown$ part of the specific adaptive policy we used earlier, i.e., $mDown(m) = \lfloor \frac{m}{2} \rfloor$. As for the mUp part, it is clear that the slower the block size is increased the better.

Furthermore, these results provide a way to fine-tune the adaptive policy according to the desired sensitivity to the most recent observed data. For example, assuming Pareto-distributed errors, if it is desired to give a high weight to the most recent observation, e.g., $w = 0.75$, then according to Table 4.7, the adaptive policy should divide the block size by 4 rather than by 2 when an error is detected.

4.5 Effects of Adaptive Error Recovery on Security

Given the existence and demonstrated practicality of a variety of side-channel attacks on the ECSM operation, it is essential to study the effect of frequent validation in general, and adaptive error recovery in particular, on the security of an ECSM implementation. The effects of frequent validation and partial recomputation on security have been addressed in Chapter 3. In summary, the following issues were discussed:

1. Fault Analysis Attacks: The validation tests used in the preceding examples are known to be immune to a wide variety of fault attacks mounted by less-sophisticated attackers, i.e., attackers who do not have full control over the location and timing of injected faults. On the other hand, it is essential to avoid the single point of failure created by the validation tests being logical tests. This can be achieved using infective computation at the end of the computation to mask any faulty results.
2. Timing and Simple Power Analysis Attacks: Both fixed-block and adaptive error recovery introduce variability in the time required to complete the computation.

However, this variability is dependent only on the location of errors and is independent of the secret information. Hence, if the underlying algorithm is immune to timing analysis, e.g., a double-and-add-always algorithm, then the use of frequent validation will not enable timing attacks.

3. Differential Power or Timing Analysis Attacks: Randomization can be used to counter differential attacks. Both fixed-block and adaptive error recovery schemes allow for the randomization of the whole computation, as well as the separate randomization of blocks, to avoid various types of differential attacks.

4.6 Conclusion

The advantage of using frequent validation with partial recomputation for error recovery in ECC implementations has been established in Chapter 3. However, this advantage depends mainly on identifying a good trade-off between reliability and overhead, which requires knowledge of the parameters of the error statistical model. This can be overcome partially by selecting smaller block sizes that generally provide higher reliability while not increasing the overhead significantly.

In this chapter, we have introduced another approach to address this issue. Instead of fixing the block size to an optimal value, the block size is allowed to vary adaptively as a response to the occurrence of errors. We have shown, using an analytical model and using simulation, that this can give near-optimal reliability and efficiency while not requiring prior knowledge of the statistical parameters of errors. We have also discussed the selection of the policy and used statistical inference to justify the general adaptive policy proposed earlier. These results should motivate designers to incorporate adaptive error recovery in their designs.

Chapter 5

Cost-Effective Validation Tests for Elliptic Curve Scalar Multiplication

5.1 Introduction

Earlier in this document, we have discussed ways to enhance the reliability of cryptographic implementations using repeated validation testing. We have addressed the issue of the selection of the validation frequency, when fixed blocks are used, and its sensitivity to the parameters of the error statistical model. We have also proposed adaptive testing as an alternative that relaxes the need for a precise error model. Although our proposals require less overhead than comparable approaches, validation testing still constitutes a significant part of the overhead. It is always desirable to reduce the testing overhead as much as possible while maintaining the reliability gain.

A key observation is that validation tests discussed earlier aim mainly to have the maximum coverage possible, and do not give the same weight to computational cost. This is justified by the fact that these tests are constructed to be used once right before the final result is provided, and hence, their cost becomes insignificant relative to that of a scalar multiplication.

However, when a validation test is applied frequently, the overhead of testing can become a significant part of the total overhead. We show in this chapter that some validation tests, and particularly those based on coherency checking, can offer a trade-off between the cost of the test and its error detection effectiveness. As a result, the effectiveness of error detection can be decreased in order to reduce the resulting overhead without threatening the security of the system.

In this chapter, we propose some cost-effective validation tests and evaluate their error detection effectiveness and the associated cost. We take two approaches to achieve this. The first comprises the modification of the coherency-based validation test presented in [23] in order to reduce its cost. Then, we propose two related coherency-based validation tests that offer different cost and effectiveness trade-offs. For all of these tests, we investigate their cost and error detection effectiveness and the impact of using a varying number of point validations in the test. We also show how the cost of these tests can be reduced further by using them in a repeated validation setting.

The second approach is to apply coherency-based validation tests to elliptic curves defined over rings in order to both reduce the cost of validation tests and present a flexible trade-off between the test cost and its effectiveness. Essentially, validation tests are performed on the smaller curve rather than the original or the combined curve, thus saving significantly on the testing cost while still being able to detect a high proportion of errors. The size of the smaller curve, however, has an effect on both the computation cost and the testing cost, so the selection of the smaller curve size has to be made with these issues in mind.

The goal here is to make validation tests as inexpensive as possible. The corresponding increase in the proportion of undetected errors is mitigated by both repeated testing and a comprehensive error detection test at the end of the computation. As shown in Chapter 3, the reduction of the cost of the validation test results in smaller block sizes, which in turn lead to a higher reliability and reduced recomputations. While our discussion will focus on general elliptic curves defined over prime fields and on homogeneous projective coordinates, the ideas presented here are applicable to other elliptic curves, finite fields and projective coordinates.

5.2 Coherency-based Validation Testing

Some scalar multiplication algorithms, e.g., double-and-add-always [18] and Montgomery's ladder [52], have an inherently redundant internal state that satisfies some invariants. The occurrence of errors will generally disturb the algorithm's internal state. As such, it is possible to detect those errors by checking the coherency of the algorithm's state as illustrated in [23] and [25]. There are error detection solutions for RSA that use the same idea like those proposed in [28] and [29]. In this section, we focus on the coherency-based validation testing proposed by Domínguez-Oviedo and Hasan in [23], which will serve as the foundation for countermeasures introduced in this chapter.

5.2.1 Threat Model and Assumptions

We start by describing the threat model that we consider when evaluating the countermeasure discussed in this chapter.

A fault can result in an error either by flipping a bit or more in a register, or by corrupting a computation resulting in a faulty output. While any variable used in the algorithm can be corrupted by a natural or deliberate fault, we focus our discussion on variables that result from finite field computations. Other variables, i.e., the scalar, base point, loop counter, and field and curve parameters, can be protected by integrity checking mechanisms like parity bits or cyclic redundancy checks. Also, it is assumed that the attacker is not able to inject faults in a way that sets a specific bit to a specific value. While the countermeasures we discuss can in some cases detect permanent faults, we generally limit our discussion to transient errors. The countermeasures discussed here cover natural and deliberate invalid-curve faults as well as sign change and safe error attacks.

When *decisional* validation tests are considered, it is assumed that the test is performed in a way that is protected from fault injection. This assumption will be relaxed when infective variants of decisional validation tests are discussed as countermeasures to double-fault attacks.

In terms of notation, when there is a potential for confusing a validation test with a statement, a question mark will be used to differentiate the two as follows. The form $a = b$ states that a is equal to b, while the form $a \overset{?}{=} b$ indicates a decisional test that returns true when a and b are equal and false otherwise.

5.2.2 Domínguez-Oviedo–Hasan Coherency-based Validation Test

Coherency checking can be used to validate the state of the algorithm is a variety of ways depending on the specific algorithm and the threat model.

In [23], a collection of validation tests based on this approach have been presented. Among these validation tests, Algorithm 4, which is discussed in Chapter 2 and is based on an analogous test for RSA presented in [29], has some interesting properties. First, it is based on a double-and-add-always scalar multiplication algorithm that is inherently resistant to simple side-channel attacks. Second, it can detect a wide range of errors that have been exploited in fault analysis attacks. Third, it can be easily modified to counter double-fault attacks.

In Algorithm 4, the validation test has two parts, namely, point validations and a coherency test that is based on point additions. It was shown in [23] that the combination of these two parts is resistant to invalid-curve, sign change and safe-error attacks under their respective models, and under the assumption that all variables other than the coordinates of Q_0, Q_1 and Q_2 are protected by an integrity checking mechanism.

5.2.2.1 A Simple Modification to Domínguez-Oviedo–Hasan Validation Test

The validation test discussed above can be modified to reduce its cost. Note that Q_0 is a dummy variable and that its value is not part of the final result of the algorithm. Since the coherency test has the form $Q_2 \overset{?}{=} Q_0 + Q_1 + P$, we can reduce its cost by initializing Q_0 to P instead of $\mathcal{O}$, resulting in Algorithm 8 in which the coherency test becomes $Q_2 \overset{?}{=} Q_0 + Q_1$. This reduces the cost of the validation test by one point addition, which amounts to about 25% of the cost of the original test. The significance of this simple modification is two-fold. First, when this validation test is used repeatedly, a saving of one point addition will accumulate to result in a significantly large saving in the testing overhead. Second, this simplified coherency test, which involves a single point addition, enables other cost-effective validation tests that will be discussed later in this chapter.

We use Algorithm 8 as the reference point of the work presented in this chapter. In particular, we discuss some alternative coherency tests beside point addition, and we also discuss the use of point validations alongside the coherency test in terms of cost and error detection. In our discussion, we will use a notation in which this test is denoted by PA+2PV, i.e., point addition with two point validations. We will also discuss its cost and error detection effectiveness when we discuss point-addition–based coherency tests.

5.2.3 Evaluation of Error Detection Effectiveness

Later in this chapter, we will compare validation tests according to their cost and error detection effectiveness. In this section, we discuss how the error detection effectiveness of a validation test can be evaluated.

When evaluating the error detection effectiveness of a validation test, two issues need to be considered. The first issue is the ability of the validation test to detect errors used in the sign-change and safe-error attacks. The second issue is the proportion of faulty state vectors that can pass the test. In this context, the state vector of the algorithm includes the coordinates of the points Q_0, Q_1 and Q_2. The Proportion of Undetected

Algorithm 8 Double-and-add-always ECSM algorithm with point validation and modified coherency checking.

```
Input:  $P \in E$, $k = \sum_{i=0}^{n-1} k_i 2^i$
Output:  $kP$
1: $Q_0 \leftarrow P$, $Q_1 \leftarrow \mathcal{O}$, $Q_2 \leftarrow P$
2: for $i \leftarrow 0$ to $n-1$ do
3:     $Q_{k_i} \leftarrow Q_{k_i} + Q_2$
4:     $Q_2 \leftarrow 2Q_2$
5: end for
6: if $Q_0 \in E$ and $Q_1 \in E$ and $Q_2 = Q_0 + Q_1$ then
7:     return $Q_1$
8: else
9:     return $\mathcal{O}$
10: end if
```

Errors (PUE) is estimated relative to the total number of state vectors. For the scalar multiplication algorithm used here, the total number of state vectors is p^6, where p is the size of the underlying field, since each of three points has two coordinates. Estimating the PUE helps in evaluating the validation test ability to detect invalid-curve attacks that target the intermediate points or the field operations. If the state vectors are assumed to be uniformly distributed, the PUE can be interpreted as the probability of an undetected error.

The PUE is a similar concept to the code rate, which is defined as the ratio of the effective bit length of a code to its total bit length, and is used to evaluate error control codes [47]. The larger the code rate, the less the redundancy available on the code, and hence, the less its error detection capability [70]. Similarly for the PUE, the larger the proportion of undetected errors, the less the error detection capability of a validation test.

To evaluate the cost-effectiveness of a test, both its computational cost and its PUE are taken into account. However, while the cost of a combination of tests is additive, their

PUEs are generally multiplicative. It follows that cost-effectiveness of a validation test can be measured by its normalized cost which can be defined as follows.

$$\text{Normalized cost} = \frac{\text{Absolute cost}}{-\log_p \text{PUE}} \tag{5.1}$$

Another key fact that is useful when evaluating the effect of point validations on a validation test is the number of points on an elliptic curve relative to the size of the underlying field. As stated by Hasse's theorem [66], the number of points on an elliptic curve, denoted by $\#E$, defined over a field of size p satisfies the relationship

$$|\#E - (p+1)| \leq 2\sqrt{p} \tag{5.2}$$

which can be rewritten as

$$\#E \in [p - 2\sqrt{p} + 1, p + 2\sqrt{p} + 1] \tag{5.3}$$

Taking the logarithm of the interval limits, we get

$$\begin{aligned} \log(p + 1 \pm 2\sqrt{p}) &= \log\left(p\left(1 + \frac{1 \pm 2\sqrt{p}}{p}\right)\right) \\ &= \log(p) + \log\left(1 + \frac{1}{p} \pm \frac{2}{\sqrt{p}}\right) \\ &\approx \log(p) \end{aligned} \tag{5.4}$$

since p is a large prime and hence $\frac{1}{p}$ and $\frac{2}{\sqrt{p}}$ are very close to zero. This indicates that $\#E$ has the same bit size as p.

5.3 Approaches to Coherency Checking

In this section, we discuss three approaches to check the internal state of Algorithm 8. The approaches covered here are:

1. Point Addition: This part expands on the coherency check presented earlier, which in turn is a modification of the Domínguez-Oviedo–Hasan coherency-based validation test [23].

2. Collinearity: Here, a collinearity-based test is used in place of point addition. This reduces both the cost of testing and its coverage as we will show later.

3. Function Evaluation: In this approach, a function is constructed such that it evaluates to a known constant when the algorithm's state is valid. This can be seen as a generalization of the approaches discussed earlier, but it has a significantly higher computational cost. However, we present this approach for the sake of completeness.

Each of these approaches is evaluated with a varying number of point validations. We discuss the error detection effectiveness as well as the absolute and normalized cost of each approach.

5.3.1 Coherency Checking using Point Addition

Assuming error-free operation, after the *i*th iteration of the right-to-left double-and-add-always scalar multiplication illustrated in Algorithm 8, the three intermediate points Q_0, Q_1 and Q_2 will have the following values.

$$Q_0 = \left(\sum_{j=0}^{i} \overline{k}_j 2^j + 1 \right) P \tag{5.5}$$

$$Q_1 = \sum_{j=0}^{i} k_j 2^j P \tag{5.6}$$

$$Q_2 = 2^{j+1} P \tag{5.7}$$

where $\overline{k}_j$ is the binary complement of k_j. It can be readily seen that

$$Q_0 + Q_1 = \left(\sum_{j=0}^{i} \overline{k}_j 2^j + \sum_{j=0}^{i} k_j 2^j + 1 \right) P = \left(\sum_{j=0}^{i} 2^j + 1 \right) P = Q_2 \tag{5.8}$$

This is similar to the test used in [23] to check the coherency of the algorithm's internal state. In the modified validation test presented earlier, the cost of this test is reduced by one point addition. This results in a test of the form "Accept Q_1 if $Q_2 = Q_0 + Q_1$." We denote this test henceforth by the PA test, and discuss its error detection effectiveness and cost using a varying number of point validations. Since three points are involved in this test, there can be at most three point validations. A point validation is denoted by PV, e.g., PA+2PV denotes a validation test that uses a point addition and two point validations.

With respect to the sign change attack, it was shown in [23] that a sign change fault injected in either Q_1 or Q_2 will break the state invariant, and hence can be detected by coherency checking. This applies to the PA test as well since point validations are not effective in detecting sign-change errors.

With respect to the safe-error attack, which generally targets dummy operations and/or variables, we distinguish between safe errors that move Q_0 off the curve and those that move it within the curve. It was shown in [23] that coherency checking can detect a safe error that moves Q_0 within the curve. It was assumed, however, that a safe-error that moves Q_0 from the curve will be detected by point validation. Here, we address the case where point validation is not used, and discuss the effect on safe-error attacks as well as general invalid-curve attacks.

5.3.1.1 Point Addition with no Point Validations (PA+0PV)

We find the PUE for PA+0PV test, which covers both invalid-curve errors and safe errors that move the point Q_0 from the curve. As mentioned earlier, there are p^6 possible state vectors. If two of the three points are selected arbitrarily, only a single value for the third point can pass the test. In other words, only p^4 state vectors can pass the test. This will give a proportion of undetected errors of $\frac{1}{p^2}$.

5.3.1.2 Point Addition with One Point Validation (PA+1PV)

The use of one point validation, which can be applied to any of the 3 points, increases both the error coverage and cost of the test. Without loss of generality, let us assume that Q_1 is the point to be validated, which is the natural choice since it is the point to be returned as the final result. We can see that the among the p^4 state vectors that will pass the PA test, only the ones where Q_1 is on E will pass the point validation. Hence, the proportion of undetected errors becomes

$$\left(\frac{p^4}{p^6}\right)\left(\frac{\#E}{p^2}\right) = \frac{\#E}{p^4} \approx \frac{1}{p^3} \tag{5.9}$$

5.3.1.3 Point Addition with Two Point Validations (PA+2PV)

This case is very similar to the original test proposed in [23]. Using the same arguments above, we can see that the proportion of undetected errors will be

$$\left(\frac{p^4}{p^6}\right)\left(\frac{\#E}{p^2}\right)^2 = \frac{(\#E)^2}{p^6} \approx \frac{1}{p^4} \tag{5.10}$$

which matches the error detection capability given in [23].

5.3.1.4 Point Addition with Three Point Validations (PA+3PV)

Recall that the sum of two points on the curve is always a point that lies on the curve. Since the coherency checking part is performed using a point addition, the third point validation is redundant as it will only detect errors that can be detected by the other parts of the validation test. In other words, the proportion of undetected errors is the same as that in PA+2PV, namely, $\frac{1}{p^4}$.

5.3.1.5 Computational Cost

Cost of Point Addition Test using the Affine Representation. While a point addition in affine coordinates requires an inversion, this test can be performed without the need of an inversion. That is because the coordinates of the potential resulting point are known, and what is required is comparing the addition result with the known point.

To find out the cost of the test $Q_0 + Q_1 \overset{?}{=} Q_2$, we write it as follows.

$$\begin{aligned} x_{Q_2} &\overset{?}{=} \frac{(y_{Q_1} - y_{Q_0})^2}{(x_{Q_1} - x_{Q_0})^2} - x_{Q_0} - x_{Q_1} \\ y_{Q_2} &\overset{?}{=} (2x_{Q_0} + x_{Q_1})\frac{(y_{Q_1} - y_{Q_0})}{(x_{Q_1} - x_{Q_0})} - \frac{(y_{Q_1} - y_{Q_0})^3}{(x_{Q_1} - x_{Q_0})^3} - y_{Q_0} \end{aligned} \tag{5.11}$$

We multiply both equations by their respective denominators to get the following.

$$\begin{aligned} &x_{Q_2}(x_{Q_1} - x_{Q_0})^2 \overset{?}{=} (y_{Q_1} - y_{Q_0})^2 - (x_{Q_0} + x_{Q_1})(x_{Q_1} - x_{Q_0})^2 \\ &y_{Q_2}(x_{Q_1} - x_{Q_0})^3 \overset{?}{=} (2x_{Q_0} + x_{Q_1})(y_{Q_1} - y_{Q_0})(x_{Q_1} - x_{Q_0})^2 - (y_{Q_1} - y_{Q_0})^3 - y_{Q_0}(x_{Q_1} - x_{Q_0})^3 \end{aligned} \tag{5.12}$$

After some simplifications and rearrangement.

$$\begin{aligned} &(x_{Q_2} + x_{Q_0} + x_{Q_1})(x_{Q_1} - x_{Q_0})^2 \overset{?}{=} (y_{Q_1} - y_{Q_0})^2 \\ &(x_{Q_1} - x_{Q_0})^2((y_{Q_2} + y_{Q_0})(x_{Q_1} - x_{Q_0}) - (2x_{Q_0} + x_{Q_1})(y_{Q_1} - y_{Q_0})) \overset{?}{=} -(y_{Q_1} - y_{Q_0})^3 \end{aligned} \tag{5.13}$$

which is an equivalent test that requires no inversion. Using these transformations, a point addition that costs $\mathcal{I} + 3\mathcal{M} + \mathcal{S}$ is replaced by a PA test that costs $5\mathcal{M} + 2\mathcal{S}$, where $\mathcal{I}$, $\mathcal{M}$ and $\mathcal{S}$ denote a field inversion, multiplication and squaring, respectively.

Cost of Point Addition Test using the Homogeneous Projective Representation. Since by definition, point addition in projective coordinates does not require a field inversion, it is not necessary to manipulate the conventional point addition formulas to avoid the need for an inversion. In projective coordinates, point addition requires $12\mathcal{M} + 2\mathcal{S}$ [9]. Let $\tilde{Q}_2$ be the sum of Q_0 and Q_1. To test the equality of Q_2 and $\tilde{Q}_2$ it is required to test both coordinates as follows.

$$\begin{aligned} \frac{X_{Q_2}}{Z_{Q_2}} &\overset{?}{=} \frac{X_{\tilde{Q}_2}}{Z_{\tilde{Q}_2}} \\ \frac{X_{Q_2}}{Z_{Q_2}} &\overset{?}{=} \frac{X_{\tilde{Q}_2}}{Z_{\tilde{Q}_2}} \end{aligned} \tag{5.14}$$

which can be rewritten as

$$\begin{aligned} X_{Q_2} Z_{\tilde{Q}_2} &\overset{?}{=} X_{\tilde{Q}_2} Z_{Q_2} \\ Y_{Q_2} Z_{\tilde{Q}_2} &\overset{?}{=} Y_{\tilde{Q}_2} Z_{Q_2} \end{aligned} \tag{5.15}$$

So, it requires $4\mathcal{M}$ to compare the two points in projective coordinates. The total cost for the PA test using the homogeneous projective representation becomes $16\mathcal{M} + 2\mathcal{S}$.

Cost of Point Validation. Point validation can be done by applying the curve equation to the point coordinates. In affine representation, this requires $2\mathcal{M} + 2\mathcal{S}$.

In projective representation, however, the use of the Z coordinate increases the cost of point validation.

$$\frac{Y^2}{Z^2} \overset{?}{=} \frac{X^3}{Z^3} + a\frac{X}{Z} + b \tag{5.16}$$

which can be rewritten as

$$Y^2 Z \overset{?}{=} X^3 + aXZ^2 + bZ^3 \tag{5.17}$$

This requires $6\mathcal{M} + 3\mathcal{S}$.

5.3.2 Coherency Checking using Collinearity Testing

Point addition is not the only way to check the coherency of the three points. We discuss here how the collinearity of the points can be used in coherency checking. This test is based on the fact that if the relationship $Q_0 + Q_1 = Q_2$ holds, then the points Q_0, Q_1 and $-Q_2$ lie on the same line, as illustrated in Figure 5.1. The converse is not true, however, so this test will not have the same error detection ability as the point addition test.

Let $Q_0 = (x_{Q_0}, y_{Q_0})$, $Q_1 = (x_{Q_1}, y_{Q_1})$, $Q_2 = (x_{Q_2}, y_{Q_2})$ and $-Q_2 = (x_{Q_2}, -y_{Q_2})$. Also, assume without the loss of generality that Q_0, Q_1 and $-Q_2$ are distinct. We want to test whether these points lie on the same line. This can be done by finding the function of the line that passes through two of the points and testing whether it passes through the third. Let $l_{Q_0,-Q_2}$ be the line that connects the points Q_0 and $-Q_2$. The function defined by this

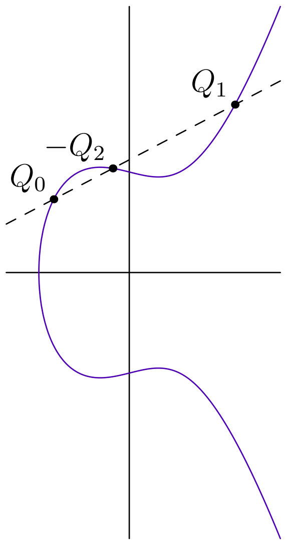

Figure 5.1: Collinearity testing

line evaluates to 0 for a given point if and only if that point lies on the line. So, we apply this function to the point Q_1, i.e., we find $l_{Q_0,-Q_2}(Q_1)$. The three points are collinear if and only if $l_{Q_0,-Q_2}(Q_1) = 0$. This test, denoted by the CL test, can be expressed as follows.

$$l_{Q_0,-Q_2}(Q_1) = y_{Q_1} - \mathfrak{m}_{Q_0,-Q_2} x_{Q_1} - \mathfrak{b}_{Q_0,-Q_2} \overset{?}{=} 0 \tag{5.18}$$

where $\mathfrak{m}_{Q_0,-Q_2} = \frac{-y_{Q_2} - y_{Q_0}}{x_{Q_2} - x_{Q_0}}$ is the slope of the line and $\mathfrak{b}_{Q_0,-Q_2} = -y_{Q_2} - \mathfrak{m}_{Q_0,-Q_2} x_{Q_2}$ is its y-intercept. By substitution for $\mathfrak{b}_{Q_0,-Q_2}$, it follows that

$$\begin{aligned} l_{Q_0,-Q_2}(Q_1) &= y_{Q_1} - \mathfrak{m}_{Q_0,-Q_2} x_{Q_1} + y_{Q_2} + \mathfrak{m}_{Q_0,-Q_2} x_{Q_2} \\ &= y_{Q_1} + y_{Q_2} - \mathfrak{m}_{Q_0,-Q_2}(x_{Q_1} - x_{Q_2}) \overset{?}{=} 0 \end{aligned} \tag{5.19}$$

This can be rearranged as follows.

$$y_{Q_1} + y_{Q_2} \overset{?}{=} \mathfrak{m}_{Q_0,-Q_2}(x_{Q_1} - x_{Q_2}) = \frac{y_{Q_0} + y_{Q_2}}{x_{Q_0} - x_{Q_2}}(x_{Q_1} - x_{Q_2}) \tag{5.20}$$

Dividing both sides by $(x_{Q_1} - x_{Q_2})$, we get the following.

$$\frac{y_{Q_1} + y_{Q_2}}{x_{Q_1} - x_{Q_2}} \overset{?}{=} \frac{y_{Q_0} + y_{Q_2}}{x_{Q_0} - x_{Q_2}} \tag{5.21}$$

In other words, if the lines $l_{Q_1,-Q_2}$ and $l_{Q_0,-Q_2}$ have the same slope then they are the same line.

5.3.2.1 Collinearity Test with no Point Validations (CL+0PV)

We evaluate the error coverage of the collinearity test when no point validations are used. With respect to sign change errors, we already know that a sign change error in any of the intermediate points will break the state's invariant by ensuring that $Q_2 \neq Q_0 + Q_1$ [23]. We also know that such an error will not move any of the points off the curve. Since the three points lie on the curve and $Q_2 \neq Q_0 + Q_1$, it follows that Q_0, Q_1 and $-Q_2$ can not lie on the same line. Hence, CL test will detect sign change errors. A similar argument indicates that safe errors that do not move point off the curve will be detected by the CL test.

Now we find the CL test's PUE. Assume that the points Q_0 and Q_1 are selected arbitrarily, which implies p^4 possibilities. Then, any point $-Q_2$ that lies on the line defined by Q_0 and Q_1 will pass the test. Since each line has p points, the proportion of undetected errors for the CL+0PV test becomes

$$\frac{p^5}{p^6} = \frac{1}{p} \tag{5.22}$$

5.3.2.2 Collinearity Test with One Point Validation (CL+1PV)

The use of one point validation will guarantee that one of the three points lies on the curve. As before, let Q_1 be the validated point tested. Then, the number of combinations of Q_0 and Q_1 that would pass the test is reduced to $\#Ep^2$. Since any point on the line defined by Q_0 and Q_1 can pass the test as $-Q_2$, the proportion of undetected errors becomes

$$\frac{\#Ep^3}{p^6} \approx \frac{1}{p^2} \tag{5.23}$$

The same argument applies when another point is tested, but Q_1 is the natural candidate since it will be returned as the final result.

5.3.2.3 Collinearity Test with Two Point Validations (CL+2PV)

When two point validations are used, the proportion of undetected errors will decrease. Assuming that Q_0 and Q_1 are tested, the number of valid combinations is reduced to $(\#E)^2$. Then, any point on the line defined by Q_0 and Q_1 will pass the collinearity test, so the proportion of undetected errors becomes

$$\frac{(\#E)^2 p}{p^6} \approx \frac{1}{p^3} \tag{5.24}$$

5.3.2.4 Collinearity Test with Three Point Validations (CL+3PV)

Unlike the case of point addition, the third point validation is not redundant here and will further reduce the proportion of undetected errors. Similar to the argument above, the number of combinations of Q_0 and Q_1 that will pass the test is $\#E^2$. However, only a point $-Q_2$ that lies on both the curve and the line will pass the test. Since a line intersects an elliptic curve in no more than three points, and since we already know two of these points, there is only one point that can pass the test. It follows that the proportion of undetected errors becomes

$$\frac{(\#E)^2}{p^6} \approx \frac{1}{p^4} \tag{5.25}$$

Note that this depends on the fact that the three point are distinct, which is implied by the algorithm given error-free operation. As such, if two of the points are equal, this by itself indicates an error.

5.3.2.5 Computational Cost

Cost of Collinearity Test using the Affine Representation. When the CL test is applied in affine representation, (5.21) can be rewritten as

$$(y_{Q_1} + y_{Q_2})(x_{Q_0} - x_{Q_2}) \overset{?}{=} (y_{Q_0} + y_{Q_2})(x_{Q_1} - x_{Q_2}) \tag{5.26}$$

to avoid field inversions. This computation requires $2\mathcal{M}$.

Cost of Collinearity Test using the Homogeneous Projective Representation. In projective coordinates, (5.26) can be written as follows.

$$\left(\frac{Y_{Q_1}}{Z_{Q_1}} + \frac{Y_{Q_2}}{Z_{Q_2}}\right)\left(\frac{X_{Q_0}}{Z_{Q_0}} - \frac{X_{Q_2}}{Z_{Q_2}}\right) \stackrel{?}{=} \left(\frac{Y_{Q_0}}{Z_{Q_0}} + \frac{Y_{Q_2}}{Z_{Q_2}}\right)\left(\frac{X_{Q_1}}{Z_{Q_1}} - \frac{X_{Q_2}}{Z_{Q_2}}\right) \quad (5.27)$$

which can be expanded and simplified to

$$(Y_{Q_1}Z_{Q_2}+Y_{Q_2}Z_{Q_1})(X_{Q_0}Z_{Q_2}-X_{Q_2}Z_{Q_0}) \stackrel{?}{=} (Y_{Q_0}Z_{Q_2}+Y_{Q_2}Z_{Q_0})(X_{Q_1}Z_{Q_2}-X_{Q_2}Z_{Q_1}) \quad (5.28)$$

which requires $10\mathcal{M}$.

5.3.3 Coherency Checking using Function Evaluation

In this section, we present a way to construct a function that evaluates to a known constant value for all valid state vectors. In order to do this, we need a working knowledge of divisors, so we provide a cursory introduction to the subject. A reader interested in the details can refer to a textbook like [66].

A *divisor* of a function f in a function field defined over an algebraic curve C is a way to encode its zeros and poles. A function f is constant if and only if it has no zeros or poles, which can be written as $\mathrm{div}(f) = \mathbf{0}$, where $\mathbf{0} = \sum_P 0(P)$ for all points $P \in C$ is the empty divisor. The coefficients in the divisor of a function indicate the order of the function at a specific point. Positive coefficients indicate zeros while negative coefficients indicate poles. In both cases, the absolute value of a coefficients indicates the multiplicity of a zero or a pole. The sum of coefficients in a divisor of a function is always zero, i.e., it has the same number of zeros and poles. For example, let $P_1, P_2, P_3 \in E$ be such that $P_1 + P_2 = P_3$. Then, the function of the line that connects P_1 and P_2 has the divisor $(P_1) + (P_2) + (-P_3) - 3(\mathcal{O})$, i.e., it has a zero of order 1 at each of P_1, P_2 and $-P_3$ and a pole of order 3 at $\mathcal{O}$.

We can use this to construct a function that evaluates to a constant for all error-free state vectors. Let E be an elliptic curve and Q_0, Q_1 and Q_2 be points on E such that

$Q_0 + Q_1 = Q_2$. As before, let $l_{Q_0,-Q_2}$ be the function of the line that connects Q_0 and $-Q_2$, i.e., for a point $M = (x_M, y_M)$,

$$l_{Q_0,-Q_2}(M) = y_M - \mathfrak{m}_{Q_0,-Q_2} x_M - \mathfrak{b}_{Q_0,-Q_2} \tag{5.29}$$

where $\mathfrak{m}_{Q_0,-Q_2}$ and $\mathfrak{b}_{Q_0,-Q_2}$ are the slope and y-intercept of the line $l_{Q_0,-Q_2}$, respectively. Define a function $F_{Q_0,Q_1,Q_2}(M)$ as follows.

$$F_{Q_0,Q_1,Q_2}(M) = \frac{l_{Q_0,-Q_2}(M) l_{-Q_1,Q_2}(M)}{l_{Q_0,\mathcal{O}}(M) l_{Q_1,\mathcal{O}}(M) l_{Q_2,\mathcal{O}}(M)} \tag{5.30}$$

where $M \notin \{\pm Q_0, \pm Q_1, \pm Q_2\}$. The line $l_{Q_0,\mathcal{O}}(M)$ is the line connecting Q_0 and $-Q_0$ and its function is $l_{Q_0,\mathcal{O}}(M) = x_{Q_0} - x_M$. The same applies to $l_{Q_1,\mathcal{O}}(M)$ and $l_{Q_2,\mathcal{O}}(M)$. Figure 5.2 illustrates the points and lines involved in this function.

The divisor of F_{Q_0,Q_1,Q_2} can be written as follows.

$$\begin{aligned} \text{div}(F_{Q_0,Q_1,Q_2}) =& (Q_0) + (Q_1) + (-Q_2) - 3(\mathcal{O}) + (-Q_0) + (-Q_1) + (Q_2) - 3(\mathcal{O}) \\ & - ((Q_0) + (-Q_0) - 2(\mathcal{O})) - ((Q_1) + (-Q_1) - 2(\mathcal{O})) \\ & - ((Q_2) + (-Q_2) - 2(\mathcal{O})) \\ =& \mathbf{0} \end{aligned} \tag{5.31}$$

Hence, if Q_0, Q_1 and Q_2 lie on the curve and satisfy the addition condition, $F_{Q_0,Q_1,Q_2}(M)$ will evaluate to a constant $c \in \mathbb{F}_p^*$ for any $M \in E$ that is not in $\{\pm Q_0, \pm Q_1, \pm Q_2\}$. However, the converse is not true, i.e., F might evaluate to the same constant for other combinations of these points.

5.3.3.1 Function Evaluation with no Point Validations (FE+0PV)

We now evaluate the error coverage of this test. Let Q_0 and Q_1 be selected arbitrarily, and let $M \in E$ be known and fixed. For the points Q_0, Q_1 and Q_2 to pass the test, the

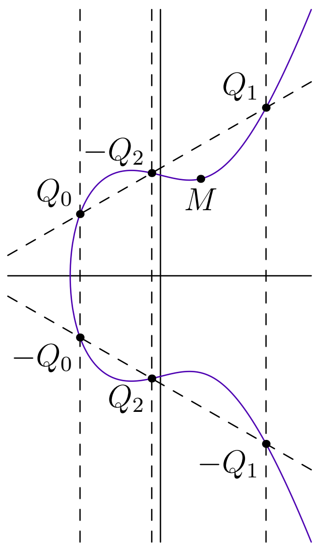

Figure 5.2: Function evaluation on an elliptic curve

function $F_{Q_0,Q_1,Q_2}(M)$ has to evaluate to the known constant c.

$$\begin{aligned} F_{Q_0,Q_1,Q_2}(M) &= \frac{l_{Q_0,-Q_2}(M)l_{-Q_1,Q_2}(M)}{l_{Q_0,\mathcal{O}}(M)l_{Q_1,\mathcal{O}}(M)l_{Q_2,\mathcal{O}}(M)} \\ &= \frac{(y_M - \mathfrak{m}_{Q_0,-Q_2}x_M - \mathfrak{b}_{Q_0,-Q_2})(y_M - \mathfrak{m}_{-Q_1,Q_2}x_M - \mathfrak{b}_{-Q_1,Q_2})}{(x_{Q_0} - x_M)(x_{Q_1} - x_M)(x_{Q_2} - x_M)} \\ &\overset{?}{=} c \end{aligned} \tag{5.32}$$

which can be reorganized as

$$(y_M - \mathfrak{m}_{Q_0,-Q_2}x_M - \mathfrak{b}_{Q_0,-Q_2})(y_M - \mathfrak{m}_{-Q_1,Q_2}x_M - \mathfrak{b}_{-Q_1,Q_2}) \overset{?}{=} c(x_{Q_0} - x_M)(x_{Q_1} - x_M)(x_{Q_2} - x_M) \tag{5.33}$$

After expansion and some manipulations, we get the following.

$$\begin{aligned} &((y_M + y_{Q_2})(x_{Q_0} - x_{Q_2}) - (y_{Q_0} + y_{Q_2})(x_M - x_{Q_2})) \\ &\times ((y_M - y_{Q_2})(x_{Q_2} - x_{Q_1}) - (y_{Q_2} + y_{Q_1})(x_M - x_{Q_2})) \\ &\overset{?}{=} c(x_{Q_0} - x_M)(x_{Q_1} - x_M)(x_{Q_2} - x_M)(x_{Q_0} - x_{Q_2})(x_{Q_2} - x_{Q_1}) \end{aligned} \tag{5.34}$$

This can be rewritten as

$$y_{Q_2}^2 + a_1 x_{Q_2} y_{Q_2} + a_3 y_{Q_2} \overset{?}{=} a_0 x_{Q_2}^3 + a_2 x_{Q_2}^2 + a_4 x_{Q_2} + a_6 \tag{5.35}$$

where a_0, a_1, a_2, a_3, a_4 and a_6 are defined in terms of x_{Q_0}, y_{Q_0}, x_{Q_1}, y_{Q_1}, x_M, y_M and c. This implies that of the possible p^2 values for Q_2, only points that lie on this elliptic curve curve can pass the test, i.e., approximately p points. Then, the proportion of undetected errors using this test is approximately $\frac{p^5}{p^6} = \frac{1}{p}$.

5.3.3.2 Function Evaluation with One Point Validation (FE+1PV)

The use of a single point validation will reduce the proportion of undetected faults. As before, let the tested point be Q_1. Then, the number of combinations of Q_0 and Q_1 becomes $\#Ep^2$. Following the above analysis for Q_2, the proportion of undetected errors will then become

$$\frac{\#Ep^3}{p^6} \approx \frac{1}{p^2} \tag{5.36}$$

5.3.3.3 Function Evaluation with Two Point Validations (FE+2PV)

When two points are validated, the proportion of undetected errors can be reduced further. Let the tested points be Q_0 and Q_1. Then, the number of accepted combinations of Q_0 and Q_1 will be $(\#E)^2$. By the same argument given above for Q_2, the proportion of undetected errors will then become

$$\frac{(\#E)^2 p}{p^6} \approx \frac{1}{p^3} \tag{5.37}$$

5.3.3.4 Function Evaluation with Three Point Validations (FE+3PV)

Since the function evaluation test can be passed by a point Q_2 that is not on the original curve even when Q_0 and Q_1 are on the curve, a third point validation will reduce the proportion of undetected errors further. Since the three points are validated, the number of combinations that will pass the point validation is $(\#E)^3$. Only $\frac{1}{p}$ of these points will pass the function evaluation test, so the proportion of undetected errors for the whole test becomes

$$\frac{\#E^3}{p^7} \approx \frac{1}{p^4} \tag{5.38}$$

5.3.3.5 Computational Cost

Cost of Function Evaluation Test using the Affine Representation. Equation (5.34) can be applied directly in affine representation. It requires $10\mathcal{M}$ to perform this test in affine representation.

Cost of Function Evaluation Test using the Homogeneous Projective Representation. Because we select M, we assume that we have its affine representation. Equation (5.34) can be written as follows in projective representation.

$$\begin{aligned}
&\frac{Y_M Z_{Q_2} + Y_{Q_2}}{Z_{Q_2}} \frac{X_{Q_0} Z_{Q_2} - X_{Q_2} Z_{Q_0}}{Z_{Q_0} Z_{Q_2}} - \frac{Y_{Q_0} Z_{Q_2} + Y_{Q_2} Z_{Q_0}}{Z_{Q_0} Z_{Q_2}} \frac{X_M Z_{Q_2} - X_{Q_2}}{Z_{Q_2}} \\
&\times \frac{Y_M Z_{Q_2} - Y_{Q_2}}{Z_{Q_2}} \frac{X_{Q_2} Z_{Q_1} - X_{Q_1} Z_{Q_2}}{Z_{Q_1} Z_{Q_2}} - \frac{Y_{Q_2} Z_{Q_1} + Y_{Q_1} Z_{Q_2}}{Z_{Q_1} Z_{Q_2}} \frac{X_M Z_{Q_2} - X_{Q_2}}{Z_{Q_2}} \\
&- c \frac{X_{Q_0} - X_M Z_{Q_0}}{Z_{Q_0}} \frac{X_{Q_1} - X_M Z_{Q_1}}{Z_{Q_1}} \frac{X_{Q_2} - X_M Z_{Q_2}}{Z_{Q_2}} \frac{X_{Q_0} Z_{Q_2} - X_{Q_2} Z_{Q_0}}{Z_{Q_0} Z_{Q_2}} \frac{X_{Q_2} Z_{Q_1} - X_{Q_1} Z_{Q_2}}{Z_{Q_1} Z_{Q_2}} \\
&\overset{?}{=} 0
\end{aligned} \tag{5.39}$$

By expanding and manipulating the expressions, we get the following.

$$\begin{aligned}
&Z_{Q_0} Z_{Q_1} ((Y_M Z_{Q_2} + Y_{Q_2})(X_{Q_0} Z_{Q_2} - X_{Q_2} Z_{Q_0}) - (Y_{Q_0} Z_{Q_2} + Y_{Q_2} Z_{Q_0})(X_M Z_{Q_2} - X_{Q_2})) \\
&\times ((Y_M Z_{Q_2} - Y_{Q_2})(X_{Q_2} Z_{Q_1} - X_{Q_1} Z_{Q_2}) - (Y_{Q_2} Z_{Q_1} + Y_{Q_1} Z_{Q_2})(X_M Z_{Q_2} - X_{Q_2})) \\
&- c Z_{Q_2} (X_{Q_0} - X_M Z_{Q_0})(X_{Q_1} - X_M Z_{Q_1})(X_{Q_2} - X_M Z_{Q_2}) \\
&\times (X_{Q_0} Z_{Q_2} - X_{Q_2} Z_{Q_0})(X_{Q_2} Z_{Q_1} - X_{Q_1} Z_{Q_2}) \overset{?}{=} 0
\end{aligned} \tag{5.40}$$

This requires $25\mathcal{M}$.

5.3.4 Summary

In this section, we have discussed three different approaches to validate the coherency of the state of Algorithm 8. Each of these approaches has been evaluated in terms of cost and error detection capability when used alongside a varying number of point validations.

Table 5.1: Summary of the PUE and cost for the decisional coherency-based validation tests.

Test	PUE	Absolute cost		Normalized cost	
		Affine	Proj.	Affine	Proj.
CC+2PV [23]	$\frac{1}{p^4}$	$14\mathcal{M}+8\mathcal{S}$	$40\mathcal{M}+10\mathcal{S}$	$3\frac{1}{2}\mathcal{M}+2\mathcal{S}$	$10\mathcal{M}+2\frac{1}{2}\mathcal{S}$
PA+0PV	$\frac{1}{p^2}$	$5\mathcal{M}+2\mathcal{S}$	$16\mathcal{M}+2\mathcal{S}$	$2\frac{1}{2}\mathcal{M}+1\mathcal{S}$	$8\mathcal{M}+\mathcal{S}$
PA+1PV	$\frac{1}{p^3}$	$7\mathcal{M}+4\mathcal{S}$	$22\mathcal{M}+5\mathcal{S}$	$2\frac{1}{3}\mathcal{M}+1\frac{1}{3}\mathcal{S}$	$7\frac{1}{3}\mathcal{M}+1\frac{2}{3}\mathcal{S}$
PA+2PV	$\frac{1}{p^4}$	$9\mathcal{M}+6\mathcal{S}$	$28\mathcal{M}+8\mathcal{S}$	$2\frac{1}{4}\mathcal{M}+1\frac{1}{2}\mathcal{S}$	$7\mathcal{M}+2\mathcal{S}$
PA+3PV	$\frac{1}{p^4}$	$11\mathcal{M}+8\mathcal{S}$	$34\mathcal{M}+11\mathcal{S}$	$2\frac{3}{4}\mathcal{M}+2\mathcal{S}$	$8\frac{1}{2}\mathcal{M}+2\frac{3}{4}\mathcal{S}$
CL+0PV	$\frac{1}{p}$	$2\mathcal{M}$	$10\mathcal{M}$	$2\mathcal{M}$	$10\mathcal{M}$
CL+1PV	$\frac{1}{p^2}$	$4\mathcal{M}+2\mathcal{S}$	$16\mathcal{M}+3\mathcal{S}$	$2\mathcal{M}+1\mathcal{S}$	$8\mathcal{M}+1\frac{1}{2}\mathcal{S}$
CL+2PV	$\frac{1}{p^3}$	$6\mathcal{M}+4\mathcal{S}$	$22\mathcal{M}+6\mathcal{S}$	$2\mathcal{M}+1\frac{1}{3}\mathcal{S}$	$7\frac{1}{3}\mathcal{M}+2\mathcal{S}$
CL+3PV	$\frac{1}{p^4}$	$8\mathcal{M}+6\mathcal{S}$	$28\mathcal{M}+9\mathcal{S}$	$2\mathcal{M}+1\frac{1}{2}\mathcal{S}$	$7\mathcal{M}+2\frac{1}{4}\mathcal{S}$
FE+0PV	$\frac{1}{p}$	$10\mathcal{M}$	$25\mathcal{M}$	$10\mathcal{M}$	$25\mathcal{M}$
FE+1PV	$\frac{1}{p^2}$	$12\mathcal{M}+2\mathcal{S}$	$31\mathcal{M}+3\mathcal{S}$	$6\mathcal{M}+1\mathcal{S}$	$15\frac{1}{2}\mathcal{M}+1\frac{1}{2}\mathcal{S}$
FE+2PV	$\frac{1}{p^3}$	$14\mathcal{M}+4\mathcal{S}$	$37\mathcal{M}+6\mathcal{S}$	$4\frac{2}{3}\mathcal{M}+1\frac{1}{3}\mathcal{S}$	$12\frac{1}{3}\mathcal{M}+2\mathcal{S}$
FE+3PV	$\frac{1}{p^4}$	$16\mathcal{M}+6\mathcal{S}$	$43\mathcal{M}+9\mathcal{S}$	$4\mathcal{M}+1\frac{1}{2}\mathcal{S}$	$10\frac{3}{4}\mathcal{M}+2\frac{1}{4}\mathcal{S}$

Table 5.1 summarizes the results of this section, and compares them to each other and to the coherency test presented in [23] in terms of PUE and both absolute and normalized cost. Recall that the normalized cost is defined in (5.1) as a function of the absolute cost and the PUE. It can be seen that in the homogeneous projective representation, the CL+0PV test provides the least absolute cost while PA+2PV provides the least normalized cost.

5.4 Repeated Validation using Coherency-based Validation Tests

When the tests discussed earlier are used to implement repeated validation, it is possible to reduce their cost significantly by reusing some of the field operations that are performed in the original computation. This mainly applies to computations in the projective representation. Note that this applies only for intermediate tests in the error-free case, but the majority of performed validation tests fall under this description.

Repeated Validation using Point Addition. The idea here is to replace the addition test $Q_0 + Q_1 \overset{?}{=} Q_2$ with either one of the equivalent tests $Q_2 - Q_0 \overset{?}{=} Q_1$ or $Q_2 - Q_1 \overset{?}{=} Q_q$ based on whether the next bit of the key is 0 or 1, respectively. This way, the test will share most of the field multiplications needed for with the following point addition.

Let us consider the case of the next bit of k being 0, so the next addition operation will be $Q_0 \leftarrow Q_2 + Q_0$. In this case, the addition test is transformed to $Q_2 - Q_0 \overset{?}{=} Q_1$. Note that the point addition in the test and the point addition in the regular computation share most of the field operations since the only different inputs coordinates are Y_{Q_0} in the main addition and $-Y_{Q_0}$ in the validation test. A look at the explicit formulas for point addition in homogeneous projective coordinates [9] indicates that $9\mathcal{M} + \mathcal{S}$ operations are common between the point addition $Q_2 + Q_0$ and the point subtraction $Q_2 - Q_0$. So, the cost of the addition test along with the point comparison becomes $7\mathcal{M} + \mathcal{S}$, which is less than half the cost of the same test when done independently, i.e., $16\mathcal{M} + 2\mathcal{S}$.

Repeated Validation using Collinearity Testing. For this test, a side-by-side look at the explicit formulas and (5.28) shows that there are $4\mathcal{M}$ common operations regardless of the next bit of k, so the cost of the test become $6\mathcal{M}$.

Table 5.2 summarizes the effect of reusing the field operations on the absolute and normalized costs of the validation tests.

Table 5.2: The absolute and normalized costs of the PA and CL coherency tests for projective representation when field operations are shared with the following point addition.

Test	PUE	Absolute cost	Normalized cost
PA+0PV	$\frac{1}{p^2}$	$7\mathcal{M} + 1\mathcal{S}$	$3\frac{1}{2}\mathcal{M} + \frac{1}{2}\mathcal{S}$
PA+1PV	$\frac{1}{p^3}$	$13\mathcal{M} + 4\mathcal{S}$	$4\frac{1}{3}\mathcal{M} + 1\frac{1}{3}\mathcal{S}$
PA+2PV	$\frac{1}{p^4}$	$19\mathcal{M} + 7\mathcal{S}$	$4\frac{3}{4}\mathcal{M} + 1\frac{3}{4}\mathcal{S}$
PA+3PV	$\frac{1}{p^4}$	$25\mathcal{M} + 10\mathcal{S}$	$6\frac{1}{4}\mathcal{M} + 2\frac{1}{2}\mathcal{S}$
CL+0PV	$\frac{1}{p}$	$6\mathcal{M}$	$6\mathcal{M}$
CL+1PV	$\frac{1}{p^2}$	$12\mathcal{M} + 3\mathcal{S}$	$6\mathcal{M} + 1\frac{1}{2}\mathcal{S}$
CL+2PV	$\frac{1}{p^3}$	$18\mathcal{M} + 6\mathcal{S}$	$6\mathcal{M} + 2\mathcal{S}$
CL+3PV	$\frac{1}{p^4}$	$24\mathcal{M} + 9\mathcal{S}$	$6\mathcal{M} + 2\frac{1}{4}\mathcal{S}$

5.5 Infective Computation Variants

Decisional validation tests are vulnerable to a class of fault attacks commonly called the double-fault attacks, which attempt to bypass a validation test and allow a faulty value to be observed by the attacker. Double-fault attacks are generally countered by avoiding decisional validation test that can be easy to manipulate [73]. An effective approach to prevent these attacks is the use of tests based on infective computation, which works by masking invalid results randomly and letting valid results pass through unmodified. Hence, the outcome of the test is not dependent on a single bit that represents the logical outcome of a decisional test.

In general, any decisional test can converted to an infective test. This can be achieved by replacing the comparison in the test with a subtraction, i.e., the two sides are equal if and only if the result of the subtraction is zero. Then, the outcome of the subtraction is multiplied by a random number and added to the potential final result, which is then returned as the outcome of the test. This way, whenever the result of the subtraction is not zero, the returned result is masked by a random number, and hence, is uninformative to the attacker. We apply this idea to the tests discussed earlier and evaluate their effectiveness and cost.

5.5.1 Infective Point Validation

Point validation is performed by substituting the coordinates of a point into the curve equation. In affine coordinates, this implies testing the validity of the equation $y^2 \overset{?}{=} x^3 + ax + b$, or equivalently,

$$y^2 - x^3 - ax - b \overset{?}{=} 0 \tag{5.41}$$

To convert this to an infective test, a random nonzero field element r is selected and the value T is computed as

$$T = y^2 - x^3 - ax - b \tag{5.42}$$

Then, the return value of the infective test is computed as $(rT + x, rT + y)$. Clearly, rT is zero if and only if the point (x, y) lies on the curve. The cost of this computation is $3\mathcal{M} + 2\mathcal{S}$, which requires one field multiplication more than a decisional point validation.

A similar approach applies to projective coordinates. In particular, T can be computed as

$$T = Y^2Z - X^3 - aXZ^2 - bZ^3 \tag{5.43}$$

and then the return value of the test becomes $(rT + X, rT + Y, rT + Z)$. This requires $7\mathcal{M} + 3\mathcal{S}$, which is again one multiplication more costly that its decisional counterpart.

5.5.2 Infective Point Addition

The coherency test based on point addition can be converted to infective computation in a similar way. The main difference between this case and the earlier discussion is that we have more than one equality condition to be tested simultaneously.

Let us consider the affine case where the two equality tests

$$\begin{aligned}&(x_{Q_2}+x_{Q_0}+x_{Q_1})(x_{Q_1}-x_{Q_0})^2 \overset{?}{=} (y_{Q_1}-y_{Q_0})^2\\ &(x_{Q_1}-x_{Q_0})^2((y_{Q_2}+y_{Q_0})(x_{Q_1}-x_{Q_0})-(2x_{Q_0}+x_{Q_1})(y_{Q_1}-y_{Q_0})) \overset{?}{=} -(y_{Q_1}-y_{Q_0})^3\end{aligned} \tag{5.44}$$

are evaluated and if they are both true then the result (x_{Q_1}, y_{Q_1}) is accepted as valid. The two equations above can be readily replaced by the computations

$$\begin{aligned}T_1 &= (x_{Q_2}+x_{Q_0}+x_{Q_1})(x_{Q_1}-x_{Q_0})^2-(y_{Q_1}-y_{Q_0})^2\\ T_2 &= (x_{Q_1}-x_{Q_0})^2((y_{Q_2}+y_{Q_0})(x_{Q_1}-x_{Q_0})-(2x_{Q_0}+x_{Q_1})(y_{Q_1}-y_{Q_0}))+(y_{Q_1}-y_{Q_0})^3\end{aligned} \tag{5.45}$$

However, there are more than one way to use the values T_1 and T_2 in an infective computation, and not all approaches have the same effectiveness. We will consider three approaches here to illustrate this fact.

The first approach is a direct extension of the one used for the point validation earlier. In particular, the two values T_1 and T_2 are combined by addition and then used to mask both coordinates, i.e., the returned result is of the form $(r(T_1+T_2)+x_{Q_1}, r(T_1+T_2)+y_{Q_1})$. To evaluate the effectiveness of this approach, we note that T_1+T_2 is equal to zero not only when both values are zero, which is the intended test, but whenever $T_1=-T_2$. This is true for $\frac{1}{p}$ of the p^6 possible state vectors. For comparison, the corresponding decisional test passes only $\frac{1}{p^2}$ of the possible state vectors. In other words, this infective test is considerably less effective than the corresponding decisional validation test.

Instead, alternative forms like $(r(T_1+T_2)+x_{Q_1}, r(T_1-T_2)+y_{Q_1})$ or $(rT_1+x_{Q_1}, rT_2+y_{Q_1})$ can be used. Both forms have the advantage of passing only $\frac{1}{p^2}$ of the state vectors unmodified as it requires both T_1 and T_2 to be equal to zero. As such, they are as effective as the decisional version. Both of these forms requires two multiplications in addition to the operations required by the decisional point addition test, for a total of $7\mathcal{M}+2\mathcal{S}$.

The same idea applies in the projective case. In particular, (5.15) shows that two element comparisons are required to establish the equality of points Q_2 and $\tilde{Q}_2$. These two element comparisons can be replaced by subtractions and used to mask X_{Q_1} and Y_{Q_1}. This will require two additional multiplication compared to the decisional test for a total of $18\mathcal{M} + 2\mathcal{S}$.

5.5.3 Invective Collinearity Testing

The earlier discussion applies to validation tests based on collinearity testing. In particular, for affine coordinates, let T be as follows.

$$T = (y_{Q_1} + y_{Q_2})(x_{Q_0} - x_{Q_2}) - (y_{Q_0} + y_{Q_2})(x_{Q_1} - x_{Q_2}) \tag{5.46}$$

Then, the returned result of the test is of the form $(rT + x_{Q_1}, rT + y_{Q_1})$, which requires one multiplication more than the corresponding decisional test, for a total of $3\mathcal{M}$.

Similarly, collinearity testing in projective coordinates requires one equality test, and hence can be easily converted to an infective version. Let T be defined such that

$$T = (Y_{Q_1}Z_{Q_2} + Y_{Q_2}Z_{Q_1})(X_{Q_0}Z_{Q_2} - X_{Q_2}Z_{Q_0}) - (Y_{Q_0}Z_{Q_2} + Y_{Q_2}Z_{Q_0})(X_{Q_1}Z_{Q_2} - X_{Q_2}Z_{Q_1}) \tag{5.47}$$

Then, the returned result is of the form $(rT + X_{Q_1}, rT + Y_{Q_1}, rT + Z_{Q_1})$. In total, this requires $11\mathcal{M}$.

5.6 Validation Testing using Combined Curves

Computation over combined curves, i.e., elliptic curves defined over rings, is another way, along with point validation, to create and use information redundancy. This section discusses the use of combined curves in validation testing. We start by reviewing the definition and properties of elliptic curves defined over rings. Then, we review two known countermeasures that use this idea, one presented in [11] and the other presented in [6], and evaluate them. After that, we present our countermeasure that uses this idea along with coherency-based validation testing to reduce the cost of validation.

5.6.1 Elliptic Curves over Rings

It is possible to define an elliptic curve over a ring instead of a field, but such a curve would have some key differences, some of which will be discussed here. Let $\mathbb{Z}_{pt}$ be a ring of size pt where p and t are distinct primes greater than 3. Let $a_{pt} \in \mathbb{Z}_{pt}$ be such that $a_{pt} \equiv a_p \bmod p$ and $a_{pt} \equiv a_t \bmod t$, and similarly for b_{pt}. We denote this relationship henceforth by $a_{pt} = [a_p, a_t]$. Recall that the Chinese Remainder Theorem (CRT) can be used to compute a_{pt} from a_p and a_t as follows.

$$a_{pt} = a_p t (t^{-1})_p + a_t p (p^{-1})_t \tag{5.48}$$

where $(t^{-1})_p$ denotes the multiplicative inverse of t modulo p, and similarly for $(p^{-1})_t$.

Let $E_p(a_p, b_p)$ and $E_t(a_t, b_t)$ be two elliptic curves defined over the fields $\mathbb{F}_p$ and $\mathbb{F}_t$, respectively. An elliptic curve over the ring $\mathbb{Z}_{pt}$, denoted by $E_{pt}(a_{pt}, b_{pt})$, is the set of points $(x, y) \in \mathbb{Z}_{pt} \times \mathbb{Z}_{pt}$ satisfying the Weierstrass equation plus the point at infinity $\mathcal{O}_{pt}$.

According to the CRT, there exists a unique point $P_{pt} \in E_{pt}(a_{pt}, b_{pt}) - \{\mathcal{O}_{pt}\}$ for every pair of points $P_p \in E_p(a_p, b_p) - \{\mathcal{O}_p\}$ and $P_t \in E_t(a_t, b_t) - \{\mathcal{O}_t\}$ such that $P_{pt} \equiv P_p \bmod p$ and $P_{pt} \equiv P_t \bmod t$, which we denote by $P_{pt} = [P_p, P_t]$.

The set $E_{pt}(a_{pt}, b_{pt})$ does not form a group since it is not closed under the point addition operation. This is due to the fact that it has no points of the forms $[P_p, \mathcal{O}_t]$ or $[\mathcal{O}_p, P_t]$ as they have no affine representation [39]. On the other hand, the direct product of the groups $E_p(a_p, b_p)$ and $E_t(a_t, b_t)$, denoted by $\widetilde{E}_{pt}(a_{pt}, b_{pt})$, constitutes a group under the point addition operation. It has all the points in $E_{pt}(a_{pt}, b_{pt})$ along with points of the form $[P_p, \mathcal{O}_t]$ or $[\mathcal{O}_p, P_t]$. Since some of the points on $\widetilde{E}_{pt}(a_{pt}, b_{pt})$ have no affine representation, all arithmetic on this curve has to be performed in projective coordinates. Moreover, when performing the point addition operation, it is essential to check for special cases where the point addition formulas do not apply. In particular, when one of the added points is of the form $[P_p, \mathcal{O}_t]$, the Z coordinate of the resulting point will be equal to $0 \bmod t$ regardless of the other point. This applies generally to all projective representations where it is assumed that neither of the points to be added is the point at infinity. Hence, it is essential to check intermediate points for the case $Z \equiv 0 \bmod t$ and handle this case correctly. In particular, the addition of the points $P_1 = [P_{1,p}, \mathcal{O}_t]$ and $P_2 = [P_{2,p}, P_{2,t}]$ should result in $P_3 = [P_{1,p} + P_{2,p}, P_{2,t}]$, where the coordinates of P_3 need to be computed using the CRT from the coordinates of $P_{1,p} + P_{2,p}$ and $P_{2,t}$. This problem does not apply modulo p since the scalar k is typically less than p, i.e., $\mathcal{O}_p$ will not appear during the computation while $\mathcal{O}_t$ might.

As a result of this issue, performing the addition operation correctly over $\widetilde{E}_{pt}(a_{pt}, b_{pt})$ requires a reduction of the Z-coordinates mod t at the beginning of each point addition. If neither or both of the points has $Z \equiv 0 \bmod t$, then addition is performed using the addition or doubling formula over $\mathbb{Z}_{pt}$. Otherwise, the addition is performed over $\mathbb{F}_p$ and the CRT is used to construct the resulting point. This requires the precomputation of $p^{-1} \bmod t$ and $t^{-1} \bmod p$ in order to reduce the cost of the CRT step.

Another solution to this problem is to use a form of elliptic curves in which the regular point addition formula applies to special cases like doubling and addition to $\mathcal{O}$. A curve that has this feature is described as a *strongly-unified* curve. An example of a strongly-unified curve is the Edwards curve which was introduced in [35]. In this curve, the point addition formula can be used to perform point doubling as well as addition to $\mathcal{O}$. Moreover, since in this setting the curve is defined over a ring, a projective representation needs to be used. Examples of strongly-unified projective representations for Edwards curves include homogeneous projective coordinates and inverted coordinates [8].

5.6.2 Earlier Countermeasures using Combined Curves

There are two countermeasures based on combined curves that have been proposed earlier. Here, we review both and evaluate them.

Algorithm 9 Combined-curve countermeasure against sign change attacks. [11]

```
Input:  $P \in E_p$, $k \leq \text{ord}(P)$
Output: $kP \in E_p$
1: Setup: Choose $t$, $E_t$ and $P_t$. Compute $E_{pt}$ and $P_{pt}$ using the CRT.
2: Compute $R_{pt} \leftarrow kP_{pt}$ on $E_{pt}$.
3: Compute $R_t \leftarrow kP_t$ on $E_t$.
4: if $R_t \neq R_{pt} \bmod t$ then
5:     return $\mathcal{O}$
6: else
7:     return $R_{pt} \bmod p$
8: end if
```

5.6.2.1 Blömer et al. Combined-curve Countermeasure

In [11], the combined-curve countermeasure is presented as a way to detect sign-change errors that can not be detected by point validation. The size of the co-curve is chosen to achieve a high probability of detection, and as such tends to be on the same order of magnitude as the original curve. In particular, it is recommended to select t to be a prime of size 60–80 bits.

Let $\mathbb{F}_p$ and $\mathbb{F}_t$ be the original field and the additional, smaller field, respectively, and let E_p and E_t be the original curve and the smaller curve, respectively. Similarly, let P_p and P_t be the base points on E_p and E_t, respectively. Using the CRT, a curve E_{pt} can be constructed as well as a base point P_{pt}. Projective point arithmetic has to be used since inversion is not defined for some of the elements of the resulting ring.

The countermeasure is illustrated in Algorithm 9. This countermeasure depends on the fact that the injected fault will break the consistency between the scalar multiplication on the two curves, and hence can be detected by comparing their results.

The overhead introduced by this countermeasure can be divided into two parts. First, it requires an additional scalar multiplication over E_t. While E_t is significantly smaller than E_p, it still can not be ignored especially with the size of t being 60–80 bits. Let t and p be of sizes 70 and 192 bits, respectively, as suggested in [11]. Assuming that the field multiplication and squaring algorithms used have a complexity of $O(n^2)$ where n is the bit length of the modulus, the field operations on $\mathbb{F}_t$ can have about 13% of the cost of the corresponding operations on $\mathbb{F}_p$. However, the number of iterations in the scalar multiplication over E_t is smaller since k can be reduced modulo t. It follows that the cost of a scalar multiplication on E_t is about 5% of the corresponding cost on E_p.

The second part of the overhead is the arithmetic over $\mathbb{Z}_{pt}$ required to perform the scalar multiplication over E_{pt}. The modulus pt will have a length of 262 bits, so a multiplication or a squaring in this ring will incur 186% of the cost of the corresponding operation on $\mathbb{F}_p$. So in total, this countermeasure has an overhead of about 91% relative to a scalar multiplication over E_p.

5.6.2.2 Baek-Vasyltsov Combined-curve Countermeasure

The countermeasure presented in [6] is described as a unified countermeasure in the sense that it is a countermeasure against both DPA and fault attacks. This countermeasure combines the use of ring extension and point validation with a modified curve equation that enables point validation to detect the normally undetectable sign change errors. This idea is presented for both prime and binary curves. We focus here on prime curves since the binary case is very similar.

The curve equation used is $y^2 + py = x^3 + ax + b$. Note that over $\mathbb{F}_p$, this equation reduces to the more familiar $y^2 = x^3 + ax + b$. However, over $\mathbb{Z}_{pt}$, this curve has the property that an inverse of a point is not obtained in general by negating its y-coordinate. It follows that sign-change errors can be detected by point validation. Moreover, the cost of the point validation is reduced by performing it on $\mathbb{Z}_t$ rather than $\mathbb{Z}_{pt}$. Algorithm 10 illustrates the details of this countermeasure.

The main problem with this countermeasure is that it does not take care of the issue discussed earlier regarding the special cases in point addition when performed over a ring.

Algorithm 10 Unified combined-curve countermeasure against sign change and invalid-curve attacks. [6]

```
Input: $E : Y^2Z = X^3 + aXY + bZ^6$ an elliptic curve over $\mathbb{F}_p$. $P(x, y) \in E$. $k$ a scalar.
Output: $kP \in E(\mathbb{F}_p)$
1: Choose a small random integer $r$.
2: $B \leftarrow y^2 + py - x^3 - ax \bmod pr$
3: Let $E' : Y^2Z + pYZ^3 = X^3 + aXZ^4 + BZ^6$.
4: Compute $kP$ over $E'(\mathbb{Z}_{pr})$.
5: if $Y^2Z + pYZ^3 = X^3 + aXZ^4 + BZ^6 \bmod r$ for $kP(X, Y, Z)$ then
6:     return $(X, Y, Z) \bmod p$
7: else
8:     return $\mathcal{O}$
9: end if
```

This problem is compounded by the proposed approach to select t as a random integer rather than a carefully-selected prime. This results in a very high probability of failure for the validation test even when t has a relatively large bit length. For example, it is illustrated in [39] that for t of size 40-bits, the probability of failure of this test ranges from 0.4–1.4%, which is much higher that $\frac{1}{t} = 2^{-40} \approx 10^{-12}$. Moreover, a t of size 20-bits results in a probability of failure within the range 23–37%.

Another issue with this countermeasure is the overhead incurred by the modified addition formula that is required to perform point arithmetic over the non-conventional curve form. According to the authors, the modified addition formula has an overhead of 23% relative to the addition formula over a conventional curve form. This estimate captures only the increase in the number of primitive operations, and does not include the overhead of doing arithmetic over $\mathbb{Z}_{pt}$ compared to $\mathbb{F}_p$, which can be found separately as shown earlier when the combined-curve countermeasure by Blömer et al. was discussed.

5.6.3 Coherency-based Combined-curve Countermeasure

As shown earlier, coherency-based tests can be used to detect a variety of errors, including sign change errors, in a cost-effective way. Here, we aim to reduce their cost further using the combined-curve construction.

In a similar approach to Baek and Vasyltsov, we perform the computation over the combined curve and the validation tests over the smaller curve. However, we avoid the overhead of the modified addition formula by the use of conventional curve forms. Moreover, in our countermeasures, the smaller curve parameters, t, E_t and P_t, are selected carefully rather than at random to avoid the security weaknesses discussed earlier.

Compared to the countermeasure by Blömer et al., our countermeasure aims to reduce the cost by avoiding the need for a second scalar multiplication and by using a significantly smaller values of t. This is possible since we mainly use the smaller-curve validation tests for testing the intermediate state of the algorithm. Before final output is returned, a validation test can be performed in $\mathbb{F}_p$ to assure a very low PUE.

Algorithm 11 Coherency-based combined-curve validation.

```
Input:  $P_p \in E_p$, $k$ a scalar.
Output: $kP_p \in E_p$
1: Initialization: Select $t$, $E_t$ and $P_t$. Find $E_{pt}$ and $P_{pt}$ using the CRT.
2: Scalar Multiplication on $E_{pt}$: Compute $kP_{pt}$ over $\mathrm{E}_{pt}$ using Algorithm 8 and obtain
   $Q_{0,pt}$, $Q_{1,pt}$ and $Q_{2,pt}$, where $Q_{0,pt}$ is $Q_0$ on $\mathrm{E}_{pt}$.
3: Validation on $E_t$: If $Q_{0,t}$, $Q_{1,t}$ and $-Q_{2,t}$ are collinear then accept $Q_{1,p}$ as the result,
   else reject.
```

Algorithm 11 illustrates our combined-curve countermeasure. In the initialization step, t is selected to be a prime and E_t and P_t are selected such that P_t generates a group on E_t that has a prime order of size close to t. Also, we use the CL test as a validation test, but any of the validation tests discussed earlier will work. Note that, for the sake of clarity, we do not show the details of the repeated testing or the final comprehensive validation test. The former can be performed at fixed intervals or adaptively as discussed in Chapters 3 and 4, while the latter is similar to the validation tests discussed earlier when performed over E_p.

Since we use coherency of the internal state of the algorithm as a basis for the validation tests instead of the consistency between the results of two scalar multiplications, as in the countermeasure by Blömer et al., there is no need to perform the scalar multiplication twice. In other words, no scalar multiplication is performed on E_t, and hence all the points on E_t, with the exception of P_t that is selected in the initialization step, are derived from the corresponding points on E_{pt}. This is the key behind using points on E_t to validate points on E_{pt} since any error in a point on E_{pt} will likely affect the corresponding point on E_t.

5.6.3.1 Proportion of Undetected Errors

The proportion of undetected errors depends on both p and t, and on the choice of the validation test among the ones discussed earlier. For a validation test on the smaller curve, we define the PUE as the proportion of undetected errors that could have been detected by a similar test on the original curve. This measure captures the loss of effectiveness that is incurred in order to reduce the testing cost.

Let us assume that a collinearity test without point validation, i.e., CL+0PV, is used. In the original setting, i.e., an elliptic curve over a field of size p, we have shown that the proportion of state vectors that can pass this test is $\frac{1}{p}$. In the case of an elliptic curve over a ring $\mathbb{Z}_{pt}$, the test performed modulo p will have the same proportion of undetected errors while the proportion of detected errors will be the complement $1 - \frac{1}{p}$. When the test is performed modulo t, the proportion of undetected errors will be $\frac{1}{t}$ by the same argument.

Table 5.3 shows how the number of detected and undetected errors will look like on a combined curve E_{pt}. To obtain a proportion, any of these figures can be divided by $p^6t^6 - 1$, i.e., the total number of invalid states. We use the proportion of errors

Table 5.3: Number of detected and undetected invalid state vectors using the CL+0PV test on E_p and E_t. Note that the total number of invalid state vectors is $p^6t^6 - 1$ since one of the state vectors is the valid one.

	Detected on E_p	Undetected on E_p	Total
Detected on E_t	$p^5t^5(t-1)(p-1)$	$p^5t^5(t-1)$	$p^6t^5(t-1)$
Undetected on E_t	$p^5t^5(p-1)$	$p^5t^5 - 1$	$p^6t^5 - 1$
Total	$p^5t^6(p-1)$	$p^5t^6 - 1$	$p^6t^6 - 1$

that can be detected on E$_p$ but go undetected on E_t as the PUE of the combined-curve countermeasure. It can be seen from the table that this proportion is

$$\frac{p^5t^5(p-1)}{p^6t^6-1} \approx \frac{p^5t^5(p-1)}{p^6t^6} = \frac{p-1}{pt} = (1-\frac{1}{p})\frac{1}{t} \tag{5.49}$$

This can be generalized for all the tests discussed earlier as follows.

$$\text{PUE}_{pt} = \text{PUE}_t(1 - \text{PUE}_p) \tag{5.50}$$

5.6.3.2 Overhead Associated with Computation on a Combined Curve

It is essential to discuss the cost aspect of using a coherency-based combined-curve countermeasure. The overhead associated with combined curves can be divided into the following parts:

1. Precomputation overhead: This includes the computations of p^{-1} mod t and t^{-1} mod p as well as the use of the CRT to find E_{pt} and P_{pt}. Since t, E_t and P_t are selected off-line and can be used in many scalar multiplications, this part of the overhead can be left out from the operational overhead estimates.

2. Ring operations over $\mathbb{Z}_{pt}$ versus field operations over $\mathbb{F}_p$: Since the point additions will be performed over $\mathbb{Z}_{pt}$ instead of $\mathbb{F}_p$, the extra bit length will increase the cost of the primitive operations and particularly of multiplication and squaring. The

overhead will also depend on the multiplication algorithm used. If an algorithm of complexity $O(n^2)$ is used, where n is the bit length of the modulus, then the overhead of multiplication over $\mathbb{Z}_{pt}$ relative to $\mathbb{F}_p$ will be

$$\frac{(\log p + \log t)^2 - (\log p)^2}{(\log p)^2} \tag{5.51}$$

3. Field reductions of $\mathbb{Z}_{pt}$ elements to $\mathbb{F}_t$: As discussed earlier, it is required to test the Z coordinate of each of the added points mod t. In each iteration of the algorithm, this implies the need for two modular reductions modulo t, i.e., one for Z_{Q_2} and one for either Z_{Q_0} or Z_{Q_1}. When validation testing is used, coordinates of the modified points need to be reduced mod t for each validation test. This implies six modular reductions modulo t for each validation test as only two of the three points are updated every iteration.

4. Validation testing over $\mathbb{F}_t$: This covers the cost of coherency checking and point validations that are performed using points of E_t. The cost for this part follows directly from our earlier analysis.

5.7 Conclusion

This chapter discussed the cost and effectiveness of a collection of coherency-based approaches to validity testing as well as variants of these approaches in terms of the number of point validations. Moreover, it discussed the further saving that can be achieved when these tests are used is a repeated-validation setting. Beside the decisional variants of the tests, infective variants that are resistant to double-fault attacks were introduced and evaluated. As a way to further reduce the cost of testing and provide a flexible trade-off between cost and effectiveness, the extension of the above mentioned tests to the combined-curve setting has been introduced and discussed.

For the countermeasures discussed in this chapter to be effective, they need to be used in a way that does not give the attacker an advantage compared to other countermeasures.

In particular, a cost-effective validation test should be used frequently to increase the reliability and reduce the potential cost of recomputation while care should be taken to validate the output results using a low-PUE validation test to prevent any errors from escaping the tests. Moreover, the final validation test should use infective computation to avoid the compromise of the validation test by double-fault attacks. This setup combines the best of both methods, namely, the reliability gain of frequent validation and the high-probability and constant-cost of a single, comprehensive validation test.

Chapter 6

Conclusion

6.1 Summary

This work addresses some issues in the area of error detection and recovery for elliptic curve cryptosystems. It has been motivated by the need to reduce wasteful computations between an error occurrence and its detection, and reduce the wasteful recomputations of error-free iterations when error recovery is sought. An intuitive solution to this problem is to test frequently for errors rather than test once at the end before the output is provided. However, this simple solution introduces its own problem, namely, the increased overhead of testing.

In Chapter 3 of this document, we have addressed the problem of the fixed-block size selection for both error detection and recovery. In the case of error detection, we have constructed a model that estimates the cost of repeated testing and the wasteful computations between the occurrence of an error and its detection. This model can be used to find a block size that minimizes the testing cost and the wasteful computations when the statistical model of errors is fully known. However, it is not always practical to know the error model upfront. To mitigate this, we have presented another model that estimates the worst-case rather than the expected cost. This model does not depend on the statistical error model, and hence can be used to find a block size that is minimizes the worst-case cost while being close to the optimal block size in terms of expected cost.

Then, we have discussed the case of error recovery. We started by constructing a model that estimates the error recovery overhead, which includes cost of testing and recomputation, as a function of the system parameters. Given a statistical error model, this cost model can be used to find a block size that minimizes the expected overhead of error recovery. We have also discussed alternatives to minimizing the expected value, namely, minimizing a combination of the expected overhead and its standard deviation, and minimizing the reliability threshold. Both of these alternatives offer the advantage of reducing the sensitivity of the block size to the parameters of the error model. In effect, using either of these alternatives alleviates the need to know the error model accurately in advance. As a whole, the analysis in Chapter 3 has indicated that smaller blocks have a relative advantage in terms of reliability and efficiency under a variety of statistical error models.

Chapter 4 has followed by discussing the idea of error recovery using a block size that adapts based on whether an error was detected or not. This has been presented as an alternative to the fixed-block error recovery approach with the aim of reducing or eliminating the need to know the parameters of the error model in advance. To evaluate this idea, we have constructed a model that estimates the error recovery overhead as a function of the system parameters and the adaptive policy used in changing the block size. The results of this model, which were evaluated under a variety of error models and confirmed by simulation, have indicated that adaptive blocks achieve their intended goal of efficient error recovery without being tied to a specific error model. When discussing the above-mentioned model, an adaptive policy that inherently prefers smaller blocks was used based on the findings of Chapter 3. To evaluate this policy and help set its parameters, we have modeled the progressive changes in the block size using a Bayesian inference procedure that aims to learn the parameters of the statistical error model from the observed data. By applying this procedure to two different statistical error models, we have shown that an adaptive policy with preference for smaller blocks makes statistical sense. We have also provided a way to select the parameters of the adaptive policy based on the weight given to more recent observations relative to older ones.

Chapter 5 has addressed the issue of efficient error detection and recovery from a different angle. It has been recognized that, although Chapters 3 and 4 present solutions to reduce the overhead of validation testing, it still can be a significant part of the total overhead in the context of repeated testing. Chapter 5 has targeted this problem by presenting a collection of cost-effective tests that provide a trade-off between the cost of testing and its error detection effectiveness. We started by discussing the alternative approaches to coherency-based validation testing. For each of the approaches presented, we have evaluated error detection effectiveness as well as the absolute and normalized costs when used with a varying number of point validations. Then, we have discussed the usage of these validation tests in the context of repeated testing and shown how their costs can be reduced by exploiting the common primitive operations between the test and underlying scalar multiplication. Moreover, we have presented infective variants of these tests that are resistant to errors affecting the testing implementation. To allow for a more flexible trade-off, we have used the coherency-based testing approach in the context of combined curves to result in a comprehensive counter measure that avoids the problems associated with earlier, related countermeasures. The use of the validation tests presented in Chapter 5 in the repeated testing paradigm discussed earlier results in error detection and recovery structures that achieve high reliability without sacrificing efficiency and without being tied to a specific statistical error model.

6.2 Open Problems

In what follows, we discuss some of the related open problems that can provide a basis for future work.

- This work has focused solely on the point arithmetic level for both attacks considered as well as countermeasures presented and evaluated. This provides a degree of platform independence which leads to wider applicability. However, it is important to address attacks that target other layers of the implementation and their corresponding countermeasures. It is conceivable that the ideas and models discussed here can be applied at the field arithmetic level, or even at the machine level. Cost models that are tailored to these level could provide a better insight and more flexible trade-offs.

- Most of the available literature on fault analysis attacks on curve-based systems targets elliptic curve cryptosystems. This is understandable since the majority of these attacks and countermeasures can be directly extended to systems like hyperelliptic curve-based or pairing-based systems. It is interesting, however, to see whether these systems have attacks that are not applicable to elliptic curves. This will probably lead to countermeasures that are inherently different from the ones we currently use for elliptic curves.

- In Chapter 4, and specifically when we discussed the Bayesian inference procedure applied to Pareto-distributed errors, we faced the problem of the nonexistence of the Pareto distribution's expected value. While an expression that would temporarily solve this problem could have been substituted, this is neither honest nor aesthetically appealing. So the question remains, is there a way to estimate the occurrence time of an error when the expected value of the error distribution does not exist?

- In Chapter 4, we have discussed the use of an adaptive policy that has only one possible outcome depending on the result of the validation test. It is possible, however, to have a adaptive policy with a stochastic component such that the outcome is still influenced by the result of the test but is no longer limited to a deterministic function of the current block size. One way to set this policy is to use the posterior distribution resulting from the Bayesian inference step. This way, the adaptive policy can be influenced by both the estimated value of the model parameter as well as the system's confidence in that estimate, which is implied by the variance of the posterior distribution.

- While this work has addressed the issue of error detection and recovery in elliptic curve cryptosystems, we are not aware of any work on error-correcting elliptic curve primitives. This can be an extension of the existing work on error-correcting field arithmetic, e.g., [62] and [34], and algebraic geometric codes [30].

References

[1] IEEE Std 1363-2000. IEEE standard specifications for public-key cryptography. IEEE Computer Society, August 2000. 22

[2] Abdulaziz Alkhoraidly, Agustín Domínguez-Oviedo, and M. Anwar Hasan. Fault attacks on elliptic curve cryptosystems. In Marc Joye and Michael Tunstall, editors, *Fault Analysis in Cryptography*. Springer-Verlag's Information Security and Cryptography Series, March 2011. 7

[3] Abdulaziz Alkhoraidly and M. Anwar Hasan. Error detection and recovery for transient faults in elliptic curve scalar multiplication. CACR Technical Report 2009-06, University of Waterloo, May 2009. 34

[4] Abdulaziz Alkhoraidly and M. Anwar Hasan. Adaptive error recovery for transient faults in elliptic curve scalar multiplication. CACR Technical Report 2011-23, University of Waterloo, June 2010. 74

[5] Adrian Antipa, Daniel R. L. Brown, Alfred Menezes, René Struik, and Scott A. Vanstone. Validation of elliptic curve public keys. In Yvo Desmedt, editor, *Public Key Cryptography – PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 211–223. Springer, 2003. 2, 17

[6] Yoo-Jin Baek and Ihor Vasyltsov. How to prevent DPA and fault attack in a unified way for ECC scalar multiplication: ring extension method. In *Proceedings of the 3rd*

international conference on Information security practice and experience, ISPEC'07, pages 225–237. Springer-Verlag, 2007. 26, 29, 134, 138, 139

[7] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings the IEEE*, 94(2):370–382, 2006. 2, 3, 15, 29

[8] Daniel J. Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50. Springer, 2007. 47, 136

[9] Daniel J. Bernstein and Tanja Lange. Explicit-formulas database. Online, June 2011. 119, 130

[10] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault analysis of elliptic curve cryptosystems. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2000. 2, 3, 16, 17, 23, 29

[11] Johannes Blömer, Martin Otto, and Jean-Pierre Seifert. Sign change fault attacks on elliptic curve cryptosystems. In Luca Breveglieri, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2006*, volume 4236 of *Lecture Notes in Computer Science*, pages 36–52. Springer, 2006. 2, 24, 26, 29, 134, 137, 138

[12] W.M. Bolstad. *Introduction to Bayesian statistics*. John Wiley, 2007. 97

[13] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14(2):101–119, 2001. Earlier version published in EUROCRYPT '97. 20

[14] Luca Breveglieri, Shay Gueron, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors. *Fault Diagnosis and Tolerance in Cryptography – FDTC 2008*. IEEE Computer Society, 2008. 154, 155

[15] Mathieu Ciet and Marc Joye. Elliptic curve cryptosystems in the presence of permanent and transient faults. *Designs, Codes and Cryptography*, 36(1):33–43, 2005. 2, 17, 18, 19, 21, 22

[16] Henri Cohen and Gerhard Frey, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, volume 34 of *Discrete Mathematics and Its Applications*. Chapman & Hall/CRC, 2005. 8, 10

[17] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '98, pages 51–65, London, UK, 1998. Springer-Verlag. 10

[18] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999. 13, 111

[19] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Trans. Netw.*, 5(6):835–846, December 1997. 37

[20] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. 100

[21] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976. 1

[22] Agustín Domínguez-Oviedo. *On Fault-based Attacks and Countermeasures for Elliptic Curve Cryptosystems*. PhD thesis, University of Waterloo, 2008. 2, 20, 21

[23] Agustín Domínguez-Oviedo and M. Anwar Hasan. Algorithm-level error detection for ECSM. CACR Technical Report 2009-05, University of Waterloo, 2009. 3, 26, 27, 28, 29, 33, 46, 70, 110, 111, 112, 115, 116, 117, 118, 122, 129

[24] Agustín Domínguez-Oviedo and M. Anwar Hasan. Error detection and fault tolerance in ECSM using input randomization. *IEEE Transactions on Dependable and Secure Computing*, 6(3):175–187, 2009. 3, 29, 30, 58

[25] Agustín Domínguez-Oviedo and M. Anwar Hasan. Algorithm-level error detection for Montgomery ladder-based ECSM. *Journal of Cryptographic Engineering*, 1(1):57–69, 2011. 3, 111

[26] FIPS PUB 186-3. Digital signature standard (DSS). Federal Information Processing Standards Publication 186-3, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, June 2009. 22

[27] Pierre-Alain Fouque, Reynald Lercier, Denis Réal, and Frédéric Valette. Fault attack on elliptic curve Montgomery ladder implementation. In Breveglieri et al. [14], pages 92–98. 2, 19

[28] Christophe Giraud. Fault resistant RSA implementation. In Luca Breveglieri and Israel Koren, editors, *2nd Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC 2005*, pages 142–151, Edinburgh, UK, September 2005. 111, 154

[29] Christophe Giraud. An RSA implementation resistant to fault attacks and to simple power analysis. *IEEE Transactions on Computers*, 55(9):1116–1120, 2006. Extended abstract in [28]. 21, 26, 111, 112

[30] V. D. Goppa. Codes associated with divisors. *Problems of Information Transmission*, 13(1):22–27, 1977. 149

[31] G. Grimmett and D. Stirzaker. *Probability and random processes.* Texts from Oxford University Press. Oxford University Press, 2001. 58

[32] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004. 8

[33] Michael Hardy. Pareto's law. *The Mathematical Intelligencer*, 32(3):38–43, 2010. 38

[34] Arash Hariri and Arash Reyhani-Masoleh. Fault detection structures for the Montgomery multiplication over binary extension fields. In Luca Breveglieri, Shay Gueron, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2007*, pages 37–46. IEEE Computer Society, 2007. 149

[35] Harold M. Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*, 44:393–422, July 2007. 136

[36] C. Howson and P. Urbach. *Scientific reasoning: the Bayesian approach*. Open Court, 1993. 97

[37] Tetsuya Izu, Bodo Möller, and Tsuyoshi Takagi. Improved elliptic curve multiplication methods resistant against side-channel attacks. In Alfred Menezes and Palash Sarkar, editors, *Progress in Cryptology – INDOCRYPT 2002*, volume 2551 of *Lecture Notes in Computer Science*, pages 296–313. Springer, 2002. 14

[38] Marc Joye. Highly regular right-to-left algorithms for scalar multiplication. In *Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, CHES '07, pages 135–147, Berlin, Heidelberg, 2007. Springer-Verlag. 13

[39] Marc Joye. On the security of a unified countermeasure. In Breveglieri et al. [14], pages 87–91. 26, 135, 139

[40] Marc Joye. Fault attacks on elliptic curve cryptosystems. Presented at Crypto'Puces 2009, June 2009. Presented at Crypto'Puces 2009. 34

[41] Marc Joye and Sung-Ming Yen. The Montgomery powering ladder. In Kaliski Jr. et al. [42], pages 291–302. 14, 19

[42] Burton S. Kaliski Jr., Çetin K. Koç, and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*. Springer, 2002. 155, 158

[43] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987. 1

[44] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. 2

[45] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999. 2

[46] Israel Koren and C. Mani Krishna. *Fault-Tolerant Systems*. Morgan Kaufmann, 2007. 35, 58

[47] S. Lin and D.J. Costello. *Error Control Coding: Fundamentals and Applications*. Pearson-Prentice Hall, 2004. 114

[48] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page, D. Wright, J. Dobson, J. Mcdermid, and D. Gollmann. Towards Operational Measures of Computer Security. *Journal of Computer Security*, 2:211–229, 1993. 35

[49] Julio López and Ricardo Dahab. Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. In *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '99, pages 316–327, London, UK, 1999. Springer-Verlag. 14

[50] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freedman and Co., New York, 1983. 38

[51] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology – CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985. 1

[52] Peter L. Montgomery. Speeding up the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987. 13, 14, 25, 29, 111

[53] François Morain and Jorge Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Theoretical Informatics and Applications*, 24:531–543, 1990. 12

[54] W. Keith Nicholson. *Introduction to Abstract Algebra.* PWS Publishing Company, 1993. 8

[55] David M. Nicol, William H. Sanders, and Kishor S. Trivedi. Model-based evaluation: From dependability to security. *IEEE Trans. Dependable Secur. Comput.*, 1(1):48–65, January 2004. 35

[56] Vern Paxson and Sally Floyd. Wide-area traffic: The failure of Poisson modeling. *IEEE/ACM Trans. Netw.*, 3(3):226–244, June 1995. 37

[57] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over and its cryptographic significance (corresp.). *Information Theory, IEEE Transactions on*, 24(1):106–110, January 1978. 17

[58] J. M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):pp. 918–924, 1978. 17

[59] Michael Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, MIT Laboratory for Computer Science, January 1979. 1

[60] William J Reed. The Pareto, Zipf and other power laws. *Economics Letters*, 74(1):15–19, 2001. 38

[61] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. Method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. 1

[62] Siavash Bayat Sarmadi and M. Anwar Hasan. On concurrent detection of errors in polynomial basis multiplication. *IEEE Transactions on Very Large Scale Integration Systems*, 15(4):413–426, 2007. 149

[63] Bianca Schroeder, Sotirios Damouras, and Phillipa Gill. Understanding latent sector errors and how to protect against them. *ACM Transactions on Storage*, 6(3):9:1–9:23, September 2010. 37

[64] Adi Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks. US Patent #5,991,415, November 1999. Presented at the rump session of EUROCRYPT '97. 26

[65] Daniel Shanks. Class number, a theory of factorization and genera. In *Proc. Sympos. Pure Math., Vol. XX*, pages 415–440. American Mathematical Society, 1971. 17

[66] J.H. Silverman. *The arithmetic of elliptic curves.* Graduate texts in mathematics. Springer, 2009. 10, 115, 124

[67] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Kaliski Jr. et al. [42], pages 2–12. 20

[68] Jerome A. Solinas. Generalized Mersenne numbers. Technical Report CORR99-39, University of Waterloo, 1999. 22

[69] Martijn Stam. On Montgomery-like representations for elliptic curves over $GF(2^k)$. In Yvo Desmedt, editor, *Public Key Cryptography — PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 240–254. Springer Berlin / Heidelberg, 2002. 14

[70] R.B. Wells. *Applied coding and information theory for engineers.* Prentice-Hall information and system sciences series. Prentice Hall, 1999. 114

[71] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000. 2, 27

[72] Sung-Ming Yen, Seungjoo Kim, Seongan Lim, and Sang-Jae Moon. A countermeasure against one physical cryptanalysis may benefit another attack. In Kwangjo Kim, editor, *Information Security and Cryptology – ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, pages 269–294. Springer, 2002. 27

[73] Sung-Ming Yen, Seungjoo Kim, Seongan Lim, and Sang-Jae Moon. RSA speedup with Chinese Remainder Theorem immune against hardware fault cryptanalysis. *IEEE Transactions on Computers*, 52(4):461–472, 2003. 3, 28, 30, 131