Error Detection Techniques Against Strong Adversaries

by
Kahraman Daglar Akdemir

A Dissertation
Submitted to the Faculty
of the

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the
Degree of Doctor of Philosophy
in
Computer Engineering

November, 2010

Approved:

Prof. Berk Sunar ECE Department Dissertation Advisor	Prof. Xinming Huang ECE Department
Dr. Gunnar Gaubatz Intel Corporation	Prof. Fred J. Looft ECE Department Head

For my wife.

Abstract

"Side channel" attacks (SCA) pose a serious threat on many cryptographic devices and are shown to be effective on many existing security algorithms which are in the black box model considered to be secure. These attacks are based on the key idea of recovering secret information using implementation specific side-channels. Especially active fault injection attacks are very effective in terms of breaking otherwise impervious cryptographic schemes.

Various countermeasures have been proposed to provide security against these attacks. Double-Data-Rate (DDR) computation, dual-rail encoding, and simple concurrent error detection (CED) are the most popular of these solutions. Even though these security schemes provide sufficient security against weak adversaries, they can be broken relatively easily by a more advanced attacker. In this dissertation, we propose various error detection techniques that target strong adversaries with advanced fault injection capabilities.

We first describe the "advanced attacker" in detail and provide its characteristics. As part of this definition, we provide a generic metric to measure the strength of an adversary.

Next, we discuss various techniques for protecting finite state machines (FSMs) of cryptographic devices against active fault attacks. These techniques mainly depend on nonlinear robust codes and physically unclonable functions (PUFs). We show that due to the nonuniform behavior of FSM variables, securing FSMs using nonlinear codes is an important and difficult problem. As a solution to this problem, we propose error detection techniques based on nonlinear codes with different randomization methods. We also show how PUFs can be utilized to protect a class of FSMs. This solution provides security on the physical level as well as the logical level. In addition, for each technique, we provide possible hardware realizations and discuss area/security performance.

Furthermore, we provide an error detection technique for protecting elliptic curve point addition and doubling operations against active fault attacks. This technique is based on nonlinear robust codes and provides nearly perfect error detection capability (except with exponentially small probability). We also conduct a comprehensive analysis in which we apply our technique to different elliptic curves (i.e. Weierstrass and Edwards) over different coordinate systems (i.e. affine and projective).

Acknowledgments

First of all, I would like to give my sincerest thanks to my advisor, Professor Berk Sunar. I learned a lot from him. I will never forget the effort, time, and energy he put in my academic life. I also would like to say a few things in Turkish to show my appreciation for all his help: "Hocam, uzerimde cok hakkiniz ve emeginiz var. Fazla soze hacet yok, her sey icin cok tesekkur ederim."

I would also like to thank Prof. Xinming Huang and Dr. Gunnar Gaubatz for agreeing to be on my dissertation committee. I really appreciate the time they took to significantly improve my dissertation. Additionally, I would like to thank National Science Foundation for partially funding this research through the Cybertrust CNS Award No. 0831416.

A considerable amount of my time at WPI was funded through being a TA. I thank our department head Professor Fred Looft, for keeping me on board for all these years. I also want to thank my lab mates and coauthors whom I prefer to call my academic siblings: Deniz Karakoyunlu, Ghaith Hammouri, Yin Hu, Chenguang Yang, and Michael Moukarzel.

Also, I would like to thank my best friends Ismail Onur Filiz, Emrah Deniz, Erdinç Öztürk, Emrah Durulan and Volkan Kurt for always being there for me.

I also would like to thank my family, but I want to do it in my own language: "Annem Aytan Akdemir, babam Mehmet Akdemir, ve canimdan cok sevdigim cadi kardesim Burcu Caglar Akdemir'e her zaman yanimda olduklari ve beni destekledikleri icin tesekkur ederim. Sizleri cok ozluyorum. Umarim nese ve umut dolu bir hayatiniz, mutluluk dolu bir omrunuz olur; cunku ben mutlu olmak icin hep sizden guc aliyorum.."

Recently, I had a another father, mother, and brother whom I love as much as my own. I also would like to thank them: "Annem Mine Keceli, babam Sedat Keceli, ve ici tertemiz kardesim Mustafa Keceli'ye beni kucaklayip ailelerine dahil ettikleri icin tesekkur ederim. Her seyin en guzelini hakediyorsunuz. Umarim her sey gonlunuzce olur."

Last but certainly not least I want to thank my best friend and my wife Merve Keceli. "Seni ilk tanidigim gunden beri eksik etmedigin emegin, destegin, ve guler yuzun icin cok sagol Mervem. Sen oldugun icin, senin gibi oldugun icin, ve hep benimle olacagin icin cok tesekkur ederim."

Worcester, MA, Fall 2010

Kahraman Daglar Akdemir

Contents

Abstract **i**

Acknowledgments **iii**

Table of Contents **v**

List of Tables **viii**

List of Figures **ix**

1 Introduction **1**
1.1 Motivation 4
1.2 Dissertation Outline 5

2 Adversarial Faults and Detection Model **7**
2.1 Characterizing Adversarial Faults 7
2.1.1 Structure Obfuscation 7
2.1.2 Injection Methodology 8
2.1.3 Resolution 9
2.1.4 Abstract Error Model 10
2.1.5 Adaptive Injection 10
2.1.6 Multiplicity 10
2.1.7 Effective Duration 11
2.1.8 Invasiveness 11
2.2 The Advanced Attacker 12

3 Background on Robust Nonlinear Codes **15**

4 Hardening FSMs Against Strong Adversaries **21**
4.1 Motivation 21
4.2 The Error Detection Technique 24
4.2.1 Why Is It Difficult to Secure FSMs? 24
4.2.2 Proposed Solution 26
4.2.3 Security Proof 29

4.3 Proposed Next-State Logic Construction . . . 31
4.3.1 Generic Technique for Next-State Logic Construction . . . 31
4.3.2 Case Study Utilizing the Proposed Construction . . . 34
4.4 Hardware Realizations of the Next-State Logic . . . 37
4.5 Comparison with other FSM security schemes . . . 39
4.6 Implementation Results . . . 41
4.7 Summary . . . 45

5 MUX-based Error Detection for FSMs 47
5.1 Motivation . . . 47
5.2 The Error Detection Technique . . . 48
5.3 Security Analysis . . . 54
5.4 Hardware Scalability . . . 57
5.5 Comparison with other FSM security schemes . . . 58
5.6 Summary . . . 59

6 PUF-based Error Detection Methods in FSMs 61
6.1 Motivation . . . 61
6.2 Physically Unclonable Functions . . . 63
6.3 Securing Known-Path State Machines . . . 66
6.4 Key Integrity . . . 73
6.5 Error Detection Network Security . . . 77
6.6 Summary . . . 79

7 Nonlinear Error Detection for ECC 81
7.1 Motivation . . . 81
7.2 Background . . . 83
7.2.1 Elliptic Curve Cryptography Overview . . . 84
7.2.1.1 Simplified Weierstrass Formulation for Elliptic Curves . . . 84
7.2.1.2 Edwards Formulation for Elliptic Curves . . . 85
7.2.2 Existing Error Detection Techniques in ECC . . . 87
7.3 The Error Detection Technique . . . 89
7.3.1 Security Analysis . . . 90
7.4 Proposed Point Addition and Doubling Constructions . . . 94
7.4.1 Edwards Projective Unified Addition . . . 94
7.4.2 Weierstrass Affine Addition and Doubling . . . 100
7.5 Results and Discussion . . . 103
7.6 Summary . . . 109

8 Conclusion 111

Bibliography 113

A Non-redundant Datapath Formulas in Dataflow Format 123

B Predictor Designs **126**

List of Tables

4.1 Grouped state assignment table 34
4.2 First level Lagrange 35
4.3 Second level Lagrange 36
4.4 Hardware implementation results for different FSM protection schemes. . . 42
4.5 Breakdown of the gate count for different Montgomery multipliers. 44
4.6 Overhead caused by our protection scheme for different MMs. 44

7.1 Overhead analysis for the error detection technique proposed in this chapter 104
7.2 Operation counts of robust modular arithmetic functions proposed in [22] . 105
7.3 Overhead analysis for the error detection technique proposed in [22] 106
7.4 Estimated overhead comparison 108

List of Figures

4.1 Fault injection example on the control unit of the Montgomery Ladder Algorithm with Point of Attack indicated by the dashed transition [24] 22
4.2 Arithmetic parallel hardware implementation of the next-state logic 38
4.3 Time redundant (serial) arithmetic hardware implementation of the next-state logic 39
4.4 Error masking probability per unit area 43

5.1 Proposed error detection technique 49

6.1 A basic delay based PUF circuit 64
6.2 State Diagram Representation of Left-to-Right Exponentiation Algorithm with Point of Attack 68
6.3 PUF-based circuit for protecting FSMs 69
6.4 Key integrity check using PUF 74
6.5 PUF based EDN 78

7.1 Secure Edwards projective unified point addition 98
7.2 Secure Weierstrass affine point addition 100
7.3 Secure Weierstrass affine point doubling 101

Chapter 1

Introduction

"Side channel" attacks (SCA) pose a serious threat on many cryptographic devices and are shown to be effective on many existing security algorithms which are in the black box model considered to be secure. These attacks are based on the key idea of recovering secret information using implementation specific side-channels. SCA can be broadly classified as passive and active attacks.

In passive attacks, the adversary is restricted to only observing implementation specific characteristics of the chip to reveal the secret information located in the device. Power consumption, timing information, electromagnetic (EM) emissions, and acoustic emissions are the most popularly utilized sources of passive attacks.

Power and timing analysis attacks mainly exploit information leaked through power and timing channels in order to deduce the secret data [37, 38]. Note that while a cryptographic device operates, it consumes power. The amount of power consumed is related to the amount of computation conducted by the device. Consequently, by tapping into the power pin of the device and observing the power consumption profile, an attacker can deduce information about the operations conducted in that device. Also note that when these operations are data dependent (e.g. dependent on the secret key), the adversary can gain information about the key used inside this device.

A similar approach applies to timing information as well when operation durations are data dependent.

Similarly, while electronic circuits (i.e. ICs, sensors, connection cables, etc.) operate, they leak EM waves at different frequencies. An attacker could use this leaked information to retrieve internal/secret data that is located on that device. Various versions of this side-channel attack are proven to be effective against different applications. For example, the project code named TEMPEST investigates the utilization of unintentional emissions (electromagnetic, mechanical, and acoustic) from a source to disclose secret information about that source [1]. Similarly, Kuhn showed that by collecting the EM waves leaked from the monitor cables of computers, the displayed data on a CRT monitor can be reconstructed by an adversary [40, 41, 42]. In other words, using the leaked EM information, the attacker can see what we have on our monitor even through walls. Furthermore, EM emanations are used to reach to secret keys in cryptographic devices such as RFIDs and FPGAs [17, 52]. In this case, the attacker analyzes the EM spectrum while the chip is running using a coil and processes this information to retrieve the secret key.

Acoustic attacks can also be used in order to leak information about a cryptographic device. These attacks are very similar to electromagnetic attacks, yet use acoustic waves instead of electromagnetic waves. Acoustic side-channel attacks are already proven to be effective in extracting the executed instruction information from CPUs [61]. In this case, by listening to the acoustic sound emanations of a CPU, the attacker can obtain the executed instruction flow and break the system. [5, 69] show that a similar attack could be conducted on keyboards as well. In this case, the attacker can identify the typed characters on a keyboard even through walls. Note that each time a key is pressed, it leaks acoustic information which has key specific characteristics. As a result, an attacker can steal passwords using this technique.

Boneh et al. [13] demonstrated that the active fault injection attacks are also very

effective in terms of breaking otherwise impervious cryptographic schemes. The main idea of active attacks is to inject faults into specific parts of an integrated circuit (IC). This faulty execution is used to gain access to the secret information, such as the cryptographic key. More specifically, the seminal paper of Boneh et al. [13] demonstrated that it is relatively easy to break the Chinese Remainder Theorem (CRT) based RSA algorithm in the presence of faults. They mainly proposed and showed how to break this scheme by injecting a fault into one of the two exponentiations required to generate the RSA signature. Following this work, Biham et al. [11] introduced differential fault attacks (DFA) and showed that secret key cryptosystems such as DES can also be compromised using active fault injection attacks.

External clock transients, variations in the external voltage lines, temperature variations, EM fault injection, and bit-flips using optical laser beams are some of the many active fault attack techniques found in the literature.

In case of the external clock transients, the clock line is toggled in an unexpected nature and this in turn causes a faulty operation on the sequential portion of the circuit. Mainly, the flip-flops in the circuit are affected from this attack. Due to the faulty clock transitions, faulty digital values are latched by these flip-flops. External voltage line attacks are similar to clock transient attacks but use power lines instead. Due to variations on the voltage lines, some circuit elements behave unexpectedly and cause faults.

Similarly, temperature variations can be used as the source to create faults. Since temperature is a very important parameter in the performance of electronic components, by changing the temperature of an integrated circuit (i.e. heating or cooling), some components on the device can be forced to perform unexpectedly.

EM-based fault injection techniques depend on the formation of Eddy current on the surface of a chip due to an adversarial EM pulse [60]. The EM pulse is generated using a spark generator. Essentially, the Eddy current passes through the silicon

and causes bit-flips on the register/gate level. In case of the optical fault injection attacks, the first step is to remove the package of the device under attack (DUA). Next, an optical laser beam is shined on the silicon in order to cause bit-flips on the register/gate level [60, 63].

More information about side-channel attacks could be found in the survey papers of Bar-El et al. [6] and Naccache [54]. In this dissertation, we mainly focus on active fault attacks and propose various error detection techniques against them.

1.1 Motivation

Side-channel attacks pose a serious threat for many cryptographic applications, such as smartcards. Typically active attacks are harder to introduce but can be more effective. Various countermeasures have been proposed to provide security against these attacks. Double-Data-Rate (DDR) computation, dual-rail encoding and concurrent error detection (CED) are the most popular countermeasures proposed in the literature to counter active attacks.

In the Double-Data-Rate (DDR) computation technique [50, 51], both edges of the clock are used to make the same computation twice and check for errors by comparing the two results.

Another countermeasure proposed against side-channel attacks is the dual-rail encoding. In these schemes, a data value is represented by two wires. In this case, two out of four states represent the data and the two extra states which are "quiet" and "alarm" can be used for security purposes. Consequently, utilizing the dual railing for the cryptographic designs can potentially provide security against active and passive side-channel attacks [64, 68, 16, 43].

The most commonly used fault detection technique is CED which employ circuit level coding techniques, e.g. parity schemes, modular redundancy, temporal redun-

dancy, etc. to produce and verify check digits after each computation [9, 23, 35, 53]. For example, in triple modular redundancy (TMR), three copies of the non-redundant circuit are run in parallel and a majority voting decides if the operation of the device is fault free or not. Quadruple modular redundancy (QMR) works exactly the same way but with four copies. Temporal redundancy is based on a similar idea. In this case, recomputations are done by the same hardware but at different times. If the results of these different computations do not match, this points to a possible fault injection attack.

Non-linear robust error detection codes [34], a more advanced CED technique, provide the most effective defense mechanism against active fault attacks. The details of this coding scheme and its possible applications, e.g. protection of the Advanced Encryption Standard (AES) datapath, are described in [33, 32, 44, 45, 46].

Even though these protection schemes provide sufficient security against weak adversaries, they can be broken relatively easily (except nonlinear robust codes) by a more advanced attacker. Hence, in this dissertation, we first define the "advanced attacker" in detail and provide its characteristics. As part of this definition, we provide a generic metric to measure the strength of an adversary. Next, we propose various error detection schemes on different cryptographic blocks that will now be secure even against this highly talented adversary.

1.2 Dissertation Outline

This dissertation consists of four major chapters (4 through 7) that represent the major results of the authors research. It is organized as follows: In Chapter 2 we will provide the reader with an overview of our adversarial fault model that forms the basis of our research and we will define the "advanced attacker" in detail. In Chapter 3, we will give the required background information associated with robust

nonlinear codes. Chapter 4 contains the results of our research on hardening state machines against strong adversaries. We show how robust nonlinear codes can be used to provide redundancy and error detection mechanisms for finite state machines (FSMs). In Chapter 5, we consider a different nonlinear error detection technique for FSMs. This is a multiplexer-based technique which is inspired from the efficient FSM implementations. Chapter 6 provides a high level description of a technique which uses physically unclonable functions (PUFs) to secure a class of FSMs against fault injection attacks. Finally, in Chapter 7, we discuss how to apply systematic nonlinear error detection codes to protect elliptic curve point addition and doubling operations against an advanced attacker.

Chapter 2

Adversarial Faults and Detection Model

All the proposed error detection schemes in this dissertation target an advanced attacker with high fault injection capabilities. In this chapter, we define this "advanced attacker" model and provide the details of its characteristics.

2.1 Characterizing Adversarial Faults

In this section, we provide a general taxonomy of the adversarial faults. We mainly describe the important features associated with an adversary.

2.1.1 Structure Obfuscation

This feature essentially is a measure of the amount of information the attacker has of the system he is attacking to. From an attacker's point of view, while conducting an attack on a cryptographic system, having information about the structure of the system becomes very crucial. In what follows, we show some of the important questions that are relevant from an attacker's perspective when attacking a system.

- What are the main arithmetic/logical blocks in this system?
- What specific function does each block implement?
- How is the circuit laid-out on the silicon?
- What kind of a timing behavior does the system show?

Most of the time, even though the attacker will have some idea about the structure of the system, all these details will not be public. However, this in turn becomes some form of security through obfuscation. Ideally, a cryptographic system should still survive when there is no obfuscation about the structure and the attacker has a detailed understanding of the system.

2.1.2 Injection Methodology

This feature describes the possible fault injection methods an attacker can use to conduct an attack on a cryptographic system. In the following, we cover some of these crucial methodologies that are used to implement active attacks on cryptographic chips.

- **Optical:** In this method, the first step is to remove the package of the device under attack (DUA). Next, an optical laser beam is shined on the silicon in order to cause bit-flips on the register/gate level.
- **Electromagnetic:** This method depends on the Eddy current that is formed on the surface of a chip due to an EM pulse. The EM pulse is generated using a spark generator. Essentially, the Eddy current passes through the silicon and causes bit-flips on the register/gate level.
- **Clock transients:** In this method, the clock line is toggled in an unexpected nature and this in turn causes a faulty operation on the sequential portion of

the circuit. Mainly, the flip-flops in the circuit are affected because due to the faulty clock transitions, wrong values can be latched in this case.

- **Variations in the external power lines:** This fault injection method is similar to clock transient attacks but uses power lines instead. Due to variations on the voltage lines, some circuit elements behave unexpectedly and cause faults.

- **Temperature variations:** Since temperature is a very important parameter in the performance of electronic components, by changing the temperature of an integrated circuit, some components on the device can be forced to perform unexpectedly.

An attacker can use any of these fault injection methodologies depending on his resources to break a system. Ideally, a cryptographic circuit should be secure against any of these attacks.

2.1.3 Resolution

This feature is a measure of the attacker's capability in injecting a fault to a specific part of the circuit at a specific time. Resolution can be broadly discussed under the following two categories:

- **Temporal resolution** defines the accuracy of the attacker in the time domain, i.e. fault insertion at a particular time.

- **Spatial resolution** on the other hand is a measure of the attacker's precision in injecting a fault at a very specific location on the device.

An attacker with limited resources probably will not have high temporal and spatial resolution. On the other hand, an advanced attacker with various resources, tools, and machines can inject faults with high precision.

2.1.4 Abstract Error Model

The basic active fault injection model assumes that when the attacker injects a fault into the circuit, an erroneous result will be observed at the output. Consequently, the error e becomes the difference between the expected output x and the observed output $\tilde{x}=x+e$. This error can be characterized either as **logical** or **arithmetic** depending on the functions implemented by the target device. If the target area of the device is mostly dominated by flip-flops, RAM, registers, etc. then using the logical model, where the error is expressed as an XOR operation ($\tilde{x}=x \oplus e$), is more appropriate. If on the other hand, the targeted region of the device is an arithmetic circuit, then it is more useful to use the arithmetic model where the error may be expressed as an addition with carry ($\tilde{x}=x+e \bmod 2^k$, where k is the data width).

2.1.5 Adaptive Injection

We define the **adaptive fault injection** feature as follows: If the attacker can observe any existing data on the circuit at the time of fault injection i.e. in the same clock cycle, and can adjust his fault injection accordingly, then he is told to have an adaptive fault injection capability. This means that the attacker will be able to adaptively attack the circuit by first reading the existing data and then choosing the appropriate error vector.

2.1.6 Multiplicity

Error multiplicity is a measure of how many bits of the output is affected due to the implemented active attack. For example, if 8-bits of the output are affected (i.e. changed value), this is called to be an error with a multiplicity of 8. It is usually reasonable to assume that injecting errors with higher multiplicities requires more effort. Hence, an advanced attacker will be able to inject faults with various

multiplicities while an attacker with limited resources will not be able to pass a threshold.

2.1.7 Effective Duration

This feature is a measure of the duration for which the injected fault stays in effect. In general, faults can be broadly classified into the following two classes when effective duration is concerned:

- **Permanent Faults** are the faults that are irreversible. In this case, the effect of the injected fault on the device is permanent.
- **Transient Faults** are on the other hand non-permanent. In other words, the effect of this type of faults are temporary and the device returns to normal operation usually after a couple of clock cycles.

In general, it can be argued that the transient faults are usually more useful for the attacker because in this case he can try many different fault injections without affecting the device permanently.

2.1.8 Invasiveness

In an adversarial setting, invasiveness is a measure of the DUA's mechanical integrity. Invasiveness of an attack can be categorized into the following classes:

- **Non-invasive faults** preserves the mechanical integrity of the DUA. The package, any protective coating, etc. are not removed. The fault injection is mainly limited to the electromagnetic radiation or to the pins of the IC.
- **Invasive faults** on the other hand sever the mechanical integrity of the DUA. Potentially, the package of the device is removed and any form of tamper-

proofing circuitry is disabled. This is probably the most advanced attack applicable because it gives the attacker the highest level of control over the device.

- **Semi-invasive faults** provide a compromise between the invasive and non-invasive cases. In this case, the package is removed but the rest of the circuit remains intact.

Invasiveness is essentially a measure of how advanced an attacker is. More advanced attackers want a higher level of control on the device. As a result, they need to be more invasive while injecting faults.

2.2 The Advanced Attacker

In this section, we define the "advanced attacker" which is the main target of the proposed error detection techniques in this dissertation. The following assumptions in connection with the characteristics we discussed in Section 2.1 define our fault model.

1. The details of the particular structure/function of the device is public.
2. There is no limit on the attacker's particular fault injection methodology.
3. The attacker is an advanced one with high temporal and spatial fault injection capability.
4. The errors observed at the output of the device are always additive in nature and the attacker cannot conduct his attack by overwriting the output values.
5. The attacker cannot observe any existing data on the circuit at the time of fault injection i.e. in the same clock cycle. This means that the attacker will not be able to adaptively attack the circuit by first reading the existing data and then choosing the appropriate error vector.

6. The attacker can configure the fault injection process to reflect any specific error vector he desires at the output, i.e. he can pick the value of e.

7. Every error vector (all multiplicities) can be observed at the output of the device. [1]

The detection model we assume disables the device or resets the secret information after an injected fault is detected. Hence, the attacker will not be able to try many different error vectors until he breaks the device. He has only one chance to successfully inject a fault.

[1] Also, a small study we conducted suggests that for limited fault injection multiplicities on a specific circuit (e.g. logical values of only m nodes are inverted with a fault), error distribution at the output is not uniform and depends on the specifics of the logic design. However, since we are trying to construct a generic method that is applicable to all FSMs and since the attackers will always aim for the most vulnerable error vector in a circuit, we will conduct our error detection study assuming that all the errors can be observed at the output of the proposed design with the same frequency.

Chapter 3

Background on Robust Nonlinear Codes

Karpovsky and Taubin proposed a new class of nonlinear error detection codes in [34]. These codes minimize over all (n, k, r)-codes the maxima of the fraction of undetectable errors. Note that the (n, k, r)-codes are coding schemes where k is the bit length of the information, r is the bit length of the redundant check-sum associated with the information, and $n{=}k + r$ is the bit length of the codewords in these codes. The following definition from [34] provides the structural details of the binary version of these codes.

Definition 3.0.1. *[34] Let V be a binary linear (n, k) code with $n \leq 2k$ and check matrix $H = [P|I]$ with rank$(P) = n - k$. Then $C_V = \{(x, w)|x \in GF(2^k), w = (Px)^3 \in GF(2^r)\}$.*

An attacker essentially aims to inject a fault into the circuit and cause modifications on the original data values. In order to model this theoretically, the errors caused by the attacker are assumed to affect the original codewords in an additive nature. According to Definition 3.0.1, each data value in the circuit is represented as the pair (x, w), where x is the original data and w is the check-sum associated with it.

Hence, in order to be successful, the attacker will have to inject a pair of error vectors on both x and w. Note that a non-zero error e is masked for a codeword (x, w) when the erroneous message (i.e. fault injected codeword) is still a valid codeword in C_V. In other words, the error $e \in \{(e_x, e_w)|e_x \in GF(2^k), e_w \in GF(2^r)\}$ is masked for the message (x, w) when $(x + e_x, w + e_w) \in C_V$, i.e. iff

$$(P(x + e_x \bmod 2^k))^3 = (Px)^3 + e_w \bmod 2^r. \tag{3.1}$$

In this case, the error masking probability for a given non-zero error $e = (e_x, e_w)$ may be quantified as

$$Q(e) = \frac{|\{x|(x + e_x, w + e_w) \in C_V\}|}{|C_V|} \ . \tag{3.2}$$

This definition essentially provides a metric to quantify the performance of C_V. Note that C_V is called robust, if it minimizes maxima of $Q(e)$ over all possible non-zero errors. More specifically, robustness is defined as following in [44]:

Definition 3.0.2. *[44] The code C_V is defined to be* ***robust*** iff $Q(e) < 1$ *for all possible error vectors e or equivalently if its kernel only contains the zero vector K=$\{0\}$.*

According to this definition, the robustness property states that the attacker will not be able to compute a non-zero error vector $e \in \{(e_x, e_w)|e_x \in GF(2^k), e_w \in GF(2^r)\}$ which will be masked by all the valid codewords of C_V. For robust codes, such an error pattern does not exist. The following theorem from [34] quantifies the error detection performance of the proposed nonlinear code C_V.

Theorem 3.0.1. *[34] For C_V the set $E = \{e|Q(e) = 1\}$ of undetected errors is a $(k-r)$-dimensional subspace of V, from the remaining $2^n - 2^{k-r}$ errors, $2^{n-1} + 2^{k-1} - 2^{k-r}$ are detected with probability 1 and $2^{n-1} - 2^{k-1}$ errors are detected with probability $1 - 2^{-r+1}$.*

This theorem is constructed and proved under the following uniformity assumption:

Assumption 3.0.2. *All the codewords $x \in GF(2^k)$ in C_V are uniformly distributed and have the same probability of being observed.*

The main disadvantage of these codes however is that they do not preserve arithmetic and hence cannot be used to secure arithmetic structures against active fault injection attacks. To solve this problem, [22] proposed a new type of non-linear arithmetic codes called "robust quadratic codes". In what follows, we will describe the details of this coding scheme.

As the first step, the following definition from [22] defines the generic structure of the arithmetic single residue codes in detail.

Definition 3.0.3. *[22] Let C=$\{(x, w)|x \in \mathbb{Z}_{2^k}, w = f(x) \in \mathbb{F}_p\}$ be an arithmetic single residue code where the function $f : \mathbb{Z}_{2^k} \mapsto \mathbb{F}_p$ is used to compute the check-sum w. The check-sum w is computed with respect to the prime check modulus p which has a length of r=$\lceil \log_2 p \rceil$ bits.*

Similarly, the error $e \in \{(e_x, e_w)|e_x \in \mathbb{Z}_{2^k}, e_w \in \mathbb{Z}_{2^r}\}$ is masked for the message (x, w) when $(x + e_x, w + e_w) \in C$, i.e. iff

$$f((x + e_x \bmod 2^k)) = f(x) + e_w \bmod 2^r. \tag{3.3}$$

In this case, the robustness property states that the attacker will not be able to compute a non-zero error vector $e \in \{(e_x, e_w)|e_x \in \mathbb{Z}_{2^k}, e_w \in \mathbb{Z}_{2^r}\}$ which will be masked by all the valid codewords of C. As described before, such an error pattern does not exist for robust codes. Similarly, C is called ϵ-robust if it achieves an upper bound of $\max_{e \neq 0}(Q(e)) \leq \epsilon \cdot 2^{-r}$ on the error masking probability, where constant $\epsilon << 2^r$.

In [22], Gaubatz et al. mainly choose $f(x)$=$x^2 \bmod p$ for the check-sum function of Definition 3.0.3 and defines the "robust quadratic codes" as follows:

Definition 3.0.4. *[22] Let C be an arithmetic single residue code according to Definition 3.0.3 with $f(x)$=x^2 mod p. Then $C_p = \{(x, w)|x \in \mathbb{Z}_{2^k}, w = (x^2 \ mod \ p) \in \mathbb{F}_p\}$ where $2^k - p < \epsilon$ and r=k.*

The next step is to measure the performance of the defined code in terms of its error masking probability. Similar to the Assumption 3.0.2, Gaubatz et al. [22] make the following uniformity assumption in order to quantify the error detection performance of C_p:

Assumption 3.0.3. *The values of $x \in \mathbb{Z}_{2^k}$ in C_p are uniformly distributed, i.e. each value of x appears at the output with equal probability.*

Note that due to the arithmetic nature of the circuits for which this coding scheme is suitable for, this is a valid/reasonable assumption. Using this assumption, they provide the following bound on the error masking equation associated with C_p.

Theorem 3.0.4. *[22] For C_p, $max_{e\neq 0}(Q(e))$=$2^{-k}\cdot max(4, 2^k - p + 1)$.*

This theorem quantifies the error detection capability of the nonlinear code C_p. In other words, all non-zero error patterns can be detected with a probability lower bounded by $1-\max(4, 2^k - p + 1) \cdot 2^{-k}$. For more details on the proof of Theorem 3.0.4, the reader is referred to [22].

Contrary to linear codes, the probability of missing an error is largely data-dependent in these coding schemes. Consequently, an active adversary trying to induce an undetected error in the data would need to know the value of the data before the fault injection process in order to compute an undetectable error pattern. Furthermore, he needs to have sufficient spatial and temporal accuracy to inject these error pairs. Hence, an attacker will have practically no chance of inserting an undetectable error vector, unless he reads the target data vector first, then computes an appropriate error pattern, and precisely inserts the computed pattern with high spatial and temporal resolution. In a linear scheme, the success of a specific error vector

is independent from the data because any error vector which is also a valid codeword is masked in this case. Also note that the error detection probability provided by the nonlinear codes has a uniform lower bound. In other words, error detection probability does not dramatically decrease for any specific error pattern.

Possible applications of systematic non-linear error detection codes, such as the protection of finite state machines and Advanced Encryption Standard (AES) datapath are described in [46, 45, 33, 32, 44, 2].

Chapter 4

Generic Approach for Hardening State Machines Against Strong Adversaries

4.1 Motivation

Protecting the datapath of cryptographic algorithms is the focus of CED based protection schemes. However, even if the datapath is protected with the most secure scheme, the unprotected control unit may cause a serious vulnerability on the system. For instance, by injecting specifically chosen errors into the part of the IC that implements the control units, the adversary may bypass the encryption states in an FSM to conveniently gain access to secret information. Similarly, e.g. in a cryptographic authorization protocol the state which checks the validation of login information can be skipped with minimal effort. Thereby, the adversary can directly impersonate a valid user. Such attacks are indeed practical since states in an FSM are implemented using flip-flops, which can be easily attacked using bit-flips realized through fault injections.

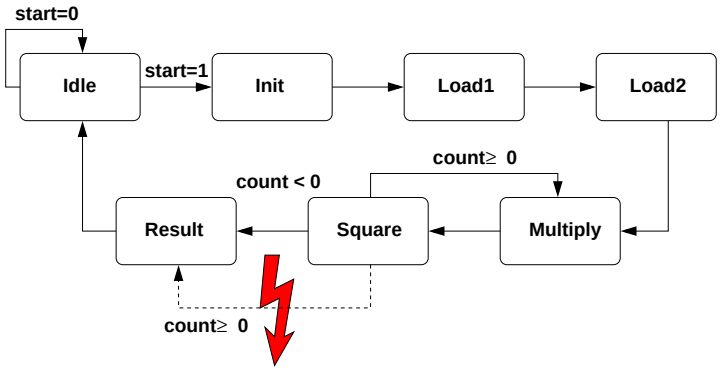

Figure 4.1: Fault injection example on the control unit of the Montgomery Ladder Algorithm with Point of Attack indicated by the dashed transition [24]

A literature review brings to light that other than a few scattered references, there is not much work done on control unit protection. [7, 39] propose simple single-bit error detection/correction schemes utilized in the areas of aerospace and communications. However, these schemes mainly target natural faults (for example due to radioactive radiation) and mostly single event upsets that occur on the control units. As a result, they are ineffective against an intelligent adversary with precise fault injection capabilities.

Another well-known solution which could be used is the application of modular redundancy, i.e. Triple Modular Redundancy (TMR) and Quadruple Modular Redundancy (QMR). Both of these schemes will provide an acceptable level of security against a weak attacker. However, when the advanced attacker defined in Chapter 2 is concerned, modular redundancy based schemes will offer **no** security at all. The reason is that the attacker can inject the same fault to all the copies of the FSM and these faults will be masked as all the copies will provide the same faulty output.

In cryptographic settings, Gaubatz et al. [24] proposed to apply linear error detection codes to prevent fault injection attacks against finite state machines (FSM). Figure 4.1 shows an example attack scenario from their paper on the FSM of the Montgomery Ladder Algorithm. They show that the attacker can recover the secret

exponent using this attack with mild effort. Even though using linear codes for FSM protection is a reasonable approach, it can only provide a limited level of robustness. Any fault injection that manifests itself with an error pattern, which is also a valid codeword in the utilized coding scheme, will be missed in a linear error detection scheme. Consequently, even though this method is sufficient to protect against weak adversaries, a more advanced attacker can still easily bypass this error detection mechanism and manipulate the device under attack.

Our Contribution: In this chapter, we propose a novel methodology to remove the vulnerability of control units against active fault attacks. As an initial step, we describe an important observation associated with the application of robust non-linear codes to FSMs. This observation states that due to the nonuniform behavior of the FSM variables (i.e. state registers, inputs, etc.), a direct implementation of non-linear codes for FSM security will not work. As a result, securing FSMs using non-linear codes is an important and difficult problem which requires additional effort. Our solution to this problem is built around two ideas. We first provide an arithmetic state machine construction for which the robustness of the applied non-linear error detection scheme can be easily measured and shown. This formulation also dramatically simplifies the predictor design. Next, we use randomized embedding to achieve unpredictability and uniformity. This two-pronged technique provides a generic solution applicable to any FSM. Consequently, the resulting FSMs will be robust even against advanced attackers with precise fault injection capabilities. At this point, it is also important to note that the control structures are very small when compared to the datapaths and occupy only a very small percentage of the whole hardware. Hence, the overall overhead of the proposed non-linear error detection technique will be minimal.

The remainder of this chapter is organized as follows: Proposed error detection technique and its robustness measure are presented in Section 4.2. Section 4.3 de-

scribes the proposed robust state machine construction. Section 4.4 discusses possible hardware implementations for the robust next-state logic. In Section 4.5, the proposed solution is compared with other existing FSM protection techniques. Finally, Section 4.6 provides the implementation results associated with our scheme.

4.2 The Error Detection Technique

In this section, we describe the proposed error detection technique to secure the next-state logic of an FSM. The main idea is to encode the variables of the FSM using a non-linear robust code (n, k, r) as in Definition 3.0.4, and to use the error detection capability of this coding scheme for fault detection. However, a direct implementation of this coding scheme for an FSM would cause a serious security problem. In the following, we will explain this problem and describe why it is inherently difficult to secure FSMs in the aforementioned adversarial setting.

4.2.1 Why Is It Difficult to Secure FSMs?

The specific error detection technique proposed by Gaubatz et al. in Definition 3.0.4 works under the uniformity assumption which states that all the codewords are observed with the same probability (Assumption 3.0.3). This is a valid assumption if the code is applied to an arithmetic structure (such as an adder or a multiplier) where the inputs, and hence the outputs tend to be uniformly distributed. However, when the FSMs are concerned, this assumption becomes invalid because

1. depending on the FSM, some states may be visited more than others while the device is in operation, and

2. the number of inputs and states in an FSM are usually relatively small over a large domain.

Due to this non-uniform behavior, the security level provided by Theorem 3.0.4 does not apply if this non-linear coding scheme is directly applied to an FSM. It is important to note that even though this naive version of the security scheme provides total robustness as defined in Definition 3.0.2, the error detection probability will be much smaller due to the non-uniformity. The reason can be explained both intuitively and mathematically.

Intuitively, the security of this particular non-linear coding scheme mainly depends on the assumption that the attacker cannot observe the output value of the device in the same clock cycle as fault injection. If the attacker knew the output value of the device before injecting a fault, he could easily choose the error pattern which would be successful because the details of the structure used in the state machine are public. Also note that the state values of an FSM are non-uniform and some states are visible most of the time. What the attacker can do at this point is to tap into the device and figure out the state that repeats with the highest frequency. This is essentially the same as knowing the output value of the next-state logic of this FSM. Consequently, the attacker can easily guess the error vector that will successfully cause faulty transitions.

Mathematically, the error detection probabilities of Theorem 3.0.4 will be much smaller because the number of valid codewords (fault free next-state values) determine the value of $|C_p|$ in the error masking probability

$$Q(e) = \frac{|\{x|(x + e_x, w + e_w) \in C_p\}|}{|C_p|} .$$

If the whole state space is being used uniformly by the valid next-state values (information carried by the code), then $|C_p|=2^k$ as in this theorem. However, in the non-uniform case of FSMs, the value of $|C_p|=t$, where t is the number of states in the FSM. This alone dramatically increases the error masking probability of the detection scheme when $t \ll 2^k$ (which is usually the case for reasonable security levels). In addition, if there are some states that are visited more than others, this will decrease

the effective value of $|C_p|$ even further, and hence will also cause an increase in the error masking probability.

In order to illustrate the problem in more detail, we will propose an attack on an FSM that is protected by the naive implementation of the error detection technique proposed in Definition 3.0.4. The objective of the attacker is to inject faults into the next-state logic of this FSM and cause state transition errors so that he can reveal information about the secret in the device. Assume that the FSM of Figure 4.1 is protected by a non-linear (n, k, r)-code that is constructed using Definition 3.0.4, where k=r is selected to be much greater than t=7 for higher security. Also assume that the states {Idle, Init, Load1, Load2, Multiply, Square, Result} are encoded as {S0, S1, S2, S3, S4, S5, S6}, respectively. Note that when the "start" signal is asserted, this FSM transitions as follows: {S0, S1, S2, S3, S4, S5, S4, S5, S4, S5, ... , S4, S5, S6, S0}. As a result, the state registers will most of the time have either the value S4 or S5. Note that the FSM and coding structures are public. Also, the advanced attacker can easily tap into the state registers and observe this non-uniformity. Consequently, all he has to do is to pick the e_{S4} or e_{S5} which are the specific error vectors that will be masked in this error detection scheme for S4 and S5, respectively. In this scenario, the attacker will be successful with a probability of one in transitioning the state machine into a wrong state without being detected.

4.2.2 Proposed Solution

Our solution to this problem is built around two innovative ideas:

- Arithmetic formulation of the next-state logic using Lagrange interpolation. This formulation, in comparison to regular Boolean next-state logic construction, dramatically simplifies the predictor design due to its arithmetic nature. Also, it is this arithmetic form of the next-state logic that makes it possible to precisely bound the error detection probability.

- Randomized embedding to solve the nonuniformity problem discussed before. In this method, each state value will have multiple images and this will provide unpredictability and uniformity.

More specifically, we first propose making state transitions work according to an arithmetic formula in a non-redundant field $\mathbb{F}_q$, i.e. $s'=f(s,i)$ where s' is the next-state value, i is the input, and s is the current-state value. In this setting, s', s, and $i \in \mathbb{F}_q$. The generation of this arithmetic formula is described in Section 4.3. Note that if the next-state logic is implemented as a set of Boolean functions, efficient predictor design and robustness quantification would be difficult.

However, this is not sufficient to solve the previously discussed non-uniformity problem because even though the state transitions are now working according to an arithmetic formula, the same non-uniform behavior is observed at the state registers. To solve this problem, we need unpredictability. In other words, the state machine variables must be practically unpredictable to the attacker. As a result, as the second step of the solution, we propose embedding the non-redundant field $\mathbb{F}_q$ into a larger redundant ring $\mathbb{R}=\mathbb{Z}_M$ using a transformation ϕ: $\mathbb{F}_q \mapsto \mathbb{R}$ (or ϕ: $\mathbb{R} \mapsto \mathbb{R}$) where ϕ is a homomorphism. Therefore, all the arithmetic operations defined for $\mathbb{F}_q$ can also be carried in $\mathbb{R}$. Utilization of *scaled embedding* for error detection in finite field arithmetic circuits is previously proposed by [22] for a similar purpose. In our case, randomized embedding is achieved by defining the ring modulus M and the scaling factor S with $M=S \times q$ where q is the prime defining the field $\mathbb{F}_q$. In this setting, we define the function ϕ as

$$\phi(s) = s + R \times q \pmod{M}$$

where R is a randomly chosen value from $\mathbb{Z}_S$. This scaling effectively partitions the ring $\mathbb{R}$ into co-sets, of which only one contains the non-redundant codewords. As a result of this scaling, each state and input value now has S images. In other words,

the same state and input will now be represented by S different values in the ring $\mathbb{R}$ depending on the value of the random R. This will increase the uniformity of the state values observed at the output of the next-state logic, and essentially increase the error detection capability of the non-linear coding technique we propose in this chapter. Note that when the state value is reduced using q, we obtain the non-redundant value of the state. We now formally define a randomized robust code by merging this embedding with the robust code definition introduced by Gaubatz et al. [22] in Definition 3.0.4 as follows.

Definition 4.2.1. *We define the coset randomized robust code* (n, k, r) *as* $C = \{(x, w)|x = y + R \times q \pmod{M}, \forall x \text{ and } \forall y \in \mathbb{Z}_M, w = x^2 \pmod{p} \in \mathbb{F}_p, R \in \mathbb{Z}_S\}$ *where* $r{=}k$, $k \geq max(\lceil \log_2 M \rceil, \lceil \log_2 p \rceil)$.

In addition, two important points for picking the scaling factor are

- to select the ring $\mathbb{R}$ large enough to provide sufficient redundancy.
- to pick an M which enables efficient reduction.

Since we are free to choose the state encoding for the robust state machine design, we can pick a suitable (q, S) pair which will satisfy the required level of security while producing an ideal M for easy reduction.

After the randomization of the next-state function f is achieved, we can use individually robust arithmetic circuits such as multipliers, adders, etc. to build the proposed robust FSM. To achieve this, we utilize the non-linear code explained in Definition 4.2.1 to encode the state variables and inputs. Next, we use the individually robust arithmetic circuits that will work in the ring $\mathbb{R}$ to implement the next-state function f. Robust adder and multiplier implementation examples that work under the utilized code are provided in [22].

4.2.3 Security Proof

In this section, we prove the robustness of the proposed scheme and give a lower bound on the error detection probability. First of all, we will encode each of the next-state logic variables using the non-linear (n, k, r) code given in Definition 4.2.1, where the value of k is determined by the size of the ring $\mathbb{R}$ and prime p. More specifically, $k \geq \max(\lceil \log_2 M \rceil, \lceil \log_2 p \rceil)$. Also note that the value of q is bounded by the number of states t in the FSM as $q = \lceil \log_2 t \rceil$.

For error detection, we need redundancy. We define the following error check function on a variable $x \in \mathbb{Z}_M$ to obtain a non-linear error check-sum

$$w = h(x) = x^2 \pmod{p} \in \mathbb{F}_p. \tag{4.1}$$

Consequently, the states and inputs are decoded as $(s, h(s))$ and $(i, h(i))$ respectively. From a practical viewpoint we have added r check digits to the state value as an error check redundancy.

In the following theorem, we establish the error detection probability of our scheme. In this theorem, we also address the nonuniform distribution observed at the output of the next-state logic.

Theorem 4.2.1. *For the nonlinear code C that is defined as in Definition 4.2.1, the error masking probability will be upper bounded by $S^{-1} \cdot max(4, 2^k - p + 1)$, where $S > max(4, 2^k - p + 1)$.*

Proof. In order to compute the error masking probability of their scheme, Gaubatz et al. [22] count the number of x values that satisfy the numerator of Equation 3.2 for each e. To achieve this, they use the following error masking equation:

$$(x + e_x \bmod 2^k)^2 \bmod p = (x^2 \bmod p) + e_w \bmod 2^k. \tag{4.2}$$

In their paper, they prove that the number of solutions to this equation is upper bounded by $\max(4, 2^k - p + 1)$. Finally, in order to bound the error masking prob-

ability, this number is normalized with the number of valid codewords, 2^k in their case.

In order to quantify the error detection capability of our code, we also need to bound the value of $Q(e)$ for all e. We achieve this in two steps. First of all, we will bound the number of solutions to the error masking equation for this new code (numerator), and as the second step, we compute the number of valid codewords (denominator).

Note that the code we define here as in Definition 4.2.1 will have the exact same error masking equation. However, also note that the domain of the x values will be different. In our case, $x \in \mathbb{Z}_M$, instead of $x \in \mathbb{Z}_{2^k}$. Since $M < 2^k$, the number of solutions to the error masking equation cannot be larger than $\max(4, 2^k - p + 1)$, even in the worst case. As a result, the upper bound of the numerator to our error masking probability will be $\max(4, 2^k - p + 1)$.

However, when the uniformity assumption is not valid, which is the case for FSMs, the normalization constant needs to be modified in order to address the nonuniformity. Note that for an FSM with nonuniform distribution, in the worst case, one of the states will be visible at the next-state logic output most of the time. In this case, we can assume uniformity at the state register for S different values due to the random mask R (e.g. each state value will have S different images due to the randomized embedding). Again in the worst case, for a specific error pair (e_x, e_w), $\max(4, 2^k - p + 1)$ solutions can be in the check-sum space. As a result, the error masking probability for this case will be $S^{-1} \cdot \max(4, 2^k - p + 1)$. □

Example 4.2.1. *Consider the FSM of Figure 4.1. As discussed before, this FSM bounces between "Square" and "Multiply" states most of the time. If we select* $\mathbb{F}_q = \mathbb{F}_{11}$ *and* $S = 390451572$*, then* $M = 4294967292$ *and we will need a* $(64, 32, 32)$*-code with* $p = 4294967291$*. In this case, injected errors are detected with a probability of at least* $1 - S^{-1} \cdot max(4, 2^k - p + 1) = 1 - (2^{32} - 4294967291 + 1)/(2 \times 390451572) \simeq 1 - 2^{-27}$*. The*

denominator in this case becomes $2 \times S$ *because the "Square" and "Multiply" states and their images will be uniformly distributed.*

4.3 Proposed Next-State Logic Construction

In this section, we describe the proposed next-state logic construction which is a generic solution applicable to any FSM. Next, the idea is clarified by an example. We assume that we are given a state diagram, and we are free to choose the state assignments. The key is to make the state transitions work according to an algebraic formula which permits us to establish its robustness. There is no reason why one may not simply construct Boolean expressions to realize the state machine in the traditional way. However, then we would need to count the number of solutions that satisfy the error detection equation to determine the degree of robustness. This is highly inconvenient as the counting process would have to be repeated for each new state machine design.

4.3.1 Generic Technique for Next-State Logic Construction

In this section we outline a general next-state logic construction approach that assumes an arbitrary state machine that is given without any state assignments. Note that the choice of state assignment does not have a direct effect on the performance of the proposed scheme. The important parameter to select is the field modulus q where the non-redundant FSM is defined. As a result, as long as we pick the most appropriate q that will provide the highest security and easy reduction for modulus M, the choice of state assignment is not an issue.

Next, we will have a state machine in the form of a state transition table which

gives us the next-state value according to the current state and input:

$$\begin{aligned} \delta(s_0, i_0) &= s'_0 \\ \delta(s_1, i_1) &= s'_1 \\ \vdots &= \vdots \\ \delta(s_{n-1}, i_{n-1}) &= s'_{n-1} \end{aligned}$$

where s'_i represents the next-state, s_i represents the current state, and the i_i represents the inputs of the state machine.

Then we may capture the transitions of the state machine using an algebraic function $s' = f(s, i)$ which explicitly computes the transitioned state from the current state and input values. To construct $f(s, i)$ we view the given set of transitions as evaluations of the function $f(s, i)$ at the points $(s_0, i_0, s'_0), (s_1, i_1, s'_1), \ldots, (s_{n-1}, i_{n-1}, s'_{n-1})$. Hence, given n evaluations we may express the function as a bi-variate polynomial of degree n.

Remember for a univariate function $f(x)$ given n evaluations $(x_0, f(x_0))$, $(x_1, f(x_1))$, $(x_{n-1}, f(x_{n-1}))$, the Lagrange interpolation can be constructed as follows

$$f(x) = \sum_{t=0}^{n} f(x_t) \prod_{j=0, j\neq t}^{n} \frac{x - x_j}{x_t - x_j}. \tag{4.3}$$

This formula basically generates a univariate polynomial, e.g. a curve in two dimensions, that passes through all the given points. However, FSM structures are bi-variate systems. This means that we need to figure out a way to compute the three dimensional surface that passes through all the given three dimensional points. In other words, for a two input system we need to derive the bi-variate polynomial $s' = f(s, i)$. The solution is to apply the Lagrange formulation in two levels. More specifically, we first group the provided points according to their current state values. In this case, we will have t different groups, where t is determined by the number of states defined in our FSM diagram. Then, for each current state value s_j, we will have

b_j number of (i_j, s'_j) pairs, where b_j is the number of transitions from each specific state and the (i_j, s'_j) pairs are the input/next-state pairs associated with this current state. An example of this grouping mechanism is shown in Table 4.1. First of all, note that there are t=7 total groups in this table because this state machine has seven states. Also note that the "Idle" and "Square" states have two input/next-state pairs (for "Idle": (000,000), (001,001) and for "Square": (010,100), (011,110)) while all the other states have a single input/next-state pair. We have two pairs for the "Idle" and "Square" states because these states have b=2 possible transitions from them. Next, for each current-state group where there are more than one input/next-state pair (i_j, s'_j), the first level Lagrange interpolation is conducted, i.e. Equation 4.3 is applied. This is shown in Table 4.2. After this first level interpolation, there will be t number of (s_j, s'_j) pairs, where s_j is the current state and the $s'_j = g(i)$ is the next-state which is a $(b_j - 1)^{th}$ degree univariate polynomial defined as a function of the input i. This case is illustrated in the first two columns of Table 4.3. Note that for the "Idle" and "Square" states, next-state value s' is a function of the input i. For these states, b=2 and hence the next-state is a $(b-1)^{th}$=1^{st} degree polynomial of i. Then, we treat these (s_j, s'_j) pairs as the given points for another Lagrange interpolation. The Lagrange formula provided in Equation 4.3 is again applied to these points as the second level of our construction. The result is a bi-variate polynomial with at most n=$(max(b_j) \times t)$ terms and is of the form

$$s'_j = f(s_j, i_j) = a_{n-1}(i)^{max(b_j)-1}(s)^{t-1} + a_{n-2}(i)^{max(b_j)-1}(s)^{t-2} + \ldots + a_2 s + a_1(i) + a_0.$$

In this equation, i variable will have at most $max(b_j)$-1^{st} degree coming from the first level Lagrange and s variable will have at most $t - 1^{st}$ degree coming from the second level Lagrange. max(b_j) in this equation is the number of transitions from the state with the maximum number of transitions going out in the FSM diagram. Note that we will have at most n=$(max(b_j) \times t)$ terms because the result of the first level Lagrange can have at most $max(b_j)$-1^{st} degree and the result of the second level

Lagrange can have at most $t - 1^{st}$ degree.

4.3.2 Case Study Utilizing the Proposed Construction

Table 4.1: Grouped state assignment table

	Input(i)	**CS**(s)	**NS**(s')
Idle	000	000	000
	001	000	001
Init	000	001	010
Load1	000	010	011
Load2	000	011	100
Multiply	000	100	101
Square	010	101	100
	011	101	110
Result	000	110	000

In this section, we present the application of the proposed technique of Section 4.3.1 to an example FSM. This example will illustrate the generation of the algebraic function $s' = f(s, i)$ which explicitly computes the transitioned state from the current state and input values for the FSM shown in Figure 4.1.

As an example binary encoding is utilized to encode this FSM. We encode the input (i) and state variables (s,s') in $\mathbb{F}_q$ with the prime q=7 (smallest prime that includes all the input and state values). As a result, it is important to note that all the numbers and variables we discuss in this part are actually elements in the field $\mathbb{F}_q$, and all the operations are conducted using modulo q.

The first step of the Lagrange method is to group and tabulate the FSM diagram and to create Table 4.1. The state transitions are grouped according to the current state value(s), and the first level Lagrange is realized using the (i, s') pairs in each

Table 4.2: First level Lagrange

For CS=000			
i	s'	Lagrange Coefficients	Interpolated Polynomial
000	000	$l_0(i) = \frac{i-001}{000-001}$	i
001	001	$l_1(i) = \frac{i-000}{001-000}$	

For CS=101			
i	s'	Lagrange Coefficients	Interpolated Polynomial
010	100	$l_0(i) = \frac{i-011}{010-011}$	$2i$
011	110	$l_1(i) = \frac{i-010}{011-010}$	

group. The values of i, s', and Lagrange coefficients associated with each current state value (s) is shown in Table 4.2. When these points are interpolated using the indicated coefficients, we get the shown interpolated polynomials in the last columns.

Next, using the results of the first level Lagrange interpolation, we create the second level Lagrange table as shown in Table 4.3. As can be observed from this table, each current state value (s) and its corresponding first level approximation polynomial (which is a function of the input variable i) form the coordinates of the points to be interpolated. The calculated Lagrange coefficient for this level are also indicated in the last column of this table. Next, we insert these coefficient values into the following Lagrange formulation

$$s' = f(s,i) = l_0(s)(i)+l_1(s)(2)+l_2(s)(3)+l_3(s)(4)+l_4(s)(5)+l_5(s)(2i)+l_6(s)(0) \quad (4.4)$$

and we get the resulting bi-variate polynomial in the open form as

$$s' = f(s,i) = (4i)s^6+(2+4i)s^5+(3+6i)s^4+(1+2i)s^3+(5+3i)s^2+(5+i)s+i. \quad (4.5)$$

This polynomial represents the next-state s' as a function of the input and current state variables i and s, respectively. It is this algebraic formulation of the next-state

variable that makes it possible to easily analyze and determine the robustness of the implemented non-linear coding scheme.

Table 4.3: Second level Lagrange

s	s'	Lagrange Coefficients l_i	
		In Ratio Form	In Extended Form
000	i	$\frac{(s-1)(s-2)(s-3)(s-4)(s-5)(s-6)}{(0-1)(0-2)(0-3)(0-4)(0-5)(0-6)}$	$6s^6+1$
001	010	$\frac{(s-0)(s-2)(s-3)(s-4)(s-5)(s-6)}{(1-0)(1-2)(1-3)(1-4)(1-5)(1-6)}$	$6s^6+6s^5+6s^4+6s^3+6s^2+6s$
010	011	$\frac{(s-0)(s-1)(s-3)(s-4)(s-5)(s-6)}{(2-0)(2-1)(2-3)(2-4)(2-5)(2-6)}$	$6s^6+5s^5+4s^4+6s^3+5s^2+3s$
011	100	$\frac{(s-0)(s-1)(s-2)(s-4)(s-5)(s-6)}{(3-0)(3-1)(3-2)(3-4)(3-5)(3-6)}$	$6s^6+4s^5+5s^4+s^3+3s^2+2s$
100	101	$\frac{(s-0)(s-1)(s-2)(s-3)(s-5)(s-6)}{(4-0)(4-1)(4-2)(4-3)(4-5)(4-6)}$	$6s^6+3s^5+5s^4+6s^3+3s^2+5s$
101	$2i$	$\frac{(s-0)(s-1)(s-2)(s-3)(s-4)(s-6)}{(5-0)(5-1)(5-2)(5-3)(5-4)(5-6)}$	$6s^6+2s^5+3s^4+s^3+5s^2+4s$
110	000	$\frac{(s-0)(s-1)(s-2)(s-3)(s-4)(s-5)}{(6-0)(6-1)(6-2)(6-3)(6-4)(6-5)}$	$6s^6+s^5+6s^4+s^3+6s^2+s$

Also, note that all the operations conducted to compute the next state polynomial of Equation 4.5 are over modulo q. Once this polynomial is computed, the next step is to encode the input and current state values using the coset randomized code of Definition 4.2.1. In this case, input will be $(i, |i^2|_p)$ and the current state will be $(s, |s^2|_p)$ where i, $s \in Z_M$. Next, using the computed function $f(s, i)$, we compute $(s', |(s')^2|_p)$. In this computation, all the main datapath operations are conducted over modulo M. Also note that each arithmetic unit (both in function f and randomizer unit) is a robust one which implements nonlinear error detection individually. For example, all the multipliers in function f and randomizer unit compute the output and its redundant checksum using the inputs and their redundant checksums. These individual components can detect an injected fault and signal an error signal. Next, using the randomizer unit, we compute $z{=}s' + R \times q \pmod{M}$. These operations are conducted using robust arithmetic units and are over modulo M as well. The resulting randomized next state value is then fed as the current state value into the

next state logic in the next iteration. Note that we do not need to recover s' for the following next state computation. Since the function $f(s,i)$ and randomizer unit work modulo M, the next state value will always be a randomized image of the correct next state value. The naked form of the next state value ($\in F_q$) can always be computed by reducing the randomized images modulo q.

With the proposed solution, we essentially push the security threat on a system from FSMs to the peripherals which require the naked form of the state and output values. As a result, the naked form of the state and output values that drive the peripherals in the system become vulnerable points for possible attacks. One possible solution to this problem is to utilize the same coding structure in all components of the system. In this case, we need secure decoder circuits that would generate the appropriate output control signals from the redundant (randomized) form of these signals. However, design of such components is beyond the scope of this thesis and can be listed as a potential future work.

4.4 Hardware Realizations of the Next-State Logic

In this section, we will outline a technique for implementing the proposed next-state logic construction. A direct arithmetic implementation of Equation 4.5 can be found in Figure 4.2. As can be observed from this figure, the next-state is algebraically formulated as a function of the input and current-state. The randomized embedding is realized using the randomizer block in the figure. For reference, the operators $\bigotimes_M$, $\bigoplus_M$, and x_M^2 in this figure stand for multiplication, addition, and squaring modulo M, respectively. All of these operations (including the coefficients at the top) are carried-out in the redundant ring $\mathbb{R}{=}\mathbb{Z}_M$ using individually robust arithmetic units. Note that since we are using the nonlinear code proposed in Definition 4.2.1 (whose robustness is proved in Theorem 4.2.1) to encode the next-state logic variables, the

security of this system will also be bounded by the security of the proposed code. In other words, each individual robust unit of this figure provides the security level proposed by Theorem 4.2.1. The reader is referred to [22] for examples of robust arithmetic units (e.g. adder, multiplier, etc.) that are used in this figure.

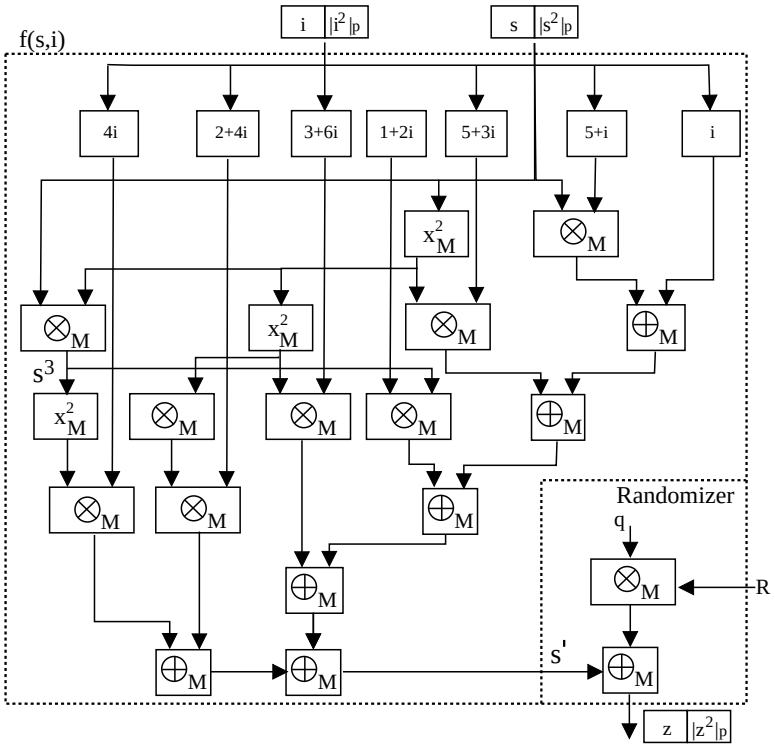

Figure 4.2: Arithmetic parallel hardware implementation of the next-state logic

We also propose a more efficient approach that can be followed in order to optimize this circuit. The idea is to add time-redundancy and reuse the expensive hardware modules (e.g. the multiplier and the adder) in a serial manner. This can be achieved by rearranging the next-state polynomial of Equation 4.5 using Horner's method. In this case, the next-state equation becomes

$$s' = ((((((4i)s + (2 + 4i))s + (3 + 6i))s + (1 + 2i))s + (5 + 3i))s + (5 + i))s + i. \quad (4.6)$$

This implementation is demonstrated in Figure 4.3. In this case, the function f is serialized and the randomization block is the same as the parallel implementation. As

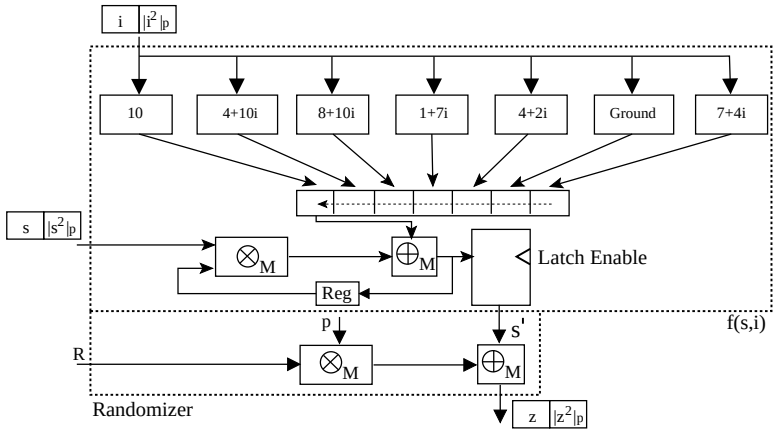

Figure 4.3: Time redundant (serial) arithmetic hardware implementation of the next-state logic

can be observed from this figure, only 2 robust multipliers combined with 7 adders, 5 robust constant multiplications, a controlled latch, and a shift register is sufficient to implement the same functionality as Figure 4.2.

4.5 Comparison with other FSM security schemes

In this section, we will compare our scheme with other error detection schemes that exist on the literature. One of the most common error detection techniques is the application of modular redundancy, i.e. Triple Modular Redundancy (TMR) and Quadruple Modular Redundancy (QMR). In TMR, three copies of the non-redundant FSM are run in parallel and a majority voting decides if the operation of the FSM is fault free or not. QMR works exactly the same way but with four copies. Under our adversarial fault model, TMR and QMR will be broken with a probability of one. The reason is that the attacker can inject the same fault to all the copies of the FSM and these faults will be masked as all the copies will provide the same faulty output.

Under our fault model, protection scheme that uses linear codes [24] will also be broken with a probability of one. An error vector which is also a valid codeword in

the utilized linear code will be missed because the sum of these two codewords is another valid codeword in this scheme.

The efficient PUF-based error detection scheme of [26], which is discussed in Chapter 6, works only for a specific class of FSMs and hence is not a generic solution that can be applied to any FSM. However, it is important to note that this solution provides error detection on both physical and logical levels.

Finally, [2] proposes a hybrid method in which non-linear codes are interleaved with a MUX-based predictor design. This is a generic scheme that will work on all FSMs. The bound of the error detection probability associated with this scheme is $1 - 9 \times 2^{-k}$, where k is the length of the information portion of the applied code. The disadvantage of this scheme is that it needs secure storage on the device for the round randomization masks that need to be transferred to the next round.

The scheme we propose is a generic solution that can be applied to any FSM. The minimum error detection probability is bounded by $1 - S^{-1} \cdot \max(4, 2^k - p + 1)$. Note that $2^k - M < \epsilon$ and q is the smallest prime that can include all the different next-state logic variables. As a result, $(2^k - \lfloor \log_2 S \rfloor) < \epsilon$. This means that the error detection probability of our scheme is on the order of the security level provided by [2]. The advantage of our scheme is that we do not need a secure location on the chip for the randomization mask that is used in each cycle because the operation of each cycle is independent from previous and next clock cycles. In addition, the proposed technique enables the utilization of robust arithmetic components for secure FSM design. This is an important point because security characteristics/performance of arithmetic structures is well studied in the literature. The algebraic formulation of the next-state logic also simplifies the predictor design in our solution.

4.6 Implementation Results

In order to compare the performance of our scheme with previously proposed FSM protection techniques, we implemented the following solutions for the FSM of the powering ladder algorithm that is shown in Figure 4.1:

1. regular FSM with no error detection or any type of redundancy.
2. FSM protected with linear error detection codes.
3. triple modular redundancy (TMR).
4. quadruple modular redundancy (QMR).
5. FSM protected with $(10, 5, 5)$, $(20, 10, 10)$, $(30, 15, 15)$, $(40, 20, 20)$, $(50, 25, 25)$, $(60, 30, 30)$ coset randomized robust codes (our solution).

We implemented these techniques using VHDL and synthesized them using the tcb013lvhptc and dw (design ware) libraries of the Synopsys design tool. We report the gate count and time delay results in Table 4.4. This table also includes the minimum error detection capability of each technique against the adversary we assume in our fault model.

First of all, note that even though they cause reasonable area overhead, the linear error detection codes, TMR, and QMR do not provide any security against the advanced attacker we assume in this thesis. On the other hand, our technique causes a large area overhead and time delay. However, we claim that this is a price which needs to be paid to secure FSMs against such advanced attackers. First of all, time delay caused by our scheme does not have a big impact on the circuit performance because FSMs are usually not in the critical path of a circuit. Critical path usually goes through the datapath which includes various arithmetic operations.

Furthermore, in our implementation of the coset randomized robust codes, we used the built in design ware multipliers and dividers to create a generic solution that can

Table 4.4: Hardware implementation results for different FSM protection schemes.

	Gate count	Time delay (ns)	Min. Error Detection Prob.
Non-red FSM	80	0.64	0
Linear	260	≈ 0.64	0
TMR	260	≈ 0.64	0
QMR	342	≈ 0.64	0
(10, 5, 5)	8305	29.99	$1 - 2^{-2}$
(20, 10, 10)	27779	88.69	$1 - 2^{-6}$
(30, 15, 15)	57310	196.42	$1 - 2^{-9}$
(40, 20, 20)	96096	294.57	$1 - 2^{-16}$
(50, 25, 25)	146038	444.04	$1 - 2^{-18}$
(60, 30, 30)	206676	618.4	$1 - 2^{-23}$

be applied to different FSMs and code sizes. We believe that for a specific FSM and code size, there will be room for gate count optimizations up to 30%. In addition, we argue that while measuring the performance of our scheme, analysing the gate count individually may be misleading. As a result, we propose a new performance measure which is called error masking probability per unit area. For the six different coset randomized robust codes we show in Table 4.4, we computed the error masking probability per unit area and we plotted the result as a function of the code size k in Figure 4.4. We plotted the synthesized results and added the interpolated version of this data as well. As can be observed in this figure, for larger code sizes, the error masking probability per unit area decreases. Note that this is a valid justification of the area overhead caused by our scheme.

It is also important to note that even though our protection scheme is not appropriate for low power cryptographic hardware, it can be applied to circuits that include large and parallel arithmetic units in the datapath. Low power cryptographic

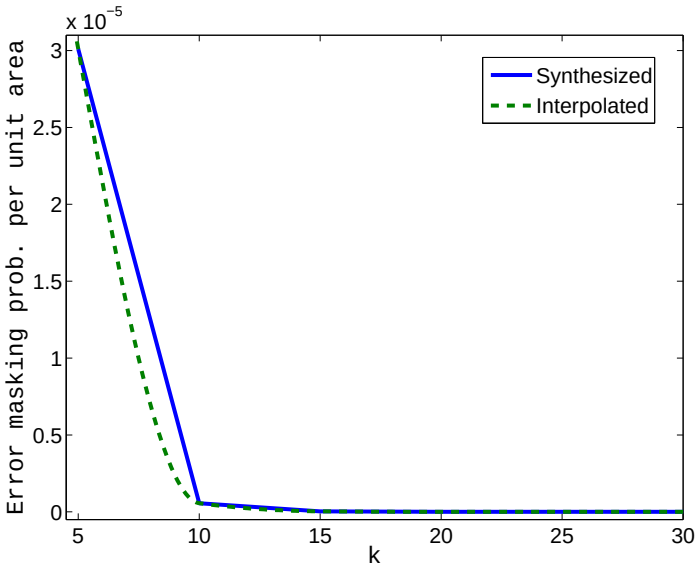

Figure 4.4: Error masking probability per unit area

units have relatively large FSMs due to their serial nature. In this case, large area overhead of our FSM protection scheme might affect the circuit area and performance in an unacceptable way. However, we argue that the overhead of our technique will be reasonable for cryptographic hardware with parallel arithmetic units (i.e. adders, multipliers, etc.) in the datapath. This kind of circuits have relatively small FSMs and hence the overhead of our protection scheme becomes minimal.

In order to clarify this idea, we investigated the effect of our FSM protection scheme on the overall circuit area. First of all, we analyzed four different Montgomery multipliers (MM) with different parameters. MM(x,y,z) indicates a Montgomery multiplier with different parameters where x is the size of the operands, y is the number of pipeline stages, and z is the word size. Table 4.5 show the breakdown of the gate count into FSM and datapath units for four different MMs. Note that as the operand and word sizes increase, the parallel nature of the circuit also increases. As a result, the ratio of the FSM to the whole design gets smaller.

Table 4.5: Breakdown of the gate count for different Montgomery multipliers.

	Non-red FSM	Non-red Datapath	FSM ratio (%)
MM(256,2,64)	3843	124750	3.08
MM(256,1,128)	3812	201588	1.89
MM(512,1,128)	3841	217168	1.77
MM(512,1,256)	7307	714557	1.02

Assume that we use the $(10, 5, 5)$ robust code to protect the FSM of these circuits. Table 4.4 indicates that this code increases the non-redundant FSM size approximately 100 times. However, remember that this number is for the generic implementation of this scheme and does not include all the possible optimization methods. If additional optimization techniques are applied, this ratio could be dropped to approximately 70. In addition, assume that we protect the datapath of these circuits with non-linear robust codes as well. Non-linear robust codes usually increase the datapath area 3 to 4 times [44]. We assume that the datapath size will increase by a factor of 3.5 in average. When we apply these numbers to the gate counts of the non-redundant designs of Table 4.5, we get the results shown in Table 4.6. Note that the overhead caused by our scheme gets smaller for circuits with large and parallel datapaths. For example, our FSM protection scheme causes an area overhead of approximately 16% for the MM(512,1,256) case.

Table 4.6: Overhead caused by our protection scheme for different MMs.

	Robust FSM	Robust Datapath	FSM ratio (%)	Overhead (%)
MM(256,2,64)	269010	436625	61.61	37.58
MM(256,1,128)	266882	705560	37.83	27.05
MM(512,1,128)	268870	760090	35.37	25.76
MM(512,1,256)	511532	2500949	20.45	16.74

4.7 Summary

In this chapter, we propose utilizing non-linear error detection codes for the control unit design of cryptographic hardware. However, the nonuniform behavior of the FSMs make it difficult to apply nonlinear codes and provide high security. Our solution to this problem includes two main steps. First, we apply Lagrange interpolation to model the state transitions using an arithmetic formula. This not only simplifies predictor design, but also makes it easier to generically quantify the security level of the applied scheme. Second, we use randomized embedding to achieve uniformity and unpredictability. Note that without the arithmetic formulation (through Lagrange interpolation) of the next-state logic, robust arithmetic units proposed by [22] could not be utilized in this scheme. In addition, note that we could also use Boolean expressions to realize the state machine in the traditional way. However, then we would need to count the number of solutions that satisfy the error detection equation to determine the degree of robustness. This is highly inconvenient as the counting process would have to be repeated for each new state machine design.

To be successful against this type of a protection, the attacker must be able to precisely read the existing data in control unit hardware, and then must select the most appropriate error vector that has the highest probability of succeeding. This, of course, is much more difficult to accomplish for an attacker and therefore this scheme is secure even against very advanced adversaries.

As future work, it may be possible to investigate the application of nonlinear codes to Boolean next-state logic functions. To achieve this, an efficient Boolean predictor design and a generic security quantification methodology must be developed.

Chapter 5

Multiplexer-based Non-linear Error Detection for Finite State Machines

5.1 Motivation

In Chapter 4, we discussed the security problems associated with Finite State Machines (FSMs) in an adversarial setting. We argued that the FSM security is a neglected area and as a solution to this problem, we proposed a generic FSM protection technique that is based on non-linear codes and arithmetic formulation of the next-state logic.

Our Contribution: In this chapter, we propose a different generic non-linear error detection technique for FSMs. This is a multiplexer-based technique which is inspired from the efficient FSM implementations. In this case, instead of formulating the next-state logic in an arithmetic fashion, we utilize the well-known (and used) multiplexers for both the non-redundant datapath and the predictor units. The main

The work presented in this chapter is a joint work with Ghaith Hammouri.

advantage of this scheme is its simplicity. Remember that protecting FSMs with systematic non-linear codes is a difficult problem due to the non-uniform characteristics of FSMs. As a solution to this problem, in this chapter we propose using randomized masking. This makes it possible to apply the nonlinear codes to FSMs and inherit their high error detection capabilities. In our security analysis, we show that any injected fault will be detected with probability exponentially close to 1.

The remainder of this chapter is organized as follows: The proposed error detection technique is described in Section 5.2. The security and robustness measure of the proposed scheme are presented in Section 5.3. In Section 5.4, hardware scaling results are provided. Finally, in Section 5.5, the proposed scheme is compared with other FSM protection schemes from a security perspective.

5.2 The Error Detection Technique

Systematic nonlinear codes described in Chapter 3 provide robustness and error detection uniformity which are crucial when error detection is concerned. As a result, the proposed error detection technique in this chapter inherits the main structure of these codes to protect next-state logics of FSMs. Note that the security level provided by the nonlinear codes is sufficient for arithmetic circuits even against strong adversaries such as the one discussed in Chapter 2. However, the structure of these codes needs to be modified to some extent to make them applicable to FSMs.

Assume that we want to apply the nonlinear (n, k, r)-code of Definition 3.0.1 to protect a specific FSM. When FSMs are concerned, the number of valid codewords is limited by the number of states in the FSM, which is usually a relatively small number. In addition to this, some states of the FSMs can be visited more than others. As a result, FSMs show a non-uniform behavior over a larger domain that is defined by the values of k and r of the applied (n, k, r)-code. Due to this non-uniform

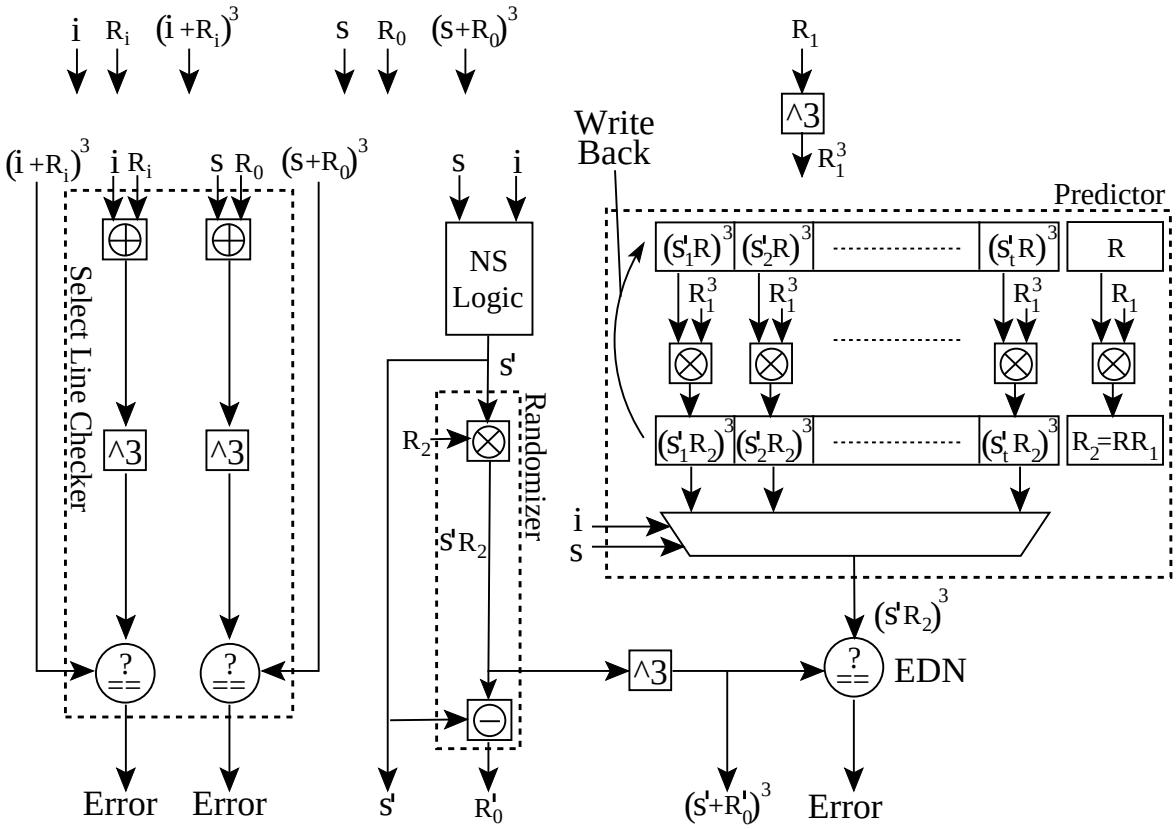

Figure 5.1: Proposed error detection technique

characteristic of the FSMs, applying the non-linear (n, k, r)-code becomes a difficult problem. In this case, the security level provided by this method cannot be quantified as in Theorem 3.0.1 because FSMs do not satisfy Assumption 3.0.2. Consequently, to inherit the useful error detection characteristics of these codes, we need to guarantee that the state register value will have a distribution that is close-to-uniform.

As a solution to this problem, we now formally define a randomized robust code by merging a randomized masking process with the robust code definition introduced by Karpovsky et al. [34] in Definition 3.0.4 as follows.

Definition 5.2.1. *Let V be a binary linear (n, k) code with $n = 2k$. We define the robust code with randomized masking as $C_V = \{(x, R, w) | x \in GF(2^k), R \in GF(2^k), w =$*

$(x + R)^3 \in GF(2^k)\}$.

In this code, the masking is achieved by the random string R. The effect of masking is to essentially remove the non-uniform characteristic of the FSM. In this case, the error detection probability can be quantified as in Theorem 3.0.1 because Assumption 3.0.2 is satisfied with the utilization of randomized masking. Note that because we are working in a finite field the above randomization can be realized by an addition as in the definition or by a multiplication.

The proposed solution which utilizes the defined randomized robust code is shown in Figure 5.1. There are five main building blocks in this solution and these blocks will be described in more detail shortly. Note that all the variables in this figure are elements of $GF(2^k)$ and all the operations are conducted over $GF(2^k)$. At a high level, the essential idea of this scheme is to provide fault free operations on the non-redundant next-state logic unit. In order to achieve this, we propose having two parallel computational paths. The first path computes the next-state value (using the non-redundant next-state logic) and randomizes the result (through the randomizer block) using the random value R_2. The second path (the predictor unit) essentially selects the appropriate check-sum associated with the randomized version of the next-state value. More specifically, using a multiplexer circuit which is controlled by the input and current state, the predictor block selects the correct randomized check-sum from a pool that is full of all possible randomized next-state check-sums. Results of these two paths are then compared (using the EDN block). A mismatch at the output potentially indicates an adversarial fault injection and is handled appropriately. This "two parallel paths" approach is usually costly when integrated with nonlinear codes. Therefore, we use a multiplexer structure which will help improve the efficiency of the circuit. Also note that when multiplexers are concerned, attacks on the select lines could lead to potential vulnerabilities. As a result, the proposed solution implements separate comparisons (in the select line checker block) to ensure that no errors were

injected into the i and s values which are the select lines of the multiplexer in the predictor unit. In what follows, we provide a more detailed description for each building block of this scheme.

- **NS Logic**: This is basically the non-redundant next-state logic of any FSM. Using the input (i) and current-state (s), this block computes the appropriate next-state value (s'). Note that i, s, and s' are padded with zeros so that they will be elements of $GF(2^k)$. In other words, i, s, and $s' \in GF(2^k)$. However, this does not affect the logic that implements the next-state function. This block uses the well-known next-state logic computation methods.

- **Randomizer**: This block applies the randomized masking on the next-state value (s'). In this case, randomization is achieved by multiplying the computed next-state value with R_2 which is the main randomizer that is used in the current cycle. Note that R_2 is computed in the predictor unit and is the product of the main randomizer from the previous cycle (i.e. R) and the input R_1. R_1 is assumed to be generated by an RNG on the device and is refreshed in every clock cycle. The refreshness of R_1 is a very crucial property that needs to be satisfied in our scheme. Otherwise, the attacker can iteratively (i.e. he can first observe the existing values at the output and then pick an appropriate error vector) inject faults into the FSM and cause masked faulty transitions.

  This block essentially computes $s'R_2$. Due to this randomization, the non-uniform behavior of the FSM is removed at the error-check (i.e. comparison) level. Even though the non-redundant next-state logic block still provides non-uniform behavior, this does not affect the error detection probability because cubing and hence the error-checking is done on the randomly masked version of the next-state value.

  We also need to compute the check-sum associated with the computed next-

state value. This check-sum will be used in the next cycle by the select line checker block to make sure that the current-state value itself is fault free. As multiplication is more expensive than addition and this multiplication has already been performed, the next state is passed the codeword (s', $R_0^{'}$, $(s'+R_0^{'})^3$) where a different version of the randomizer, namely $R_0^{'}$ is used. Note that $R_0^{'}$ will also be used in the next cycle by the select line checker block for the validation of the current-state value. This verification is very important because current-state value s controls the multiplexer circuit in the predictor unit. As a result, it needs to be fault-free. More specifically, the output of this block will be

$$s'R_2 = s' + s'(R_2 - 1), \tag{5.1}$$

where $R_0^{'} = s'(R_2 - 1)$. In order to compute $R_0^{'}$ for the following cycle, s' is subtracted from the output $s'R_2$ of the randomizer. The check-sum $(s' + R_0^{'})^3$ is essentially computed by the cubing after this block. Note that $(s'R_2)^3$=$(s' + s'(R_2-1))^3$=$(s'+R_0^{'})^3$. Keep in mind that all these addition and multiplication operations are conducted over $GF(2^k)$.

- **Predictor**: The function of the predictor block is to predict the check-sum of the randomized next-state logic value (output of the randomizer). In order to achieve this, the predictor takes the input (i) and current-state (s). In this block, there is a register that holds the $(s_j'R)^3$ values with randomizer $R \in GF(2^k)$ (this is actually the main randomizer from the previous cycle) where j indexes all possible next-state values. With every clock-cycle, the values in this register are randomized with R_1 (more specifically by multiplication with R_1^3) that is generated using a true-random number generator (TRNG) and the results are also written back into the original register. This block also computes and stores the main randomizer R_2=$R \times R_1$. This value is also written-back and replaces the previous cycle's randomizer (i.e. R in Figure 5.1). It is also sent to the

randomizer block for the randomization process. The write-back is necessary to prevent the attacker from reading the existing values of the original register, calculating the appropriate error vector that will be missed, and injecting the faults that will result with this error vector. In other words, we want to update these register values at every clock cycle because the advanced attacker we are modeling cannot read and write within the same clock cycle. After the computation of the new $(s'_j R_2)^3$ for all j, the input i and the current-state s select the appropriate $(s'R_2)^3$ using a multiplexer circuit. The multiplexer circuit is inspired from the efficient FSM implementations. This is the predicted check-sum of the randomized next-state logic.

As with the randomizer block, the multiplication and cubing operations are conducted over $GF(2^k)$.

- **Select Line Checker**: As we mentioned earlier, since s and i are used as the select lines of the multiplexer in the predictor unit, they create a potential point of attack. Hence, we need a mechanism to verify that the s and i variables in the circuit are intact and fault-free. Select line checker block is aimed to achieve this goal. Basically, to verify the integrity of the current state s, this block utilizes the codeword $(s', R'_0, (s' + R'_0)^3)$ which is generated by the randomizer block in the previous cycle. It mainly computes $(s+R_0)$ and takes the cube of this addition, which is equivalent to computing the check-sum associated with s and R_0. The result is then compared with the fault-free check-sum $(s + R_0)^3$ that is computed in the previous cycle. The same idea applies to the input i as well. However, in our solution, we assume that the input is always coupled with its randomizer R_i and check-sum $(i + R_i)^3$. In other words, the peripheral that manages the inputs always sends the codeword $(i, R_i, (i + R_i)^3)$ to our state machine implementation. Any error generated by this block indicates that either s or i inputs of our state machine implementation is exploited by a fault

injection.

- **EDN**: The error detection network (EDN) is mainly a comparator. It compares the cube of the randomized next-state logic with the predicted check-sum. If the results match, this means that the operation is fault free, yet if there is a mismatch, then an error signal is triggered. The error signal can either reset the secret information or can stop the operation. A PUF based EDN mechanism is proposed in [26]. This provides a fault free EDN block and prevents the attacker from simply attacking and disabling the error signal.

5.3 Security Analysis

The main idea of the proposed protection scheme is to prevent an attacker from forcing the state machine into an arbitrary state. The predictor's job is focused on using the checksum of the state and the input in order to generate the checksum of the next state. Building a circuit which directly performs this operation can be quite costly. Therefore, we have used a multiplexer structure as explained in the previous section. The multiplexer uses the current input and the current state to compute the check sum of the next state. This operation does not preserve the isolation between the predictor branch and the FSM which could have an effect on the security of the scheme. To solve this problem we perform an overall comparison between the current state and its checksum. We also perform separate comparisons between the actual values and the checksums of both the current state and the input. In this section we will show that the error detection probability will be exponentially close to 1. We start with the following Corollary to Theorem 3.0.1.

Corollary 5.3.1. *Given a uniformly random* $x \in \{0,1\}^k$ *and* $w = x^3 \in \{0,1\}^k$ *the probability that any chosen pair* $(e_x, e_w) \neq 0$ *satisfies* $(x + e_x)^3 = w + e_w$ *is lower bounded by* $1 - 2^{-k+1}$.

Proof. Use Theorem 3.0.1 with parameters $n = 2k$, $k = r$ and $H = I$. □

We will also need the following Lemma.

Lemma 5.3.2. *Given a uniformly random $x \in \{0,1\}^k$ and any $w \in \{0,1\}^k$ the probability that any chosen pair (e_x, e_w) such that $e_x \neq 0$ satisfies $(x + e_x)^3 = w + e_w$ is upper bounded by $3 \cdot 2^{-k}$.*

Proof. The equation $(x + e_x)^3 = w + e_w$ is a cubic equation over GF(2^k) and hence will have at most 3 solutions for x for any given w, e_x, e_w. As x is chosen uniformly at random, the probability that x will be the correct solution for a specific w, e_x, e_w will be at most $\frac{3}{2^k}$. □

Now we can prove the main theorem.

Theorem 5.3.3. *The error detection probability of the scheme depicted in Figure 5.1 is lower bounded by $1 - 9 \cdot 2^{-k}$.*

Proof. Recall that under the adversarial fault model we are assuming, the attacker will not know R_0, R_i, R and R_1 in the same clock cycle in which he will inject his error. The scheme of Figure 5.1 will perform a comparison between $s_w = (s + R_0)^3$ and the cube of $(s + R_0)$, and similarly for $i_w = (i + R_i)^3$ and the cube of $(i + R_i)$. As R_i and R_0 are unknown and uniformly random to the attacker, using Corollary 5.3.1 we can assume that except with probability at most $2 \cdot 2^{-k+1}$ no error will be injected on s or i. Now we proceed by assuming error injections in every possible location of the circuit. Let the error injected into the FSM reflect to s' as a shift of $e_{s'}$. Similarly, let the error injected into the registers holding $R, R_1, R_2, (S'R)^3$ and $(S'R_2)^3$ be $e_R, e_{R1}, e_{R2}, e_{SR}$ and e_w respectively. Also, let the error injected into the multiplication between s' and R_2 be e_x. Finally, let the error injected into the cubing of R_1 be e_{R^3}. The output of the main branch will be

$$(s' + e_{s'})\left[(R_1 + e_{R1})(R + e_R) + e_{R2}\right] + e_x.$$

The output of the predictor side will be

$$((s'R)^3 + e_{SR})[(R_1 + e_{R1})^3 + e_{R^3}] + e_w.$$

As the main goal of an attacker is to change s' to another valid state we can assume $e_{s'} \neq 0$. We will now work our way backwards. The EDN will cube the main branch and compare it to the predictor branch. Now the term to the left of e_x in the main branch equation, which we label x, is a uniformly random string in $\{0,1\}^k$ and similarly, the term to left of e_w, which we label w is some string in $\{0,1\}^k$. Therefore, using Lemma 5.3.2 if $e_x \neq 0$ the probability of finding (e_x, e_w) to satisfy the comparison will be upper bounded by $3 \cdot 2^{-k}$. If this is not the case we can assume that $e_x = 0$. Next, we assume that $(RR_1) \neq 0$ which will happen with probability at least $1 - 2 \cdot 2^{-k}$. We can now factor RR_1 from x and $(RR_1)^3$ from w. For the left side we get,

$$(RR_1)[(s' + e_{s'})[(1 + e_{R1}R_1^{-1})(1 + e_R R^{-1}) + e_{R2}(RR_1)^{-1}]] \ .$$

The right side becomes.

$$(RR_1)^3[((s')^3 + e_{SR}R^{-3})[(1 + e_{R1}R_1^{-1})^3 + e_{R^3}R_1^{-3}]] + e_w \ .$$

We can now write the comparison as

$$[(RR_1)(\hat{x} + e_{s'})]^3 \stackrel{?}{=} (RR_1)^3(\hat{w} + e_w) \ .$$

Where

$$\begin{aligned}\hat{x} = s'[(1 + e_{R1}R_1^{-1})(1 + e_R R^{-1}) + e_{R2}(RR_1)^{-1}] + e_R e_{R1} e_{s'}(RR_1)^{-1} \\ + e_R e_{s'} R^{-1} + e_R e_{s'} R_1^{-1} + e_{R2} e_{s'}(RR_1)^{-1}\end{aligned}$$

which can be seen as a uniformly random string in $\{0,1\}^k$. Similarly,

$$\hat{w} = ((s')^3 + e_{SR}R^{-3})[(1 + e_{R1}R_1^{-1})^3 + e_{R^3}R_1^{-3}] + e_2(RR_1)^{-3}]$$

which is some string in $\{0,1\}^k$ which can be dependent on $\hat{x}$. Recall, that $(RR_1) \neq 0$ therefore the comparison above will hold iff

$$(\hat{x} + e_{s'})^3 = (\hat{w} + e_w) \ .$$

However, $e_{s'} \neq 0$. Now using Lemma 5.3.2 the above will hold with probability $1 - 3 \cdot 2^{-k}$. This will make the overall probability of an error not being detected by the scheme of Figure 5.1 lower bounded by

$$1 - (2 \cdot 2^{-k+1} + 3 \cdot 2^{-k} + 2 \cdot 2^{-k}) = 1 - 9 \cdot 2^{-k}$$

□

The theorem above insures the security of the proposed scheme against fault attacks. We note here that this scheme needs to carry three separate comparisons. A simpler scheme would be for the main branch to compute $(i + R_i) \times (s + R_0) \times (sR_2)$ then cube this term and compare it to $(i + R_i)^3 \times (s + R_0)^3 \times (sR_2)^3$ coming from the predictor branch. In fact, we conjecture that the error detection probability of such a scheme will still be exponentially close to 1. However, in this chapter we do not present a formal proof of this conjecture.

5.4 Hardware Scalability

As can be observed from Figure 5.1, the scheme requires 4 k-bit finite field cubings (1 k-bit squaring + 1 k-bit multiplication), $(t+2)$ k-bit finite field multiplications where t is the number of states in our FSM, and 3 k-bit finite field additions [1]. Since addition over $GF(2)$ is just XORing, field adders can be implemented very efficiently. Squaring over $GF(2)$ can also be implemented quite efficiently. As a result, the effect of the

[1]The number of multipliers in our design can be reduced to 2 by using only 1 multiplier in the predictor unit. In this case, all t multiplications will be implemented in a serial manner.

field adders and squarings in the scheme will be minimal. However, multiplication over $GF(2)$ is an expensive operation and hence the multipliers will dominate most of the area of the whole scheme. Since this is the case, the total area of the proposed FSM will scale in parallel with the area of the field multipliers. It is a reasonable assumption to state that the area of a field multiplier will scale as $O(k^2)$. Hence the area of the whole scheme will scale as $O(k^2)$.

The scaling factor $O(k^2)$ essentially determines a trade-off between area overhead and security. When the FSM is running a very sensitive application with a high security risk, one cannot tolerate any errors. Therefore, only an exponentially small probability of failure can be accepted. Of course it does not make sense to pay for such an overhead when the underlying application is not sensitive.

An interesting perspective is to consider the overhead from a complexity point of view. With this perspective one can see that the proposed scheme requires a circuit of size polynomial in k while providing an exponentially small (in k) fault injection probability.

5.5 Comparison with other FSM security schemes

At this point, it is also important to compare the error detection capability of our scheme with other error detection schemes. Triple modular redundancy (TMR) and quadruple modular redundancy (QMR) are two of the most common error detection techniques against active fault attacks. In TMR, the non-redundant FSM is replicated three times and a majority voting circuit determines the correct result. QMR works in the exact same way, but the non-redundant FSM is replicated four times. Applying linear codes for error detection is another proposed method [24]. Under a weak attacker model, linear codes, TMR, and QMR may provide limited security with minimum error detection probability greater than zero. However, in the advanced

attacker model considered in this chapter, an attacker can with %100 probability cause invalid state transitions on an FSM protected by these schemes. For example, in TMR and QMR, the attacker can inject the exact same error to all replicas of the original design. This attack will clearly go undetected as all replicas of the circuits will behave in an identical fashion. Similarly, in the linear scheme, the attacker will choose error vectors which are also valid codewords in the utilized code and hence the injected error will be undetected. It should be clear that the strength of the scheme proposed in this chapter stems from the exponentially small error detection probability even against an advanced attacker.

Another interesting FSM security scheme based of physically unclonable functions (PUF) was proposed in [26]. This PUF-based scheme is quite efficient with an exponentially small error detection probability even against an advanced attacker. However, it is only applicable to known-path FSMs, which is a specific class of FSMs. The approach we present in this chapter is a generic one that can be applied to any FSM.

5.6 Summary

We presented a fault detection scheme in FSMs based on multiplexers and systematic nonlinear error detection codes. Our scheme detects any injected fault with probability exponentially close to 1 even against an advanced attacker with high temporal and spatial fault injection capability. Furthermore, the work here presents a new approach to handling the non-uniform output of a state machine in a way which enables the usage of some classical error detection techniques.

Chapter 6

Novel PUF-based Error Detection Methods in Finite State Machines

6.1 Motivation

Utilizing Physically Unclonable Functions (PUFs) [21, 57, 62] for protection against active attacks in authentication schemes is an interesting approach. A PUF is a physical pseudo-random function which exploits the small variances in the wire and gate delays inside an integrated circuit (IC). Even when the ICs are designed to be logically identical, the delay variances will be unique to each IC. These variances depend on highly unpredictable factors, which are mainly caused by the inter-chip manufacturing variations. Hence, given the same input, the PUF circuit will return a different output on different chipsets [49]. Additionally, if the PUFs are manufactured with high precision, any major external physical influence will change the PUF function and therefore change the output. These features are indeed very attractive for any low cost scheme hoping to achieve tamper-resilience against active attacks.

In this chapter, we focus our attention on active attacks and present a CED

The work presented in this chapter is a joint work with Ghaith Hammouri.

design based on PUFs. We mainly focus on error detection in FSMs of hardware implementations. Basically, we propose a different approach to protect the FSMs against active fault attacks in an adversarial setting. This approach is different than the previously proposed techniques becuase it is not built on systematic non-linear codes. Instead, the main building blocks of the scheme proposed in this chapter are PUFs.

Our Contribution: In this chapter, we present a high level description of a technique which uses PUFs to secure a class of finite state machines against fault injection attacks. Our proposal offers a two-layer security approach. In the first layer the PUF's functionality is used to produce a checksum mechanism on the logical level and in the second layer, the intrinsic sensitivity of the PUF construction is used as a protection mechanism on the physical level. An injected error has a high probability of either causing a change in the checksum, or causing a change in the PUF characteristics, therefore signaling an attack. With this dual approach our proposal opens an interesting area of research which explores hardware specific features of state machines. The proposed design integrates with a finite state machine in three different ways: 1) It provides a checksum mechanism for the state transitions 2) It provides key integrity protection for any secrets used by the state machine 3) It provides a novel fault-resilient implementation of the error detection network. Our work here is the first study on utilizing PUFs to secure the control logic in hardware implementations. In addition, the utilization of PUFs proved to be extremely suitable when the hardware resources are limited.

The remainder of this chapter is organized as follows: Section 6.2 introduces the necessary background on PUF circuits. Our PUF-based approach to secure known-path state machines is discussed in Section 6.3. Next, the key integrity scheme utilizing PUFs is described in Section 6.4. In Section 6.5, a PUF-based secure error detection network (EDN) is presented.

6.2 Physically Unclonable Functions

A PUF is a challenge-response circuit which takes advantage of the interchip variations. The idea behind a PUF is to have the same logical circuit produce a different output depending on the actual implementation parameters of the circuit. The variations from one circuit implementation to another are not controllable and are directly related to the physical aspects of the fabrication environment. These aspects include temperature, pressure levels, electromagnetic waves and quantum fluctuations. Therefore, two identical logical circuits sitting right next to each other on a die in a fabrication house might still have quite different input-output behavior due to the nature of a PUF.

The reason one might seek to explore such a property is to prevent any ability to clone the system. Additionally, because of the high sensitivity of these interchip variations it becomes virtually impossible for an attacker to accurately reproduce the hardware. Another major advantage of the PUF's sensitivity is to prevent physical attacks on the system. Trying to tap into the circuit will cause a capacitance change therefore changing the output of the PUF circuit. Removing the outer layer of the chip will have a permanent effect on these circuit variables and again, it will change the output of the PUF circuit. Briefly, we can say that a well-built PUF device will be physically tamper-resilient up to its sensitivity level. Multiple designs have been proposed in the literature to realize PUFs. Optical PUFs were first introduced in [58]. Delay-based PUFs or more known as silicon PUFs were introduced in [21]. Coating PUFs were introduced in [66]. More recently, FPGA SRAM PUFs were introduced in [25]. Surface PUFs were proposed in [57, 67] and further developed in [62]. In general, so far the usage of PUFs has focused on the fact that they are unclonable. In this chapter we focus our attention to the delay-based PUF first introduced in [21].

A delay based PUF [21] is a $\{0,1\}^n \rightarrow \{0,1\}$ mapping, that takes an n-bit challenge (C) and produces a single bit output (R). The delay based PUF circuit consists

of n stages of switching circuits as shown in Figure 6.1. Each switch has two input and two output bits in addition to a control bit. If the control bit of the switch is logical 0, the two inputs are directly passed to the outputs through a straight path. If on the other hand, the control bit to the switch is 1, the two input bits are switched before being passed as output bits. So based on the control bit of every switch, the two inputted signals will take one of two possible paths. In the switch-based delay PUF, there are n switches where the output of each switch is connected to the input of the following switch. At the end, the two output bits of the last switch are connected to a flip-flop, which is called the *arbiter*. The two inputs to the first of these switches are connected to each other, and then connection is sourced by a pulse generator.

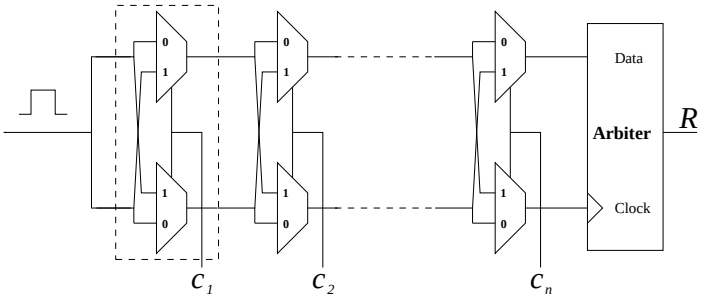

Figure 6.1: A basic delay based PUF circuit

The delay PUF can be described using the following linear model [21, 27],

$$R = \mathrm{PUF}_Y(C) = \mathrm{sign}\left(\sum_{i=1}^{n}(-1)^{p_i}y_i + y_{n+1}\right) \quad . \tag{6.1}$$

where $Y = [y_1 \dots y_{n+1}]$ with y_i as the delay variation between the two signal paths in the i^{th} stage and y_{n+1} captures the setup time and the mismatch of the arbiter. $\mathrm{Sign}(x) = 1$ if $x > 0$, and 0 if $x \leq 0$. $p_i = c_i \oplus c_{i+1} \oplus \dots \oplus c_n$, where c_i is the i^{th} bit of C. Note that the relation between $P = [p_1 \dots p_n]$ and $C = [c_1 \dots c_n]$ can be described using the equation $(P = UC)$. The strings C and P are represented as column vectors, U is the upper triangular matrix with all non-zero entries equal to 1 and the matrix multiplication is performed modulo 2. Equation 6.1 captures the

ideal PUF response. However, due to race conditions which will sometimes cause the two signals inside the PUF paths to have very close delays, the output of the PUF will sometimes be random. We refer to these random outputs as *metastable* outputs. This happens with a certain probability depending on the sensitivity of the arbiter. For typical flip-flops a 64-bit PUF will have about 1 metastable output for every 1000 outputs [47].

The variables y_i capture the secret maintained in the hardware, and which cannot be measured directly. These variables are usually assumed to be independent with each following a normal distribution of mean 0 and some variance which can be assumed to be 1 without loss of generality [49]. We note here that the independence of these variables will highly depend on the manufacturing process. For example in an FPGA implementation it is much harder to get almost independent y_i variables. In an ASIC implementation the y_i variables seem to be closer to independence. However, to simulate an independent response one might average over multiple independent challenges. In this study, we will work under the assumption that the y_i variables are independent.

With the independence assumption one can derive the probability distribution of two inputs $C^{(1)}$ and $C^{(2)}$ producing the same PUF output over all possible outputs as follows,

$$\text{Prob}[\text{PUF}_Y(C^{(1)}) = \text{PUF}_Y(C^{(2)})] = 1 - \frac{2}{\pi} \arctan\left(\sqrt{\frac{d}{n+1-d}}\right). \tag{6.2}$$

where d is the Hamming distance between $P^{(1)} = UC^{(1)}$ and $P^{(2)} = UC^{(2)}$. For the full derivation the reader is referred to [28, 27]. This relation carries the important fact that the correlations in the PUF output are solely dependent on the Hamming distance.

It is important to mention that due to the linear nature of the PUF circuit, given a number of challenge-response pairs (C, R), an attacker can use linear programming

[59, 55, 21] to approximate for the unknown vector Y. To solve this problem we have one of two options. First, hide the output R such that it is not accessible to an adversary. Our second option is to use non-linearization techniques such as the feed-forward scheme presented in [21, 47]. For simplicity we will assume that the output R is not available to an adversary. This only means that the attacker cannot read R, but he still can inject a fault. We will shortly see from the coming sections that this is quite a reasonable assumption.

Note on the Adversarial Fault Model: In this chapter, we are mostly concerned with secure FSMs and key storage. As a result, it is more appropriate to use the logical fault model. It is also important to note that since the analysis conducted in this chapter does not assume attacker's inability to read the existing data on the circuit before injecting a fault, overwriting and jamming errors can also be modeled as logical error additions. In addition, note that one additional assumption for the fault model of Section 6.4 is described by Assumption 6.4.1.

6.3 Securing Known-Path State Machines

In this section we address the security of state machines against adversarial fault attacks. We focus our attention to the state machines which are not dependent upon input variables. Such machines are quite common in cryptographic applications which typically contain a limited number of states and perform functions which require a long sequence of states. We now formally define the class of state machines which we address in this section.

Definition 6.3.1. *A known-path state machine, is a state machine in which state transitions are not dependent upon the external input. The state-sequence which the state machine goes through is known beforehand, and can be considered a property of the state machine.*

Algorithm 6.1 Always multiply Right-to-Left Binary Exponentiation Algorithm [29]

Require: x, $e{=}(e_{t-1}, ..., e_0)_2$

Ensure: $y = x^e$

$\mathrm{R}_0 \leftarrow 1$	`INIT`
$\mathrm{R}_1 \leftarrow x$	`LOAD`
for $i = 0$ upto $t - 1$ **do**	
$b = 1 - e_i$	
$\mathrm{R}_b \leftarrow \mathrm{R}_b^2$	`SQUARE`
$\mathrm{R}_b \leftarrow \mathrm{R}_b \cdot \mathrm{R}_{e_i}$	`MULTIPLY`
end for	
$y \leftarrow \mathrm{R}_0$	`RESULT`

An example of an algorithm that can be implemented with a known-path state machine is the "always multiply right-to-left binary exponentiation" [29] which is shown above. The associated state diagram for this algorithm is shown in Figure 6.2 with a possible fault injection attack (indicated by the dotted line). As can be observed, once the start signal is received, the transitions will follow a specific pattern and are independent from any kind of input except i which is a predetermined value. Another example of a known-path state machine is the "Montgomery ladder exponentiation" which is commonly used for RSA signature generation [30]. The PUF based security mechanism which we describe in this section defines a generic approach to secure this class of state machines. We now rigorously define our approach and derive the probability of detecting an injected error.

Let F be a known-path state machine with m states. We refer to the known-path of states which the state machine goes through in a full operation as the *state-sequence* and we denote it by S_Ω. Here, Ω represents the encoded states in the state-sequence observed by F. We define f to be the encoding function for our states. The function f is also assumed to produce a binary output. Let n be the bit size of the output of

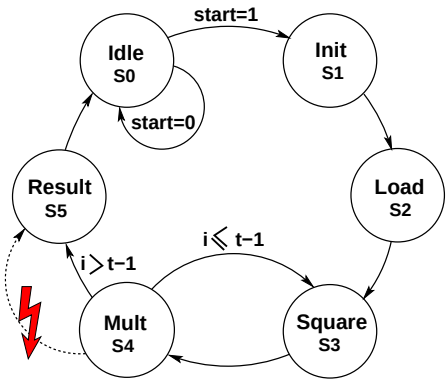

Figure 6.2: State Diagram Representation of Left-to-Right Exponentiation Algorithm with Point of Attack

f and k the number of states in Ω. So for example, if F enters the state-sequence $s_1, s_2, s_3, s_4, s_3, s_4, s_5$ then $\Omega = [f(1), f(2), f(3), f(4), f(3), f(4), f(5)]$. If the encoding scheme is a simple binary encoding then $\Omega = [001, 010, 011, 100, 011, 100, 101]$, $n = 3$ and $k = 7$.

With the above definitions our proposal's main goal becomes to finger-print the state-sequence S_Ω. The way we achieve this is by adding a PUF circuit to the state machine logic. The straightforward idea of the scheme works as follows: at initialization time the circuit calculates the PUF output for each of the encodings in Ω. This output which we label ω is then securely stored for future comparisons. The i^{th} bit of ω which we label ω_i is calculated as

$$\omega_i = \mathrm{PUF}_Y\left(\Omega(i)\right)$$

where $\Omega(i)$ is the i^{th} entry in Ω. This equation means that an n-bit PUF needs to be utilized, and that ω will be a k-bit string. This straightforward approach captures the essential idea of the proposal. However, there are problems with the efficiency of this approach. One can imagine a simple state machine going into a 3-state loop for 1000 cycles. This would mean that ω contains at least 3000 bits of a repeating 3-bit sequence. If secure storage is not an issue, or if k is a small number, then the

straightforward approach should suffice. However, when a state machine is expected to have large iterations over various cycles a different approach should be explored.

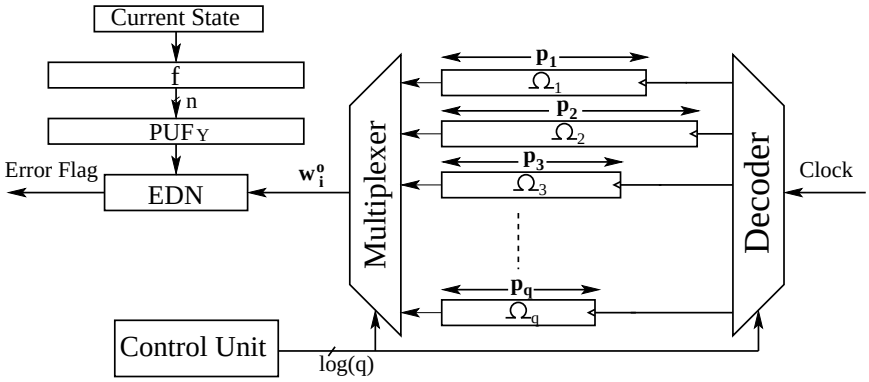

Figure 6.3: PUF-based circuit for protecting FSMs

To solve this problem we take a deeper look into the state-sequences that we expect to observe in the known-path state machines. Such state-sequences can be broken into q different sub-sequences each of which contain p_i states and is repeated for c_i cycles. We can now rewrite the overall state-sequence as

$$\Omega = [\{\Omega_1\}^{c_1} \mid \{\Omega_2\}^{c_2} \mid \ldots \mid \{\Omega_q\}^{c_q}] \ ,$$

where $\mid$ stands for concatenation of sequences, and $\{\Omega_i\}^{c_i}$ indicates the repetition of the sub-sequence Ω_i for c_i times. Note that $k = c_1p_1 + c_2p_2 + \ldots + c_qp_q$. With the new labeling, we can see that the checksum does not need to be of length k. It should suffice for the checking circuit to store the constants c_i and p_i in addition to the bits $\omega_{i,j} = \mathrm{PUF}_Y(\Omega_i(j))$ for $i = 1 \ldots q$ and $j = 1 \ldots c_i$. We can write

$$\omega = [\ \omega_1 \mid \omega_2 \mid \ \ldots \ \mid \omega_q \ ] \ ,$$

where ω_i will contain p_i bits. With only having to store the ω_i strings, the scheme will be efficient in terms of storage. We next explain how the checking circuit works.

As can be seen in Figure 6.3, the checking circuit stores each of the ω_i strings in a separate shift register with the output (the most significant bit) connected to

the input (the least significant bit) of the register. The outputs of the shift registers are also connected to a multiplexer, whose $\log(q)$ select signals connected to a small control unit. The control unit's main task is to maintain a counter C which will indicate how far along the state-sequence is the state machine, therefore generating the select signals for the multiplexer. When $C \leq c_1 p_1$ the first register's output is selected. Once the counter exceeds $c_1 p_1$ the control unit will select the second register, and will maintain the same output until the counter reaches $c_1 p_1 + c_2 p_2$. The control unit will essentially chose the i^{th} register as long as $c_1 p_1 + c_2 p_2 + \ldots + c_{i-1} p_{i-1} < C \leq c_1 p_1 + c_2 p_2 + \ldots + c_i p_i$. The checking circuit continues in this fashion until the counter reaches $k = c_1 p_1 + c_2 p_2 + \ldots + c_q p_q$ at which point the counter resets to zero since the state machine will be back to its initial state. We will label the output of the multiplexer at i^{th} state of the state-sequence as ω_i^o. For the registers to generate the right output, the select signals produced by the control unit also need to be fed into a decoder which will produce the clock signals of the registers. The input of the decoder will be the master clock signal, and the outputs of the decoder will be connected to the clock inputs of the shift registers. Note that these signals will only be high when the corresponding register is being used, therefore causing the register to shift accordingly.

At every clock cycle the current check bit $x_i = \mathrm{PUF}_Y\left(f(S_i)\right)$ is generated, where S_i is the current state of the state machine. The checking circuit will verify the condition $\omega_i^o = x_i$. Whenever this condition is violated the checking circuit can issue a signal to indicate a fault injection. We have mentioned earlier that the output of a PUF circuit will not be consistent for a certain percentage of the inputs. This percentage will set a tolerance threshold labeled L for the checking circuit. If the number of violations detected by the checking circuit is more than L, the checking circuit can signal an attack, therefore halting the circuits operation.

To calculate the probability of an attack actually being detected, we note that

the PUF output is uniform. A fault injected by the attacker will change the current state, and will consequently change the following states. We label the states in the fault-free sequence S_Ω as ideal states, and we label the states which are different as a result of the fault injection as faulty states. The fault-free sequence and the new faulty sequence will have t different states, $t \leq k$. We are interested in calculating the probability of the new faulty states actually yielding a different PUF output than that of the ideal states. Equation 6.2 shows that when the Hamming distance between the two PUF inputs is about $n/2$, this probability is 0.5. With this in mind, we can choose the encoding function f and the size of its output n such that the encodings of any two states have a Hamming distance of $n/2$. Even more efficiently, if the encoding is assumed to be secret, the Hamming distance between the encoded state vectors would be averaged over all possible encodings, therefore also yielding an effective Hamming distance of $n/2$ between any two state vectors. In either case, the probability of a faulty state generating the same PUF output as an ideal state will effectively be 0.5. With these factors, we can expect the detection probability of an injected fault which causes a total of t state changes to be

$$P_t = 1 - 2^{-(t-L)} \quad .$$

Naturally, t is assumed to be larger than L since otherwise the detection probability would be zero. If the fault injected causes a small number of state changes t, this probability will not be sufficient to secure the system. Although it is expected that an injected fault will cause a large number of state changes, for completeness we next handle the case when t is small.

We propose two approaches to solve this problem. The first is to utilize a number of PUF circuits each of which storing a separate array of checksums. And the second is to use a single PUF but calculate the check bits for different encoding functions of the states. Essentially one can use a single encoding and then apply a permutation to generate a variant encoding. Whether we use d PUFs or we use d different encodings,

in either case we will be adding d-levels of check bits. Regardless of which of the two approaches we use the detection probability of a fault causing t state changes will be

$$P_t = 1 - 2^{-(td-L)} \quad .$$

Using error control techniques for the PUF circuit such as majority voting [47], the error rate in the PUF output can be reduced to as low as 1 in 1000. This means that for state machine where $k < 1000$ states, the probability of error detection becomes $P_t = 1 - 2^{1-td}$. Naturally, this probability does not take into account the probability of inducing a change in the internal PUF parameters. Such a change will have an effect on the PUF output and will therefore increase the error detection probability.

In order to estimate the hardware overhead incurred by the proposed error detection mechanism, we carry out the following analysis. The number of flip-flops required for storing the checksums will be equivalent to the number of flip-flops used in the current state register. As mentioned earlier, the counter C constitute the main part of the control unit, which is also the case for the state machine. Therefore, we can argue that the size of the control unit and the checksum storage will be approximately the same as the state machine, implying a 100% overhead.

The encoding function f will typically have an output size which is on the order of the total number of states m. Consequently, we can assert that the function f will on average use about $2m$ combinational gates. The same applies to the PUF circuit which will also require about $2m$ gates. Finally, the size of the decoder and the multiplexer shown in Figure 6.3 is expected to be on the order of $\log(q)$ gates. Although q can be of any size, in a typical state machine q will not be larger than 2^m. Hence, the number of gates associated with the multiplexer and the decoder will be about $2m$. Adding these numbers results in a total gate count of $6m$. This is the same number of gates used by the current state register (Note that m flip-flops are approximately composed of $6m$ universal gates). In general, it is safe to assume that the current state register will consume approximately 50% of the total state machine

area, which implies an area overhead of 50%.

As a result, the total area overhead introduced by the proposed error detection scheme will be approximately 150%, with a high error detection rate even against strong adversaries. When compared to the simpler error detection schemes such as Triple Modular Redundancy (TMR) and Quadruple Modular Redundancy (QMR) (which only replicate the existing hardware, implement the same function concurrently, and do a majority voting to check if an error has been injected), the proposed scheme provides a higher level of security even against advanced adversaries because an attacker can simply inject the same error to all replicas of the original hardware and mask the error in these detection schemes. The area overhead associated with these trivial detection mechanisms will be at least 200% for TMR and 300% for QMR which is also higher than the overhead of the proposed mechanism. As a comparison to a more advanced error detection scheme, the study conducted by Gaubatz et al. [24], which utilizes linear codes for error detection, reports an area overhead of more than 200%. However, their fault model assumes weak adversaries and the error detection scheme becomes vulnerable against strong attackers. It is important to note that, finite state machines usually constitute a very small part of the entire circuit. Therefore, although the reported area overhead might appear to be large, the effective increase in the overall area is reasonable. To sum up, PUF-based error detection mechanism discussed in this chapter accomplishes a higher level of security with a reduced area overhead.

6.4 Key Integrity

In [11], Biham and Shamir extended the fault injection attacks to block-ciphers and reported that they can recover the full DES key of a tamper-proof encryptor by examining 50 to 200 cipher-texts. In their paper, they also described a method to

break an unknown (unspecified) cryptosystem by utilizing the asymmetric behavior associated with the used memory device. Basically, their fault model assumes that the applied physical stress on the memory device, which contains the key bits, could only cause one to zero transitions[1]. Using this attack, the secret key can be obtained using at most $O(n^2)$ encryptions. Similarly, Kocar [36] reports a method to estimate the key bits of a cryptographic device by employing the charge shifting characteristic of EEPROM cell transistors after baking. In addition, in [4] authors describe an EEPROM modification attack where they can recover the DES key by overwriting the key bits one by one and checking for parity errors. Same authors, in [3], also discuss example attacks on PIC16C84 microcontroller and DS5000 security processor in which security bits can be reset by modifying the source voltages. The key overwrite attacks also constitute a crucial risk on smart cards where the key is stored inside the EEPROM. To summarize, fault injection attacks on secret keys stored on-chip memory pose a serious threat in many cryptographic hardware implementations.

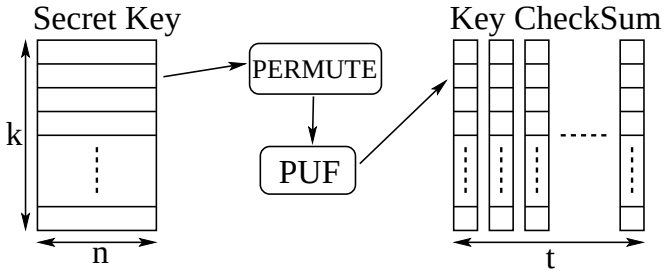

Figure 6.4: Key integrity check using PUF

In this section, we propose utilizing PUFs as a solution to this important problem. The main idea here is similar to that of the previous section. Basically, we generate a secret key fingerprint or checksum for the correct secret key using a PUF circuit. As outlined earlier, the checksums are assumed to be stored secretly while allowing fault

[1]This one-way characteristic can also cause zero to one transitions depending on the asymmetry of technology used to fabricate the memory device.

injection. Regularly, the checking circuit can check and verify the integrity of the key. If the checksum value for the current key does not match the checksum value for the correct secret key, this can be interpreted as an error injection to the key. As a result, an error message can be issued and the secret data can be flushed or the device can be reseted to prevent any kind of secret leakage. This mechanism is briefly shown in Figure 6.4. This figure shows part of the memory which contains a secret key of size $k \times n$. Each row of this key block is labeled r_i and is treated as an input to the PUF circuit. If the rows are directly fed to the PUF circuit, an attacker can carefully choose his errors such that the Hamming distance between the actual variables (P) defined in Equation 6.1 is minimal. Recall that this would mean that the PUF output will not be able to detect the injected error. If the size of the checksum for each key row is a single bit, the error detection probability for an injected error would be 0.5 as the PUF can only provide an output of $\{0, 1\}$. In this case, the success rate for the attacker is considerably high. This is why we utilize a permutation block as shown in Figure 6.4.

The permutation block will essentially permute each input row r_i by a pre-determined permutation ρ_j where $j = (1, \ldots, t)$ and $t < n$. Consequently, $\rho_j(r_i)$ is fed to the PUF in order to generate the (i, j) bit of the checksum. In short, for the secret key array S with size $k \times n$ and rows r_i, we calculate the (i, j) bit of the checksum S_w as

$$S_w(i, j) = \mathrm{PUF}_Y(\rho_j(r_i)) \tag{6.3}$$

where the ρ_j's are random permutations pre-chosen secretly and S_w is of size $k \times t$.

When this model is applied to secure the cryptographic devices against memory overwrite attacks, the robustness and security measure of the error detection scheme becomes a direct function of t, the number of the permutations used for each row. The probability of an error being detected is essentially the probability of an error changing the PUF output. However, we have seen in the previous section that the PUF output

will sometimes be metastable. Therefore, we will again define an acceptable level of errors which will be a property of the system and which will not raise an alarm. Similar to the previous section we define this level as L. Now we can define the event for an error being detected. In particular, an error injected to row r_i will be detected provided that the following equation will not hold for more than L of the row's t checking bits.

$$\mathrm{PUF}_Y(\rho_j(r_i)) = \mathrm{PUF}_Y(\rho_j(r_i + e)) \tag{6.4}$$

where e indicates an error injected to the i^th row, e.g. bit flip of some memory cells. We now calculate this probability. Equation 6.4 is essentially the probability calculated in Equation 6.2. Therefore, we will again have to refer back to the Hamming distance between the ideal and the attacked PUF inputs. Because the permutations ρ_j are taken over all possible permutations, the attacker cannot control the effective location of his injected errors. To simplify the calculation we make the following assumption.

Assumption 6.4.1. *We assume an attacker model where the number of faults injected by the attacker is uniform over all possible number of faults.*

With Assumption 6.4.1 we can calculate the expected value of the Hamming distance between the P values of the original data and the faulty data when taken over all permutations to be equal to $n/2$. Going back to Equation 6.2 for this particular Hamming distance the probability for Equation 6.3 to hold will be 0.5. With this, the detection probability of an error becomes $1 - 2^{-(t-L)}$.

At this point, it is important to note the trade-off between the area overhead and security level of the suggested mechanism. As the number of permutations t for each row increases, the security of the device gets stronger because the error detection probability increases. However, the area overhead also increases linearly with t due to the checksum storage space. The optimal value for t is an application dependent

issue and can be adjusted according to the required security level or allowed area overhead.

Note that one can use error correcting codes to address the integrity issue. However, such a solution would require substantially more hardware for decoding the code words. Moreover, the PUF circuit has built-in fault resilience due to its sensitive characteristics. Consequently, any kind of fault injection or perturbation of the hardware will modify the result of the PUFs. This brings an additional level of security to the proposed key integrity and protection scheme. In addition, the solution we present here views the separate memory rows as independent entities. It is an interesting problem to explore combinations of the rows and columns, which might improve the error detection probability. Finally, the model here assumes that the checksum is hidden secretly. If a designer wishes to relax this condition different PUF designs should then be explored.

6.5 Error Detection Network Security

Concurrent error detection (CED) is one of the most common solutions against active fault attacks. The basic idea in these schemes is to calculate the expected result using predictor circuits in parallel to the main hardware branch, and compare if the predicted value of the result matches the actual value calculated in the main branch. A good overview along with elaborate examples about this mechanism can be found in [22]. An error signal is issued when these two paths do not produce agreeing results. This comparison along with the error signal generation are conducted by the error detection network (EDN) modules.

While the attackers are always assumed to target the main or predictor branch of a cryptographic device, the EDN network which is in fact the weakest link in the design is always assumed to be completely fault resilient. An attacker can deactivate

the EDN by preventing an error signal from being issued or by just simply attacking the inputs of the EDN. Therefore, totally disabling/bypassing the error detection mechanism. To tackle this problem, we propose utilizing PUF structures to design secure and fault tolerant EDN blocks.

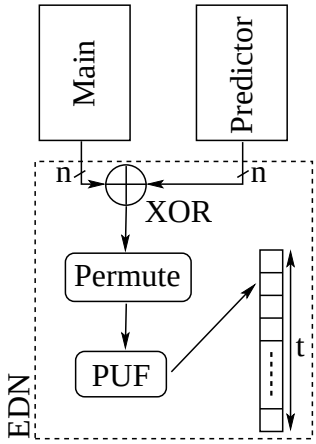

Figure 6.5: PUF based EDN

The suggested PUF based EDN mechanism is shown in Figure 6.5. Basically, the results coming from the main and predictor branches of the computation are first XOR-ed together. In the absence of an injected fault, the result will be the zero vector. The circuit starts by producing a fingerprint of the PUF response to an all zero bit challenge right before the circuit is deployed. This fingerprint is stored as the checksum bit. Throughout the circuit's operation, the XOR-ed results are continuously permuted and fed into the PUF circuit. This permutation block implements the same functionality as in Section 6.4. In the absence of an error, the permutation will have no effect on the all zero vector, Therefore the output should always match the stored checksum bit. However, when an error is injected the output of the XOR will not be the all zero vector. This will cause the permutations to generate different challenge vectors which will consequently produce PUF outputs which are different from the checksum bit.

When the circuit detects a mismatch between the output of the PUF and the checksum bit an injected fault is assumed and an error signal can be issued. Similar to the analysis conducted in the previous two sections, the error detection capability of the EDN is dependent upon the number of applied permutations t, and can be formulated as $1 - 2^{-(t-L)}$. As in the previous sections, L here is the acceptable threshold of errors in the PUF response. The trade-off between the security and area overhead discussed in Section 6.4 also exists in this PUF based EDN methodology too.

Note that any attempt by the attacker to modify the voltage levels of the wires located inside the EDN will affect and change the result of the PUF due to its high sensitivity. This intrinsic tamper resistance of the PUF circuit acts as assurance against fault attacks targeting the EDN.

6.6 Summary

In this chapter, we explored the integration of PUFs into the building blocks of finite state machines to provide security. In particular, we addressed the security of state-transitions (next-state logic) against fault-injection attacks, the integrity of secret information, and finally fault-resilience in error detection networks. We proposed PUF-based architectures for the security of these modules in a control unit, and showed that the probability of error detection is high. More importantly, the solution we propose provides security on the physical level as well as the logical level. Even if the adversary can find the appropriate fault to inject, there will still be a good chance of being detected by the change in the PUF behavior. The designs we propose in this chapter are described from a higher level and therefore are far from being final solutions ready for implementation. Rather, these solutions are mainly intended to open a new door for research in the area of unclonable protection of FSMs.

Such mechanisms can provide strong error detection with a relatively low hardware overhead. Our work here is a first step in this direction.

Chapter 7

Nonlinear Error Detection for Elliptic Curve Cryptosystems

7.1 Motivation

Elliptic curve cryptosystems (ECC) offer the highest security per bit among the existing public key cryptography systems such as RSA, Diffie-Hellman and ElGamal. In [48], Lenstra and Verheul reported that ECC using a 130-bit key offers comparable security as RSA with a key length of 1024 bits. As a result, it is a reasonable alternative to RSA especially for embedded applications. However, such devices are more vulnerable to side-channel attacks, since the attacker can procure, isolate, and test such a system without being detected [3, 4].

Elliptic curves are conventionally represented using Weierstrass formulation in the most general form. In 2007, Edwards proposed a novel formulation of elliptic curves and associated point arithmetic operations defined over all non-binary fields [19]. Edwards elliptic curves, which have no point at infinity, are the normal form of birationally equivalent Weierstrass elliptic curves. In addition, Edwards formulation

The work presented in this chapter is a joint work with Deniz Karakoyunlu.

may provide performance benefits especially when projective coordinates are used in the absence of an efficient modular divider. Moreover, point addition and doubling operations can be handled with the same unified operation. This facilitates the design of side-channel resistant ECC systems [8, 31].

ECC based systems have also been a target of active fault attacks. In [10], Biehl et al. showed that using fault injection, ECC point multiplication can be forced to be computed over a less secure elliptic curve. As a result, it becomes relatively easy to solve the discrete log problem over this less secure curve. They also proposed implementing bit faults during random moments of a multiplication operation and showed that it is possible to reveal the secret key d in a bit-by-bit fashion. In [15], authors relaxed the assumptions of Biehl et al. in terms of the location and precision of the injected faults. They used two main fault models: 1) A permanent fault (in an unknown location on the device) on any of the system parameters defining the elliptic curve 2) A transient fault that occurs while the system parameters are being transferred to the working memory (RAM). Even with this new attacker model, their attack essentially recovers the (partial) secret in ECC discrete log problem. Note that the attacks described in these two papers can be prevented by checking whether the points used in these computations are on the utilized elliptic curve. However, in [12], Blomer et al. proposed the so-called "Sign Change Fault" attacks on the ECC based systems. These attacks do not change the original curve and work with points on this curve. Basically, they propose changing the signs of intermediate points during a scalar multiplication using fault injection. This leads to a faulty output which is also on the curve. Then they use a similar algorithm to the one described in [13] to recover the secret scalar in polynomial time.

Our Contribution: In this chapter, we propose applying systematic nonlinear error detection codes to protect elliptic curve point addition and doubling operations against active fault attacks. We analyze the security level provided by this error

detection technique and show that these codes provide nearly perfect error detection capability (except with exponentially small probability) for elliptic curve point operations. In addition, we discuss how to design low-overhead predictor circuits as part of our nonlinear error detection scheme. In this comprehensive analysis, we include both Weierstrass and Edwards curves over different coordinate systems (i.e. affine and projective). Moreover, we compare our technique with the method discussed in [22] where the authors proposed an error detection technique for robust public key arithmetic. They did not explicitly apply their technique to secure ECC point operations. However, by using their robust arithmetic units to implement each arithmetic operation in ECC, their security level can be achieved for ECC point operations as well. When compared with their method, our technique provides approximately the same level of security and robustness with much less overhead. The overhead of our scheme is less than half (42%-46%) of the overhead of [22] for Edwards curves and is 52% to 81% of the overhead of [22] for different versions of the Weierstrass curves.

This chapter is organized as follows: In Section 7.2, we give a short overview of the ECC and existing error detection techniques for ECC in the literature. The proposed error detection technique is described in Section 7.3. In Section 7.4, we discuss the proposed point addition and doubling constructions. Finally, we discuss the area overhead associated with our technique in Section 7.5.

7.2 Background

In this section, we initially present the required background on elliptic curves and ECC. We also provide the existing error detection strategies proposed to prevent fault injection attacks on ECC implementations.

7.2.1 Elliptic Curve Cryptography Overview

This section briefly describes the elliptic curve discrete logarithm problem (ECDLP) and ECC formulations over finite fields of prime characteristics. A point P of order n, selected over an elliptic curve E defined over a finite field $\mathbb{F}_p$, can be used to generate a cyclic subgroup $\langle P\rangle = \{\infty, P, 2P, 3P, \ldots, (\#n-1)P\}$ of $E(\mathbb{F}_p)$.

The ECDLP is the underlying number theoretical problem used by ECC, and it is defined as determining the value $k \in [1, \#n-1]$, given a point $P \in E(\mathbb{F}_p)$ of order $\#n$, and a point $Q = kP \in \langle P\rangle$. In a cryptosystem, the private key is obtained by selecting an integer k randomly from the interval $[1, \#n-1]$. Then corresponding public key is the result of scalar point multiplication $Q = kP$, which is computed by a series of point additions and doublings.

7.2.1.1 Simplified Weierstrass Formulation for Elliptic Curves

An elliptic curve E defined over a prime field $\mathbb{F}_p$ (with $p > 3$) can be written in the simplified Weierstrass form as:

$$E(\mathbb{F}_p) \,:\, y^2 = x^3 + ax + b \tag{7.1}$$

where $a, b \in \mathbb{F}_p$, and the discriminant of the curve $\Delta = -16(4a^3 + 27b^2) \neq 0$. A point addition operation can be defined for adding two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ in $E(\mathbb{F}_p)$ resulting in a third point $P + Q = (x_3, y_3)$ in $E(\mathbb{F}_p)$ with the point at ∞ serving as identity element $(P + \infty = P)$. Assuming that $P \neq \pm Q$, the point $P + Q = (x_3, y_3)$ can be computed as:

$$\begin{aligned} x_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \pmod p \\ y_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1 \pmod p \end{aligned} \tag{7.2}$$

For $P = Q$ the operation is called doubling, and the calculation of $2P = (x_3, y_3)$ is slightly different:

$$\begin{aligned} x_3 &= \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \pmod{p} \\ y_3 &= \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1 \pmod{p} \end{aligned} \tag{7.3}$$

Finally, if $P = -Q$ the operation results in point at infinity, and it should be handled separately.

A Weierstrass elliptic curve defined in Equation (7.1) is converted to Jacobian coordinates as follows:

$$E(\mathbb{F}_p) : Y^2 = X^3 + aXZ^4 + bZ^6$$

where $X = xZ^2$, $Y = yZ^3$. Then the point addition (Equation 7.4) and doubling (Equation 7.5) formulations with Jacobian coordinates become [14]:

$$\begin{aligned} X_3 &= (Y_2Z_1^3 - Y_1Z_2^3)^2 - (X_2Z_1^2 - X_1Z_2^2)^2(X_2Z_1^2 + X_1Z_2^2) \pmod{p} \\ 2Y_3 &= (Y_2Z_1^3 - Y_1Z_2^3)[(X_2Z_1^2 - X_1Z_2^2)^2(X_2Z_1^2 + X_1Z_2^2) - 2X_3] \\ &\quad -(X_2Z_1^2 - X_1Z_2^2)^3(Y_2Z_1^3 + Y_1Z_2^3) \pmod{p} \\ Z_3 &= (X_2Z_1^2 - X_1Z_2^2)Z_1Z_2 \pmod{p} \end{aligned} \tag{7.4}$$

$$\begin{aligned} X_3 &= (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2 \pmod{p} \\ Y_3 &= (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4 \pmod{p} \\ Z_3 &= 2Y_1Z_1 \pmod{p} \end{aligned} \tag{7.5}$$

7.2.1.2 Edwards Formulation for Elliptic Curves

An elliptic curve E defined over a prime field $\mathbb{F}_p$ (with $p > 3$) can be written in the Edwards normal form as:

$$E(\mathbb{F}_p) : x^2 + y^2 = c^2(1 + dx^2y^2) \tag{7.6}$$

where the parameter c can be chosen as 1 without loss of generality. Addition of two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ in $E(\mathbb{F}_p)$ resulting in a third point $P + Q = (x_3, y_3)$ in $E(\mathbb{F}_p)$ can be computed as:

$$\begin{aligned} x_3 &= \frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2} \pmod p \\ y_3 &= \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2} \pmod p \end{aligned} \tag{7.7}$$

This equation is valid even if $P = Q$, and it never results in point at infinity. An Edwards elliptic curve defined in Equation (7.6) is converted to homogeneous projective coordinates as follows:

$$E(\mathbb{F}_p) : X^2 + Y^2 = Z^4 + dX^2Y^2$$

where $X = xZ$, $Y = yZ$. The following formulas compute the unified point addition and doubling (Equation 7.8), and optimized doubling (Equation 7.9) operations with projective coordinates [8]:

$$\begin{aligned} X_3 &= Z_1Z_2(X_1Y_2 + Y_1X_2)(Z_1^2Z_2^2 - dX_1X_2Y_1Y_2) \pmod p \\ Y_3 &= Z_1Z_2(Y_1Y_2 - X_1X_2)(Z_1^2Z_2^2 + dX_1X_2Y_1Y_2) \pmod p \\ Z_3 &= (Z_1^2Z_2^2 - dX_1X_2Y_1Y_2)(Z_1^2Z_2^2 + dX_1X_2Y_1Y_2) \pmod p \end{aligned} \tag{7.8}$$

$$\begin{aligned} X_3 &= 2X_1Y_1(X_1^2 + Y_1^2 - 2Z_1^2) \pmod p \\ Y_3 &= (X_1^2 - Y_1^2)(X_1^2 + Y_1^2) \pmod p \\ Z_3 &= (X_1^2 + Y_1^2)(X_1^2 + Y_1^2 - 2Z_1^2) \pmod p \end{aligned} \tag{7.9}$$

We provide the dataflow of point operations associated with both Weierstrass and Edwards curves in Appendix A. The dataflow format makes it easier to count the number of operations that are used for each point operation.

7.2.2 Existing Error Detection Techniques in ECC

In this section, we will give a brief overview of the existing techniques that can be used for error detection in ECC systems. One countermeasure against active fault attacks in ECC is to check if the resulting point is on the elliptic curve or not. This method is called point validation (PV). However, Blomer et al. showed that PV is not sufficient against sign change fault attacks [12]. As a countermeasure against sign change fault attacks, they proposed performing scalar multiplication over a combined curve. The additional curve is then used to verify the final result of the computation.

[65] proposed using the mathematical properties of the register transfer level field primitives, i.e. field inversion, field addition, and squaring, for concurrent error detection. For example, for the field inversion operation, they propose feeding the output of the first inversion operation to the same inversion block and compare it to the original input. Since inversion is an involutional operation, the comparison should hold in the absence of any faults. A similar time redundant approach is applied to multiplication, addition, and squaring as well. Even though this is an interesting approach, it may not be suitable for time-critical ECC implementations due to its high time overhead which is on the order of 120%.

In [20], Francq et al. proposed the integration of the parity preserving logic gates to some blocks of an elliptic curve unit. More specifically, borrow-save adders (BSAs) in the ECC unit utilize these parity checking gates in order to provide error detection capabilities. This is a gate level security technique and the ratio of the undetected faults is between 5 to 12% depending on the multiplicity of the injected errors. Also, note that this technique only protects the BSA blocks in the ECC circuit and causes high latency on the computations.

[56] proposed error detection for point multiplication based on "Euclidean Addition Chain" implementations where a sequence of additions are used to compute the result of a point multiplication. In this method, at each point addition of the

sequence, the difference between the points is also computed and compared with one of the operands stored from the previous addition in the chain. Any mismatch after this comparison indicates a fault injection attack on the device. To implement this additional subtraction operation, they propose using shared logic with the point addition. Even though this could be an interesting approach against a weak adversary, this scheme can be broken by the advanced attacker that is assumed in this thesis. By reflecting the same error vector (e) on both $U_c(i)$ and $U_1(i)$ computations, the attacker can bypass the ($if\ U_D \neq U_C$) check and mask the error. In this case, the error vector e is the effect of the injected fault on a fault-free variable, $U_c(i)$, $U_1(i)$, $U_2(i)$, and $U_D(i)$ are partial products in the addition chain. By implementing such an attack, the attacker can successfully change the output of the whole point multiplication operation. This algorithm also requires the secure transfer of intermediate values such as $U_1(i-1)$, $U_2(i-1)$, and $U_C(i-1)$ between each round to be successful against fault injection attacks.

Another approach proposed by [18] is based on time and hardware redundancy. In this paper, authors proposed the application of encoding techniques called "input point and scalar randomization" to ECC operations. More specifically, a point P on the projective coordinates of an elliptic curve E has multiple representations. By encoding this point to one if its images and using time redundant/parallel computation, error detection can be achieved. The outputs of the actual and redundant computation are then compared. If there is a mismatch, this points to a fault injection attack. A similar encoding approach can be used to randomize the scalar k of the multiplication operation as well. The error masking probability of the proposed scheme is on the order of $1/q^2$, where $q \approx 2^{160}$ for standard NIST curves. It is also important to note that the parallel computation method described in this paper brings a hardware overhead of approximately %137. In this paper, authors assume the secure implementation of point and scalar randomization modules, multiplexers,

and registers that are used in the proposed error detection scheme. Moreover, the randomization approach they use requires a random number generator (RNG) that will generate the encoding masks. We do not incorporate any kind of RNG in our technique.

7.3 The Error Detection Technique

We mainly propose applying nonlinear codes to secure operations conducted over elliptic curves, i.e. point addition and doubling operations against active fault injection attacks. This is a generic method that can be applied to all elliptic curve structures (Weierstrass and Edwards) and all coordinate systems (projective and affine). In this chapter, we are focusing on ECC structures based on prime fields $\mathbb{F}_p$, yet a similar idea can be applied to protect elliptic curves that are defined over binary fields as well.

The main idea is to encode the coordinates of elliptic curve points using the systematic nonlinear (n, k, r)-code of Definition 3.0.4. This code essentially uses redundancy for error detection. We define the following error check function on a point coordinate $X \in \mathbb{F}_p$ to obtain a non-linear error check-sum

$$w = f(X) = X^2 \pmod{p} \in \mathbb{F}_p. \tag{7.10}$$

Consequently, the point coordinate X is encoded as $(X, f(X))$. From a practical viewpoint we have added r check digits to the point coordinate value as an error check redundancy. We now formally define a robust code by embedding the nonlinear code definition introduced by Gaubatz et al. [22] into elliptic curves as follows.

Definition 7.3.1. *We define the prime field robust code* (n, k, r) *as* $C_p = \{(x, w)|x \in \mathbb{F}_p, w = x^2 \pmod{p} \in \mathbb{F}_p\}$ *where* $2^k - p < \epsilon$ *and* $r{=}k$.

In this case, $k \geq \lceil \log_2 p \rceil$ because essentially k-bits are used to represent the x and w values where $x, w \in \mathbb{F}_p$. Furthermore, note that in this coding structure,

x and w are generic variables in $\mathbb{F}_p$, and do not represent any coordinates of an elliptic curve point. This code will essentially be applied to all the coordinates of an elliptic curve point. In the non-redundant case, a point P on an elliptic curve E is represented as $P=(X, Y, Z)$ in projective and Jacobian coordinates and as $P=(x, y)$ in affine coordinates. However, after the robust code of Definition 7.3.1 is applied, each point will be represented as $P=(X, X_w, Y, Y_w, Z, Z_w)$ in projective and Jacobian coordinates and $P=(x, x_w, y, y_w)$ in affine coordinates, where subscript w is used to show the check-sum portions. For the details on applying this coding technique to elliptic curves point operations, please see Section 7.4 and appendices.

7.3.1 Security Analysis

In this section, we prove the robustness of the proposed scheme. To quantify the performance of the nonlinear code described in Definition 7.3.1, we need to provide a lower bound of its error detection probability as a function of all error vectors $e \neq 0$. The following theorem establishes this bound for any elliptic curve E.

Theorem 7.3.1. *For the nonlinear code C of Definition 7.3.1, the error masking probability is upper bounded by $max(4, 2^k - p + 1) \cdot (p + 1 - 2\sqrt{p})^{-1}$.*

Proof. In order to quantify the performance of their scheme, Gaubatz et al. [22] needed to bound the error masking probability associated with the nonlinear code they used. Consequently, they count the number of x values that satisfy the numerator of Equation 3.2 for each $e \neq 0$. They use the following error masking equation to count these x values.

$$(x + e_x \bmod 2^k)^2 \bmod p = (x^2 \bmod p) + e_w \bmod 2^k. \tag{7.11}$$

In their paper, they provide a detailed proof which shows that the number of solutions to this equation is upper bounded by $\max(4, 2^k - p + 1)$. As the final step, this number

is normalized with the number of valid codewords, 2^k in their case, to compute the maximum error masking probability of their scheme.

We use a similar approach to quantify the error detection capability of our code. Essentially, for the nonlinear coding structure we propose in this chapter, we need to bound the value of $Q(e)$ for all $e = (e_x, e_w) \neq 0$. We achieve this in two steps.

First of all, we will bound the number of solutions to the error masking equation (numerator). In order to achieve this, we initially need to identify the error masking equation for our specific coding structure. Note that even though $x \in \mathbb{F}_p$ in our code, it is still represented using k-bits. Hence, when the attacker injects the error vector $e{=}(e_x, e_w)$ into the circuit, its additive effect on the x (datapath side) will be $x + e_x \bmod 2^k$. Similarly, values of w (check-sum associated with x) are also elements of the domain $\mathbb{F}_p$ and since $r{=}k$ in our code, redundant check-sum (w) values are also represented using k-bits. In this case, the effect of the injected fault on the check-sum side will be represented as $w + e_w \bmod 2^k{=}(x^2 \bmod p) + e_w \bmod 2^k$. In order to check if this fault injected data and check-sum pair is a valid codeword, we apply the check-sum function $f(x) = x^2 \bmod p$ on the resulting data $x + e_x \bmod 2^k$ and compare it with the resulting check-sum $(x^2 \bmod p) + e_w \bmod 2^k$. Note that this is the exact same error masking equation shown in Equation 7.11. This proves that the proposed code in this chapter will use the same error masking equation which is used by [22]. Next step is to count the number of valid (x, w) pairs that will satisfy this equation for each nonzero $e{=}(e_x, e_w)$. In [22], authors compute the number of solutions to the error masking equation for $x \in \mathbb{Z}_{2^k}$. However, the domain of the x values defined in our coding scheme will be different. In our case, $x \in \mathbb{F}_p$, instead of $x \in \mathbb{Z}_{2^k}$. The important observation we made at this point is the following. Since $p < 2^k$, even in the worst case, the number of solutions to the error masking equation cannot be larger than $\max(4, 2^k - p + 1)$ which is the bound computed by [22]. Hence, the upper bound of the numerator of the error masking probability of our code will

also be $\max(4, 2^k - p + 1)$.

As the second step, we compute the number of valid codewords (denominator) that will be observed at the output of the ECC functional units. At this point, our coding scheme has two important deviations from the scheme described in [22]:

- $x \in \mathbb{F}_p$, instead of $x \in \mathbb{Z}_{2^k}$.
- For an elliptic curve, the X, Y, and Z coordinate values (assuming projective coordinates) will not be uniformly distributed over $\mathbb{F}_p$.

In this case, Assumption 3.0.3 becomes invalid. As a result, we cannot use the same normalization constant (2^k) used by [22] to compute the error masking probability of our scheme. Note that, only the X, Y, and Z values that satisfy the elliptic curve equation can be visible at the output of the point addition and doubling operations. As a result, the new normalization constant of the error masking equation will be the number of valid points on the elliptic curve E. This can be bounded using Hasse's theorem which states that

$$p + 1 - 2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p} \ .$$

Now, we will reformulate the uniformity assumption used by [22] to make it suitable for our coding scheme and its security proof.

Assumption 7.3.2. *For an elliptic curve $E(\mathbb{F}_p)$, the valid point coordinate values are uniformly distributed, i.e. each possible value of the point coordinates appears at the output with equal probability. The minimum number of these unique coordinates is $p + 1 - 2\sqrt{p}$.*

Under this assumption, the denominator of the error masking equation will be $p + 1 - 2\sqrt{p}$. Remember that in the worst case, for a specific error pair (e_x, e_w), $\max(4, 2^k - p + 1)$ solutions can be in the check-sum space. As a result, the error masking probability for our coding scheme will be $\max(4, 2^k - p + 1) \cdot (p + 1 - 2\sqrt{p})^{-1}$.

At this point, it is also important to discuss the two biases we have in our analysis. The first bias is the difference $2^k - p$. In the first step of our proof (numerator), we show that the number of solutions to the error masking equation of our code is upper bounded by $\max(4, 2^k - p + 1)$. Note that this bound is computed by [22] in great detail. In this section, we proved that even in the worst case the error masking equation associated with our code will have less solutions than $\max(4, 2^k - p + 1)$. The crucial point is to make this difference as small as possible because it directly affects the attacker's success chance in injecting and masking a fault in our scheme. By choosing the appropriate k and p values, the designer of the circuit can reduce the number of solutions to a minimum of 4 in our scheme.

The second bias is the difference $p - \#E$. This bias indicates the difference between the size of the finite field $\mathbb{F}_p$ and the number of valid points in the elliptic curve $E(\mathbb{F}_p)$. This difference essentially affects the second step of our proof (denominator). Note that as the valid number of points in an elliptic curve decreases, the denominator of our error masking probability will get smaller. As a result, the attacker's success rate in inducing a successful fault will increase as well. Hence, to increase the performance of our coding scheme, the circuit designer needs to pick the appropriate p and curve parameters which will increase the valid number of points defined over this curve. □

Example 7.3.1. *Consider the NIST recommended prime field curve P-192. For this curve,* $(2^k - p + 1) = 18446744073709551618 \approx 2^{64}$. *In this case, according to Hasse's theorem, the number of valid points on this curve will be at least* $(p + 1 - 2\sqrt{p}) \approx 2^{192}$. *As a result, minimum error detection capability proposed by our scheme in this case will be* $1 - 2^{64}/2^{192} = 2^{-128}$.

7.4 Proposed Point Addition and Doubling Constructions

In this section, we provide the secure implementations of point addition and doubling operations for Edwards and Weierstrass curves. These implementations utilize the error detection technique that is described in Section 7.3. The main idea of the nonlinear error detection is to create two computation paths that are nonlinear to each other. As the first step to achieve this, the coordinates of the input points in an operation (point addition or doubling) are encoded using the nonlinear code described in Definition 7.3.1. One of the nonlinear paths is the original non-redundant datapath. The second path, which is called the "predictor" block, runs in parallel to the non-redundant path, and essentially predicts the check-sum of the results of the original computation. At this point, it is important to note that we do not simply replicate the original hardware to implement the predictor. Predictor block uses the fault-free check-sums of the inputs to compute the check-sum of the outputs that will be computed by the original block. For each datapath, the total operation count is expressed in terms of multiplications, divisions and addition/subtractions, where **M** stands for multiplication, **D** stands for division, and **A** stands for addition or subtraction.

7.4.1 Edwards Projective Unified Addition

As the first example, we will explain how this method can be applied to the unified addition in Edwards curves that are using projective coordinates. Essentially, the unified point addition operation computes the point P_3=(X_3, Y_3, Z_3) using the input points P_1=(X_1, Y_1, Z_1) and P_2=(X_2, Y_2, Z_2). The explicit formula that implements the point addition is shown in Equation 7.8. In the following, we show how the predictor block works. It mainly computes the expected X_{3w}, Y_{3w}, Z_{3w} using the

inputs and their check-sums. Note that the subscript w is used to specify the check-sum of an information portion for a codeword as described in Definition 7.3.1. For example, $(X_3, X_{3w}) \in C_p$ constitute a codeword where $X_{3w} = X_3^2 \pmod p$, X_3 is the information portion, and X_{3w} is the check-sum. More specifically, the expected check-sums should be

$$X_{3w} = (X_3)^2 = [Z_1 Z_2 (X_1 Y_2 + Y_1 X_2)(Z_1^2 Z_2^2 - dX_1 X_2 Y_1 Y_2)]^2$$

$$Y_{3w} = (Y_3)^2 = [Z_1 Z_2 (Y_1 Y_2 - X_1 X_2)(Z_1^2 Z_2^2 + dX_1 X_2 Y_1 Y_2)]^2$$

$$Z_{3w} = (Z_3)^2 = [(Z_1^2 Z_2^2 - dX_1 X_2 Y_1 Y_2)(Z_1^2 Z_2^2 + dX_1 X_2 Y_1 Y_2)]^2$$

Next, we need to express the terms on the right hand side as a function of the inputs and their check-sums. As an example, we will show how to achieve this for X_3. The same method can also be applied to the Y and Z coordinates as well.

$$\begin{aligned} X_{3w} &= [Z_1 Z_2 (X_1 Y_2 + Y_1 X_2)(Z_1^2 Z_2^2 - dX_1 X_2 Y_1 Y_2)]^2 \\ &= Z_1^2 Z_2^2 (X_1^2 Y_2^2 + Y_1^2 X_2^2 + 2X_1 Y_2 Y_1 X_2)(Z_1^4 Z_2^4 - 2Z_1^2 Z_2^2 dX_1 X_2 Y_1 Y_2 + d^2 X_1^2 X_2^2 Y_1^2 Y_2^2) \\ &= Z_{1w} Z_{2w} (X_{1w} Y_{2w} + Y_{1w} X_{2w} + 2\alpha)(Z_{1w}^2 Z_{2w}^2 - 2Z_{1w} Z_{2w} d\alpha + d^2 X_{1w} X_{2w} Y_{1w} Y_{2w}) \end{aligned}$$

where α=$X_1 X_2 Y_1 Y_2$. After some algebra, we get the following equation array for each coordinate of the resulting point P_3. These equations mainly represent the function implemented by the predictor unit in our design.

$\alpha = X_1 X_2 Y_1 Y_2$;

$A = Z_{1w} Z_{2w}$; $B = X_{1w} Y_{2w}$; $C = Y_{1w} X_{2w}$; $D = X_{1w} X_{2w}$;

$E = Y_{1w} Y_{2w}$; $F = d\alpha$; $G = B + C + 2\alpha$; $H = D + E - 2\alpha$;

$K = A^2 + F^2$; $L = AF$; $M = K - 2L$; $N = K + 2L$;

$X_{3w} = AGM$;

$Y_{3w} = AHN$;

$Z_{3w} = MN$;

The total operation count for this predictor unit will be $14\mathbf{M} + 7\mathbf{A}$, where $\mathbf{M}$ is multiplication and $\mathbf{A}$ is addition. Note that, all the operations in this setup are modulo p, where p is the prime that generates the finite field the elliptic curve is defined over. Similarly, in the remaining of this chapter, all the arithmetic operations are modulo p even though it is not explicitly stated in each equation.

At this point, it is important to note that $\alpha = X_1X_2Y_1Y_2$ is taken from the non-redundant datapath. One might think that the security of the system might decrease in this case because a fault injected into α on the non-redundant datapath will be automatically carried into the predictor unit as well. However, we argue that the security level of the system will not be affected due to α. The following theorem and its proof describes our reasoning.

Theorem 7.4.1. *Let $\alpha = X_1X_2Y_1Y_2$ be a computation result shared between the non-redundant datapath and the predictor unit as shown above. This computation sharing does not decrease the security of the proposed system and the error masking probability is still upper bounded by $max(4, 2^k - p + 1) \cdot (p + 1 - 2\sqrt{p})^{-1}$.*

Proof. Note that when the device operates correctly, the X_3 computed by the non-redundant datapath will have the following equation:

$$X_3 = Z_1Z_2(X_1Y_2 + Y_1X_2)(Z_1^2Z_2^2 - dX_1X_2Y_1Y_2) \pmod p$$

Assume that the attacker injected an error vector $e \in \mathbb{Z}_{2^k}$ to the portion of the circuit which computes the α. In this case, the result of this computation will be $\alpha + e$. Let $\tilde{X}_3$ be the faulty X_3 value caused by this fault injection. Then,

$$\tilde{X}_3 = Z_1Z_2(X_1Y_2 + Y_1X_2)(Z_1^2Z_2^2 - d(\alpha + e)) \pmod p$$

where $\alpha = X_1X_2Y_1Y_2$. Let $\beta = Z_1Z_2(X_1Y_2 + Y_1X_2)$. Note that

$$\begin{aligned} \tilde{X}_3 &= \beta(Z_1^2Z_2^2 - d(\alpha + e)) \pmod p \\ &= \beta(Z_1^2Z_2^2 - d\alpha) - \beta de \\ &= X_3 + e_x \end{aligned}$$

where $e_x = \beta de \in \mathbb{Z}_{2^k}$.

Note that this error will also propagate into the predictor unit as well. As a result, the next step is to analyze the effect of e on the result of the predictor unit, i.e. X_{3w}. Note that in the fault free operation

$$X_{3w} = Z_{1w}Z_{2w}(X_{1w}Y_{2w} + Y_{1w}X_{2w} + 2\alpha)(Z_{1w}^2Z_{2w}^2 - 2Z_{1w}Z_{2w}d\alpha + d^2\alpha^2).$$

Let $\tilde{X}_{3w}$ be the faulty check-sum result due to e that is transmitted through the shared α. Then

$$\begin{aligned}
\tilde{X}_{3w} &= Z_{1w}Z_{2w}(X_{1w}Y_{2w} + Y_{1w}X_{2w} + 2(\alpha + e))(Z_{1w}^2Z_{2w}^2 - 2Z_{1w}Z_{2w}d(\alpha + e) + d^2(\alpha + e)^2) \\
&= Z_{1w}Z_{2w}([X_{1w}Y_{2w} + Y_{1w}X_{2w} + 2\alpha] + 2e)([Z_{1w}^2Z_{2w}^2 - 2Z_{1w}Z_{2w}d\alpha + d^2\alpha^2] + 2d^2\alpha e + \\
&\quad d^2e^2 + 2Z_{1w}Z_{2w}de) \\
&= X_{3w} + e_w
\end{aligned}$$

where $e_w \in \mathbb{Z}_{2^k}$ is the addition of all extra terms that appear in this equation. Note that the final check that is implemented by the error detection network will be as following:

$$(X_3 + e_x \bmod 2^k)^2 \bmod p = X_{3w} + e_w \bmod 2^k,$$

which is identical to the error masking equation we solved in Theorem 7.3.1. Hence, the error masking probability of the system is still bounded by $\max(4, 2^k - p + 1) \cdot (p + 1 - 2\sqrt{p})^{-1}$. Note that a similar analysis also applies to Y_3 and Z_3 as well. However, due to space considerations, we will not provide the details for these cases. □

In terms of efficiency, pulling $\alpha = X_1X_2Y_1Y_2$ from the non-redundant datapath is very beneficial, since we do not have to compute this value in the predictor datapath. Moreover, to be able to compute this value in the predictor datapath using the inputs X_{1w}=$X_1^2 \bmod p$, Y_{1w}=$Y_1^2 \bmod p$, X_{2w}=$X_2^2 \bmod p$, and Y_{2w}=$Y_2^2 \bmod p$, we would have to implement modular square-root operation. Modular square-root is a very costly operation; therefore, we chose to take this value from the non-redundant datapath.

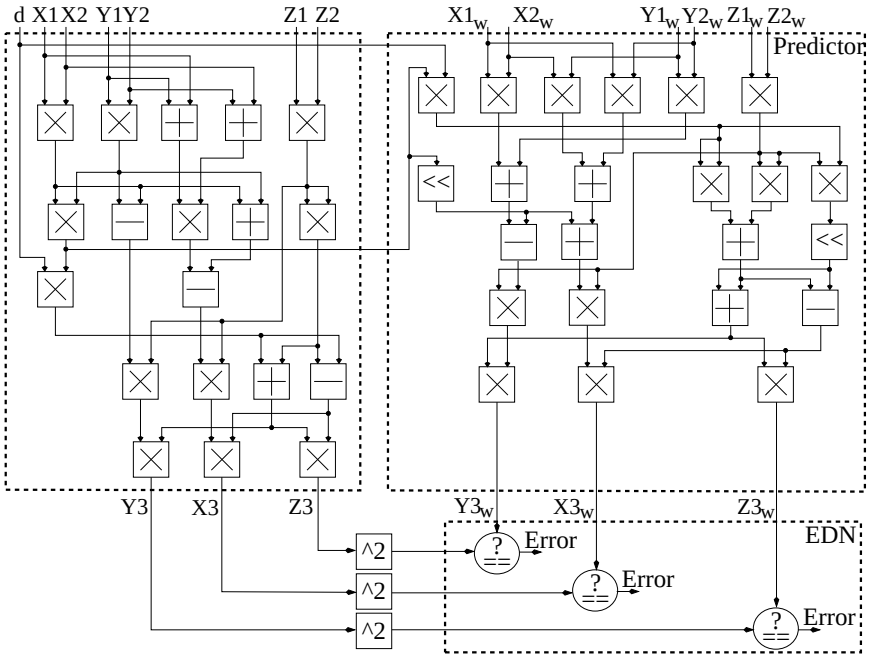

Figure 7.1: Secure Edwards projective unified point addition

In addition, to ease the security analysis of our scheme, we conduct the security checks at the outputs of the non-redundant datapath and the predictor. This is achieved by reflecting the effect of each fault injection to the outputs. This is a reasonable assumption because otherwise, one would need to write all possible error detection equations that would reflect to the outputs. However, this requires exponentially many equations and makes the security analysis extremely difficult.

The hardware implementation of this technique is shown in Figure 7.1. In this figure, the block on the left is the original, non-redundant datapath that computes the unified point addition. The predictor block mainly implements the X_{3w}, Y_{3w}, and Z_{3w} computations defined above. Next, the output coordinates are squared to compute their check-sums. Finally, the error detection network (EDN) compares the results of these two paths. If all the results match, this means that the conducted operation is fault free. However, if there is a mismatch in any one of the coordinate comparisons, this points to an injected fault. Hence, an error signal is asserted. Once

the error signal is asserted, either the secret can be flushed or the device can be reset.

Example 7.4.1. *Consider the following Edwards elliptic curve*

$$E(\mathbb{F}_p) \,:\, X^2 + Y^2 = Z^4 + dX^2Y^2$$

in homogeneous projective coordinates with 160-bit parameters where $c = 1$*,*

$$\begin{aligned} p &= 904237709503056132134073013251929604353336630099, \\ d &= 654173458343505972456671157852824043318841457169. \end{aligned}$$

Let $P_1{=}(X_1, X_{1w}, Y_1, Y_{1w}, Z_1, Z_{1w},)$ *and* $P_2{=}(X_2, X_{2w}, Y_2, Y_{2w}, Z_2, Z_{2w})$ *be projective representation of two points on this elliptic curve with*

$$\begin{aligned} X_1 &= 870127788413360698968854496841390328966899692865, \\ Y_1 &= 574645271602872238900537148484954656697831740108, \\ Z_1 &= 345690248650413035085013184802176684592198761837, \\ X_2 &= 451150124911200084467566472191656051656543909608, \\ Y_2 &= 632698684792777393199017690844497399704298703004, \\ Z_2 &= 396991703048768867254892739709047646980786148617, \\ X_{1w} &= 182483089008170957111891478528504386053106733877, \\ Y_{1w} &= 134813777183292608845385299789608703898099046583, \\ Z_{1w} &= 282113222159106554505926145024027727198697475921, \\ X_{2w} &= 420952235496534850961769608406544657885631784595, \\ Y_{2w} &= 710441061377976621430641814219746918373154886080, \\ Z_{2w} &= 749997069021309398716883073899546932396100433538. \end{aligned}$$

Then, the Edwards point addition computes $P_3{=}P_1 + P_2{=}(X_3, X_{3w}, Y_3, Y_{3w}, Z_3, Z_{3w})$

in $E(\mathbb{F}_p)$ *as:*

$$
\begin{aligned}
X_3 &= 2826070446337215353564348988876823426624634348 85, \\
Y_3 &= 1514377331013004395667830135110049455097019139 43, \\
Z_3 &= 6762808584941458902162576395894408588743707554 86, \\
X_{3w} &= 8867206157712426351589180433162205178993621084 25, \\
Y_{3w} &= 5397733325760478664390504831582964202235331281 15, \\
Z_{3w} &= 2947913160236605039730464972388881784223495792 42.
\end{aligned}
$$

7.4.2 Weierstrass Affine Addition and Doubling

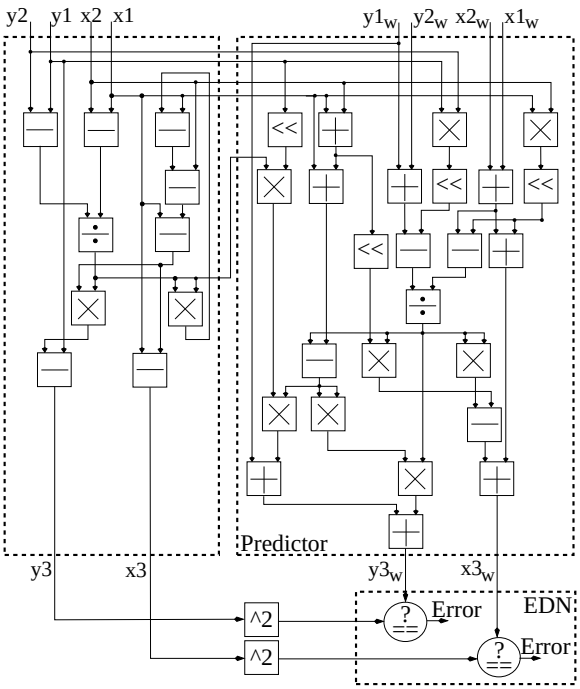

Figure 7.2: Secure Weierstrass affine point addition

In this section, we describe how the same approach can be applied to Weierstrass curves in affine coordinates. In the following we provide secure implementations of point addition and doubling using Weierstrass formulation. In order to show that our

approach can also be applied to different coordinate systems in a similar manner, we pick the affine coordinates this time.

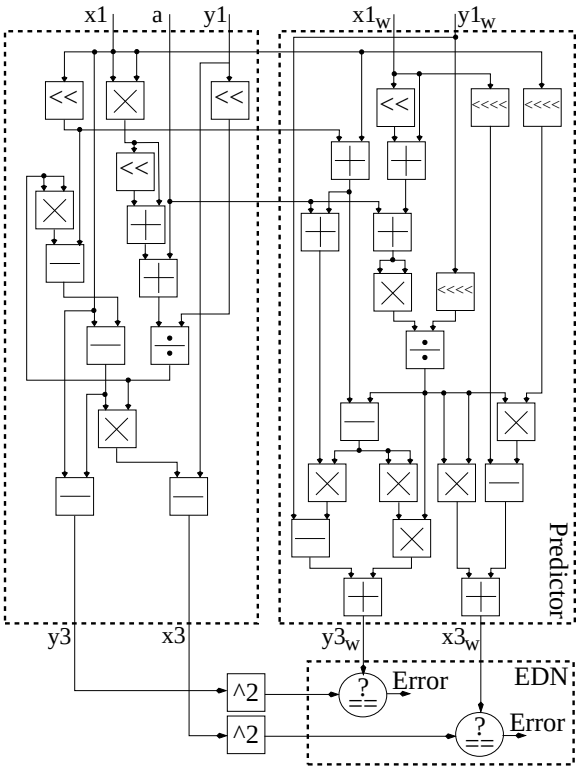

Figure 7.3: Secure Weierstrass affine point doubling

Using the same methodology discussed in Section 7.4.1, we first compute the following predictor functions for point addition.

$\alpha = \frac{y_2 - y_1}{x_2 - x_1}$;

$A = y_1 y_2$; $\quad B = x_1 x_2$; $\quad C = x_1 + x_2$; $\quad D = x_1 + C$;

$E = \frac{y_{2w} + y_{1w} - 2A}{x_{2w} + x_{1w} - 2B}$; $\quad F = E - D$; $\quad G = x_{2w} + x_{1w} + 2B$;

$x_{3w} = E^2 + G - 2CE$;

$y_{3w} = EF^2 + y_{1w} + 2y_1 \alpha F$;

where $\alpha = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)$ is obtained from the non-redundant datapath. Total operation

count for this predictor unit is $8\mathbf{M} + 13\mathbf{A} + 1\mathbf{D}$, where $\mathbf{D}$ is used to indicate division.

A similar analysis for the point doubling produces the following predictor unit.

$$A = \frac{(3x_{1w}+a)^2}{4y_{1w}}; \quad B = x_1 A; \qquad C = 3x_1;$$
$$D = C - A; \qquad E = C + a;$$

$$x_{3w} = A^2 + 4x_{1w} - 4B;$$
$$y_{3w} = AD^2 + y_{1w} - DE;$$

In this case, the total operation count for the predictor becomes $6\mathbf{M} + 9\mathbf{A} + 1\mathbf{D}$.

We show the hardware implementations of these secure operations in Figures 7.2 and 7.3. The main blocks used in these figures implement the same functions as in Figure 7.1.

Example 7.4.2. *Consider the following Weierstrass elliptic curve*

$$E(\mathbb{F}_p) \; : \; y^2 = x^3 + ax + b$$

in affine coordinates with 160-bit parameters where

$$\begin{aligned} p &= 777645715940287777910005789305681810797687415809, \\ a &= 567500170552851441844054828293673707780907112235, \\ b &= 703455979504253314740690805725708046205894842210. \end{aligned}$$

Let $P_1 = (x_1, x_{1w}, y_1, y_{1w})$ *be a point on this elliptic curve with*

$$\begin{aligned} x_1 &= 24521122072027183747660901827345179826975892027 2, \\ y_1 &= 48824646576197515542209490050440158857272439005 2, \\ x_{1w} &= 69508556806391595977548408290103553586210631327 5, \\ y_{1w} &= 70293328921427552718062963363026722030435676726. \end{aligned}$$

The Weierstrass point doubling unit computes $P_2 = 2P_1 = (x_2, x_{2w}, y_2, y_{2w})$ *in* $E(\mathbb{F}_p)$ *as:*

$$\begin{aligned} x_2 &= 728609857396342373439877711694115269199147865869, \\ y_2 &= 532357780097897790333060514664411836014525549756, \\ x_{2w} &= 576489689937230253513047354510609128256109419771, \\ y_{2w} &= 333980457248136285466037412958831355718690903835. \end{aligned}$$

Due to space considerations, results of the similar analysis for the Weierstrass Jacobian addition, Weierstrass Jacobian doubling, and Edwards affine unified addition are presented in appendix B. We provide the predictor functions associated with each of these cases and compute the total operation counts for their predictor units.

7.5 Results and Discussion

The area overhead caused by the application of our scheme is dependent on the areas of arithmetic unit implementations in a particular system. Without knowing the relative area ratios of division, multiplication and addition/subtraction units, it is not possible to provide an exact overhead measure. However, given the higher complexity of divisions and multiplications with respect to additions/subtractions, it is reasonable to ignore additions and subtractions to obtain an estimation of the overhead. Also, we assume that the area of a multiplication unit is on the order of a division unit. This is a reasonable assumption because it does not make sense to have an affine system where the area of a divider is much larger than the area of a multiplier.

Having made these assumptions, the estimated percentage overheads of the non-linear error detection scheme we propose for point operations are presented in Table 7.1. Note that this table provides results for both Weierstrass and Edwards curves

Table 7.1: Overhead analysis for the error detection technique proposed in this chapter

ECC System	Non-Redundant Datapath	Our Predictor Datapath	Our Complete Datapath	Our Estimated % Overhead
Weier. Affine Point Addition	1**D** + 2**M** + 6**A**	1**D** + 8**M** + 13**A**	2**D** + 10**M** + 19**A**	**300%**
Weier. Affine Point Doubling	1**D** + 3**M** + 5**A**	1**D** + 6**M** + 9**A**	2**D** + 9**M** + 14**A**	**175%**
Weier. Jacobian Point Addition	16**M** + 7**A**	27**M** + 13**A**	43**M** + 20**A**	**169%**
Weier. Jacobian Point Doubling	10**M** + 5**A**	13**M** + 8**A**	23**M** + 13**A**	**130%**
Edwards Affine Unified Point Addition	2**D** + 5**M** + 7**A**	2**D** + 6**M** + 8**A**	4**D** + 11**M** + 15**A**	**114%**
Edwards Projective Unified Point Addition	12**M** + 7**A**	14**M** + 7**A**	26**M** + 14**A**	**117%**

for different coordinate systems. We observe that the Weierstrass based elliptic curve systems can be protected with reasonable area overhead. Note that the application of our scheme cause 175%, 169%, 130% for the affine point doubling, Jacobian point addition, and Jacobian point doubling operations, respectively. The worst case for securing Weierstrass operations is the affine point addition which causes an area overhead of 300%. In addition, we also observe that the predictor unit for point additions incur more overhead than point doublings in Weierstrass systems. This is also an expected result since point addition operations are more complex than point doubling operations.

On the other hand, the balanced normal form of Edwards formulation provides simpler predictor designs with less overhead when compared to the Weierstrass systems. In other words, the Edwards formulation is more appropriate for the non-linear error detection technique that is proposed in this chapter. The secure unified point addition in affine coordinates require an overhead of 114% while in projective coordinates, the overhead ratio is 117%.

Table 7.2: Operation counts of robust modular arithmetic functions proposed in [22]

Operation	Non-Redundant Datapath	Predictor Datapath
Robust Addition	1**A**	1**M** + 3**A**
Robust Multiplication	1**M**	2**M** + 2**A**
Robust Division	1**D**	1**D**

In the proposed error detection technique for robust public key arithmetic [22], Gaubatz et al. designed robust arithmetic units (i.e. adders, multipliers, etc.) that work over integers. These are individually secure arithmetic units which can be utilized to design secure public key crypto systems. Note that this is a generic approach which targets the elementary modular arithmetic operations and it can be applied to any modular arithmetic based system. In order to implement this security measure for their system, designers need to replace all the arithmetic units in their design with their robust counterparts proposed by [22]. The elliptic curve point addition and doubling operations presented in Section 7.2.1 are also based on modular arithmetic operations. Hence, the technique described in [22] can also be applied to implement secure ECC point operations. By using the individually robust arithmetic units proposed by [22] as the building blocks of the ECC system, approximately the same level of security we propose in this chapter can be achieved for ECC point operations. However, due to its generic nature, this solution causes high overhead when applied to public key algorithms.

Table 7.3: Overhead analysis for the error detection technique proposed in [22]

ECC System	Non-Redundant Datapath	Predictor Datapath of [22]	Complete Datapath of [22]	Estimated % Overhead of [22]
Weier. Affine Point Addition	1**D** + 2**M** + 6**A**	1**D** + 10**M** + 22**A**	2**D** + 12**M** + 28**A**	**367%**
Weier. Affine Point Doubling	1**D** + 3**M** + 5**A**	1**D** + 11**M** + 21**A**	2**D** + 14**M** + 26**A**	**300%**
Weier. Jacobian Point Addition	16**M** + 7**A**	39**M** + 53**A**	55**M** + 60**A**	**244%**
Weier. Jacobian Point Doubling	10**M** + 5**A**	25**M** + 35**A**	35**M** + 40**A**	**250%**
Edwards Affine Unified Point Addition	2**D** + 5**M** + 7**A**	2**D** + 17**M** + 31**A**	4**D** + 22**M** + 38**A**	**271%**
Edwards Proj. Unified Point Addition	12**M** + 7**A**	31**M** + 45**A**	43**M** + 52**A**	**258%**

On the other hand, our solution mainly focuses on the elliptic curve point addition or doubling operations. We target these operations as a whole entity. Consequently, we design robust elliptic curve point operations by implementing a predictor circuitry for the complete point operation instead of targeting individual arithmetic operations. In other words, the approach we propose is optimized for elliptic curve point operations. This essentially causes a considerable overhead reduction with approximately the same level of security.

In order to compare the area overhead of our scheme with the overhead caused by the scheme presented in [22], we computed the predictor overheads for each ECC point

operation using the error detection scheme of [22]. Table 7.2 shows the non-redundant datapath and predictor arithmetic operation counts for the robust modular arithmetic functions proposed in [22]. As is shown in this table, a robust arithmetic unit consists of two main parts: 1) non-redundant datapath and 2) predictor. Essentially, each individual robust arithmetic unit shown in this table, provide the security level of the proposed non-linear code in [22]. Note that, one way of applying this security scheme to elliptic curve point operations (which we cover in this chapter) is to replace all the arithmetic units with the robust arithmetic units shown in this table. For example, the non-redundant path of the Edwards projective unified point addition shown in Figure 7.1 has 12**M** and 7**A**. In order to apply their non-linear error detection scheme, Gaubatz et al. in [22] propose replacing each of these arithmetic units with their robust counterparts. In this case, each multiplication and addition operation will have the predictor overheads shown in Table 7.2. Hence, the robust Edwards unified point addition will have $12\times(2\mathbf{M} + 2\mathbf{A}) = 24\mathbf{M} + 24\mathbf{A}$ in the robust multiplier predictors and $7\times(1\mathbf{M} + 3\mathbf{A})=7\mathbf{M} + 21\mathbf{A}$ in the robust adder predictors. In total, the robust version of the Edwards unified point addition proposed by [22] will have $31\mathbf{M} + 45\mathbf{A}$ in the predictors. Similarly, for the Weierstrass affine point addition shown in Figure 7.2 which has $1\mathbf{D} + 2\mathbf{M} + 6\mathbf{A}$ in the non-redundant datapath, the technique presented in [22] will have a total of $1\mathbf{D} + 2\times(2\mathbf{M} + 2\mathbf{A}) + 6\times(1\mathbf{M} + 3\mathbf{A}) = 1\mathbf{D} + 10\mathbf{M} + 22\mathbf{A}$ in the predictors. By applying this approach for each ECC system, we obtain the predictor overheads of [22] as presented in the third column (Predictor Datapath) of Table 7.3. We also present the operation count of the complete datapath (sum of the operation counts in non-redundant and predictor datapaths), and the estimated overhead percentage. Note that we ignore the effect of adders for the overhead percentage computation as we did in the overhead percentage computation of our scheme. However, as it can be observed in the fourth column (Complete Datapath) of Table 7.3, the number of adders reach to non-negligible

levels for this case.

Table 7.4: Estimated overhead comparison

Estimated % Overhead	Our Approach	Gaubatz et al. 2006 [22]	Ratio of our approach to [22]
Weierstrass Affine Point Addition	**300%**	**367%**	**82%**
Weierstrass Affine Point Doubling	**175%**	**300%**	**58%**
Weierstrass Jacobian Point Addition	**169%**	**244%**	**69%**
Weierstrass Jacobian Point Doubling	**130%**	**250%**	**52%**
Edwards Affine Unified Point Addition	**114%**	**271%**	**42%**
Edwards Projective Unified Point Addition	**117%**	**258%**	**45%**

Even when we ignore the impact of adders on their overhead computation, our technique provides approximately the same level of security and robustness with much less overhead. For Edwards curves, the overhead of our scheme is less than the half (42%-46%) of the overhead of [22]. In addition, the overhead of our scheme is 52% to 81% of the overhead of [22] for different versions of the Weierstrass curves. The overall summary of the estimated overhead percentages and their comparison is presented in Table 7.4.

Furthermore, remember that in all of these secure point addition and doubling operations, the utilized error detection technique provides an error detection capability that is nearly perfect (except with exponentially small probability).

7.6 Summary

In this chapter, we conducted a comprehensive analysis about integrating nonlinear error detection techniques to ECC. We propose the application of systematic nonlinear error detection codes to protect elliptic curve point addition and doubling operations against active fault attacks. These codes provide nearly perfect error detection capability (except with exponentially small probability) at reasonable overhead. We observe that the Weierstrass based elliptic curve systems can be protected with reasonable area overhead. However, the balanced normal form of Edwards formulation provides simpler predictor designs with less overhead. In other words, Edwards formulation is more appropriate for the non-linear error detection technique that is proposed in this chapter. We also compared our error detection technique with the method proposed by [22]. Results indicate that our technique provides approximately the same level of security and robustness with much less overhead. For Edwards curves, the overhead of our scheme is less than half (42%-46%) of the overhead of [22]. In addition, the overhead of our scheme is 52% to 81% of the overhead of [22] for different versions of the Weierstrass curves.

Chapter 8

Conclusion

"Side channel" attacks (SCA) pose a serious threat on many cryptographic devices and are shown to be effective on many existing security algorithms which are in the black box model considered to be secure. These attacks are based on the key idea of recovering secret information using implementation specific side-channels. Especially active fault injection attacks are very effective in terms of breaking otherwise impervious cryptographic schemes.

Various countermeasures have been proposed to provide security against these attacks. Double-Data-Rate (DDR) computation, dual-rail encoding, and simple concurrent error detection (CED) are the most popular of these solutions. Even though these security schemes provide sufficient security against weak adversaries, they can be broken relatively easily by a more advanced attacker. In this dissertation, we proposed various error detection techniques that target strong adversaries with advanced fault injection capabilities.

We first described the "advanced attacker" in detail and provided its characteristics. As part of this definition, we provided a generic metric to measure the strength of an adversary.

Next, we discussed various techniques for protecting finite state machines (FSMs)

of cryptographic devices against active fault attacks. These techniques mainly depend on nonlinear robust codes and physically unclonable functions (PUFs). We showed that due to the nonuniform behavior of FSM variables, securing FSMs using nonlinear codes is an important and difficult problem. As a solution to this problem, we proposed error detection techniques based on nonlinear codes with different randomization methods. We also showed how PUFs can be utilized to protect a class of FSMs. This solution provides security on the physical level as well as the logical level. In addition, for each technique, we provided possible hardware realizations and discussed area/security performance.

Bibliography

[1] Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/TEMPEST`.

[2] K. D. Akdemir, G. Hammouri, and B. Sunar. Non-linear Error Detection for Finite State Machines. *Proceedings of The 10th International Workshop on Information Security Applications, Busan, Korea*, 5932:226–238, 2009.

[3] R. J. Anderson and M. G. Kuhn. Tamper Resistance–a Cautionary Note. *The Second USENIX Workshop on Electronic Commerce, Oakland, CA, USA*, 2:1–11, 1996.

[4] R.J. Anderson and M. Kuhn. Low cost attacks on tamper resistant devices. *Proceedings of the 5th International Workshop on Security Protocols, Paris, France*, 1361:125–136, 1997.

[5] D. Asonov and R. Agrawal. Keyboard acoustic emanations. In *IEEE Symposium on Security and Privacy*, pages 3–11. Citeseer, 2004.

[6] Bar-El H. and Choukri H. and Naccache D. and Tunstall. M. and Whelan C. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94:370–382, 2006.

[7] M. Berg. Fault tolerant design techniques for asynchronous single event upsets within synchronous finite state machine architectures. In *7th International Mili-*

tary and Aerospace Programmable Logic Devices (MAPLD) Conference. NASA, Sep 2004.

[8] D.J. Bernstein and T. Lange. Faster Addition and Doubling on Elliptic Curves. *International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology, Kuching, Malaysia*, 4833:29–50, 2007.

[9] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. *IEEE Transactions on Computers*, 52(4):492–505, 2003.

[10] I. Biehl, B. Meyer, and V. Muller. Differential fault attacks on elliptic curve cryptosystems. *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, California, USA*, 1880:131–146, 2000.

[11] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, California, USA*, 1294:513–525, 1997.

[12] J. Blomer, M. Otto, and J.P. Seifert. Sign change fault attacks on elliptic curve cryptosystems. *Proceedings of the 2nd Workshop on Fault Diagnosis and Tolerance in Cryptography, Edinburgh, Scotland, UK*, 4236:25–40, 2005.

[13] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy, editor, *Advances in Cryptology - EuroCrypt'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51, Heidelberg, 1997. Springer. Proceedings.

[14] D.V. Chudnovsky and G.V. Chudnovsky. Sequences of Numbers Generated by Addition in Formal Groups and New Primality and Factorization Tests. *Advances in Applied Mathematics*, 7(4):385–434, 1986.

[15] M. Ciet and M. Joye. Elliptic curve cryptosystems in the presence of permanent and transient faults. *Designs, Codes and Cryptography*, 36(1):33–43, 2005.

[16] P. Cunningham, R. Anderson, R. Mullins, G. Taylor, and S. Moore. Improving Smart Card Security Using Self-Timed Circuits. In *Proceedings of the 8th International Symposium on Asynchronus Circuits and Systems*. IEEE Computer Society Washington, DC, USA, 2002.

[17] E. De Mulder, P. Buysschaert, SB Ors, P. Delmotte, B. Preneel, G. Vandenbosch, and I. Verbauwhede. Electromagnetic analysis attack on an FPGA implementation of an elliptic curve cryptosystem. In *Proceedings of the International Conference on Computer as a tool (EUROCON)*, pages 21–24. Citeseer, 2006.

[18] A. Dominguez-Oviedo and M.A. Hasan. Error Detection and Fault Tolerance in ECSM Using Input Randomization. *IEEE Transactions on Dependable and Secure Computing*, 6(3):175–187, 2009.

[19] H.M. Edwards. A Normal Form for Elliptic Curves. *Bulletin of the American Mathematical Society*, 44(3):393–422, 2007.

[20] J. Francq, J.B. Rigaud, P. Manet, A. Tria, and A. Tisserand. Error Detection for Borrow-Save Adders Dedicated to ECC Unit. *Proceedings of the 5th Workshop on Fault Diagnosis and Tolerance in Cryptography, Washington DC, USA*, pages 77–86, 2008.

[21] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Delay-based Circuit Authentication and Applications. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 294–301, 2003.

[22] G. Gaubatz, B. Sunar, and M.G. Karpovsky. Non-linear residue codes for robust public-key arithmetic. *Proceedings of the 3rd Workshop on Fault Tolerance and Diagnosis in Cryptography Yokohama, Japan*, 4236:173–184, 2006.

[23] Gunnar Gaubatz and Berk Sunar. Robust finite field arithmetic for fault-tolerant public-key cryptography. In Luca Breveglieri and Israel Koren, editors, *2nd Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2005*, September 2005.

[24] Gunnar Gaubatz, Berk Sunar, and Erkay Savas. Sequential circuit design for embedded cryptographic applications resilient to adversarial faults. *IEEE Transactions on Computers*, 57(1):126–138, 2008.

[25] Jorge Guajardo, Sandeep S. Kumar, Geert-Jan Schrijen, and Pim Tuyls. Fpga intrinsic pufs and their use for ip protection. In *CHES '07: Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, pages 63–80, Berlin, Heidelberg, 2007. Springer-Verlag.

[26] G. Hammouri, K. Akdemir, and B. Sunar. Novel PUF-Based Error Detection Methods in Finite State Machines. *Lecture Notes In Computer Science*, pages 235–252, 2009.

[27] Ghaith Hammouri, Erdinç Öztürk, and Berk Sunar. A tamper-proof and lightweight authentication scheme. *Pervasive Mob. Comput.*, 4(6):807–818, 2008.

[28] Ghaith Hammouri and Berk Sunar. PUF-HB: A Tamper-Resilient HB based Authentication Protocol. In *to appear in Proceedings of the Applied Cryptography and Network Security Conference – ACNS08*, 2008.

[29] M. Joye. Highly Regular Right-to-Left Algorithms for Scalar Multiplication. *LECTURE NOTES IN COMPUTER SCIENCE*, 4727:135, 2007.

[30] M. Joye and S.M. Yen. The Montgomery Powering Ladder. *Cryptographic Hardware and Embedded Systems-Ches 2002: 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002: Revised Papers*, 2002.

[31] D. Karakoyunlu, F. Gurkaynak, B. Sunar, and Y. Leblebici. Efficient and Side-Channel-Aware Implementations of ECC Cryptosystems over Prime Fields . *Information Security, IET*, 4(1):30–43, 2010.

[32] Mark Karpovsky, Konrad Kulikowski, and Alexander Taubin. Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard. *Proceedings of the 2004 International Conference on Dependable Systems and Networks, Florance, Italy*, page 93, 2004.

[33] Mark Karpovsky, Konrad J. Kulikowski, and Alexander Taubin. Differential fault analysis attack resistant architectures for the advanced encryption standard. *Smart Card Research and Advanced Applications VI*, 153:177–192, 2004.

[34] Mark Karpovsky and Alexander Taubin. A new class of nonlinear systematic error detecting codes. *IEEE Trans Info Theory*, 50(8):1818–1820, 2004.

[35] Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim. Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 21(12):1509–1517, 2002.

[36] Osman Kocar. Estimation of keys stored in cmos cryptographic device after baking by using the charge shift. Cryptology ePrint Archive, Report 2007/134, 2007. http://eprint.iacr.org/.

[37] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. *Advances in Cryptology-Crypto'99: 19th Annual International Cryptology Conference, Santa Barbara, California, USA August 15-19, 1999 Proceedings*, 1999.

[38] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 104–113, London, UK, 1996. Springer-Verlag.

[39] Andrzej Krasniewski. Concurrent error detection in sequential circuits implemented using fpgas with embedded memory blocks. In *Proceedings of the 10th IEEE International On-Line Testing Symposium (IOLTS'04)*, 2004.

[40] M.G. Kuhn. Optical time-domain eavesdropping risks of CRT displays. In *IEEE Symposium on Security and Privacy*, pages 3–18. Citeseer, 2002.

[41] M.G. Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. *Lecture Notes in Computer Science*, 3424:88–107, 2005.

[42] M.G. Kuhn. Security limits for compromising emanations. *Lecture notes in computer science*, 3659:265, 2005.

[43] K.J. Kulikowski, V. Venkataraman, Z. Wang, A. Taubin, and M. Karpovsky. Asynchronous balanced gates tolerant to interconnect variability. In *IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008*, pages 3190–3193, 2008.

[44] K.J. Kulikowski, Z. Wang, and M.G. Karpovsky. Comparative Analysis of Robust Fault Attack Resistant Architectures for Public and Private Cryptosystems. *Proceedings of the 5th Workshop on Fault Diagnosis and Tolerance in Cryptography, Washington DC, USA*, pages 41–50, 2008.

[45] Konrad Kulikowski, Mark Karpovsky, and Alexander Taubin. Robust codes for fault attack resistant cryptographic hardware. *Proceedings of the 2nd Workshop on Fault Tolerance and Diagnosis in Cryptography, Edinburgh, Scotland, UK*, 4236:1–12, 2005.

[46] Konrad J. Kulikowski, Mark Karpovsky, and Alexander Taubin. Fault attack resistant cryptographic hardware with uniform error detection. *Proceedings of the 3rd Workshop on Fault Tolerance and Diagnosis in Cryptography, Yokohama, Japan*, 4236:185–195, 2006.

[47] J. W. Lee, L. Daihyun, B. Gassend, G. E. Suh amd M. van Dijk, and S. Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *Symposium of VLSI Circuits*, pages 176–179, 2004.

[48] A.K. Lenstra and E.R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14:255–293, 2001.

[49] Daihyun Lim, Jae W. Lee, Blaise Gassend, G. Edward Suh, Marten van Dijk, and Srinivas Devadas. Extracting secret keys from integrated circuits. *IEEE Trans. VLSI Syst.*, 13(10):1200–1205, 2005.

[50] P. Maistri and R. Leveugle. Double-Data-Rate Computation as a Countermeasure against Fault Analysis. *IEEE Transactions on Computers*, 57(11):1528–1539, 2008.

[51] P. Maistri, P. Vanhauwaert, and R. Leveugle. Evaluation of Register-Level Protection Techniques for the Advanced Encryption Standard by Multi-Level Fault Injections. In *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems, 2007. DFT'07*, pages 499–507, 2007.

[52] A. Matthews. Low cost attacks on smart cards: The electromagnetic side-channel. *Next Generation Security Software, Sept*, 2006.

[53] Subhasish Mitra and Edward J. McCluskey. Which concurrent error detection scheme to choose? In *Proc. Int. Test Conference (ITC)*, pages 985–994. IEEE, IEEE Press, 2000.

[54] David Naccache. Finding faults. *IEEE Security and Privacy, Oakland, California, USA*, 3(5):61–65, 2005.

[55] Erdinc Ozturk, Ghaith Hammouri, and Berk Sunar. Towards robust low cost authentication for pervasive devices. In *PERCOM '08: Proceedings of the Sixth IEEE International Conference on Pervasive Computing and Communications*, 2008.

[56] S. Pontarelli, G.C. Cardarilli, M. Re, and A. Salsano. Error detection in addition chain based ecc point multiplication. *IEEE International On-Line Testing Symposium, Sesimbra-Lisbon, Portugal*, 0:192–194, 2009.

[57] R. Posch. Protecting Devices by Active Coating. *Journal of Universal Computer Science*, 4(7):652–668, 1998.

[58] P.S. Ravikanth. *Physical One-Way Functions*. PhD thesis, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2001.

[59] Cornelis Roos, Tamas Terlaky, and Jean-Philippe Vial. *Interior Point Methods for Linear Optimization*. Springer, second edition, 2005.

[60] J.M. Schmidt and M. Hutter. Optical and em fault-attacks on crt-based rsa: Concrete results. *Austrochip '07: Proceedings of the 15th Austrian Workshop on Microelectronics, Graz, Austria*, pages 61–67, 2007.

[61] A. Shamir and E. Tromer. Acoustic cryptanalysis: on nosy people and noisy machines. *Online at http://people. csail. mit. edu/tromer/acoustic*.

[62] B. Skoric, S. Maubach, T. Kevenaar, and P. Tuyls. Information-theoretic Analysis of Coating PUFs. Cryptology ePrint Archive, Report 2006/101, 2006.

[63] S.P. Skorobogatov and R.J. Anderson. Optical Fault Induction Attacks. *Cryptographic Hardware and Embedded Systems: 4th International Workshop, Redwood Shores, CA, USA*, 2523:2–12, 2002.

[64] Danil Sokolov, Julian Murphy, Alexandre V. Bystrov, and Alexandre Yakovlev. Design and analysis of dual-rail circuits for security applications. *IEEE Trans. Computers*, 54(4):449–460, 2005.

[65] R. Stern, N. Joshi, K. Wu, and R. Karri. Register Transfer Level Concurrent Error Detection in Elliptic Curve Crypto Implementations. *Proceedings of the 4th Workshop on Fault Diagnosis and Tolerance in Cryptography, Vienna, Austria*, pages 112–119, 2007.

[66] P. Tuyls, G.J. Schrijen, B. Skoric, J. van Geloven, N. Verhaegh, and R. Wolters. Read-Proof Hardware from Protective Coatings. *Cryptographic Hardware and Embedded SystemsCHES*, pages 10–13, 2006.

[67] P. Tuyls and B. Skoric. Secret Key Generation from Classical Physics: Physical Uncloneable Functions. In S. Mukherjee, E. Aarts, R. Roovers, F. Widdershoven, and M. Ouwerkerk, editors, *AmIware: Hardware Technology Drivers of Ambient Intelligence*, volume 5 of *Philips Research Book Series*. Springer-Verlag, Sep 2006.

[68] Jason Waddle and David Wagner. Fault attacks on dual-rail encoded systems. In *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*, pages 483–494, Washington, DC, USA, 2005. IEEE Computer Society.

[69] L. Zhuang, F. Zhou, and JD Tygar. Keyboard acoustic emanations revisited. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 373–382. ACM New York, NY, USA, 2005.

Appendix A

Non-redundant Datapath Formulas in Dataflow Format

Weierstrass Affine Addition Non-redundant Datapath:

$A = y_2 - y_1; \quad B = x_2 - x_1; \quad C = \frac{A}{B}; \quad D = x_1 - x_3;$

$x_3 = C^2 - x_1 - x_2;$

$y_3 = CD - y_1;$

Total operation count: $2\mathbf{M} + 6\mathbf{A} + 1\mathbf{D}$.

Weierstrass Affine Doubling Non-redundant Datapath:

$A = 3x_1^2 + a; \quad B = \frac{A}{2y_1}; \quad C = x_1 - x_3;$

$x_3 = B^2 - 2x_1;$

$y_3 = BC - y1;$

Total operation count: $3\mathbf{M} + 5\mathbf{A} + 1\mathbf{D}$.

Weierstrass Jacobian Addition Non-redundant Datapath:

$A = X_1 Z_2^2; \quad B = X_2 Z_1^2; \quad C = Y_1 Z_2^3; \quad D = Y_2 Z_1^3;$
$E = Z_1 Z_2; \quad F = B - A; \quad G = B + A; \quad H = D - C;$
$K = D + C; \quad L = F^2 G; \quad M = F^3 K; \quad N = L - 2X_3;$

$X_3 = H^2 - L;$
$Y_3 = \frac{HN - M}{2};$
$Z_3 = EF;$

Total operation count: $16\mathbf{M} + 7\mathbf{A}$.

Weierstrass Jacobian Doubling Non-redundant Datapath:

$A = 3X_1^2 + aZ_1^4; \quad B = Y_1^2 \quad C = X_1 B; \quad D = 4C - X_3;$

$X_3 = A^2 - 8C;$
$Y_3 = AD - 8B^2;$
$Z_3 = 2Y_1 Z_1;$

Total operation count: $10\mathbf{M} + 5\mathbf{A}$.

Edwards Affine Unified Addition Non-redundant Datapath:

$A = x_1 + y_1; \quad B = x_2 + y_2; \quad C = x_1 x_2; \quad D = y_1 y_2;$
$E = AB - C - D; \quad F = D - C; \quad G = dCD; \quad H = 1 - G;$
$K = 1 + G;$

$x_3 = \frac{E}{K};$
$y_{3w} = \frac{F}{H};$

Total operation count: $5\mathbf{M} + 7\mathbf{A} + 2\mathbf{D}$.

Edwards Projective Unified Addition Non-redundant Datapath:

$A = Z_1 Z_2$; $B = X_1 + Y_1$; $C = X_2 + Y_2$; $D = X_1 X_2$;
$E = Y_1 Y_2$; $F = BC - D - E$; $G = E - D$; $H = dED$;
$K = A^2 + H$; $L = A^2 - H$;

$X_3 = AFL$;
$Y_{3w} = AGK$;
$Z_{3w} = KL$;

Total operation count: $12\mathbf{M} + 7\mathbf{A}$.

Appendix B

Predictor Designs

Weierstrass Jacobian Addition Predictor:

$$\alpha = X_1 Z_2^2; \quad \beta = X_2 Z_1^2; \quad \gamma = Y_1 Z_2^3; \quad \delta = Y_2 Z_1^3;$$
$$A = \alpha\beta; \quad B = \gamma\delta; \quad C = \beta - \alpha; \quad D = Z_{1w} Z_{2w};$$
$$E = X_{1w} Z_{2w}^2; \quad F = X_{2w} Z_{1w}^2; \quad G = Y_{1w} Z_{2w}^3; \quad H = Y_{2w} Z_{1w}^3;$$
$$K = F + E - 2A; \quad L = F - E; \quad M = H + G - 2B; \quad N = H + G + 2B;$$
$$P = H - G; \quad R = 3CL - 2M;$$

$$X_{3w} = M^2 + KL^2 - 2CML;$$
$$Y_{3w} = \frac{MR^2 + K^3 N - 2CPRK}{4};$$
$$Z_{3w} = KD;$$

where $\alpha = X_1 Z_2^2$, $\beta = X_2 Z_1^2$, $\gamma = Y_1 Z_2^3$, $\delta = Y_2 Z_1^3$ are obtained from the non-redundant datapath.

Total operation count: 27**M** + 13**A**.

Weierstrass Jacobian Doubling Predictor:

$A = X_1 Y_{1w}; \quad B = 3X_{1w} + aZ_{1w}^2; \quad C = Y_{1w}^2; \quad D = X_{1w} C;$
$E = Y_{1w} Z_{1w}; \quad F = 12A - B^2; \quad G = BF;$

$X_{3w} = B^4 + 64D - 16AB^2;$
$Y_{3w} = G^2 + 64C^2 - 16GC;$
$Z_{3w} = 4E;$

Total operation count: 13**M** + 8**A**.

Edwards Affine Unified Addition Predictor:

$\alpha = x_1 x_2 y_1 y_2;$
$A = x_{1w} y_{2w}; \quad B = y_{1w} x_{2w}; \quad C = x_{1w} x_{2w}; \quad D = y_{1w} y_{2w};$
$E = A + B + 2\alpha; \quad F = C + D - 2\alpha; \quad G = d\alpha; \quad H = G^2;$
$K = 1 + H + 2G; \quad L = 1 + H - 2G;$

$x_{3w} = \frac{E}{K};$
$y_{3w} = \frac{F}{L};$

where $\alpha = x_1 x_2 y_1 y_2$ is obtained from the non-redundant datapath.

Total operation count: 6**M** + 8**A** + 2**D**.