March 2021

Efficient Hardware Constructions for Error Detection of Post-Quantum Cryptographic Schemes

Alvaro Cintas Canto
University of South Florida

Efficient Hardware Constructions for Error Detection of Post-Quantum Cryptographic Schemes

by

Alvaro Cintas-Canto

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Mehran Mozaffari Kermani, Ph.D.
Sriram Chellappan, Ph.D.
Srinivas Katkoori, Ph.D.
Nasir Ghani, Ph.D.
Reza Azarderakhsh, Ph.D.

Date of Approval:
March 16, 2021

Keywords: Fault Detection, Field-Programmable Gate Array, Finite Field Arithmetic, Post-Quantum Cryptography, Side-Channel Attacks

Dedication

For my kind parents, Alfonso and Maria Eugenia;

My awesome siblings, Alfonso and Miriam;

And my best love, Kenzie

Acknowledgments

First and foremost, I would like to thank my supervisor Dr. Mehran Mozaffari-Kermani for his priceless support, advice, and the knowledge shared with me during my graduate years. You are the true definition of an amazing mentor. You have not only helped me to become a Doctor, but you have helped me to grow as a person, to not give up, and to think outside of the box. I want to thank the Department of Computer Science for their valuable assistance during my years at USF. Also, I would like to thank Dr. Reza Azarderakhsh, Dr. Sriram Chellappan, Dr. Srinivas Katkoori, and Dr. Nasir Ghani for their amazing support.

Lastly, I would also like to thank my love Kenzie and my family for their unconditional love and support. Dad, I know how proud you would be. You always wanted me to be a Doctor and I wish you could still be here to see me becoming one. Thanks to you, to mom, and to my brother and sister for always being next to me, even in the distance.

Table of Contents

List of Tables

List of Figures

Abstract

Quantum computers are presumed to be able to break nearly all public-key encryption algorithms used today. The National Institute of Standards and Technology (NIST) started the process of soliciting and standardizing one or more quantum computer resistant public-key cryptographic algorithms in late 2017. It is estimated that the current and last phase of the standardization process will last till 2022-2024. Among those candidates, code-based and multivariate-based cryptography are a promising solution for thwarting attacks based on quantum computers. Nevertheless, although code-based and multivariate-based cryptography, e.g., McEliece, Niederreiter, and Luov cryptosystems, have good error correction capabilities, research has shown their hardware architectures are vulnerable to faults due to the complexity and large footprint of the finite field arithmetic architectures used in those architectures. In this dissertation, error detection schemes on various post-quantum cryptosystems that use finite fields are derived, proving the high efficiency and error coverage of such schemes, and the acceptable overhead needed to implement them in deeply-embedded architectures. Moreover, general error detection schemes are derived for finite field arithmetic with polynomial and normal basis, applicable to any classical or post-quantum cryptographic algorithms that use finite field block in their designs.

Chapter 1: Introduction

1.1 Overview of Cryptography

In the IoT-connected world, cryptography plays a crucial role when transmitting sensitive information from one device to another in the presence of third parties. The main idea behind cryptography is to encrypt sensitive information through a mathematical algorithm in a way that only a specific key can decrypt and unveil the original data, providing confidentiality, data integrity, authentication, and non-repudiation. Cryptography has existed for many years and many different cryptographic algorithms have been studied, each one with its own advantages and drawbacks. There are two main classes of cryptography: Symmetric-key cryptography and asymmetric-key cryptography.

Symmetric-key cryptography uses the same key for both encryption and decryption. This type of cryptography is traditionally much faster than asymmetric-key cryptography and it is use to transmit data in bulk. However, it requires both the sender and the recipient to have the secret key and it does not provide data integrity and authentication. The concept of data integrity is to ensure full confidence that the data sent to the recipient is the actual valid data from the sender and has not been tampered with or manipulated, while authenticity means that the recipient can establish that the data is originated from a trusted entity. Currently, the Advanced Encryption Standard (AES) [1] is the symmetric-key cryptographic algorithm chosen by NIST to protect sensitive information. NIST started the development of the AES in 1997 since the Data Encryption Standard (DES) [2] was becoming vulnerable to brute-force attacks.

Asymmetric-key cryptography uses a pair of keys, i.e., public key and private key, instead of just a secret key, tackling the problem of data integrity and authentication that

symmetric-key cryptography has. Public keys are published in the public domain (e.g., Internet), while the private key is unique and infeasible to obtain from the encryption or decryption operations or from the public key. Some public key algorithms provide digital signatures (e.g., Digital Signature Algorithm [3]), some provide key distribution and secrecy (e.g., Diffie–Hellman key exchange [4]), and there are others that provide both (e.g., RSA [5]). Asymmetric encryption is slower than good symmetric encryption, which is inconvenient for many applications. The elliptic curve cryptography (ECC) was a pioneer for public key cryptography till the announcement by NSA and NIST to transition to post-quantum algorithms. As a result, today's cryptographic systems use both symmetric and asymmetric encryption techniques (e.g., TLS, Secure Shell).

1.2 Post-Quantum Cryptography

The prospects of emergence of low-power and low-energy quantum computers have brought the need for developing public-key cryptosystems secure against attacks potentially enabled by these computers. For instance, the quantum Shor's algorithm allows breaking traditional asymmetric-key cryptosystems by solving integer-factorization problems in polynomial time [6]. The NIST started the process of soliciting and standardizing one or more quantum computer resistant public-key cryptographic algorithms in late 2017 [7]. Several solutions have been proposed for this post-quantum era, called post-quantum cryptography (PQC).

There are five popular PQC algorithm classes: Code-based, hash-based, isogeny-based, lattice-based, and multivariate-based cryptosystems. Code-based cryptography, such as the McEliece [8] and Niederreiter cryptosystems [9], differs from others by that its security relies on the hardness of decoding in a linear error correcting code. Hash-based cryptography creates signature algorithms based on the security of a selected cryptographic hash function [10], [11]. The security of isogeny-based cryptography is based on the hard problem to find an isogeny between two given supersingular elliptic curves [12], [13], [14]. Lattice-based cryptography is capable of creating a public-key cryptosystem based on lattices [15],[16], [17].

Lastly, multivariate-based cryptography is founded on asymmetric cryptography primitives using multivariate polynomials over a finite field [18], [19]. Many traditional and post-quantum cryptographic algorithms, e.g., ECC, AES, McEliece, and Luov, use finite field operations in their schemes. Among the different finite field operations, multiplication and inversion are the most studied due to their complexity.

1.3 Fault Diagnosis

Finite field arithmetic, especially finite field multiplications and inversions, are very complex and require large area footprint. Therefore, it is a complex task to implement such operations resilient to natural and malicious faults [20], [21]. Side-channel attacks are a type of attack that relies on knowledge obtained from the execution of a computer system rather than flaws in the system's cryptographic algorithm. Among the different side-channel attacks, the most known are cache attacks, where the third party monitors the cache accesses that the sender/receiver makes in a shared physical system; power-monitoring attacks, where the third party monitors the differences in power consumption during the systems' computation; timing attacks, where the third party measures the time needed for a specific computation; and electromagnetic attacks, where the third party analyzes the leaked electromagnetic radiation to obtain the secret key or the plaintext.

These type of side-channel attacks are passive, or in other words, the attacker mainly observes the different leakages or aspects of a system to discover the key or the sensitive data. Differential fault analysis (DPA), on the other hand, are active, meaning that the attacker induces faults, tampering the system, to compare the faulty outputs with the original outputs and reveal information. Depending on the duration of such faults (as a result from the physical stress applied), there are three types of faults: Transient faults, permanent faults, and destructive faults.

Transient faults are the most common ones and they only last one or a few clock cycles, the system will retry the same operation and the fault will have disappeared. Permanent

faults last longer than transient faults, the system will have to be reset or the value where the fault injection took place will have to be overwritten for the device to output the correct values. Lastly, destructive faults are those which destroy a physical structure of the device, causing a specific variable or bit to be fixed in all the device runs. The main difference between transient faults, permanent faults, and destructive faults is that the latter one is not reversible. Transient faults are also known as soft faults and can be classified into parametric or logical depending on the effect of the fault. Parametric faults affect voltage, current, or speed, while logical faults affect the Boolean function through the circuit.

To determine if the fault injections are being performed in a system, fault detection schemes are applied. There are many works on fault detection analysis [22], [23], [24], [25], [26], [27], and [28]. The different fault detection techniques can be classified into: Concurrent error detection, Off-Line Detection, and Roving Fault Detection. In this dissertation, concurrent error detection schemes are studied. Concurrent error detection adds additional logic to the system to detect faults. By performing a set of derivations, the system will be compare the actual output with the predicted output and if the error flag is high, a mismatch has been found among the two outputs (fault detected). Redundancy is a type of concurrent error detection which is used in hardware implementations to achieve error control and reliability. These redundancy schemes can be divided into: Hardware, time, hybrid, and information redundancy. Hardware redundancy duplicates the function and compares both outputs, time redundancy computes the function twice and compares both results, hybrid redundancy performs the inverse of one part of the function, or the inverse of the whole function and compares it, and information redundancy adds some check bits along the function and validates it when the result is obtained.

1.4 Objectives

Through this dissertation, different error detection schemes based on normal, interleaved, two-part, three-part, and CRC signatures are proposed. The choice of the utilized signatures

can be tailored based on the reliability requirements and the overhead to be tolerated. In other words, for applications such as game consoles in which performance is critical (and power consumption is not because these are plugged in), one can increase the size of the signatures. However, for deeply-embedded systems such as implantable and wearable medical devices, smaller signatures are preferred. The main contributions of this dissertation are summarized as follows:

• Error detection schemes based on normal, interleaved, two-part, three-part, and CRC signatures for different finite field arithmetic operation blocks, which include addition, subtraction, multiplication, squaring, and inversion (both Fermat's little theorem (FLT) and the Itoh-Tsujii algorithm (ITA)) used in code-based and multivariate-based cryptosystems are proposed. Additionally, such schemes are applied to the code-based McEliece and Niederreiter cryptosystems and the multivariate-based Luov cryptosystem.

• New formulations for the error detection schemes performing software implementations are derived for the sake of verification. Such derivations cover a wide range of applications and security levels. Nevertheless, the presented schemes are not confined to these case studies.

• The error coverage of the proposed error detection schemes is analyzed by embedding the error detection schemes into the original constructions. The implementations are performed using different Xilinx field-programmable gate array (FPGA) families to confirm that the schemes are overhead-aware and that they provide high error coverage.

1.5 Dissertation Outline

In this dissertation, error detection for different cryptosystems are presented through the following chapters:

• Chapter 2: This chapter presents countermeasures against side-channel attacks for the implementation of different composite field arithmetic units used in the McEliece cryptosystem.

• Chapter 3: In this chapter, error detection schemes are applied to the key generation of the McEliece cryptosystem.

• Chapter 4: Error detection schemes for the different blocks of the key generator in the Niederreiter cryptosystem, e.g., $GF(2^m)$ multiplication, squaring, inversion, and addition, are proposed in this chapter.

• Chapter 5: This chapter introduces efficient hardware architectures for the Luov cryptographic algorithm.

• Chapter 6: Through this chapter, CRC signatures are derived for both FLT and ITA algorithms, used to perform finite field inversion over $GF(2^m)$ with normal basis field element representation.

• Chapter 7: This chapter concludes the dissertation by summarizing the different error detection schemes that have been implemented on the various cryptosystems.

Chapter 2: Reliable Architectures for Composite-Field-Oriented Constructions of McEliece Post-Quantum Cryptography on FPGA

2.1 McEliece Cryptosystem

[1]The McEliece cryptosystem is successfully advanced to the third and last round of the post-quantum cryptography standardization competition in 2020. Since the McEliece cryptosystem uses public-key encryption, it can serve in a wide variety of applications such as digital signatures, authentication protocols, exchange of a secret key over an insecure channel, or even digital cash systems such as Bitcoin. Even though there are different alternatives to Goppa codes such as LDPC and MDPC codes, Reed-Solomon codes, or convolutional codes, they are not as secure as the binary Goppa codes [30]. Since the key size of the McEliece scheme is very large, other slight variations of binary Goppa codes such as quasi-cyclic and quasi-dyadic codes had also been explored [31], [32]. These alternate codes allow a smaller key size, but they are more sensitive to fault injection attacks [33]. Having the architectures of [34] as example, the McEliece cryptosystem uses Goppa codes in its different arithmetic units. The McEliece cryptosystem is based on linear error-correcting codes; however, it is vulnerable to fault injection attacks as it has been previously indicated [35].

Side-channel attacks such as differential power analysis (DPA) for the McEliece cryptosystems have been studied in a number of previous works, e.g., [36] and [37]. In [36], the authors present a successful DPA of a state-of-the-art McEliece implementation based on quasi-cyclic MDPC codes. Authors in [37] successfully demonstrate side-channel attacks of the McEliece cryptosystem implemented on constrained devices. Approaches on countering fault attacks such as [38], [39], [40], [41], [42], and [43] have been the center of research

[1]This chapter was published in the IEEE Transactions on Computer-Aided Design Integr. Circuits Syst. [29]

attention for cryptography. In this chapter, error detection schemes are derived for different Goppa arithmetic units that the McEliece cryptosystem uses, e.g., evaluation, multiplication, squaring, division, square root, inversion, and greatest common division. These error detection schemes are based on signatures providing high error coverage. The term signature here refers to appended bits used for error detection through error detecting codes and not the typical signatures commonly used for proof of authenticity in cryptography. Additionally, the overhead of the proposed schemes is calculated by implementing the arithmetic units used in the McEliece cryptosystem on FPGA.

The McEliece cryptosystem includes three operations: Key generation, which generates a pair of keys (public key and private key) needed to keep the message secret; encryption, which creates the ciphertext using the public key; and decryption, which allows to obtain the original message using the private key. The key generation in the McEliece cryptosystem uses the dimension of the code subspace m, the maximum number of errors that can be corrected t, the code length n, and code rank k. In this work, the security parameters used are *m=13*, *t=128*, and *n=8,192* (k can be calculated by performing $k = n - mt$), which are one of the possible security parameters submitted to NIST in late 2017 [8]. However, the proposed approaches are oblivious of the sizes of these three parameters. The McEliece cryptosystem produces the pair of keys by first constructing a basic finite field $GF(2^m)$. Since *m=13*, this field contains 8,192 elements, i.e., α_0, α_1,..., α_{8191}, where each element is a vector of 13 bits. Next, a random monic irreducible polynomial $g(x) = x^t + g_{t-1}x^{t-1} + ... + g_1x + g_0$, with degree t is also generated, called Goppa polynomial. This monic irreducible polynomial conforms part of the private key, and all its coefficients are elements of the basic finite field. After producing the Goppa polynomial, a control matrix H is constructed by multiplying three auxiliary matrices based on the private key, denoted as X, Y, and Z. The control matrix is permutated by using a random permutation matrix called P. Then, it is expanded into a binary form H_2 over $GF(2)$, converted into the systematic form $\tilde{G}$, and transposed into G to obtain its public key.

Table 1: McEliece operations and corresponding processes.

Operation	Process
Goppa Division	Key Generation, Decryption
Goppa Multiplication/Addition	Key Generation, Encryption, Decryption
Goppa Squaring	Key Generation
Goppa Square Root	Decryption
Goppa GDC, Goppa Inversion and Goppa Polynomial Decomposition	Key Generation, Decryption
Goppa Polynomial Evaluation	Key Generation, Decryption

To get the ciphertext z, a random n-bit error vector e is created. The error vector e has to contain a total of t bits having the value 1. Then, z is calculated by performing $z = pG \oplus e$, where p refers to plaintext. The decryption process is fairly more complex than the encryption process. First, an error locator polynomial $\sigma(x)$ is created, which will reveal all errors of the elements α_i. Then, the error vector is reconstructed and gone through the *CCA2*-safe ciphertext to obtain the *CCA2*-safe plaintext. Lastly, if the *CCA2*-safe padding data is valid after the *CCA2*-related padding is removed, the plaintext is correct and ready to be returned; if not, an error is given. Since this chapter focuses mainly on the key generation, readers interested in other aspects of the McEliece cryptosystem can refer to [34].

2.2 Proposed Fault Detection Schemes

The McEliece cryptosystem is based on three different Galois fields: The Goppa field $GF((2^m)^t)$ used by the Goppa polynomial (i.e., $GF((2^{13})^{128})$ in this chapter); $GF(2^{13})$, which is the field polynomial (we use $p(x) = x^{13} + x^4 + x^3 + x + 1$ as described by [8]); and the binary field *GF(2)*. The McEliece cryptosystem uses (based on the underlying composite fields) different Goppa field arithmetic units to perform a number of its operations, e.g., evaluation, multiplication, squaring, division, square root, inversion, and greatest common division. These Goppa field operations work on polynomials of degree *t-1* with coefficients from $GF(2^m)$. In Table 1, the McEliece operations and corresponding processes are shown.

2.2.1 Goppa Division

Polynomial division is required to find the greatest common divisor. It follows the long division method by first, inverting the highest coefficient of the polynomial divisor. This inverted coefficient is then multiplied by the highest-degree coefficient of the dividend to obtain the quotient. Next, the quotient coefficient is multiplied by all the coefficients of the polynomial divisor and the product is finally subtracted from the dividend polynomial using modulo-2 addition. Since each coefficient is in $GF(2^{13})$, signatures for inversion, multiplication, and XORing in the field $GF(2^{13})$ are needed. Authors in [44] perform error detection over binary extension fields by adding parities which can only detect an odd number of faults and therefore, to obtain higher error coverage, two other signatures are derived in this chapter. The first alternative, called two-part signature, divides the 13 bits into two blocks (from bit 0 to bit 6, and from bit 7 to bit 12), and the other alternative, called three-part signature, divides the bits into three blocks (from bit 0 to bit 4, from bit 5 to bit 9, and from bit 10 to bit 12).

2.2.2 Goppa Addition and Goppa Multiplication

Goppa addition and Goppa multiplication need a total of t $GF(2^m)$ additions and t $GF(2^m)$ multiplications, respectively. As it is shown in [44], the multiplication of any two elements A and B of $GF(2^m)$ can be represented as

$$A \cdot B \bmod p(x) = \sum_{i=0}^{m-1} b_i \cdot ((A\alpha^i) \bmod p(x)) = \sum_{i=0}^{m-1} b_i \cdot X^{(i)},$$

where the set of α^i's is the polynomial basis of element A, the set of b_i's is the B coefficients, *p(x)* is the field polynomial, $X^{(i)} = \alpha \cdot X^{(i-1)} \bmod p(x)$, and $X^{(0)} = A$. To perform polynomial basis multiplications over binary extension fields, α, *sum*, and *pass-thru* modules are used. The *pass-thru* module multiplies a $GF(2^m)$ element by a *GF(2)* element, the α module multiplies an element of $GF(2^m)$ by α and it reduces the result modulo *p(x)*, and the *sum*

module adds two elements in $GF(2^m)$ using m two-input XOR gates. The latter one is used for addition of polynomials in $GF(2^{13})$.

In the *sum* module, the parity bits of the inputs A and B, and the predicted parities of the output P are divided into two or three blocks, depending on if two-part signature or three-part signature is used. If two-part signature is used, the parity bits of A and B are divided into p_{A1} and p_{A2}, and p_{B1} and p_{B2}, respectively. The addition of elements A and B in $GF(2^m)$ produce the predicted parities

$$\hat{p}_{P1} = p_{A1} + p_{B1}$$

and

$$\hat{p}_{P2} = p_{A2} + p_{B2}.$$

On the other hand, if three-part signature is used, the parity bits of A are divided into p_{A1}, p_{A2}, and p_{A3}, and p_{B1}, p_{B2}, and p_{B3}, obtaining the predicted parities

$$\hat{p}_{P1} = p_{A1} + p_{B1},$$

$$\hat{p}_{P2} = p_{A2} + p_{B2},$$

and

$$\hat{p}_{P3} = p_{A3} + p_{B3}.$$

In the *pass-thru* module, the parity bits of the input A and the predicted parities of the output P are also divided into two or three blocks. If two-part signature is used, the parity bits of A are multiplied by an element b of *GF(2)* to obtain the predicted parities

$$\hat{p}_{P1} = b \cdot p_{A1}$$

and

$$\hat{p}_{P2} = b \cdot p_{A2}.$$

On the other hand, if three-part signature is used, the predicted parities of output P are

$$\hat{p}_{P1} = b \cdot p_{A1},$$

$$\hat{p}_{P2} = b \cdot p_{A2},$$

and

$$\hat{p}_{P3} = b \cdot p_{A3}.$$

Next, for the α module, Theorems 2.1 and 2.2 are derived for two-part signature and three-part signature, respectively.

Theorem 2.1 *Let* $p_{A1} = \sum_{i=0}^{\frac{m-1}{2}} a_i$ *be the first parity for bits of* A*, and* $p_{A2} = \sum_{i=\frac{m-1}{2}+1}^{m-1} a_i$ *be the last parity for bits of* A*, where* m *is assumed to be odd, used for the dimension of the code subspace, and* $f_i \in GF(2) for i = 0, 1, \ldots, m-1$*. Then, the predicted parities* $\hat{p}_{X1}$ *and* $\hat{p}_{X2}$*, are*

$$\hat{p}_{X1} = a_{m-1} + \sum_{i=1}^{\frac{m-1}{2}} (a_{i-1} + a_{m-1} \cdot f_i)$$

and

$$\hat{p}_{X2} = \sum_{i=\frac{m-1}{2}+1}^{m-1} (a_{i-1} + a_{m-1} \cdot f_i).$$

For m=13, to the following concise formulations are obtained:

$$\hat{p}_{X1} = p_{A1} + a_6$$

and

$$\hat{p}_{X2} = p_{A2} + a_6 + a_{12}.$$

Proof. Using the below formulations to calculate the X coordinates [44],

$$x_i = \begin{cases} a_{i-1} + a_{m-1} \cdot f_i & 1 \leq i \leq m-1, \\ a_{m-1} & i = 0, \end{cases}$$

the predicted parity $\hat{p}_X$ can be split as

$$\hat{p}_X = a_{m-1} + \sum_{i=1}^{\frac{m-1}{2}} (a_{i-1} + a_{m-1} \cdot f_i) + \sum_{i=\frac{m-1}{2}+1}^{m-1} (a_{i-1} + a_{m-1} \cdot f_i) = \hat{p}_{X1} + \hat{p}_{X2}.$$

Since $p(x) = x^{13} + x^4 + x^3 + x + 1$, one obtains $f_{13} = f_4 = f_3 = f_1 = f_0 = 1$, achieving the concise formulations. This completes the proof.

Theorem 2.2 *Let $p_{A1} = \sum_{i=0}^{\lfloor \frac{m-1}{3} \rfloor} a_i$ be the first parity for bits of A, $p_{A2} = \sum_{i=\lfloor \frac{m-1}{3} \rfloor+1}^{2(\lfloor \frac{m-1}{3} \rfloor)+1} a_i$ be the second parity for bits of A, and $p_{A3} = \sum_{i=2(\lfloor \frac{m-1}{3} \rfloor+1)}^{m-1} a_i$ be the last parity for bits of A. Then, the predicted parities of output X, referred to as $\hat{p}_{X1}$, $\hat{p}_{X2}$, and $\hat{p}_{X3}$), are*

$$\hat{p}_{X1} = a_{m-1} + \sum_{i=1}^{\lfloor \frac{m-1}{3} \rfloor} (a_{i-1} + a_{m-1} \cdot f_i),$$

$$\hat{p}_{X2} = \sum_{i=\lfloor \frac{m-1}{3} \rfloor+1}^{2(\lfloor \frac{m-1}{3} \rfloor)+1} (a_{i-1} + a_{m-1} \cdot f_i),$$

and

$$\hat{p}_{X3} = \sum_{i=2(\lfloor \frac{m-1}{3} \rfloor+1)}^{m-1} (a_{i-1} + a_{m-1} \cdot f_i).$$

For the case study of m=13, the following concise formulations are obtained:

$$\hat{p}_{X1} = p_{A1} + a_4,$$

$$\hat{p}_{X2} = p_{A2} + a_4 + a_9,$$

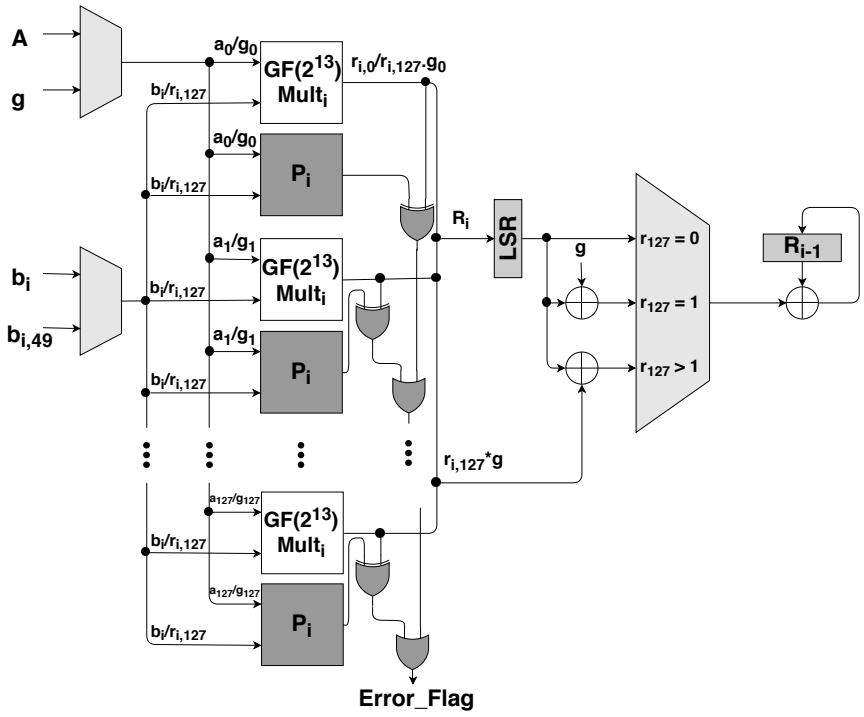


Figure 2.1: Goppa multiplication unit with the proposed scheme.

and

$$\hat{p}_{X3} = p_{A3} + a_9 + a_{12}.$$

Proof. The predicted parity $\hat{p}_X$ can be split as

$$\hat{p}_X = a_{m-1} + \sum_{i=1}^{\lfloor \frac{m-1}{3} \rfloor}(a_{i-1} + a_{m-1} \cdot f_i) + \sum_{i=\lfloor \frac{m-1}{3} \rfloor + 1}^{2(\lfloor \frac{m-1}{3} \rfloor)+1}(a_{i-1} + a_{m-1} \cdot f_i) +$$
$$\sum_{i=2(\lfloor \frac{m-1}{3} \rfloor + 1)}^{m-1}(a_{i-1} + a_{m-1} \cdot f_i) = \hat{p}_{X1} + \hat{p}_{X2} + \hat{p}_{X3}.$$

This completes the proof.

Fig. 2.1 illustrates the architecture of the Goppa multiplication unit with the proposed fault detection schemes. It follows a *shift-and-add* approach with a chain of 127 $GF(2^{13})$ multiplications, and therefore, a total of 127 signatures are needed. These signatures are shown as P_i for *i=0,1,...,127*, and they can be implemented as normal, two-part, or three-part signatures (depicted by dark-grey, shaded blocks). a_i, b_i, and g_i's are coefficients of the field $GF(2^{13})$. R_i is the intermediate result of the 127 multiplications, expressed as

$R_i = \sum_{j=0}^{127} r_{i,j} x^j$, where $r_{i,j} = b_i a_j \; mod \; p(x)$. The multiplication in Fig. 2.1 is essentially a schoolbook multiplication. First, a_i is multiplied with b_i, obtaining R_i. If $i \neq 0$, R_i is shifted 13 bits using the left shift register (shown as LSR in Fig. 2.1). Then, R_i is reduced depending on the value of $r_{i,127}$: If $r_{i,127} = 0$, there is no reduction; if $r_{i,127} = 1$, R_i is XOR-ed with g; and if $r_{i,127} > 1$, R_i is XOR-ed with $r_{i,127} \cdot g$. Lastly, the reduced result is XOR-ed with the previous result R_{i-1}. If a faulty output is detected in any of the 127 $GF(2^{13})$ multiplication modules, *Error_Flag* is asserted.

2.2.3 Goppa Inversion

To perform Goppa inversion, t $GF(2^m)$ inversions are needed. The polynomial variant of Fermat's Little Theorem (FLT) is used. FLT achieves higher performance, allowing to calculate the inverse $GF(2^{13})$ using 12 squarings and 12 multiplications. Next, Theorem 2.3 and Theorem 2.4 are derived for signatures of squaring.

Theorem 2.3 *Let* $p_{A1} = \sum_{i=0}^{\frac{m-1}{2}} a_i$ *be the first parity for bits of A, and* $p_{A2} = \sum_{i=\frac{m-1}{2}+1}^{m-1} a_i$ *be the last parity for bits of A. Then, the predicted parities of output* ν *(instead of X to avoid confusion), referred to as* $\hat{p}_{\nu 1}$ *and* $\hat{p}_{\nu 2}$*, are*

$$\hat{p}_{\nu 1} = a_{m-1} \cdot f_{m-1} + a_{m-2} + a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1 + \sum_{i=2}^{\frac{m-1}{2}} (a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2})$$

and

$$\hat{p}_{\nu 2} = \sum_{i=\frac{m-1}{2}+1}^{m-1} (a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2}).$$

For the case study of m=13, the following concise formulations are obtained:

$$\hat{p}_{\nu 1} = p_{A1} + a_5 + a_6$$

and

$$\hat{p}_{\nu 2} = p_{A2} + a_5 + a_6 + a_{11} + a_{12}.$$

Proof. The predicted parity $\hat{p}_\nu$ can be split as

$$\begin{gathered}\hat{p}_\nu = a_{m-1} \cdot f_{m-1} + a_{m-2} + a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1 + \\ \textstyle\sum_{i=2}^{\frac{m-1}{2}} (a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2}) + \\ \textstyle\sum_{i=\frac{m-1}{2}+1}^{m-1} (a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2}) \\ = \hat{p}_{\nu 1} + \hat{p}_{\nu 2}.\end{gathered}$$

This completes the proof.

Theorem 2.4 *Let $p_{A1} = \sum_{i=0}^{\lfloor \frac{m-1}{3} \rfloor} a_i$ be the first parity for bits of A, $p_{A2} = \sum_{i=\lfloor \frac{m-1}{3} \rfloor + 1}^{2(\lfloor \frac{m-1}{3} \rfloor)+1} a_i$ be the second parity for bits of A, and $p_{A3} = \sum_{i=2(\lfloor \frac{m-1}{3} \rfloor + 1)}^{m-1} a_i$ be the last parity for bits of A. Then, the predicted parities of output ν, referred to as $\hat{p}_{\nu 1}$, $\hat{p}_{\nu 2}$, and $\hat{p}_{\nu 3}$, are*

$$\begin{gathered}\hat{p}_{\nu 1} = a_{m-1} \cdot f_{m-1} + a_{m-2} + a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1 + \\ \textstyle\sum_{i=2}^{\lfloor \frac{m-1}{3} \rfloor} (a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2}),\end{gathered}$$

$$\hat{p}_{\nu 2} = \sum_{i=\lfloor \frac{m-1}{3} \rfloor + 1}^{2(\lfloor \frac{m-1}{3} \rfloor)+1} (a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2}),$$

and

$$\hat{p}_{\nu 3} = \sum_{i=2(\lfloor \frac{m-1}{3} \rfloor + 1)}^{m-1} (a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2}).$$

For the case study of m=13, the following concise formulations are obtained:

$$\hat{p}_{\nu 1} = p_{A1} + a_3 + a_4 + a_{12},$$

$$\hat{p}_{\nu 2} = p_{A2} + a_8 + a_9 + a_{12},$$

and

$$\hat{p}_{\nu 3} = p_{A3} + a_{11} + a_{12}.$$

*Proof.*The predicted parity $\hat{p}_\nu$ can be split as

$$\begin{aligned}\hat{p}_\nu = a_{m-1} \cdot f_{m-1} + a_{m-2} + a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1 + \\ \sum_{i=2}^{\lfloor \frac{m-1}{3} \rfloor} (a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2}) + \\ \sum_{i=\lfloor \frac{m-1}{3} \rfloor + 1}^{2(\lfloor \frac{m-1}{3} \rfloor)+1} (a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2}) + \\ \sum_{i=2(\lfloor \frac{m-1}{3} \rfloor + 1)}^{m-1} (a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2}) \\ = \hat{p}_{\nu 1} + \hat{p}_{\nu 2} + \hat{p}_{\nu 3}.\end{aligned}$$

This completes the proof.

2.2.4 Goppa Square Root

Square root is needed in the decryption process. It is a rather time-consuming task because first, it multiplies the input polynomial by a matrix Q^{-1} (a polynomial of order t and with coefficients of $GF(2^m)$ where its rows are formed by the square of a monomial modulo the Goppa polynomial), and then, it does the square root of each coefficient $GF(2^m)$ obtained. In total, 16,384 finite field multiplications and 128 square roots are needed for the case study of *m=13*. Since all the elements are in the field $GF(2^{13})$, 12 $GF(2^{13})$ squarings are needed to perform the square root of each element.

2.2.5 GGCD, Goppa Inversion, and GCDIPD

Goppa Polynomial Decomposition (GCDIPD) unit performs three different operations in the Goppa field $GF((2^m)^t)$. It finds the Goppa greatest common divisor (GGCD) of two polynomials of order t, it inverts the syndrome, and it does polynomial decomposition. To perform such operations, Goppa division, Goppa multiplication, and $GF(2^{13})$ inversion are needed. The architecture of the proposed fault detection constructions for this unit is shown in Fig. 2.2, where r stands for remainder, q stands for quotient, α is the intermediate

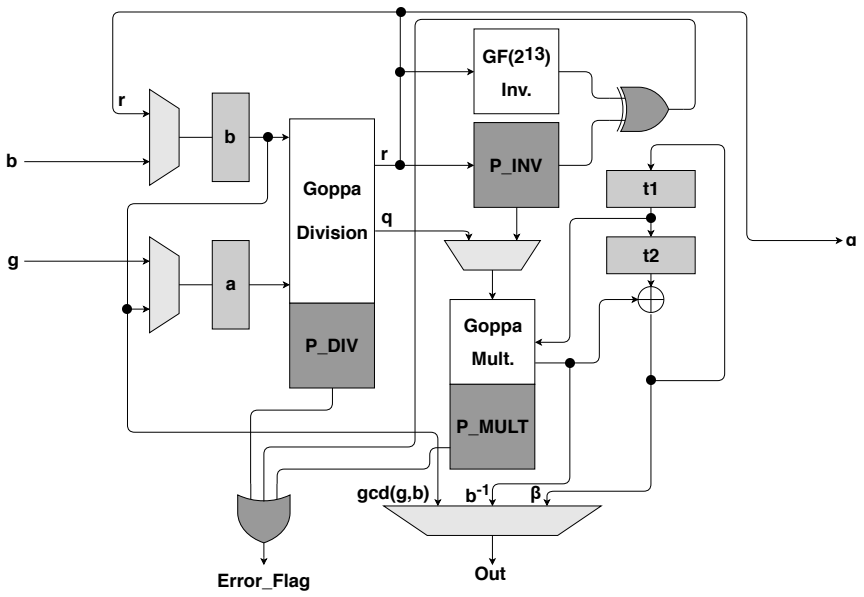


Figure 2.2: Goppa greatest common division, Goppa inversion, and Goppa polynomial decomposition with the error detection scheme.

remainder, and $\beta = (b(x)\ mod\ a(x))^{-1} = t(x)$. Goppa division and Goppa multiplication have their own signatures inside their blocks denoted as P_DIV and P_MULT, respectively, while the signatures for inversion block are separated in Fig. 2.2, since it performs $GF(2^{13})$ inversion, instead of Goppa inversion. First, the polynomials $t_1(x)$ and $t_2(x)$ are set to 1 and 0, respectively. While $b(x) \neq 0 : g(x)$ is divided over $b(x)$ to obtain $q(x)$; $g(x)$ is *mod* with $b(x)$ to obtain $r(x)$; $t(x)$ is obtained by performing $t_2(x) - q(x)t_1(x)$; and lastly, $g(x)$, $b(x)$, $t_2(x)$, and $t_1(x)$ are set to $b(x)$, $r(x)$, $t_1(x)$, and $t(x)$, respectively. If the degree of $r(x)$ is less than 25 and the mode is Patterson (which reconstructs the error vector needed in the process of decryption), $\alpha(x)$ and $\beta(x)$ are set to $r(x)$ and $t(x)$, respectively. If the mode is gcd, $gcd(g(x), b(x))$ is returned. Moreover, if the mode is inverse, $r^{-1}(x)$ is multiplied with $t_1(x)$ to obtain $b^{-1}(x)$. The *Error_Flag* becomes high for faults in any of the polynomial operations.

2.2.6 Goppa Polynomial Evaluation (GPE)

Goppa Polynomial Evaluation (GPE) unit is utilized in the key generation and decryption processes. Adopting the Horner scheme, it only needs to use a $GF(2^{13})$ multiplier because it allows removing high-degree polynomial multiplications, e.g., the element $f_1 + f_2\alpha_2 + f_3\alpha_2^2$ is written as $f_1 + (f_2 + f_3\alpha_2)\alpha_2$.

2.3 Error Coverage and FPGA Implementations

Since the GPE unit uses less signatures than most of the other units, an analysis is perform to show the efficiency of the proposed error detection schemes even for the smallest units of the McEliece cryptosystem. The GPE unit uses the Horner scheme over the Goppa polynomial, which has order t and coefficients of $GF(2^m)$. To perform the Goppa polynomial evaluation, a total of 128 $GF(2^{13})$ finite field multiplications and 128 $GF(2^{13})$ finite field additions are required. Each finite field multiplication and addition requires α, *sum*, and *pass-thru* modules. More precisely, a total of 12 α, 12 *sum*, and 13 *pass-thru* modules are needed for each finite field multiplication, and a total of 12 *sum* modules are required for each finite field addition. Moreover, the total number of signatures needed in the GPE unit is $128_{mult.} \cdot (12_{\alpha} + 12_{sum} + 13_{pass-thru}) + 128_{add.} \cdot (12_{sum})$ or a total of close to $6 \cdot 10^3$ signatures for the normal signature scheme. The percentage of error detection is calculated by applying the formula $100 \cdot (1 - (\frac{1}{2})^{\#sign})\%$, where $\#sign$ is as the total number of signatures. Therefore, the percentage error coverage by this unit is approximately $100 \cdot (1 - (\frac{1}{2})^{6 \cdot 10^3})\%$, which is close to 100%. In the two-part signature scheme, for every module, there are two error flags instead of one (as it is with normal signatures). Moreover, two-part signature scheme uses $2_{error-flags} \cdot (128_{mult.} \cdot (12_{\alpha} + 12_{sum} + 13_{pass-thru}) + 128_{add.} \cdot (12_{sum}))$ or a total of close to $1.2 \cdot 10^4$ signatures, which makes an error percentage of approximately $100 \cdot (1 - (\frac{1}{2})^{1.2 \cdot 10^4})\%$. Lastly, in the three-part signature scheme, for every module, there are three error flags needing $3_{error-flags} \cdot (128_{mult.} \cdot (12_{\alpha} + 12_{sum} + 13_{pass-thru}) + 128_{add.} \cdot (12_{sum}))$ or a total of close to $1.8 \cdot 10^4$ signatures with error coverage of $100 \cdot (1 - (\frac{1}{2})^{1.8 \cdot 10^4})\%$.

Table 2: Overheads of the proposed error detection schemes based of normal signatures (NS), two-part signatures (2PS), and three-part signatures (3PS) for the GPE unit.

Architecture	Area (occupied slices)	Delay (ns)	Power (mW) @50 MHz
GPE (Kintex-7)	1370	4.205	0.205
GPE-NS. (Kintex-7)	1447 (5.62%)	4.494 (6.87%)	0.213 (3.90%)
GPE-2PS (Kintex-7)	1484 (8.32%)	4.415 (4.99%)	0.213 (3.90%)
GPE-3PS (Kintex-7)	1487 (8.54%)	4.402 (4.68%)	0.213 (3.90%)
GPE (Spartan-7)	1339	5.386	0.219
GPE-NS (Spartan-7)	1470 (9.78%)	5.461 (1.39%)	0.225 (2.74%)
GPE-2PS (Spartan-7)	1491 (11.35%)	5.431 (0.84%)	0.225 (2.74%)
GPE-3PS (Spartan-7)	1467 (9.57%)	5.440 (1.00%)	0.225 (2.74%)

Error detection schemes are implemented on two different Xilinx FPGA families. The target platform does not necessarily affect the results because the chosen FPGAs belong to the same series (Xilinx series 7), being very similar from a technological point of view. The design entry is Verilog. The proposed error schemes are added to the GPE unit. In Table 2, the overheads in terms of area (occupied slices), delay, and power (at the frequency of 50 MHz) of the different signatures on the GPE unit are presented. The overheads obtained are very acceptable, especially since the error detection coverage is close to 100%. There are several hardware implementations of the McEliece cryptosystem, e.g., [45] and [46]. The authors in [45] produce a hardware/software implementation which is able to decipher 8,192 bit-length in 47.39 ms. In [46], the authors propose a new variant of McEliece cryptosystem and its encryption-decryption co-processor based on the generalized non-binary Orthogonal Latin Square Code (OLSC). The error detection schemes proposed in this work are suitable for any cryptosystem that uses any of the mentioned Goppa modules such as the works in [45] and [46]. There has not been any prior work done on this type of error detection methods for the McEliece cryptosystem to the best of the author's knowledge. For qualitative comparison to verify that the overheads incurred are acceptable, let us go over some case studies. The work in [47] presented signature-based fault diagnosis for cryptographic block ciphers LED and HIGHT, obtaining a combined area and delay overhead of 21.9% and 31.9% for LED

and HIGHT, respectively. Additionally, the authors in [38] propose efficient error detection architectures of hash-counter-hash tweakable enciphering schemes, obtaining a combined area and throughput overhead of less than 13.5%. The proposed schemes in this chapter have combined area and delay overheads of less than 12% (worst-case scenario). Such prior works on classical cryptography verify that the proposed error detection architectures obtain acceptable overhead.

Chapter 3: Efficient and Low-Power Hardware Constructions for Error Detection of Key Generation in McEliece Cryptosystem

3.1 Key Generator in the McEliece Cryptosystem

Most of the runtime in the McEliece cryptosystem is spent performing operations in finite fields for the key generation. Finite field arithmetic operations include addition, subtraction, multiplication, squaring, and inversion, among which the computation of inversion is the most time-consuming one. To perform inversion over $GF(2^m)$, many solutions have been proposed to increase the performance of such operation for polynomial basis field element representation. FLT and ITA are among the main approaches to compute finite field inversion over $GF(2^m)$. The latter approach was first intended to be applied over binary extension fields with normal basis [48]; however, in more recent studies, it has been shown that it can be used for other field element representations [49], [50]. These approaches extensively use multiplication and squaring, requiring thousands of gates. Thus, these constructions are vulnerable to faults and it is a complex task to implement such architectures resilient to natural and malicious faults. Not only do these structures need low overhead but the error coverage needs to be acceptable. Deteriorated performance can lead to catastrophic results for sensitive applications; accordingly, research has focused on ways to eliminate errors and achieve greater reliability with reasonable overhead [39], [40], [43], [51], [52], [53], [54], [55].

This chapter focuses on providing countermeasures oblivious to the source of the faults. Mounting attacks (or having natural defects) in such finite field arithmetic blocks is one among other possibilities to cause erroneous outputs which is considered in this chapter. Protecting such blocks in the key generator is highly important, especially when the private

key is re-computed from a seed before each decryption operation, so the attacker cannot target the generation of the same private key repeatedly.

In this dissertation, the first work on error detection based on normal, interleaved, and CRC signatures for the key generation of code-based cryptosystems is proposed. Error detection and correction is crucial in the key generation, especially for remote systems, e.g., satellites, where fault-free key generation is needed for the reliability of the overall system. Key generation has the highest area complexity and its hardware implementation is thus the most involved within McEliece. The proposed approach is applicable to the control matrix H generation in key generation and error detection schemes are also added for other remaining parts of the key generation block to account for the entire block, with a special focus on the inversion block. Nevertheless, the underlying blocks which perform operations in finite field are utilized not only in this architecture but are used in other cryptographic implementations. Therefore, the derivations here can be extended to be used beyond key generation in McEliece. In classical cryptography, there has been previous work on countering fault attacks [38], [43], [54], [55], [56], [57], [58]. Although the approach is derived in this chapter for the matrix generation in the encryption of McEliece and implemented the result on FPGA, the proposed approach is applicable to the other important variants of code-based cryptography, e.g., Niederreiter public key cryptography, and is also oblivious of the platform, expecting similar results on application-specific integrated circuit (ASIC) hardware platform. Previous research work [59] implements and analyzes each block of the key encryption independently to prove that the key generation process is the most expensive operation in the Niederreiter cryptosystem. The main contributions in this chapter are summarized as follows:

• Derivation of closed formulations for normal, interleaved, and CRC signatures for the different finite field arithmetic operation blocks, which include addition, subtraction, multiplication, squaring, and inversion, of the McEliece public-key cryptosystem. Moreover, error detection schemes for other remaining parts of the key generation to account for the entire block are proposed.

• To maximize the probability of error detection, the proposed error detection approaches are used in different blocks of key generation since it is, in general, formed by multiplication and inversion.

• Analysis of the error coverage by the proposed error detection schemes. The encryption matrix generator within McEliece cryptosystem is implemented with signatures through the proposed approaches on FPGA to benchmark the overhead of proposed schemes.

The key generation in MECS (an implementation of the McEliece cryptoprocessor performed in [34]) uses the parameters m, t, and n previously mentioned, and dimension k. MECS generates a pair of keys: a private key, produced by a generator matrix, and a public key, which uses a control matrix. First, the key generator randomly creates a monic irreducible polynomial in the form $f(\alpha) = \alpha^t + f_{t-1}\alpha^{t-1} + ... + f_1\alpha + f_0$, with degree t, called Goppa polynomial. To construct the Goppa polynomial, the coefficients of a basic finite field $GF(2^m)$ are used. Taking *m=13* as an example based on the 2017 NIST submission recommendations, the basic finite field contains 8,192 elements, i.e., $\alpha_0, \alpha_1, ..., \alpha_{8,191}$, where each element is a vector of *13* bits. The Goppa polynomial of the field $GF(2^m)$ is kept secret since it is the private key. Based on this private key and three auxiliary matrices denoted as X, Y, and Z, the control matrix H is generated. Next, to obtain the public key, a permutation transformation using a random matrix P is performed. This produces a lengthy public key $\tilde{H}$ which is reduced by first, transforming $\tilde{H}$ into its binary form H_2 over *GF(2)* and next, into the systematic form $\tilde{G}$ using the matrices Π_{mt} and R. Lastly, $\tilde{G}$ is transposed into G to obtain its public key which is denoted as R^T.

3.2 Proposed Fault Detection Schemes

The fault model in this chapter considers both transient faults and permanent internal faults (and the proposed schemes are oblivious of these). Based on the used fault model, stuck-at faults (both stuck-at zero and stuck-at one), adjacent (for interleaved case), and single/multiple stuck-at faults are considered. This is similar to the notion of predictor

in coding theory. In such constructions, for instance in cyclic redundancy check, generator polynomials are used to derive added redundancy for predicting the signatures and comparing with the actual ones. Fault analysis attacks are those which intend to recover the private key by comparing the correct output with the faulty one. The proposed detection architectures are especially important against fault injection when the private key is recomputed from a seed before each decryption operation, allowing the attacker to target the generation of the same private key repeatedly. There are different fault models depending on the type of attack, e.g., the number of bits affected, how the faults are injected, the location of the faults, and the fault duration. Therefore, specific countermeasures are needed to make the McEliece cryptosystem secure against such attacks. Fault analysis attack implementation exhibits (i) fault injection at physical level (timing, power, heating, light, electromagnetic field), (ii) fault manifestation at circuit level (logic cells, memory cells), (iii) fault propagation at micro-architecture level, and finally (iv) fault observation and exploitation at software and firmware level.

Key generation has the highest area complexity and its hardware implementation is thus the most involved within McEliece. The most complex and time-consuming sub-block of the key generator architecture is the "H-generator". The "H-generator" produces the control matrix H needed to obtain the public key of the MECS cryptosystem. This part of the cryptosystem is vulnerable to faults whose aims are to cause algorithm malfunction. In presenting the respective error detection approaches, the typical compromise between error detection capabilities of signatures and the overhead to be tolerated are taken into account.

As seen before, the control matrix H is obtained by multiplying the auxiliary matrices X, Y, and Z. The auxiliary matrix X comes from the "G-generator" which uses a field polynomial *p(x)* of degree m (utilized to derive and construct the different parities). For this architecture, the field polynomial used is $p(x) = x^{13} + x^4 + x^3 + x + 1$; however, the proposed approaches are oblivious of the field size. The auxiliary matrix Y is created inside the "H-generator" by generating the 8,192 elements of the $GF(2^m)$ where $\alpha_i, i \in \{0, 1, \ldots, 8,191\}$,

needing to calculate all the powers from 0 to 127 (we note that three fields within the composite fields of complex Goppa codes, i.e., $GF((2^{13})^{128})$, $GF(2^{13})$, and GF(2), are used for example in the NIST submission recommendation). Next, Z is constructed by performing the inversion of $f(\alpha_i)$. The first multiplication that takes place is XY. This multiplication can leverage Horner schemes, increasing the efficiency by performing easier and less time-consuming multiplications, e.g., the element $f_1 + f_2\alpha_2 + f_3\alpha_2^2$ is written as $f_1 + (f_2 + f_3\alpha_2)\alpha_2$. To perform *(XY)Z*, α_i is multiplied by the correct element available from XY to obtain $f(\alpha_i)$, e.g., to calculate $f(\alpha_0)$, the element $f_1 + f_2\alpha_2 + f_3\alpha_0^2$ is multiplied by α_0, and so on. Then, $f(\alpha_i)$ is inverted by using $GF(2^m)$ inverse.

The multiplicative inverse of an element $A \neq 0$ in the field $GF(2^m)$ is defined as the process of finding the unique element $A^{-1}\,\epsilon\,GF(2^m)$ such that $A \cdot A^{-1} = 1$. To find the multiplicative inverse of an element A, both FLT and ITA schemes are studied in this work.

FLT specifies that the inverse of an element A can be derived as follows

$$A^{2^m-1} \equiv 1 \bmod p(x)$$
$$A^{2^m-2} \cdot A \equiv 1 \bmod p(x)$$
$$A^{2^m-2} \cdot A \cdot A^{-1} \equiv A^{-1} \bmod p(x)$$
$$A^{2^m-2} \equiv A^{-1} \bmod p(x).$$

For hardware implementations, this theorem leads to a total of $2^m - 2$ finite field multiplications, and it may require additional memory to store the precomputed values. Approaches to reduce the complexity of finite field inversions have been studied, e.g., square-and-multiply algorithm [60], Kaliski inversion [61], and ITA algorithm.

The latter method, introduced by Itoh and Tsujii [48], yields dramatic reductions in the number of multiplications needed in the exponentiation by an efficient use of addition chains. The inverse can be rewritten as

$$A^{-1} = [\beta_{m-1}(A)]^2,$$

where $\beta_k(A) = A^{2^k-1} \epsilon GF(2^m)$ and $k \epsilon N$. In the work of [50], a recursive sequence is used with an addition chain for *m-1* to compute $\beta_{m-1}(A)$. To calculate an addition chain $C = \{c_1, c_2, \ldots, c_t\}$ with a field polynomial *p(x)* of m degree, $c_1 = 1$ and $c_t = m - 1$. If c_i is even, $c_{i-1} = c_i/2$ and if c_i is odd, $c_{i-1} = c_i - 1$.

Implementing the control matrix H is a costly and complex task. Since all the blocks of the "H-generator" are constructed through finite field multiplications and inversions over finite fields $GF(2^m)$, normal, interleaved, and CRC signatures are used for these two underlying finite field operations. The first contribution is to develop interleaved and CRC signatures for $GF(2^m)$ addition and multiplication, and normal, interleaved, and CRC signatures for squaring in $GF(2^m)$. The latter three are used for inversion, after the square-and-multiply scheme (or, for instance, ITMIA scheme) is applied.

Signatures used throughout this chapter refer to a footprint of the output of a block used for error detection. Signature derivation allows to monitor and detect if there is an error in the output C of the multiplication of the inputs A and B, where A, B, and C are in $GF(2^m)$. In this chapter, three different schemes for fault detection are proposed: Normal signatures, interleaved signatures, and CRC signatures. The first two are based on parity prediction, while the latter one is based on cyclic error-correcting codes. For CRC, a generator polynomial $g(\alpha)$ is required to implement CRC, which becomes the divisor in a long division of polynomials. The message becomes as the dividend, the quotient is discarded and the result is generated by the remainder. A fixed number of check bits are appended to the data and these check bits are checked when the output is obtained to detect any errors [62].

To perform polynomial basis multiplications over binary extension fields, the hardware architecture is divided into three different modules. These modules are called α, *sum*, and *pass-thru* modules. The α module multiplies an element of $GF(2^m)$ by α and it reduces the result modulo $F(\alpha)$. The *sum* module adds two elements in $GF(2^m)$ using m two-input XOR gates. Finally, the *pass-thru* module multiplies a $GF(2^m)$ element by a *GF(2)* element.

The merit of the proposed work in this chapter is that it is practical for embedded systems because of its acceptable overhead and high error coverage. Encrypting a test message with a fresh public key and then decrypting it using the private key to compare it with the original message would add high performance overhead which might not be acceptable for constrained usage models.

3.2.1 Normal and Interleaved Signatures

Normal signatures for the α module can be found in [44]. The motivation to this first derivation is to make sure adjacent faults are also detected. For interleaved parity in the α module, assuming that A and X are the input and output, respectively, the following theorem is derived. Without losing generality, let us assume m is odd in this theorem (using minor tweaks, it can be adopted for even m).

Theorem 3.1 *Let $p_{Ae} = \sum_{i=0}^{\frac{m-1}{2}} a_{2i}$ be the even parity for bits of A, and $p_{Ao} = \sum_{i=1}^{\frac{m-1}{2}} a_{2i-1}$ be the odd parity for bits of A, where m is assumed to be odd and is used for the dimension of the code subspace, and also $f_i \in GF(2)$ for $i = 0, 1, \ldots, m-1$. Then, the even and odd predicted parities of X, referred to as $\hat{p}_{Xe}$ and $\hat{p}_{Xo}$, respectively, are*

$$\hat{p}_{Xe} = a_{m-1} + \sum_{i=1}^{\frac{m-1}{2}} (a_{2i-1} + a_{m-1} \cdot f_{2i})$$

and

$$\hat{p}_{Xo} = \sum_{i=1}^{\frac{m-1}{2}} (a_{(2i-1)-1} + a_{m-1} \cdot f_{2i-1}).$$

For the case study of m=13, the following concise formulations are obtained: $\hat{p}_{Xe} = p_{Ao}$ and $\hat{p}_{Xo} = p_{Ae} + a_{12}$.

Proof. Using the below formulations to calculate the X coordinates,

$$x_i = \begin{cases} a_{i-1} + a_{m-1} \cdot f_i & 1 \leq i \leq m-1, \\ a_{m-1} & i = 0, \end{cases}$$

the predicted parity $\hat{p}_X$ can be split as

$$\hat{p}_X = a_{m-1} + \sum_{i=1}^{m-1}(a_{i-1} + a_{m-1} \cdot f_i) = a_{m-1} + \sum_{i=1}^{\frac{m-1}{2}}(a_{2i-1} + a_{m-1} \cdot f_{2i}) + \sum_{i=1}^{\frac{m-1}{2}}(a_{(2i-1)-1} + a_{m-1} \cdot f_{2i-1}) = \hat{p}_{Xe} + \hat{p}_{Xo}.$$

Since the field polynomial used in this architecture is $p(x) = x^{13}+x^4+x^3+x+1$, one obtains $f_{13} = f_4 = f_3 = f_1 = f_0 = 1$. Then, $\hat{p}_{Xe} = a_{12} + a_1 + a_3 + a_{12} + a_5 + a_7 + a_9 + a_{11} = p_{Ao}$ and $\hat{p}_{Xo} = a_0 + a_{12} + a_2 + a_{12} + a_4 + a_6 + a_8 + a_{10} = p_{Ae} + a_{12}$. This completes the proof.

In the *sum* module, the even and odd predicted parities of output D, which is the result of the addition of elements A and B in $GF(2^m)$, are referred to as $\hat{p}_{De}$ and $\hat{p}_{Do}$, respectively, and are derived as $\hat{p}_{De} = p_{Ae} + p_{Be}$ and $\hat{p}_{Do} = p_{Ao} + p_{Bo}$.

Lastly, in the *pass-thru* module, the parity bits of A are also divided into p_{Ae} and p_{Ao}, which are multiplied by an element b of *GF(2)*, resulting in the output G. Then, the even and odd predicted parities of G, referred to as $\hat{p}_{Ge}$ and $\hat{p}_{Go}$, respectively, are $\hat{p}_{Ge} = b \cdot p_{Ae}$ and $\hat{p}_{Go} = b \cdot p_{Ao}$.

Since for the multiplicative inversion in finite field squaring is used, an architecture which only uses two modules is utilized, i.e., the α^2 and the *sum* modules. In α^2 module, an element A is multiplied by α^2 to achieve:

$$A(\alpha) \cdot \alpha^2 = a_{m-1} \cdot \alpha^{m+1} + a_{m-2} \cdot \alpha^m + ... + a_0 \cdot \alpha^2,$$

where

$$\alpha^{m+1} = f_{m-1} \cdot \alpha^m + f_{m-2} \cdot \alpha^{m-1} + ... + f_0 \cdot \alpha \bmod p(x)$$

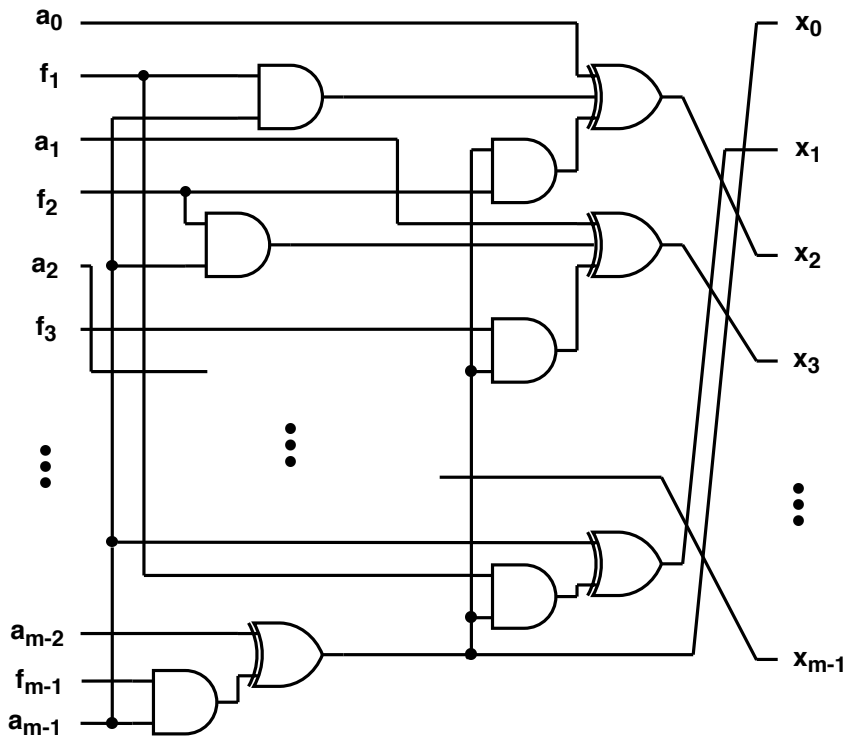


Figure 3.1: The derived architecture for α^2 module, where a_i's and f_i's represent the inputs and x_i's represent the outputs.

and

$$\alpha^m = f_{m-1} \cdot \alpha^{m-1} + f_{m-2} \cdot \alpha^{m-2} + ... + f_0 \bmod p(x),$$

considering the utilized field polynomial. Therefore, the X coordinates can be written as

$$x_i = \begin{cases} a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2} & 2 \leq i \leq m-1, \\ a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1 & i = 1, \\ a_{m-1} \cdot f_{m-1} + a_{m-2} & i = 0. \end{cases}$$

In Fig. 3.1, the architecture for the α^2 module is shown, where $a_0 - a_{m-1}$ are the coefficients of input A, $f_0 - f_{m-1}$ stand for the coefficients of the field polynomial p(x), and $\alpha_0 - \alpha_{m-1}$ are the coefficients of output X. Fig. 3.1 shows the formulations derived for α^2 module. Two

theorems are proposed below for deriving the normal and interleaved signatures of the α^2 module.

Theorem 3.2 *Let $p_A = \sum_{i=0}^{m-1} a_i$ be the parity for bits of A, where m is used for the dimension of the code subspace, and also $f_i \in GF(2)$ for $i = 0, 1, \ldots, m-1$. Then, the predicted parity of X is*

$$\hat{p}_X = a_{m-1} \cdot f_{m-1} + a_{m-2} + a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1 + \sum_{i=2}^{m-1} (a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2}).$$

For the case study of m=13, the following concise formulation is obtained: $\hat{p}_X = p_A + a_{11} + a_{12}$.

Proof. The field polynomial used in this architecture is $p(x) = x^{13} + x^4 + x^3 + x + 1$. Therefore, $f_{13} = f_4 = f_3 = f_1 = f_0 = 1$. Then, one obtains

$$\hat{p}_X = a_{11} + a_{12} + a_{11} + a_{12} + a_0 + a_{11} + a_1 + a_{12} + a_{11} + a_2 + a_{12} + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 + a_9 + a_{10} = p_A + a_{11} + a_{12}.$$

This completes the proof.

Theorem 3.3 *Let $p_{Ae} = \sum_{i=0}^{\frac{m-1}{2}} a_{2i}$ be the even parity for bits of A, and $p_{Ao} = \sum_{i=1}^{\frac{m-1}{2}} a_{2i-1}$ be the odd parity for bits of A, where m is odd and used for the dimension of the code subspace, and also f_i in GF(2) for $i = 0, 1, \ldots, m-1$. Then, the even and odd predicted parities of X, referred to as $\hat{p}_{Xe}$ and $\hat{p}_{Xo}$, respectively, are*

$$\hat{p}_{Xe} = a_{m-1} \cdot f_{m-1} + a_{m-2} + \sum_{i=1}^{\frac{m-1}{2}} (a_{m-1} \cdot f_{2i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_{2i} + a_{2i-2})$$

and

$$\hat{p}_{Xo} = a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1 + \sum_{i=1}^{\frac{m-3}{2}} (a_{m-1} \cdot f_{2i} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_{2i+1} + a_{2i-1}).$$

For the case study of m=13, the following concise formulations are obtained: $\hat{p}_{Xe} = p_{Ae} + a_{12}$ *and* $\hat{p}_{Xo} = p_{Ao} + a_{11}$.

Proof. The predicted parity $\hat{p}_X$ can be split as

$$\hat{p}_X = a_{m-1} \cdot f_{m-1} + a_{m-2} + a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1 +$$
$$\sum_{i=1}^{m-1} (a_{m-1} \cdot f_{2i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_{2i} + a_{2i-2}) = \hat{p}_{Xe} + \hat{p}_{Xo}.$$

Since the field polynomial used in this architecture is $p(x) = x^{13} + x^4 + x^3 + x + 1$, one obtains $f_{13} = f_4 = f_3 = f_1 = f_0 = 1$. Then,

$$\hat{p}_{Xe} = a_{11} + a_{12} + a_0 + a_{12} + a_{11} + a_2 + a_4 + a_6 + a_8 + a_{10} = p_{Ae} + a_{12}$$

and

$$\hat{p}_{Xo} = a_{12} + a_{11} + a_{11} + a_1 + a_{12} + a_3 + a_5 + a_7 + a_9 = p_{Ao} + a_{11}.$$

This completes the proof.

In Table 3, a summary of the different normal and interleaved signatures for the α and α^2 modules is shown. Additionally, Fig. 3.2 illustrates the architecture of the "H-generator" with the proposed error detection schemes. This architecture allows to perform the multiplication of matrices X, Y, and Z to obtain the control matrix H. This "H-generator" can use both normal and interleaved signatures denoted as $P_1 - P_4$ in Fig. 3.2. P_2 and P_4 blocks are signatures for finite field multiplication operations, as derived in Theorem 3.1. P_1 is also for finite field multiplications, after the Horner schemes are applied. P_3 block performs normal and interleaved parities for finite field multiplications and squarings, derived through Theorem 3.1 for the finite field multiplications, and through Theorem 3.2 and Theorem 3.3 for normal and interleaved signatures of finite field squarings, respectively. Using the derived formulations, one obtains the eventual error indication flag denoted as H_GEN_Error in Fig. 3.2, which is the union of four error indication flags through the four signatures presented in this figure. Below the "H-generator" in Fig. 3.2, the architectures

Table 3: Normal and interleaved signatures for the α and α^2 modules.

Module	Case Study	Normal Signatures	Interleaved Signatures
α	General	$\hat{p}_X = a_{m-1} \cdot \sum_{i=1}^{m-1} f_i + \sum_{i=0}^{m-1} a_i$	$\hat{p}_{Xe} = a_{m-1} + \sum_{i=1}^{\frac{m-1}{2}} (a_{2i-1} + a_{m-1} \cdot f_{2i})$, $\hat{p}_{Xo} = \sum_{i=1}^{\frac{m-1}{2}} (a_{(2i-1)-1} + a_{m-1} \cdot f_{2i-1})$
	m=13	$\hat{p}_X = p_A + a_{12}$	$\hat{p}_{Xe} = p_{Ao}$, $\hat{p}_{Xo} = p_{Ae} + a_{12}$
α^2	General	$\hat{p}_X = a_{m-1} \cdot f_{m-1} + a_{m-2} + a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1 + \sum_{i=2}^{m-1} (a_{m-1} \cdot f_{i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2})$	$\hat{p}_{Xe} = a_{m-1} \cdot f_{m-1} + a_{m-2} + \sum_{i=1}^{\frac{m-1}{2}} (a_{m-1} \cdot f_{2i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_{2i} + a_{2i-2})$, $\hat{p}_{Xo} = a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1 + \sum_{i=1}^{\frac{m-3}{2}} (a_{m-1} \cdot f_{2i} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_{2i+1} + a_{2i-1})$
	m=13	$\hat{p}_X = p_A + a_{11} + a_{12}$	$\hat{p}_{Xe} = p_{Ae} + a_{12}$, $\hat{p}_{Xo} = p_{Ao} + a_{11}$

for the α and α^2 modules of the normal and interleaved signature blocks are derived as an example. P_1, P_2, and P_4 use only finite field multiplications, while P_3 utilizes both finite field multiplications and squarings.

3.2.2 CRC Signatures

The choice of the utilized CRCs can be tailored based on the reliability requirements and the overhead to be tolerated. CRC signatures in the *sum* and *pass-thru* modules do not require as much derivations as the ones needed for α and α^2 modules. For the *sum* module, the predicted CRC-*1* signature $\hat{p}_x$ is equal to the *sum* of the parity bits of the input elements A and B in $GF(2^m)$, $\hat{p}_X = p_A + p_B$. Moreover, for the *pass-thru* module, $\hat{p}_X = b \cdot p_A$, where b is an element in *GF(2)*. For any CRC-X scheme, instead of summing all the parity bits (which is done in CRC-*1*), it checks X bits at a time in the *sum* and *pass-thru* modules. In the following, the NIST field $GF(2^{13})$ is used with CRC-*2* and CRC-*8*; however, the proposed

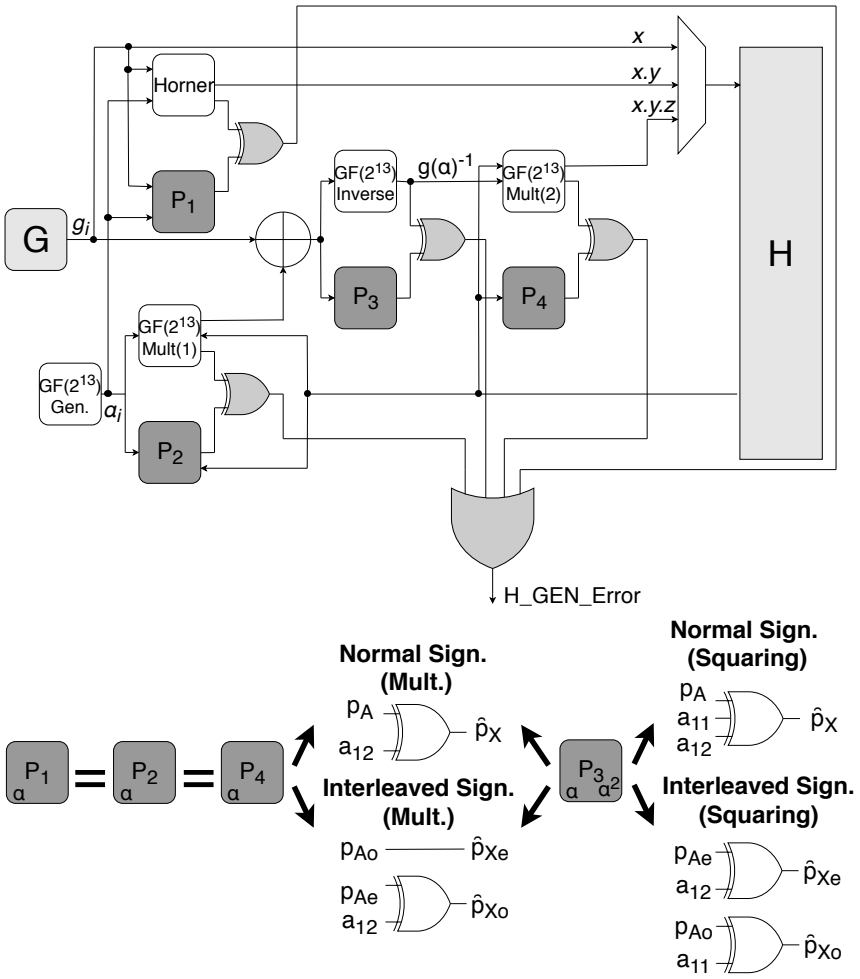


Figure 3.2: The proposed error detection of "H-generator" for MECS with the error detection blocks for normal and interleaved signatures.

fault detection schemes are applicable to any field size and CRC signature. Additionally, the field polynomial used is $p(x) = x^{13} + x^4 + x^3 + x + 1$.

1. CRC for α Module: In the α module, the multiplication of any element in $GF(2^{13})$ by x gives

$$A(x) \cdot x = a_{12} \cdot x^{13} + a_{11} \cdot x^{12} + ... + a_1 \cdot x^2 + a_0 \cdot x,$$

where

$$x^{13} = f_{12}x^{12} + f_{11}x^{11} + ... + f_1 x + f_0 \bmod p(x).$$

• For *m=13* with CRC-*2*, the generator polynomial used is $g_0(x) = x^2 + x + 1$. To find its signatures, $g_0(x)$ is used as follows,

$$x^2 \equiv x + 1 \bmod g_0(x)$$
$$x^3 \equiv 1 \bmod g_0(x)$$
$$x^4 \equiv x \bmod g_0(x)$$
$$x^5 \equiv x + 1 \bmod g_0(x)$$
$$x^6 \equiv 1 \bmod g_0(x)$$
$$x^7 \equiv x \bmod g_0(x)$$
$$x^8 \equiv x + 1 \bmod g_0(x)$$
$$x^9 \equiv 1 \bmod g_0(x)$$
$$x^{10} \equiv x \bmod g_0(x)$$
$$x^{11} \equiv x + 1 \bmod g_0(x)$$
$$x^{12} \equiv 1 \bmod g_0(x).$$

The irreducible polynomial $p(x) = x^{13} + x^4 + x^3 + x + 1$ is applied to obtain

$$A(x) \cdot x \equiv a_{12}x^4 + a_{12}x^3 + a_{12}x + a_{12} + a_{11}x^{12} + a_{10}x^{11} + a_9x^{10} + a_8x^9 +$$
$$a_7x^8 + a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x \bmod p(x).$$

Then, the generator polynomial $g_0(x)$ is applied to calculate the predicted CRC-*2* for $GF(2^{13})$ in the α module ($PCRC2_{13}$),

$$A(x) \cdot x \equiv a_{11} + a_{10}(x + 1) + a_9x + a_8 + a_7(x + 1) + a_6x + a_5 + a_4(x + 1) +$$
$$a_3x + a_2 + a_1(x + 1) + a_0x \bmod g_0(x),$$

or

$$PCRC2_{13} = (a_{10} + a_9 + a_7 + a_6 + a_4 + a_3 + a_1 + a_0)x +$$
$$(a_{11} + a_{10} + a_8 + a_7 + a_5 + a_4 + a_2 + a_1).$$

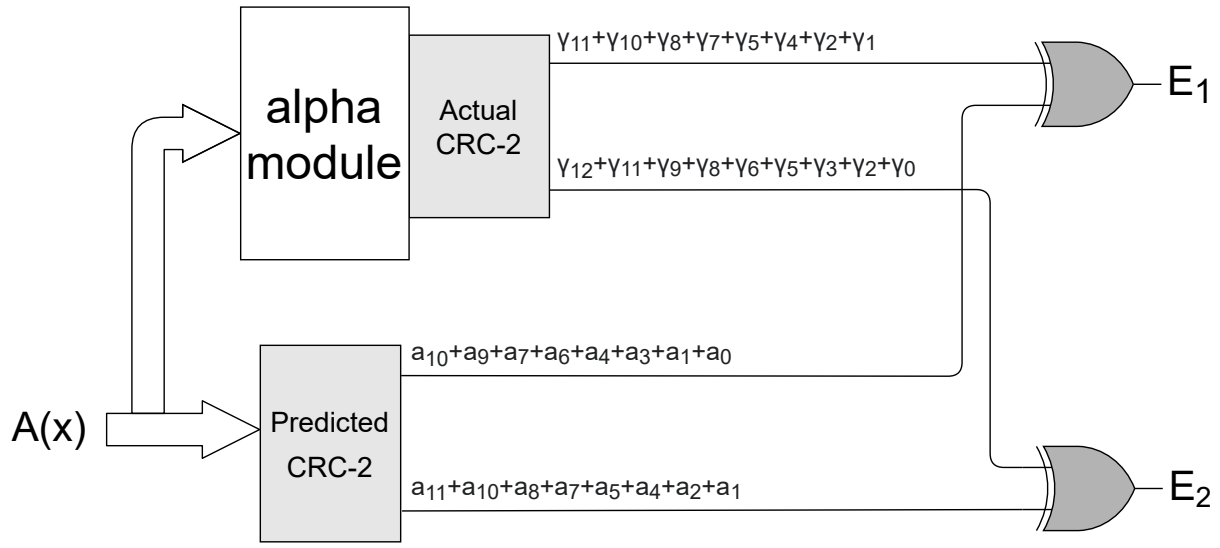


Figure 3.3: The proposed CRC-*2* error detection scheme for the α module.

We rename the coefficients to calculate the actual CRC-*2* for $GF(2^{13})$ in the α module ($ACRC2_{13}$): a_{11} as γ_{12},..., a_0 as γ_1,

$$A(x) \cdot x \equiv \gamma_{12}x^{12} + \gamma_{11}x^{11} + \gamma_{10}x^{10} + \gamma_9 x^9 + \gamma_8 x^8 + \gamma_7 x^7 + \gamma_6 x^6 + \gamma_5 x^5 + \gamma_4 x^4 + \gamma_3 x^3 + \gamma_2 x^2 + \gamma_1 x^1 + \gamma_0 \bmod g_0(x),$$

and the generator polynomial is applied as follows

$$A(x) \cdot x \equiv \gamma_{12} + \gamma_{11}(x+1) + \gamma_{10}x + \gamma_9 + \gamma_8(x+1) + \gamma_7 x + \gamma_6 + \gamma_5(x+1) + \gamma_4 x + \gamma_3 + \gamma_2(x+1) + \gamma_1 x + \gamma_0 \bmod g_0(x),$$

or

$$ACRC2_{13} = (\gamma_{11} + \gamma_{10} + \gamma_8 + \gamma_7 + \gamma_5 + \gamma_4 + \gamma_2 + \gamma_1)x + (\gamma_{12} + \gamma_{11} + \gamma_9 + \gamma_8 + \gamma_6 + \gamma_5 + \gamma_3 + \gamma_2 + \gamma_0).$$

Fig. 3.3 illustrates how the error detection scheme based on CRC-*2* works in the α module.

• For *m=13* with CRC-*8*, the generator polynomial used is $g_1(x) = x^8 + x^2 + x + 1$. To find its signatures, $g_1(x)$ is used as follows,

$$x^8 \equiv x^2 + x + 1 \bmod g_1(x)$$
$$x^9 \equiv x^3 + x^2 + x \bmod g_1(x)$$
$$x^{10} \equiv x^4 + x^3 + x^2 \bmod g_1(x)$$
$$x^{11} \equiv x^5 + x^4 + x^3 \bmod g_1(x)$$
$$x^{12} \equiv x^6 + x^5 + x^4 \bmod g_1(x).$$

The irreducible polynomial $p(x) = x^{13} + x^4 + x^3 + x + 1$ is applied to obtain

$$A(x) \cdot x \equiv a_{12}x^4 + a_{12}x^3 + a_{12}x + a_{12} + a_{11}x^{12} + a_{10}x^{11} + a_9x^{10} + a_8x^9 + a_7x^8 +$$
$$a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x \bmod p(x).$$

Then, the generator polynomial $g_1(x)$ is applied to calculate the predicted CRC-*8* for $GF(2^{13})$ in the α module ($PCRC8_{13}$),

$$A(x) \cdot x \equiv a_{12}(x^4 + x^3 + x + 1) + a_{11}(x^6 + x^5 + x^4) + a_{10}(x^5 + x^4 + x^3) +$$
$$a_9(x^4 + x^3 + x^2) + a_8(x^3 + x^2 + x) + a_7(x^2 + x + 1) + a_6x^7 + a_5x^6 +$$
$$a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x \bmod g_1(x),$$

or

$$PCRC8_{13} = a_6x^7 + (a_{11} + a_5)x^6 + (a_{11} + a_{10} + a_4)x^5 + (a_{12} + a_{11} + a_{10} +$$
$$a_9 + a_3) \cdot x^4 + (a_{12} + a_{10} + a_9 + a_8 + a_2)x^3 + (a_9 + a_8 + a_7 + a_1)x^2 +$$
$$(a_{12} + a_8 + a_7 + a_0)x + (a_{12} + a_7).$$

We rename the coefficients to calculate the actual CRC-*8* for $GF(2^{13})$ in the α module ($ACRC8_{13}$): a_{11} as γ_{12},..., a_0 as γ_1,

$$A(x) \cdot x \equiv \gamma_{12}x^{12} + \gamma_{11}x^{11} + \gamma_{10}x^{10} + \gamma_9x^9 + \gamma_8x^8 + \gamma_7x^7 + \gamma_6x^6 + \gamma_5x^5 +$$
$$\gamma_4x^4 + \gamma_3x^3 + \gamma_2x^2 + \gamma_1x + \gamma_0 \bmod g_1(x),$$

and the generator polynomial is applied as follows

$$A(x) \cdot x \equiv \gamma_{12}(x^6 + x^5 + x^4) + \gamma_{11}(x^5 + x^4 + x^3) + \gamma_{10}(x^4 + x^3 + x^2) +$$
$$\gamma_9(x^3 + x^2 + x) + \gamma_8(x_2 + x + 1) + \gamma_7 x^7 + \gamma_6 x^6 + \gamma_5 x^5 + \gamma_4 x^4 + \gamma_3 x^3$$
$$+\gamma_2 x^2 + \gamma_1 x + \gamma_0 \bmod g_1(x),$$

or

$$ACRC8_{13} = \gamma_7 x^7 + (\gamma_{12} + \gamma_6)x^6 + (\gamma_{12} + \gamma_{11} + \gamma_5)x^5 + (\gamma_{12} + \gamma_{11} + \gamma_{10} +$$
$$\gamma_4)x^4 + (\gamma_{11} + \gamma_{10} + \gamma_9 + \gamma_3)x^3 + (\gamma_{10} + \gamma_9 + \gamma_8 + \gamma_2)x^2 + (\gamma_9 + \gamma_8 + \gamma_1)x$$
$$+(\gamma_8 + \gamma_0).$$

Next, for α^2 module, the NIST field $GF(2^{13})$ is used with CRC-*2* and CRC-*8*; however, the proposed fault detection schemes are applicable to any field size and CRC signature.

2. CRC for α^2 Module: In the α^2 module, the multiplication of any element in $GF(2^{13})$ by x gives

$$A(x) \cdot x^2 = a_{12} \cdot x^{14} + a_{11} \cdot x^{13} + ... + a_1 \cdot x^3 + a_0 \cdot x^2,$$

where

$$x^{14} = f_{12}x^{13} + f_{11}x^{12} + ... + f_1 x^2 + f_0 x \bmod p(x)$$

and

$$x^{13} = f_{12}x^{12} + f_{11}x^{11} + ... + f_1 x + f_0 \bmod p(x).$$

• For *m=13* with CRC-*2*, the irreducible polynomial $p(x) = x^{13} + x^4 + x^3 + x + 1$ is applied to obtain

$$A(x) \cdot x^2 \equiv a_{12}x^5 + a_{12}x^4 + a_{12}x^2 + a_{12}x + a_{11}x^4 a_{11}x^3 + a_{11}x + a_{11} +$$
$$a_{10}x^{12} + a_9 x^{11} + a_8 x^{10} + a_7 x^9 + a_6 x^8 + a_5 x^7 + a_4 x^6 + a_3 x^5 + a_2 x^4 +$$
$$a_1 x^3 + a_0 x^2 \bmod p(x).$$

Then, the generator polynomial $g_0(x)$ is applied to calculate the predicted CRC-*2* for $GF(2^{13})$ in the α^2 module ($PCRC2_{13}$),

$$A(x) \cdot x^2 \equiv a_{10} + a_9(x+1) + a_8x + a_7 + a_6(x+1) + a_5x + a_4 + a_3(x+1) + a_2x + a_1 + a_0(x+1) \bmod g_0(x),$$

or

$$PCRC2_{13} = (a_9 + a_8 + a_6 + a_5 + a_3 + a_2 + a_0)x + (a_{10} + a_9 + a_7 + a_6 + a_4 + a_3 + a_1 + a_0).$$

We rename the coefficients to calculate the actual CRC-*2* for $GF(2^{13})$ in the α^2 module ($ACRC2_{13}$), obtaining the same formulations as for the α module.

- For *m=13* with CRC-*8*, the irreducible polynomial $p(x) = x^{13} + x^4 + x^3 + x + 1$ is applied to obtain

$$A(x) \cdot x^2 \equiv a_{12}x^5 + a_{12}x^4 + a_{12}x^2 + a_{12}x + a_{11}x^4 a_{11}x^3 + a_{11}x + a_{11} + a_{10}x^{12} + a_9x^{11} + a_8x^{10} + a_7x^9 + a_6x^8 + a_5x^7 + a_4x^6 + a_3x^5 + a_2x^4 + a_1x^3 + a_0x^2 \bmod p(x).$$

Then, the generator polynomial $g_1(x)$ is applied to calculate the predicted CRC-*8* for $GF(2^{13})$ in the α^2 module ($PCRC8_{13}$),

$$A(x) \cdot x^2 \equiv a_{12}(x^5 + x^4 + x^2 + x) + a_{11}(x^4 + x^3 + x + 1) + a_{10}(x^6 + x^5 + x^4) + a_9(x^5 + x^4 + x^3) + a_8(x^4 + x^3 + x^2) + a_7(x^3 + x^2 + x) + a_6(x^2 + x + 1) + a_5x^7 + a_4x^6 + a_3x^5 + a_2x^4 + a_1x^3 + a_0x^2 \bmod g_1(x)$$

or

$$PCRC8_{13} = a_5x^7 + (a_{10} + a_4)x^6 + (a_{12} + a_{10} + a_9 + a_3)x^5 + (a_{12} + a_{11} + a_{10} + a_9 + a_8 + a_2)x^4 + (a_{11} + a_9 + a_8 + a_7 + a_1)x^3 + (a_{12} + a_8 + a_7 + a_6 + a_0)x^2 + (a_{12} + a_{11} + a_7 + a_6)x + (a_{11} + a_6).$$

Table 4: Different CRC signatures for the α and α^2 modules.

Module	CRC	Predicted CRC signatures	Actual CRC signatures
α	CRC-2	$(a_{10}+a_9+a_7+a_6+a_4+a_3+a_1+a_0)x+(a_{11}+a_{10}+a_8+a_7+a_5+a_4+a_2+a_1)$	$(\gamma_{11}+\gamma_{10}+\gamma_8+\gamma_7+\gamma_5+\gamma_4+\gamma_2+\gamma_1)x+(\gamma_{12}+\gamma_{11}+\gamma_9+\gamma_8+\gamma_6+\gamma_5+\gamma_3+\gamma_2+\gamma_0)$
	CRC-8	$a_6x^7+(a_{11}+a_5)x^6+(a_{11}+a_{10}+a_4)x^5+(a_{12}+a_{11}+a_{10}+a_9+a_3)\cdot x^4+(a_{12}+a_{10}+a_9+a_8+a_2)x^3+(a_9+a_8+a_7+a_1)x^2+(a_{12}+a_8+a_7+a_0)x+(a_{12}+a_7)$	$\gamma_7x^7+(\gamma_{12}+\gamma_6)x^6+(\gamma_{12}+\gamma_{11}+\gamma_5)x^5+(\gamma_{12}+\gamma_{11}+\gamma_{10}+\gamma_4)x^4+(\gamma_{11}+\gamma_{10}+\gamma_9+\gamma_3)x^3+(\gamma_{10}+\gamma_9+\gamma_8+\gamma_2)x^2+(\gamma_9+\gamma_8+\gamma_1)x+(\gamma_8+\gamma_0)$
α^2	CRC-2	$(a_9+a_8+a_6+a_5+a_3+a_2+a_0)x+(a_{10}+a_9+a_7+a_6+a_4+a_3+a_1+a_0)$	$(\gamma_{11}+\gamma_{10}+\gamma_8+\gamma_7+\gamma_5+\gamma_4+\gamma_2+\gamma_1)x+(\gamma_{12}+\gamma_{11}+\gamma_9+\gamma_8+\gamma_6+\gamma_5+\gamma_3+\gamma_2+\gamma_0)$
	CRC-8	$a_5x^7+(a_{10}+a_4)x^6+(a_{12}+a_{10}+a_9+a_3)x^5+(a_{12}+a_{11}+a_{10}+a_9+a_8+a_2)x^4+(a_{11}+a_9+a_8+a_7+a_1)x^3+(a_{12}+a_8+a_7+a_6+a_0)x^2+(a_{12}+a_{11}+a_7+a_6)x+(a_{11}+a_6)$	$\gamma_7x^7+(\gamma_{12}+\gamma_6)x^6+(\gamma_{12}+\gamma_{11}+\gamma_5)x^5+(\gamma_{12}+\gamma_{11}+\gamma_{10}+\gamma_4)x^4+(\gamma_{11}+\gamma_{10}+\gamma_9+\gamma_3)x^3+(\gamma_{10}+\gamma_9+\gamma_8+\gamma_2)x^2+(\gamma_9+\gamma_8+\gamma_1)x+(\gamma_8+\gamma_0)$

We rename the coefficients to calculate the actual CRC-*8* for $GF(2^{13})$ in the α^2 module ($ACRC8_{13}$), obtaining the same formulations as for the α module.

In Table 4, the different CRC signatures proposed in this work for the α and α^2 modules are shown.

3.3 Error Coverage and FPGA Implementations

As mentioned earlier, finite field multiplications use three different modules (α, *sum*, and *pass-thru* modules), squarings use two modules (α^2 and *sum* modules), and XOR operations use only the sum module. Since *m=13*, for bit-parallel structures using normal signatures, 12 α, 12 *sum*, and 13 *pass-thru* modules are needed in each finite field multiplication; 12 α^2,

Table 5: Steps to perform the inverse of $A \, \epsilon \, GF(2^{13})$ using addition chain.

Step	$\beta_{V_i}(x)$	$\beta_{V_j+U_k}(x)$	Exponentiation
1	$\beta_1(x)$	-	A
2	$\beta_2(x)$	$\beta_{1+1}(x)$	$(\beta_1)^{2^1}\beta_1 = A^{2^2-1}$
3	$\beta_3(x)$	$\beta_{2+1}(x)$	$(\beta_2)^{2^1}\beta_1 = A^{2^3-1}$
4	$\beta_6(x)$	$\beta_{3+3}(x)$	$(\beta_3)^{2^3}\beta_3 = A^{2^6-1}$
4	$\beta_{12}(x)$	$\beta_{6+6}(x)$	$(\beta_6)^{2^6}\beta_6 = A^{2^{12}-1}$

and 12 *sum* modules in finite field squaring; and 12 *sum* modules in finite field addition. For normal signatures, the number of signatures used by each $P_i, 1 \leq i \leq 4$, block is calculated as follows:

1. For deriving the matrix multiplication XY, a 128×128 matrix X is created. Then, Horner implementation is applied to perform effective polynomial evaluations. For each column of XY (which is a $128 \times 8,192$ matrix), 127 finite field multiplications and 127 XOR operations are used. Therefore, for the Horner block, $8,192 \cdot 127 \cdot (12+12+13)$ signatures for multiplications and $8,192 \cdot 127 \cdot 12$ signatures for XOR operations are needed, which makes a total of more than $5 \cdot 10^7$ signatures.

2. For deriving Z, first, the Mult.(1) block in Fig. 3.2 performs 8,192 multiplications and 8,192 XOR operations. Therefore, $8,192 \cdot (12 + 12 + 13)$ signatures for multiplications, and $8,192 \cdot 127 \cdot 12$ signatures for XOR operations are needed, which makes a total of more than 10^7 signatures.

3. Next, for the inverse block, 8,192 inversions are needed. The addition chain C obtained is $C = \{1, 2, 3, 6, 12\}$. The computational steps to calculate the inverse of $A \, \epsilon \, GF(2^{13})$ using such addition chain are illustrated in Table 5, where V_i are the integers in the addition chain, $V_j = V_{i-1}$, and $U_k = V_i - V_j$. As it is shown in Table 5, 4 finite field multiplications and 12 finite field squaring are required. Each multiplication in $GF(2^{13})$ uses 12 *sum* modules, 12 α modules, and *13 pass-thru* modules; on the other hand, each squaring in $GF(2^{13})$ uses 12 *sum* modules and 12 α^2 modules. Therefore, the total number of operations and signatures is $8,192 \cdot (4 \cdot (12 + 12 + 13) + 12 \cdot (12 + 12))$ or close to $3.6 \cdot 10^6$.

Table 6: Implementation results on Xilinx FPGA family Kintex-7 for device xc7k70tfbv676-1.

Architecture	Area (occupied slices)	Delay (ns)	Power (mW) @50 MHz
Horner block	2976	28.267	0.144
Horner Normal Sign.	3138 (5.44%)	28.541 (Neg.)	0.147 (Neg.)
Horner Inter. Sign.	3402 (14.31%)	29.410 (Neg.)	0.157 (9.03%)
Horner CRC-2 Sign.	3285 (10.38%)	28.850 (Neg.)	0.154 (6.94%)
Horner CRC-8 Sign.	3572 (20.03%)	29.803 (5.43%)	0.158 (9.72%)
Inversion block	783	28.820	0.101
Inversion Normal Sign.	976 (24.65%)	29.013 (Neg.)	0.108 (6.93%)
Inversion Inter. Sign.	1083 (38.31%)	28.995 (Neg.)	0.110 (8.91%)
Inversion CRC-2 Sign.	1121 (43.17%)	28.720 (Neg.)	0.112 (10.89%)
Inversion CRC-8 Sign.	1166 (48.91%)	31.124 (7.99%)	0.113 (11.88%)

4. Lastly, to obtain XYZ, a total of $8,192 \cdot 128$ multiplications are performed. Therefore, for the Mult.(2) block in Fig. 3.2, $8,192 \cdot 128 \cdot (12 + 12 + 13)$ signatures for multiplications are needed, which makes a total of approximately $3 \cdot 10^7$ signatures.

5. After generating the control matrix H, other operations are performed. The control matrix is permutated by multiplying it with a random permutation matrix P. The signatures can be integrated into the $\hat{H}$ generator since it performs close to $8.6 \cdot 10^9$ multiplications $(8,192 \cdot 8,192 \cdot 128)$ and also close to $8.6 \cdot 10^9$ additions $(8,192 \cdot 8,192 \cdot 128)$ of 13 bit-vectors. The key generation also uses the Gauss Systemizer block, which performs row permutation and XOR addition of two rows. Permutation in this case can be accomplished by rewiring, not modifying the signatures. Lastly, in the XOR addition of two rows, the signatures described here are added, since it performs XOR addition between vectors of 13 bits.

To calculate the error coverage percentage of the proposed signatures, the following formula $100 \cdot (1 - (\frac{1}{2})^{\#sign})\%$ is used, where the number of signatures is denoted as $\#sign$. Therefore, the high error coverage percentage with the proposed normal signatures is approximately $100 \cdot (1 - (\frac{1}{2})^{10^8})\%$, the error coverage percentage with the proposed interleaved and CRC-*2* signatures is $100 \cdot (1 - (\frac{1}{2})^{2 \cdot 10^8})\%$, and the error coverage percentage with the proposed CRC-*8* signatures is $100 \cdot (1 - (\frac{1}{2})^{8 \cdot 10^8})\%$. Furthermore, for local faults, one will only

take into account the signatures needed for one of the four blocks in the proposed scheme depicted in Fig. 3.2. As an example, if the faults are confined to the inversion block, the error coverage is $100 \cdot (1 - (\frac{1}{2})^{3.6 \cdot 10^6})\%$ for normal signatures, $100 \cdot (1 - (\frac{1}{2})^{7.2 \cdot 10^6})\%$ for interleaved and CRC-*2* signatures, and $100 \cdot (1 - (\frac{1}{2})^{2.9 \cdot 10^7})\%$ for CRC-*8* signatures. In Table 6, the overhead of the error detection architectures in terms of area (occupied slices), delay, and power (at the frequency of 50 MHz) for Horner and inversion blocks is presented. The different architectures have been implemented on Xilinx FPGA family Kintex-7. However, the target platform does not necessarily affect the results because of platform-obliviousness of the schemes. As shown in Table 6, when larger signatures are applied to the original architectures, with higher error coverage, they end up having higher overhead in terms of area and power. In terms of delay, the overheads difference is minimal and it varies according to the gates used in each architecture. Additionally, it can be seen that the bigger the entire architecture is, the less overhead is obtained. The inversion block performs less operations than the Horner block, which makes the overall overheads larger. Interleaved and CRC-*2* signatures are very similar since both have the same number of error flags, while CRC-*8* is the most expensive error detection scheme. This is expected since CRC signatures perform more operations ending up having higher error coverage, as seen in the assessments. The proposed schemes in this chapter have combined area and delay overheads of less than 49% (worst-case scenario for the inversion block with CRC-*8* signatures) and more than 5% (best-case scenario for the Horner block with normal signatures). The choice of the utilized signatures can be tailored based on the reliability requirements and the overhead to be tolerated. The overhead achieved is acceptable taking into account the very high error coverage of close to 100%.

Chapter 4: CRC-Oriented Error Detection Schemes Assessed on FPGA for the Niederreiter Key Generator

4.1 Niederreiter Cryptosystem

The Niederreiter cryptosystem is a variant of code-based cryptography, similar to the McEliece cryptosystem. At first, the Niederreiter cryptosystem used Reed-Salomon codes, but it has been proven that the Niederreiter cryptosystem is more secure using binary Goppa codes than Reed-Salomon codes, since the latter can be broken [63]. Researchers have developed other variants of the Niederreiter cryptosystem by using different codes other than binary Goppa codes such as quasi-cyclic codes and quasi-dyadic codes. The aim has been to lower the key size. While it is shown that other variants of the Niederreiter cryptosystem do not show particular weaknesses [64], [65], [66], it has been shown that the Niederreiter cryptosystem using binary Goppa codes is more secure against fault injection attacks than its variants [67]. Moreover, the size of the parameters is a high determining factor in the security of the cryptosystem. In this work, the chosen sizes for its binary field m are $11 \leq m \leq 14$. This is one of the merits of this work, not getting confined to one field size. The security parameters submitted to NIST in late 2017 [8] were for *m=13*. However, there exist prominent other works with different security parameters based on security necessities and system constraints [30], [59], [68], [69]. Therefore, the work in this chapter provides flexibility and tunable security by covering such works as well.

There has been previous work on countering fault attacks and providing reliability for PQC [40], [55], [70], and [71]. The authors in [40] and [55] perform fault detection for stateless hash-based PQC signatures; in [70], the authors use error detection schemes of number-theoretic transform; and the authors in [71] investigate several countermeasures

based on error detection checksum codes, and spatial/temporal redundancies for the NTRU encryption algorithm. Even though Niederreiter is based on error-correcting codes, it has been previously shown that it is vulnerable to fault injection attacks in the key generation process [67]. Redundancy is used in hardware implementations to achieve error control and reliability increase at the expense of area, delay, or product. These redundancy schemes can be classified into four different categories: Hardware, time, hybrid, and information redundancy. Hardware redundancy duplicates the function and compares both outputs, highly increasing the overhead. Time redundancy computes the function twice and compares both results, obtaining a significant delay overhead on the system. Hybrid redundancy performs the inverse of one part of the function, or the inverse of the whole function, which can suffer from more than 100% throughput. Lastly, information redundancy adds some check bits along the function and it validates when the result is obtained, producing some area overhead depending on the system implemented.

The Niederreiter cryptosystem consists of three operations: key generation, encryption, and decryption. The work performed in this chapter focuses on the key generation, which is the most expensive operation in the Niederreiter cryptosystem. In this process, a public key needed for the encryption of the message and a private key needed to decrypt the message are created. There are four important parameters, which can highly affect how secure the cryptosystem is: m, which is the extension field dimension; t, which is the number of correctable errors; the code length n; and the dimension k.

To create the private and public keys, n elements in $GF(2^m)$ are chosen randomly, i.e., $\alpha_0, \alpha_1, ..., \alpha_{n-1}$. Next, a monic irreducible polynomial in the form $g(\alpha) = \alpha^t + g_{t-1}\alpha^{t-1} + ... + g_1\alpha + g_0$, with degree t is created. A parity check matrix H is then computed with the following form:

$$H = \begin{bmatrix} \frac{1}{g(\alpha_0)} & \frac{1}{g(\alpha_1)} & \cdots & \frac{1}{g(\alpha_{n-1})} \\ \frac{\alpha_0}{g(\alpha_0)} & \frac{\alpha_1}{g(\alpha_1)} & \cdots & \frac{\alpha_{n-1}}{g(\alpha_{n-1})} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\alpha_0^{t-1}}{g(\alpha_0)} & \frac{\alpha_1^{t-1}}{g(\alpha_1)} & \cdots & \frac{\alpha_{n-1}^{t-1}}{g(\alpha_{n-1})} \end{bmatrix}.$$

Computing H is a costly process since it performs addition, multiplication, and inversion in $GF(2^m)$. In this chapter, the inversion is derived with squaring and multiplications by employing the polynomial variant of FLT to obtain a better performance. To get the binary parity check matrix, each element of the matrix H is replaced with a column of m bits. Lastly, the key generator transforms the modified $mt \times n$ H matrix to its systematic form $[\Pi_{mt}|K]$, and returns the public key K and the private key $(g(x), (\alpha_0, \alpha_1, ..., \alpha_{n-1}))$.

In the process of encryption, the sender encodes the message as an error vector e of length n and weight at most t and computes the ciphertext by multiplying the public parity matrix H. Such operation produces a specific syndrome which is sent to the receiver as the ciphertext c. When the receiver obtains such ciphertext, the original plaintext is recovered by using the private key $(g(x), (\alpha_0, \alpha_1, ..., \alpha_{n-1}))$. The decryption process takes polynomial time by using an efficient syndrome decoding algorithm of the binary Goppa code. Since this chapter focuses mainly on the key generation, readers interested in other aspects of the Niederreiter cryptosystem can refer to [72].

4.2 Proposed Fault Detection Schemes

Fault analysis attacks are those which intend to recover the private key by comparing the correct output with the faulty one. The proposed detection architectures are especially important against fault injection when the private key is re-computed from a seed before each decryption operation, allowing the attacker to target the generation of the same private key repeatedly.

Table 7: Security parameters used to derive the different fault detection schemes proposed.

m	t	n	k
11	40	2,048	1,608
12	67	3,408	2,604
13	119	6,960	5,413
14	15	16,384	16,174

In this chapter, different fault detection schemes based on normal and interleaved signatures, CRC-*3*, and CRC-*4* are proposed. These schemes aim to detect transient and permanent internal faults on the key generator. In fault attacks (intentional, malicious fault injections), preferably, single-bit faults using the stuck-at model are injected. By repeatedly comparing the erroneous and error-free outputs, the last sub-key is derived, and eventually, the secret key is compromised (noting the technological constraints, an attacker may not be able to inject a single stuck-at fault; therefore, multiple bits might be flipped). The stuck-at fault model (both single and multiple) is able to model both natural and malicious faults and thus is utilized throughout this chapter to achieve this twofold goal of the proposed schemes [73]. This is one of the reasons that adjacent stuck-at faults need to be considered in fault models as well. Such fault models consider both malicious faults and also natural faults based on stuck-at faults considered in this chapter. Normal parity detects 1-bit errors, which is useful when 1-bit or an odd number of bits are affected. Interleaved parity is used for 2-bit error detection, and it is proposed when faults are adjacent since it divides the predicted parities into an odd-term and an even-term part. Lastly, the CRCs proposed in this chapter (CRC-*3* and CRC-*4*) allow multiple error detections by detecting *3*-bit and *4*-bit errors. To implement CRC, a generator polynomial $g_p(\beta)$ is required and a fixed number of check bits are appended to the data to detect any errors [62]. In the error schemes proposed in this chapter, the formulations for predicted CRC signatures are compared via XOR gates with the actual CRC.

The main operations within the key generator of the Niederreiter cryptosystem are $GF(2^m)$ addition, $GF(2^m)$ multiplication, and $GF(2^m)$ inversion. The proposed error detection architectures can be embedded in the original blocks of the key generator. The security parameters used are shown in Table 7, where it can be seen how m varies from 12 to 14 bits, which appears in [69], [59], [30], and [68], respectively. Moreover, the set for *m=13* corresponds to the Classic McEliece NIST submission [8].

4.2.1 $GF(2^m)$ Addition and $GF(2^m)$ Multiplication

To perform $GF(2^m)$ addition, the *sum* module described in [44] is used. $GF(2^m)$ multiplication is done in three different modules: *sum*, α, and *pass-thru* modules. The *sum* module adds two elements in $GF(2^m)$ using m two-input XOR gates; the α module multiplies an element of $GF(2^m)$ by α and it reduces the result modulo $F(\alpha)$; and the *pass-thru* module multiplies a $GF(2^m)$ element by a *GF(2)* element.

In multiplication using polynomial basis, the inputs A and B are elements of $GF(2^m)$ in the form of

$$A = \sum_{i=0}^{m-1} a_i \alpha^i,\ a_i \,\epsilon\, \{0,\ 1\}$$

and

$$B = \sum_{i=0}^{m-1} b_i \alpha^i,\ b_i \,\epsilon\, \{0,\ 1\},$$

where α^i and b^i are the coordinates of each input. The multiplication of these two elements can be represented as:

$$A \cdot B = A \cdot \sum_{i=0}^{m-1} b_i \alpha^i = \sum_{i=0}^{m-1} b_i (A\alpha^i).$$

This, in turn, obtains the output C as follows:

$$C = A \cdot B \bmod F(\alpha) = \sum_{i=0}^{m-1} b_i X^{(i)},$$

where

$$X^{(i)} = \alpha \cdot X^{(i-1)} \bmod F(\alpha),\ 1 \leq i \leq m-1$$

$$X^{(0)} = A.$$

1. Normal Signature: For the *sum* and *pass-thru* modules, the predicted parities of the output $\hat{p}_x$ are calculated with the same formulations for all m's. For the *sum* module, $\hat{p}_x$ is equal to the *sum* of the parity bits of the input elements A and B in $GF(2^m)$, $\hat{p}_X = p_A + p_B$. Moreover, for the *pass-thru* module, $\hat{p}_X = b \cdot p_A$, where b is an element in *GF(2)*. Lastly, for the α module, $\hat{p}_X = p_A + a_{m-1}$.

2: Interleaved Signature: To find the predicted parities of the output X, the parity bits of A and B are divided into even parity bits, $\hat{p}_{Ae}$ and $\hat{p}_{Be}$, and odd parity bits, $\hat{p}_{Ao}$ and $\hat{p}_{Bo}$. For the *sum* module, $\hat{p}_{Xe} = p_{Ae} + p_{Be}$ and $\hat{p}_{Xo} = p_{Ao} + p_{Bo}$ for all m's. Furthermore, for the *pass-thru* module, $\hat{p}_{Xe} = b \cdot p_{Ae}$ and $\hat{p}_{Xo} = b \cdot p_{Ao}$ for all m's.

The multiplication of any element in $GF(2^m)$ by α gives

$$A(\alpha) \cdot \alpha = a_{m-1}\alpha^m + a_{m-2}\alpha^{m-1} + ... + a_0\alpha.$$

In the α module, if m is odd then the parity bits of element A are divided into $p_{Ae} = \sum_{i=0}^{\frac{m-1}{2}} a_{2i}$ and $p_{Ao} = \sum_{i=1}^{\frac{m-1}{2}} a_{2i-1}$, obtaining:

$$\hat{p}_{Xe} = a_{m-1} + \sum_{i=1}^{\frac{m-1}{2}} (a_{2i-1} + a_{m-1} \cdot f_{2i})$$

and

$$\hat{p}_{Xo} = \sum_{i=1}^{\frac{m-1}{2}} (a_{(2i-1)-1} + a_{m-1} \cdot f_{2i-1}).$$

Moreover, if m is even then the parity bits of element A are divided into $p_{Ae} = \sum_{i=0}^{\frac{m-2}{2}} a_{2i}$ and $p_{Ao} = \sum_{i=1}^{\frac{m}{2}} a_{2i-1}$, obtaining:

$$\hat{p}_{Xe} = a_{m-1} + \sum_{i=1}^{\frac{m-2}{2}} (a_{2i-1} + a_{m-1} \cdot f_{2i})$$

Table 8: Predicted interleaved parities for different m's in the α module.

m	Predicted Even Parity	Predicted Odd Parity
11	p_{Ao}	$p_{Ae} + a_{10}$
12		$p_{Ae} + a_{11}$
13		$p_{Ae} + a_{12}$
14		$p_{Ae} + a_{13}$

and

$$\hat{p}_{Xo} = \sum_{i=1}^{\frac{m}{2}} (a_{(2i-1)-1} + a_{m-1} \cdot f_{2i-1}).$$

In Table 8, the predicted interleaved parities for the different m's in the α module are presented. The derivations and proofs of these are not presented for the sake of brevity.

3. Cyclic Redundancy Check: For the *sum* and *pass-thru* modules, it follows the same approach as the parity checks. However, instead of 1-bit or 2-bit checks as shown previously for the normal and interleaved signatures, respectively; it checks 3 bits and 4 bits at a time for CRC-*3* and CRC-*4*, respectively. For the α module, $g_{p_1}(\beta) = \beta^3 + \beta + 1$ is used as the generator polynomial for CRC-*3*, and $g_{p_2}(\beta) = \beta^4 + \beta + 1$ is used as the generator polynomial for CRC-*4*. To find the different signatures for each m, the fixed polynomials $g_{p_1}(\beta)$ and $g_{p_2}(\beta)$ are used, respectively.

According to $g_{p_1}(\beta)$:

$$\beta^3 \equiv \beta + 1 \bmod g_{p_1}(\beta)$$
$$\beta^4 \equiv \beta^2 + \beta \bmod g_{p_1}(\beta)$$
$$\beta^5 \equiv \beta^3 + \beta^2 \equiv \beta^2 + \beta + 1 \bmod g_{p_1}(\beta)$$
$$\vdots$$
$$\beta^{13} \equiv \beta^3 + \beta^2 + \beta \equiv \beta^2 + 1 \bmod g_{p_1}(\beta).$$

According to $g_{p_2}(\beta)$:

Table 9: Different CRC signatures for α module when *m=11* and *m=12*.

m	CRC	Predicted CRC Signature	Actual CRC Signatures
11	3	$(a_{10}+a_8+a_5+a_4+a_3+a_1)\alpha^2$ $+(a_9+a_7+a_4+a_3+a_2+a_0)\alpha$ $+(a_{10}+a_9+a_6+a_5+a_4+a_2)$	$(\gamma_9+\gamma_6+\gamma_5+\gamma_4+\gamma_2)\alpha^2$ $+(\gamma_{10}+\gamma_8+\gamma_5+\gamma_4+\gamma_3$ $+\gamma_1)\alpha+(\gamma_{10}+\gamma_7+\gamma_6$ $+\gamma_5+\gamma_3+\gamma_0)$
	4	$(a_8+a_6+a_5+a_2)\alpha^3+(a_{10}+a_9$ $+a_7+a_5+a_4+a_1)\alpha^2+(a_9+a_8$ $+a_6+a_4+a_3+a_0)\alpha+(a_{10}+a_9$ $+a_7+a_6+a_3)$	$(\gamma_9+\gamma_7+\gamma_6+\gamma_3)\alpha^3$ $+(\gamma_{10}+\gamma_8+\gamma_6+\gamma_5+\gamma_2)$ $\cdot\alpha^2+(\gamma_{10}+\gamma_9+\gamma_7+\gamma_5$ $+\gamma_4+\gamma_1)\alpha+(\gamma_{10}+\gamma_8+\gamma_7$ $+\gamma_4+\gamma_0)$
12	3	$(a_{10}+a_8+a_5+a_4+a_3+a_1)\alpha^2$ $+(a_{10}+a_9+a_7+a_4+a_3+a_2$ $+a_0)\alpha+(a_9+a_6+a_5+a_4+a_2)$	$(\gamma_{11}+\gamma_9+\gamma_6+\gamma_5+\gamma_4$ $+\gamma_2)\alpha^2+(\gamma_{11}+\gamma_{10}+\gamma_8$ $+\gamma_5+\gamma_4+\gamma_3+\gamma_1)\alpha+(\gamma_{10}$ $+\gamma_7+\gamma_6+\gamma_5+\gamma_3+\gamma_0)$
	4	$(a_{11}+a_{10}+a_8+a_6+a_5+a_2)\alpha^3$ $+(a_{11}+a_{10}+a_9+a_7+a_5+a_4+$ $a_1)\alpha^2+(a_{10}+a_9+a_8+a_6+a_4$ $+a_3+a_0)\alpha+(a_9+a_7+a_6+a_3)$	$(\gamma_{11}+\gamma_9+\gamma_7+\gamma_6+\gamma_3)\alpha^3$ $+(\gamma_{11}+\gamma_{10}+\gamma_8+\gamma_6+\gamma_5$ $+\gamma_2)\alpha^2+(\gamma_{11}+\gamma_{10}+\gamma_9+\gamma_7$ $+\gamma_5+\gamma_4+\gamma_1)\alpha+(\gamma_{10}+\gamma_8$ $+\gamma_7+\gamma_4+\gamma_0)$

$$\beta^4 \equiv \beta+1 \bmod g_{p_2}(\beta)$$
$$\beta^5 \equiv \beta^2+\beta \bmod g_{p_2}(\beta)$$
$$\beta^6 \equiv \beta^3+\beta^2 \bmod g_{p_2}(\beta)$$
$$\vdots$$
$$\beta^{13} \equiv \beta^4+\beta^3+\beta^2+\beta \equiv \beta^3+\beta^2+1 \bmod g_{p_2}(\beta).$$

Applying these generator polynomials, the CRC-*3* and CRC-*4* signatures are obtained for the different m's as shown in Table 9 and Table 10. To clarify this process, the case for *m=11* is explained below.

If *m=11*,

$$A(\alpha)\cdot\alpha = a_{10}\alpha^{11}+a_9\alpha^{10}+...+a_1\alpha^2+a_0\alpha.$$

Then, applying the irreducible polynomial $p(\alpha)=\alpha^{11}+\alpha^2+1$, one obtains

Table 10: Different CRC signatures for α module when *m=13* and *m=14*.

m	CRC	Predicted CRC Signature	Actual CRC Signatures
13	3	$(a_{12}+a_{11}+a_{10}+a_8+a_5+a_4$ $+a_3+a_1)\alpha^2+(a_{12}+a_{11}+a_{10}$ $+a_9+a_7+a_4+a_3+a_2+a_0)\alpha$ $+(a_{11}+a_9+a_6+a_5+a_4+a_2)$	$(\gamma_{12}+\gamma_{11}+\gamma_9+\gamma_6+\gamma_5$ $+\gamma_4+\gamma_2)\alpha^2+(\gamma_{12}+\gamma_{11}$ $+\gamma_{10}+\gamma_8+\gamma_5+\gamma_4+\gamma_3$ $+\gamma_1)\alpha+(\gamma_{12}+\gamma_{10}+\gamma_7+\gamma_6$ $+\gamma_5+\gamma_3+\gamma_0)$
	4	$(a_{12}+a_{11}+a_{10}+a_8+a_6+a_5$ $+a_2)\alpha^3+(a_{11}+a_{10}+a_9+a_7$ $+a_5+a_4+a_1)\alpha^2+(a_{11}+a_{10}$ $+a_9+a_8+a_6+a_4+a_3+a_0)\alpha$ $+(a_{11}+a_9+a_7+a_6+a_3)$	$(\gamma_{12}+\gamma_{11}+\gamma_9+\gamma_7+\gamma_6$ $+\gamma_3)\alpha^3+(\gamma_{12}+\gamma_{11}+\gamma_{10}$ $+\gamma_8+\gamma_6+\gamma_5+\gamma_2)\alpha^2+(\gamma_{12}$ $+\gamma_{11}+\gamma_{10}+\gamma_9+\gamma_7+\gamma_5$ $+\gamma_4+\gamma_1)\alpha+(\gamma_{12}+\gamma_{10}$ $+\gamma_8+\gamma_7+\gamma_4+\gamma_0)$
14	3	$(a_{13}+a_{12}+a_{11}+a_{10}+a_8+a_5$ $+a_4+a_3+a_1)\alpha^2+(a_{11}+a_{10}$ $+a_9+a_7+a_4+a_3+a_2+a_0)\alpha$ $+(a_{12}+a_{11}+a_9+a_6+a_5+a_4$ $+a_2)$	$(\gamma_{13}+\gamma_{12}+\gamma_{11}+\gamma_9+\gamma_6$ $+\gamma_5+\gamma_4+\gamma_2)\alpha^2+(\gamma_{12}+\gamma_{11}$ $+\gamma_{10}+\gamma_8+\gamma_5+\gamma_4+\gamma_3$ $+\gamma_1)\alpha+(\gamma_{13}+\gamma_{12}+\gamma_{10}+\gamma_7$ $+\gamma_6+\gamma_5+\gamma_3+\gamma_0)$
	4	$(a_{13}+a_{12}+a_{11}+a_{10}+a_8+a_6+a_5$ $+a_2)\alpha^3+(a_{12}+a_{11}+a_{10}+a_9+a_7$ $+a_5+a_4+a_1)\alpha^2+(a_{13}+a_{11}+a_{10}$ $+a_9+a_8+a_6+a_4+a_3+a_0)\alpha$ $+(a_{12}+a_{11}+a_9+a_7+a_6+a_3)$	$(\gamma_{13}+\gamma_{12}+\gamma_{11}+\gamma_9+\gamma_7$ $+\gamma_6+\gamma_3)\alpha^3+(\gamma_{13}+\gamma_{12}$ $+\gamma_{11}+\gamma_{10}+\gamma_8+\gamma_6+\gamma_5$ $+\gamma_2)\alpha^2+(\gamma_{12}+\gamma_{11}+\gamma_{10}+\gamma_9$ $+\gamma_7+\gamma_5+\gamma_4+\gamma_1)\alpha+(\gamma_{13}$ $+\gamma_{12}+\gamma_{10}+\gamma_8+\gamma_7+\gamma_4+\gamma_0)$

$$A(\alpha)\cdot\alpha\equiv a_{10}\alpha^2+a_{10}+a_9\alpha^{10}+a_8\alpha^9$$
$$+a_7\alpha^8+a_6\alpha^7+a_5\alpha^6+a_4\alpha^5+a_3\alpha^4$$
$$+a_2\alpha^3+a_1\alpha^2+a_0\alpha \bmod p(\alpha).$$

To calculate the predicted CRC-*3* for *m=11* in the α module ($PCRC3_{11}$), the generator polynomial is applied as follows

$$A(\alpha)\cdot\alpha\equiv a_{10}\alpha^2+a_{10}+a_9(\alpha+1)+a_8\alpha^2+a_7\alpha$$
$$+a_6+a_5(\alpha^2+1)+a_4(\alpha^2+\alpha+1)+a_3(\alpha^2+\alpha)$$
$$+a_2(\alpha+1)+a_1\alpha^2+a_0\alpha \bmod g_1(\alpha)$$

or

$$PCRC3_{11} = (a_{10} + a_8 + a_5 + a_4 + a_3$$
$$+a_1)\alpha^2 + (a_9 + a_7 + a_4 + a_3 + a_2 + a_0)$$
$$\cdot\alpha + (a_{10} + a_9 + a_6 + a_5 + a_4 + a_2).$$

Lastly, to calculate the actual CRC-*3* for *m=11* in the α module ($ACRC3_{11}$), the coefficients are renamed: a_9 as γ_{10},..., $a_{10} + a_1$ as γ_2,a_0 as γ_1, and a_{10} as γ_0,

$$A(\alpha) \cdot \alpha = \gamma_{10}\alpha^{10} + \gamma_9\alpha^9 + \gamma_8\alpha^8 + \gamma_7\alpha^7 + \gamma_6\alpha^6$$
$$+\gamma_5\alpha^5 + \gamma_4\alpha^4 + \gamma_3\alpha^3 + \gamma_2\alpha^2 + \gamma_1\alpha^1 + \gamma_0,$$

and the generator polynomial is applied as follows,

$$A(\alpha) \cdot \alpha = \gamma_{10}(\alpha + 1) + \gamma_9\alpha^2 + \gamma_8\alpha^1 + \gamma_7 + \gamma_6(\alpha^2$$
$$+1) + \gamma_5(\alpha^2 + \alpha + 1) + \gamma_4(\alpha^2 + \alpha) + \gamma_3(\alpha + 1)$$
$$+\gamma_2\alpha^2 + \gamma_1\alpha^1 + \gamma_0 \bmod g_1(\alpha),$$

or

$$ACRC3_{11} = (\gamma_9 + \gamma_6 + \gamma_5 + \gamma_4 + \gamma_2)\alpha^2$$
$$+(\gamma_{10} + \gamma_8 + \gamma_5 + \gamma_4 + \gamma_3 + \gamma_1)\alpha$$
$$+(\gamma_{10} + \gamma_7 + \gamma_6 + \gamma_5 + \gamma_3 + \gamma_0).$$

4.2.2 $GF(2^m)$ Inversion

To perform $GF(2^m)$ inversion, the schemes on this chapter are based on FLT, which applies traditional exponentiation techniques to solve $A^{-1} = A^{2^{m-2}}$, requiring *m-2* multiplications and *m-1* squarings. Another algorithm to perform $GF(2^m)$ inversion was introduced by Itoh and Tsujii (ITA), which is based on the observation that $1+2+2^2+\cdots+2^{m-2}$ can be decomposed as $1+2^n+2^{2n}+\cdots+2^{(k-2)n}$ as shown in [74], requiring $\lfloor log_2(m-1) \rfloor + H_2(m-1) - 1$ multiplications (where $H_2(m-1)$ is the Hamming weight) and *m-1* squarings. Next, an example for both FLT and ITA for *m=14* is done to clarify the differences.

- For ITA:

$$A^{-1} = A^{2^m-2} = A^{2^{14}-2} = A^{2(1+2+\cdots+2^{12})}. \; Then, (1 + 2 + \cdots + 2^{12})$$

can be written as:

$$1 + 2 + \cdots + 2^{12} = 1 + 2 \times (1 + 2) \times$$
$$(1 + 2^2) \times (1 + 2^4 \times (1 + 2^4)).$$

- For FLT:

$$A^{-1} = A^{2^m-2} = A^{2^{14}-2} = A^{16382} = ((((((((((((A^2$$
$$\times A)^2 \times A)^2 \times A)^2 \times A)^2 \times A)^2 \times A)^2 \times A)^2$$
$$\times A)^2 \times A)^2 \times A)^2 \times A)^2 \times A)^2.$$

Since both methods use multiplications and squarings in Galois fields, FLT and ITA can be used with the proposed error detection approaches for finite field multiplication presented in this chapter. The goal is not to compare such inversion methods but to emphasize that the proposed schemes for the sub-blocks of these schemes can be generally used for both. In other words, the above formulae for both schemes have multiplication as the main building blocks, for which error detection are proposed.

Fault detection schemes for $GF(2^m)$ squaring are presented below. $GF(2^m)$ squaring uses the *sum* module and an α^2 module instead of the α module presented previously. In the α^2 module, an element A is multiplied by α^2 to achieve:

$$A(\alpha) \cdot \alpha^2 = a_{m-1}\alpha^{m+1} + a_{m-2}\alpha^m + ... + a_0\alpha^2$$

.For normal and interleaved parities,

$$\alpha^{m+1} = f_{m-1}\alpha^m + f_{m-2}\alpha^{m-1} + ... + f_0\alpha \bmod p(x)$$

Table 11: Predicted parities for the different m's in the α^2 module.

m	Parity Type	Predicted Parity	
11	Normal	$p_A + a_{10} + a_9$	
	Interleaved	$p_{Ae} + a_{10}$	Even
		$p_{Ao} + a_9$	Odd
12	Normal	$p_A + a_{11} + a_{10}$	
	Interleaved	$p_{Ae} + a_{11}$	Even
		$p_{Ao} + a_{10}$	Odd
13	Normal	$p_A + a_{12} + a_{11}$	
	Interleaved	$p_{Ae} + a_{12}$	Even
		$p_{Ao} + a_{11}$	Odd
14	Normal	$p_A + a_{13} + a_{12}$	
	Interleaved	$p_{Ae} + a_{13}$	Even
		$p_{Ao} + a_{12}$	Odd

and

$$\alpha^m = f_{m-1}\alpha^{m-1} + f_{m-2}\alpha^{m-2} + ... + f_0 \bmod p(x).$$

1. Normal Signature: By following the previous equations, one can obtain the predicted parity of X as

$$\hat{p}_X = a_{m-1} \cdot f_{m-1} + a_{m-2} + a_{m-1} + (a_{m-1}$$
$$\cdot f_{m-1} + a_{m-2}) \cdot f_1 + \textstyle\sum_{i=2}^{m-1}(a_{m-1} \cdot f_{i-1}$$
$$+(a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2}).$$

In Table 11, the predicted parities for the different m's in the α^2 module are presented.

2. Interleaved Signature: To obtain the predicted even and odd parities of X in the α^2 module, the previous equations are used to obtain

$$\hat{p}_{Xe} = a_{m-1} \cdot f_{m-1} + a_{m-2} + \textstyle\sum_{i=1}^{\frac{m-1}{2}}(a_{m-1}$$
$$\cdot f_{2i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_{2i} + a_{2i-2})$$

and

$$\hat{p}_{Xo} = a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1 + \textstyle\sum_{i=1}^{\frac{m-3}{2}}(a_{m-1} \cdot f_{2i}$$
$$+(a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_{2i+1} + a_{2i-1}).$$

Table 12: Different CRC signatures for α^2 module when *m=11* and *m=12*.

m	CRC	Predicted CRC Signature
11	3	$(a_9 + a_7 + a_4 + a_3 + a_2 + a_0)\alpha^2$ $+(a_8 + a_6 + a_3 + a_2 + a_1)\alpha + (a_{10}$ $+a_9 + a_8 + a_5 + a_4 + a_3 + a_1)$
	4	$(a_{10} + a_7 + a_5 + a_4 + a_1)\alpha^3 + (a_9$ $+a_8 + a_6 + a_4 + a_3 + a_0)\alpha^2 + (a_{10}$ $+a_8 + a_7 + a_5 + a_3 + a_2)\alpha + (a_9$ $+a_8 + a_6 + a_5 + a_2)$
12	3	$(a_9 + a_7 + a_4 + a_3 + a_2 + a_0)\alpha^2$ $+(a_9 + a_8 + a_6 + a_3 + a_2 + a_1)\alpha$ $+(a_8 + a_5 + a_4 + a_3 + a_1)$
	4	$(a_{11} + a_{10} + a_9 + a_7 + a_5 + a_4$ $+a_1)\alpha^3 + (a_{10} + a_9 + a_8 + a_6$ $+a_4 + a_3 + a_0)\alpha^2 + (a_{11} + a_9$ $+a_8 + a_7 + a_5 + a_3 + a_2)\alpha$ $+(a_{11} + a_8 + a_6 + a_5 + a_2)$

In Table 11, the predicted interleaved parities for the different m's in the α^2 module are presented.

3. Cyclic Redundancy Check: The element A is multiplied by α^2; therefore, the actual and predicted CRCs for the different values of m differ. In Table 12 and Table 13, the predicted CRC signatures for the α^2 module are presented (the actual CRC signatures of the α^2 module are the same as the ones from the α module). To clarify this process, *m=14* is used as an example to show how the predicted CRC-*4* and actual CRC-*4* are calculated. First,

$$A(\alpha) \cdot \alpha^2 = a_{13}\alpha^{15} + a_{12}\alpha^{14} + ... + a_1\alpha^3 + a_0\alpha^2.$$

Then, applying the irreducible polynomial $p(\alpha) = \alpha^{14} + \alpha^8 + \alpha^6 + \alpha + 1$, one obtains

$$A(\alpha) \cdot \alpha^2 = a_{13}\alpha^9 + a_{13}\alpha^7 + a_{13}\alpha^2 + a_{13}\alpha + a_{12}\alpha^8 a_{12}\alpha^6 + a_{12}\alpha + a_{12} + a_{11}\alpha^{13} + a_{10}\alpha^{12}$$
$$+a_9\alpha^{11} + a_8\alpha^{10} + a_7\alpha^9 + a_6\alpha^8 + a_5\alpha^7 + a_4\alpha^6 + a_3\alpha^5 + a_2\alpha^4 + a_1\alpha^3 + a_0\alpha^2.$$

Table 13: Different CRC signatures for α^2 module when *m=13* and *m=14*.

m	CRC	Predicted CRC Signature
13	3	$(a_{12}+a_{11}+a_{10}+a_9+a_7+a_4$ $+a_3+a_2+a_0)\alpha^2+(a_{12}+a_{11}$ $+a_{10}+a_9+a_8+a_6+a_3+a_2$ $+a_1)\alpha+(a_{12}+a_{10}+a_8+a_5$ $+a_4+a_3+a_1)$
	4	$(a_{11}+a_{10}+a_9+a_7+a_5+a_4$ $+a_1)\alpha^3+(a_{10}+a_9+a_8+a_6$ $+a_4+a_3+a_0)\alpha^2+(a_{12}+a_{10}$ $+a_9+a_8+a_7+a_5+a_3+a_2)\alpha$ $+(a_{12}+a_{10}+a_8+a_6+a_5+a_2)$
14	3	$(a_{12}+a_{11}+a_{10}+a_9+a_7+a_4$ $+a_3+a_2+a_0)\alpha^2+(a_{13}+a_{10}$ $+a_9+a_8+a_6+a_3+a_2+a_1)\alpha$ $+(a_{13}+a_{11}+a_{10}+a_8+a_5+a_4$ $+a_3+a_1)$
	4	$(a_{12}+a_{11}+a_{10}+a_9+a_7+a_5$ $+a_4+a_1)\alpha^3+(a_{13}+a_{11}+a_{10}$ $+a_9+a_8+a_6+a_4+a_3+a_0)\alpha^2$ $+(a_{13}+a_{12}+a_{10}+a_9+a_8+a_7$ $+a_5+a_3+a_2)\alpha+(a_{13}+a_{11}$ $+a_{10}+a_8+a_6+a_5+a_2)$

To calculate the predicted CRC-*4* for *m=14* in the α^2 module ($PCRC4_{14_2}$), the generator polynomial is applied as follows:

$$A(\alpha)\cdot\alpha^2 = a_{13}(\alpha^3+\alpha)+a_{13}(\alpha^3+\alpha+1)+a_{13}\alpha^2$$
$$+a_{13}\alpha+a_{12}(\alpha^2+1)+a_{12}(\alpha^3+\alpha^2)+a_{12}\alpha+a_{12}$$
$$+a_{11}(\alpha^3+\alpha^2+1)+a_{10}(\alpha^3+\alpha^2+\alpha+1)+a_9(\alpha^3$$
$$+\alpha^2+\alpha)+a_8(\alpha^2+\alpha+1)+a_7(\alpha^3+\alpha)+a_6(\alpha^2$$
$$+1)+a_5(\alpha^3+\alpha+1)+a_4(\alpha^3+\alpha^2)+a_3(\alpha^2+\alpha)$$
$$+a_2(\alpha+1)+a_1\alpha^3+a_0\alpha,$$

or

$$PCRC4_{14} = (a_{12} + a_{11} + a_{10} + a_9 + a_7$$
$$+a_5 + a_4 + a_1)\alpha^3 + (a_{13} + a_{11} + a_{10} + a_9 + a_8$$
$$+a_6 + a_4 + a_3 + a_0)\alpha^2 + (a_{13} + a_{12} + a_{10} + a_9$$
$$+a_8 + a_7 + a_5 + a_3 + a_2)\alpha + (a_{13} + a_{11} + a_{10}$$
$$+a_8 + a_6 + a_5 + a_2).$$

Lastly, to calculate the actual CRC-*4* for *m=14* in the α^2 module ($ACRC4_{14_2}$), the coefficients are renamed: a_{12} as γ_{13}, ..., $a_{13} + a_0$ as γ_1, and a_{13} as γ_0,

$$A(\alpha) \cdot \alpha^2 = \gamma_{13}\alpha^{13} + \gamma_{12}\alpha^{12} + \gamma_{11}\alpha^{11} + \gamma_{10}\alpha^{10}$$
$$+\gamma_9\alpha^9 + \gamma_8\alpha^8 + \gamma_7\alpha^7 + \gamma_6\alpha^6 + \gamma_5\alpha^5 + \gamma_4\alpha^4$$
$$+\gamma_3\alpha^3 + \gamma_2\alpha^2 + \gamma_1\alpha^1 + \gamma_0,$$

and again, the generator polynomial is applied as follows:

$$A(\alpha) \cdot \alpha^2 = \gamma_{13}(\alpha^3 + \alpha^2 + 1) + \gamma_{12}(\alpha^3 + \alpha^2 + \alpha + 1)$$
$$+\gamma_{11}(\alpha^3 + \alpha^2 + \alpha) + \gamma_{10}(\alpha^2 + \alpha + 1) + \gamma_9(\alpha^3 + \alpha)$$
$$+\gamma_8(\alpha^2 + 1) + \gamma_7(\alpha^3 + \alpha + 1) + \gamma_6(\alpha^3 + \alpha^2) + \gamma_5(\alpha^2$$
$$+\alpha) + \gamma_4(\alpha + 1) + \gamma_3\alpha^3 + \gamma_2\alpha^2 + \gamma_1\alpha^1 + \gamma_0$$

or

$$ACRC4_{14_2} = (\gamma_{12} + \gamma_{11} + \gamma_{10} + \gamma_9 + \gamma_7 + \gamma_5$$
$$+\gamma_4 + \gamma_1)\alpha^3 + (\gamma_{13} + \gamma_{11} + \gamma_{10} + \gamma_9 + \gamma_8 + \gamma_6$$
$$+\gamma_4 + \gamma_3 + \gamma_0)\alpha^2 + (\gamma_{13} + \gamma_{12} + \gamma_{10} + \gamma_9 + \gamma_8$$
$$+\gamma_7 + \gamma_5 + \gamma_3 + \gamma_2)\alpha + (\gamma_{13} + \gamma_{11} + \gamma_{10} + \gamma_8$$
$$+\gamma_6 + \gamma_5 + \gamma_2).$$

In Fig. 4.1, the proposed architecture with CRC-*3* or CRC-*4* signatures is presented. As shown in Fig. 4.1, three or four error indication flags denoted as $e_{\alpha 0}$, $e_{\alpha 1}$, $e_{\alpha 2}$, and $e_{\alpha 3}$ are obtained to indicate if an error has been found using CRC-*3* or CRC-*4*, respectively.

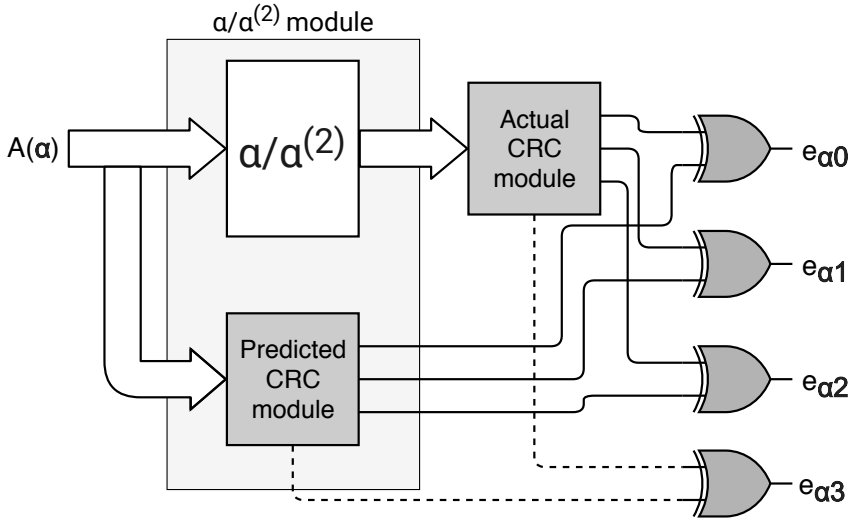


Figure 4.1: The proposed error detection of the α and α^2 modules using CRC signatures (CRC-*3* and CRC-*4*).

This figure represents the α module as well as the α^2 module. The XOR gates are used to compare the outputs of the CRC modules. The output from the actual CRC module is divided into *3* or *4* sub-outputs (CRC-*3* or CRC-*4*, respectively) and they are compared with the sub-outputs of the predicted CRC module.

4.3 Error Coverage and FPGA Implementations

To calculate the error coverage provided by the different error detection schemes presented in this chapter, the total number of operations need to be taken into account. Computing H is a costly process since it performs addition, multiplication, and inversion in $GF(2^m)$. First, to obtain a polynomial $g(\alpha)$ for $\alpha \, \epsilon \, (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$, n finite field multiplications and $n \times t$ XOR operations are needed. Each finite field multiplication uses three different modules. A total of *m-1* α modules, *m-1 sum* modules, and *m pass-thru* modules are needed to perform each finite field multiplication; and a total of *m-1 sum* modules are needed to perform an XOR operation. Next, a total of n inversions are needed. Depending on the value of m, a different number of squaring and multiplications will be needed to perform each inversion. To perform a finite field square operation, *m-1* α^2 modules and *m-1 sum*

modules are needed. After that, the results are multiplied by n distinct elements in $GF(2^m)$, needing an extra $n \times t$ finite field multiplications.

Each of the modules requires one, two, three, or four signatures depending on the choice of normal parity, interleaved parity, CRC-*3*, or CRC-*4*, respectively. For each value of m, the error coverage percentage is calculated as follows:

1. For *m=11*, a total of 2,048 finite field multiplications and 81,920 XOR operations are needed to obtain a polynomial $g(\alpha_i)$; each inverse calculation can be derived by 10 squaring and 9 multiplication operations for $f(\alpha)$, or a total of 20,480 squarings and 18,432 multiplications; and lastly, 81,920 finite field multiplications are needed to multiply the previous outputs by n distinct elements in $GF(2^m)$. Therefore, a total of $2,048 \cdot (10 + 10 + 11) + 81,920 \cdot (10) + 20,480 \cdot (10) \cdot (10) + 18,432 \cdot (9) \cdot (10) + 81,920 \cdot (10 + 10 + 11)$ or close to 7.1×10^6, 1.4×10^7, 2.1×10^7, or 2.8×10^7 normal, interleaved, CRC-*3*, or CRC-*4* signatures are needed, respectively.

2. For *m=12*, a total of $3,408 \cdot (11 + 11 + 12) + 228,336 \cdot (11) + 37,488 \cdot (11) \cdot (11) + 34,080 \cdot (10) \cdot (11) + 228,336 \cdot (11 + 11 + 12)$ or close to 1.8×10^7, 3.7×10^7, 5.6×10^7, or 7.5×10^7 normal, interleaved, CRC-*3*, or CRC-*4* signatures are needed, respectively.

3. For *m=13*, a total of $6,960 \cdot (12 + 12 + 13) + 828,240 \cdot (12) + 83,520 \cdot (12) \cdot (12) + 76,560 \cdot (11) \cdot (12) + 828,240 \cdot (12 + 12 + 13)$ or close to 6×10^7, 1.2×10^8, 1.9×10^8, or 2.5×10^8 normal, interleaved, CRC-*3*, or CRC-*4* signatures are needed, respectively.

4. For *m=14*, a total of $16,384 \cdot (13 + 13 + 14) + 245,760 \cdot (13) + 212,992 \cdot (13) \cdot (13) + 196,608 \cdot (12) \cdot (13) + 245,760 \cdot (13 + 13 + 14)$ or close to 8×10^7, 1.6×10^8, 2.4×10^8, or 3.2×10^8 normal, interleaved, CRC-*3*, or CRC-*4* signatures are needed, respectively.

The lowest percentage (representing the worst case scenario) of error coverage will be obtained by applying normal parity with *m=11*, where the error coverage percentage is $100(1 - (\frac{1}{2})^{7.1 \times 10^6})\%$. In Table 14, the overhead of the error detection architectures in terms of area (occupied slices), delay, and power (at the frequency of 50 MHz) are presented for the Horner block, where finite field multiplications and additions of a polynomial $g(\alpha)$ are

Table 14: Overheads of the proposed error detection schemes for the Horner Unit using *m=14* on Xilinx FPGA family Spartan-7 and also Xilinx FPGA family Artix-7.

Architecture	Area (occupied slices)	Delay (ns)	Power (mW) @50 MHz
Horner (Spartan-7)	191	5.97	0.116
Horner-Parity (Spartan-7)	206 (7.85%)	5.629 (0.57%)	0.117 (0.86%)
Horner-CRC3 (Spartan-7)	235 (23.03%)	5.782 (3.31%)	0.121 (4.31%)
Horner-CRC4 (Spartan-7)	242 (26.70%)	5.725 (2.29%)	0.121 (4.31%)
Horner (Artix-7)	190	5.517	0.110
Horner-Parity (Artix-7)	200 (5.26%)	5.627 (1.99%)	0.111 (0.91%)
Horner-CRC3 (Artix-7)	233 (22.63%)	5.598 (1.47%)	0.116 (5.45%)
Horner-CRC4 (Artix-7)	242 (27.37%)	5.749 (4.21%)	0.116 (5.45%)

Table 15: Overheads of the proposed error detection schemes for the entire Key Generator using the parameters *m=13*, *t=119*, and *n=6,960* on Xilinx Kintex UltraScale+ FPGA.

Architecture	Area (CLBs)	Delay (ns)	Power (mW) @50 MHz
Key Gen. (adopted from [9])	7942	9.404	0.844
Key Gen. with CRC-3	8282 (4.28%)	9.695 (3.09%)	0.849 (0.59%)
Key Gen. with CRC-4	8301 (4.52%)	9.918 (5.47%)	0.851 (0.83%)

applied. To calculate the overheads, the implementations are performed in two different FPGA families and devices. As shown in Table 14, when CRC signatures are applied to the original architecture, with higher error coverage, they end up having higher overhead in terms of area, delay, and power. This is expected since CRC signatures perform more operations ending up having higher error coverage, as seen in the assessments.

Moreover, the proposed error detection schemes are added for the "modified" Key Generator adopted from [59]. The presented implementations are performed using Xilinx Vivado with the parameters *m=13*, *t=119*, and *n=6,960* on Xilinx Kintex Ultrascale+ FPGA. The implementation results for original work and the presented error detection schemes are shown in Table 15 in terms of area, delay/frequency, power, throughput, and efficiency. As seen in this table, acceptable overheads are obtained with efficiency degradation of at most *8.8%*.

Chapter 5: Reliable CRC-Based Error Detection Constructions for Finite Field Multipliers with Applications in Cryptography

5.1 Finite Field Multipliers in Luov's Cryptosystem

[2]Many modern, sensitive applications and systems use finite field operations in their schemes. Finite field multipliers perform multiplication modulo an irreducible polynomial used to define the finite field. For post-quantum cryptography (PQC), the inputs can be very large and the finite field multipliers may require millions of logic gates. Therefore, it is a complex task to implement such architectures resilient to natural and malicious faults; consequently, research has focused on ways to eliminate errors and obtain more reliability with acceptable overhead [38], [54],, [55], [51], [76], [77]. Moreover, there has been previous work on countering fault attacks and providing reliability for PQC. In [70], Sarker *et al.* use error detection schemes of number-theoretic transform to detect both permanent and transient faults. Mozaffari Kermani *et al.* perform fault detection for stateless hash-based PQC signatures in [40]. Additionally, error detection hash trees for stateless hash-based signatures are proposed in [78] to make such schemes more reliable against natural faults and help protecting them against malicious faults. In [56], algorithm-oblivious constructions are proposed through recomputing with swapped ciphertext and additional authenticated blocks, which can be applied to the GCM architectures using different finite field multipliers in $GF(2^{128})$. Several countermeasures based on error detection checksum codes and spatial/temporal redundancies for the NTRU encryption algorithm have been presented in [71].

[2]This chapter was published in the IEEE Transactions on Very Large Scale Integration (VLSI) Systems [75]

In this chapter, finite fields $GF(2^m)$ with $m¿1$ are considered. The polynomials that such multipliers use as inputs have the form

$$p(x) = a_{m-1}x^{m-1} + ... + a_1x + a_0,$$

with degree *m-1* and $a_i \epsilon GF(2)$. To reduce the output, the finite field multiplier uses an irreducible polynomial or field polynomial

$$f(x) = x^t + f_{t-1}x^{t-1} + ... + f_1x + f_0,$$

with degree t and $f_i \epsilon GF(2)$. The multiplication of elements A and B is represented as $A(x) \times B(x) \, mod \, f(x)$.

Luov is a multivariate public key cryptosystem and an adaptation of the Unbalanced Oil and Vinegar (UOV) signature scheme, but there is a restriction on the coefficients of the public key. Instead, the scheme uses two finite fields, one is the binary field of two elements, the other is its extension of degree m. F_2 is the binary field and F_{2^m} is its extension of degree m. The central map $F : \mathrm{F}_{2^m}^n \rightarrow \mathrm{F}_{2^m}^o$ is a quadratic map, where o and v satisfy $n = o + v$, $\alpha_{i,j,k}$, $\beta_{i,k}$ and γ_k are chosen from the base field F_2, and whose components $f_1, \ldots, f_o$ are in the form:

$$fk(x) = \sum_{i=1}^{v} \sum_{j=i}^{n} \alpha_{i,j,k} x_i x_j + \sum_{i=1}^{n} \beta_{i,k} x_i + \gamma_k.$$

The aim of this chapter is to provide countermeasures against natural faults and fault injections for the finite field multipliers used in cryptosystems such as the Luov algorithm as a case study, noting that the proposed error detection schemes can be adapted to other applications and cryptographic algorithms whose building blocks need finite field multiplications. Readers interested in more details about the Luov's cryptographic algorithm are encouraged to refer to [79].

5.2 Proposed Fault Detection Schemes

As seen in previous chapters, the multiplication of any two elements A and B of $GF(2^m)$, following the approach in [44], can be presented as

$$A \cdot B \bmod f(x) = \sum_{i=0}^{m-1} b_i \cdot ((A\alpha^i) \bmod f(x)) = \sum_{i=0}^{m-1} b_i \cdot X^{(i)},$$

where the set of α^i's is the polynomial basis of element A, the set of b_i's is the B coefficients, *f(x)* is the field polynomial, $X^{(i)} = \alpha \cdot X^{(i-1)} \bmod f(x)$, and $X^{(0)} = A$. Fault injection can occur in any of these modules and formulations for parity signatures in $GF(2^m)$ are derived in [44].

In this work, the main goal is the derivation of error detection schemes that provide a broader and higher error coverage than parity signatures (the major drawback of parity signatures is that their error coverage is approximately 50%, i.e., if the number of faults is even, the approach would not be able to detect the faults) and explore the application of such schemes to the Luov algorithm. Thus, CRC signatures [80] are derived and applied to the finite field multipliers used in Luov algorithm. This would be a step forward towards detecting natural and malicious intelligent faults, especially and as discussed in this chapter, considering both primitive and standardized CRCs with different fault multiplicity coverage.

The entire finite field multiplier with the proposed error detection schemes is shown in Fig. 5.1, where ACRC and PCRC stand for actual CRC signatures and predicted CRC signatures, respectively. In Fig. 5.1, only one error flag (EF) is shown for clarity; however, for CRC-5, which is the case study proposed in this work, 5 error flags are computed on each module. In Fig. 5.2, the α module is shown more in depth to clarify how the proposed CRC signatures work in each finite field multiplier.

For the *sum* and *pass-thru* modules, it follows the approach as for parity signatures described in [44]. For the *sum* module in CRC-*1*, $\hat{p}_x$ is equal to the *sum* of the parity bits of the input elements A and B in $GF(2^m)$, $\hat{p}_X = p_A + p_B$. Furthermore, for the *pass-thru*

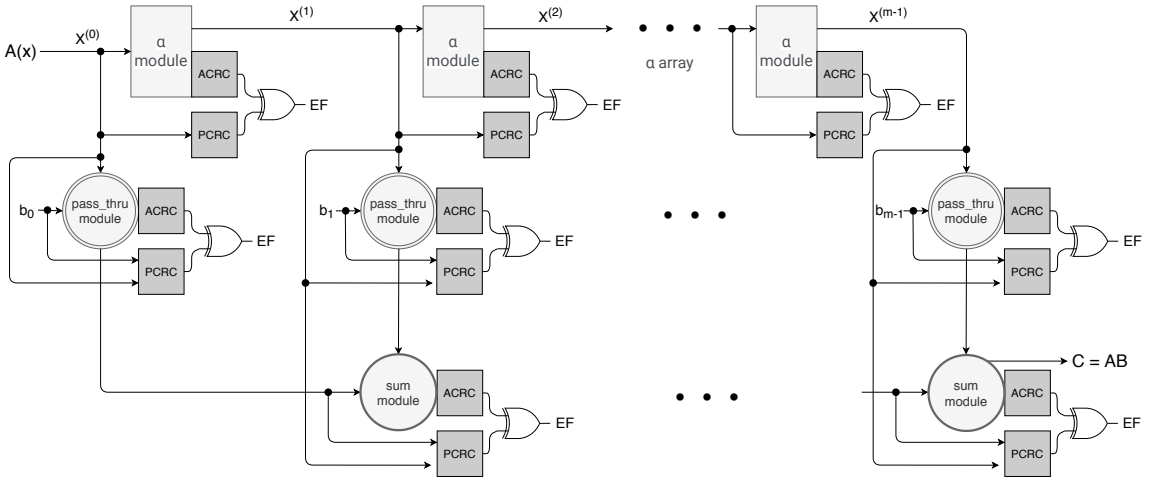


Figure 5.1: Finite field multiplier with the proposed error detection schemes based on CRC.

module in CRC-*1*, $\hat{p}_X = b \cdot p_A$, where b is an element in *GF(2)*. For any other CRC-n scheme, instead of summing all the bits, it checks n bits at a time in the *sum* and *pass-thru* modules. For the α module:

$$A(x) \cdot x = a_{m-1} \cdot x^m + a_{m-2} \cdot x^{m-1} + ... + a_0 \cdot x,$$

for which a set of derivations is needed to implement CRC-n into it. In these chapter, four different generator polynomials are studied. The generator polynomial $g_0(x) = x^5 + x^3 + 1$ is one of the standards used for radio-frequency identification [62]. The other three generator polynomials $g_1(x) = x^5 + x^2 + 1$, $g_2(x) = x^5 + x^4 + x^2 + x + 1$, and $g_3(x) = x^5 + x^4 + x^3 + x^2 + 1$ are primitive polynomials. The benefit of using a primitive polynomial as the generator that the resulting code has full total block length, which means that all 1-bit errors within that block length have separate remainders. Moreover, since the remainder is a linear function of the block, all 2-bit errors within that block length can be identified.

For the α module of the Luov's finite field multipliers, $g_0(x) = x^5 + x^3 + 1$ is used as the standardized generator polynomial for CRC-*5*. To find its CRC signatures, this fixed

polynomial is used as follows:

$$x^5 \equiv x^3 + 1 \bmod g_0(x)$$
$$x^6 \equiv x^4 + x \bmod g_0(x)$$
$$x^7 \equiv x^5 + x^2 \equiv x^3 + x^2 + 1 \bmod g_0(x)$$
$$\vdots$$
$$x^{15} \equiv x^2 + 1 \bmod g_0(x).$$

According to the previous equations, $A(x) \cdot x = a_{15} \cdot x^{16} + a_{14} \cdot x^{15} + ... + a_1 \cdot x^2 + a_0 \cdot x$. Then, applying the irreducible polynomial $f(x) = x^{16} + x^{12} + x^3 + x + 1$, one obtains

$$\begin{aligned} A(x) \cdot x \equiv\ & a_{15}x^{12} + a_{15}x^3 + a_{15}x + a_{15} + a_{14}x^{15} \\ & + a_{13}x^{14} + a_{12}x^{13} + a_{11}x^{12} + a_{10}x^{11} + a_9x^{10} \\ & + a_8x^9 + a_7x^8 + a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 \\ & + a_2x^3 + a_1x^2 + a_1x \bmod f(x). \end{aligned}$$

To calculate the predicted CRC-5 for $GF(2^{16})$ in the α module ($PCRC5_{16}$), the generator polynomial is applied as

$$\begin{aligned} A(x) \cdot x \equiv\ & a_{15}(x^4 + x^3 + x^2 + x) + a_{15}x^3 + a_{15}x + a_{15} \\ & + a_{14}(x^2 + x) + a_{13}(x + 1) + a_{12}(x^4 + x^2 + 1) + a_{11}(x^4 \\ & + x^3 + x^2 + x) + a_{10}(x^3 + x^2 + x + 1) + a_9(x^4 + x \\ & + 1) + a_8(x^4 + x^3 + x^2 + 1) + a_7(x^4 + x^3 + x) + a_6(x^3 \\ & + x^2 + 1) + a_5(x^4 + x) + a_4(x^3 + 1) + a_3x^4 + a_2x^3 \\ & + a_1x^2 + a_0x \bmod g_0(x) \end{aligned},$$

or

$$PCRC5_{16} = (a_{15} + a_{12} + a_{11} + a_9 + a_8 + a_7$$
$$+a_5 + a_3)x^4 + (a_{12} + a_{11} + a_9 + a_8 + a_7 + a_6$$
$$+a_4 + a_2)x^3 + (a_{15} + a_{14} + a_{12} + a_{11} + a_{10} + a_8$$
$$+a_6 + a_1)x^2 + (a_{14} + a_{13} + a_{11} + a_{10} + a_9 + a_7$$
$$+a_5 + a_0)x + (a_{15} + a_{13} + a_{12} + a_{10} + a_9 + a_8$$
$$+a_6 + a_4).$$

To calculate the actual CRC-*5* for $GF(2^{16})$ in the α module ($ACRC5_{16}$), the coefficients are renamed: a_{14} as γ_{15},..., a_0 as γ_1,

$$A(x) \cdot x \equiv \gamma_{15}x^{15} + \gamma_{14}x^{14} + \gamma_{13}x^{13} + \gamma_{12}x^{12} +$$
$$\gamma_{11}x^{11} + \gamma_{10}x^{10} + \gamma_9 x^9 + \gamma_8 x^8 + \gamma_7 x^7 +$$
$$\gamma_6 x^6 + \gamma_5 x^5 + \gamma_4 x^4 + \gamma_3 x^3 + \gamma_2 x^2 + \gamma_1 x^1 +$$
$$\gamma_0 \bmod g_0(x),$$

and the generator polynomial is applied as follows

$$A(x) \cdot x \equiv \gamma_{15}(x^2 + x) + \gamma_{14}(x + 1) + \gamma_{13}(x^4 + x^2 + 1)$$
$$+\gamma_{12}(x^4 + x^3 + x^2 + x) + \gamma_{11}(x^3 + x^2 + x + 1) + \gamma_{10}(x^4$$
$$+x + 1) + \gamma_9(x^4 + x^3 + x^2 + 1) + \gamma_8(x^4 + x^3 + x)$$
$$+\gamma_7(x^3 + x^2 + 1) + \gamma_6(x^4 + x) + \gamma_5(x^3 + 1) + \gamma_4 x^4$$
$$+\gamma_3 x^3 + \gamma_2 x^2 + \gamma_1 x^1 + \gamma_0 \bmod g_0(x)$$

or

$$ACRC5_{16} = (\gamma_{13} + \gamma_{12} + \gamma_{10} + \gamma_9 + \gamma_8 + \gamma_6$$
$$+\gamma_4)x^4 + (\gamma_{13} + \gamma_{12} + \gamma_{11} + \gamma_9 + \gamma_8 + \gamma_7 + \gamma_5$$
$$+\gamma_3)x^3 + (\gamma_{15} + \gamma_{13} + \gamma_{12} + \gamma_{11} + \gamma_9 + \gamma_7 + \gamma_2)$$
$$\cdot x^2 + (\gamma_{15} + \gamma_{14} + \gamma_{12} + \gamma_{11} + \gamma_{10} + \gamma_8 + \gamma_6 + \gamma_1)$$
$$\cdot x + (\gamma_{14} + \gamma_{13} + \gamma_{11} + \gamma_{10} + \gamma_9 + \gamma_7 + \gamma_5 + \gamma_0).$$

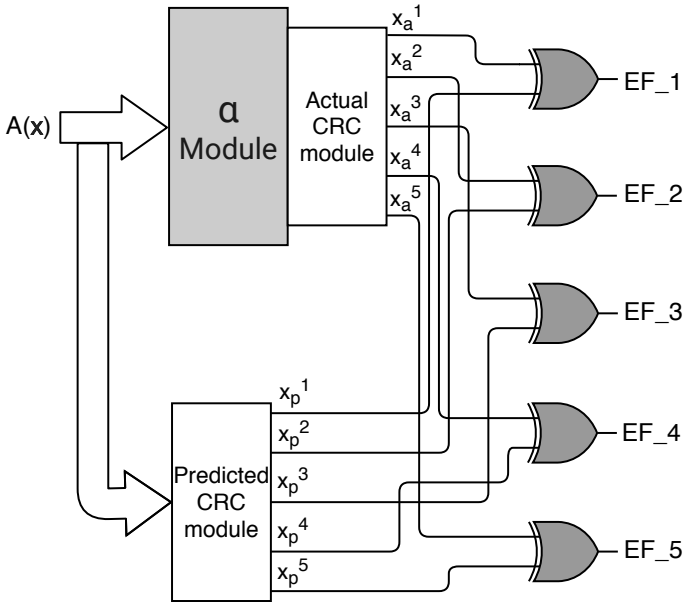


Figure 5.2: The proposed error detection constructions for α module.

The predicted output and the actual output are divided into five parity groups. These parity groups are XORed with each other to determine if there has been any fault, e.g., flip of bits, during the α module operation. In total, each α module outputs five error flags. Fig. 5.2 shows the implementation of the α module with the proposed error detection schemes. A(x) is the input with the form $p(x) = a_{m-1}x^{m-1} + ... + a_1x + a_0$, which goes to two different modules that run in parallel. The output from the α module is divided into 5 groups in the actual CRC module, which are denoted as $x_a^1 - x_a^5$ in Fig. 5.2. Meanwhile, *A(x)* is also being divided into *5* groups in the predicted CRC module, which are denoted as $x_p^1 - x_p^5$. Once the two CRC modules are done, each group is XORed with its respective one to produce *5* error flags, which are represented as $EF_1 - EF_5$. As an example, to obtain EF_1, x_p^1 (or $a_{15} + a_{13} + a_{12} + a_{10} + a_9 + a_8 + a_6 + a_4$ for $g_0(x)$) is XORed with x_a^1 (or $\gamma_{14} + \gamma_{13} + \gamma_{11} + \gamma_{10} + \gamma_9 + \gamma_7 + \gamma_5 + \gamma_0$ for $g_0(x)$). For this case study, the outputs are divided into *5* groups since CRC-*5* is used; however, if any other CRC-n is used, there will be n error flags and the actual and predicted outputs will be divided into n groups. The choice

of the utilized CRC can be tailored based on the reliability requirements and the overhead to be tolerated.

5.3 Error Coverage and FPGA Implementations

Luov polynomial generation is implemented to show that the proposed error detection schemes provide high error coverage with acceptable overhead. Such implementation produces a polynomial $p(x) = a_{m-1}x^{m-1} + ... + a_1x + a_0$, which requires *m-1* finite field multiplications and *m-1* XOR operations. As pointed out before, each finite field multiplication uses three different modules called α, sum, and *pass-thru* modules. A total of *m-1* α modules, *m-1 sum* modules, and *m pass-thru* modules are needed to perform each finite field multiplication. Moreover, a total of *m-1 sum* modules are needed to perform an XOR operation. For each architecture, the error coverage is calculated as $100 \cdot (1 - (\frac{1}{2})^{sign})\%$, where *sign* denotes the number of signatures.

Luov uses the finite field $GF(2^{16})$, or *m=16*. Implementing its polynomials in the form of $p(x) = a_{15}x^{15} + ... + a_1x + a_0$ requires 14 finite field multiplications and 15 XOR operations. Since each finite field multiplication uses *m-1* α modules, *m-1 sum* modules, and *m pass-thru* modules, 14×15 α modules, 14×15 *sum* modules, and 14×16 *pass-thru* modules are needed. Moreover, a total of $14_{multiplications} \cdot (15_{\alpha} + 15_{sum} + 16_{pass-thru}) + 15_{xor}$ or *659* signatures are implemented. The error coverage percentage for the generation of Luov's polynomial using the finite field $GF(2^{16})$ is $100 \cdot (1 - (\frac{1}{2})^{659})\%$. In Table 16, the overhead of the proposed error detection architectures is presented in terms of area (CLBs), delay, and power consumption (at the frequency of 50 MHz) for the generation of polynomial p(x) where $p(x) = a_{m-1}x^{m-1} + ... + a_1x + a_0$.

Xilinx FPGA family Kintex Ultrascale+ for device xcku5p-ffvd900-1-i is utilized, using Verilog as the hardware design entry and Vivado as the tool for the implementations. As shown in Table 16, when CRC signatures are applied to the original architecture, with higher error coverage, they end up having higher overhead in terms of area, delay, and power.

Table 16: Overheads of the proposed error detection schemes for the finite field multipliers used in the Luov algorithm during the polynomial generation on Xilinx FPGA family Kintex Ultrascale+ for device xcku5p-ffvd900-1-i.

Architecture	Area (CLBs)	Delay (ns)	Power (mW) @50 MHz
Luov Multiplier	120	4.044	0.465
Luov Multiplier-CRC5-$g_0(x)$	139 (15.83%)	4.194 (3.71%)	0.466 ($\simeq$ 0%)
Luov Multiplier-CRC5-$g_1(x)$	134 (11.67%)	4.242 (4.90%)	0.466 ($\simeq$ 0%)
Luov Multiplier-CRC5-$g_2(x)$	131 (9.17%)	4.252 (5.14%)	0.466 ($\simeq$ 0%)
Luov Multiplier-CRC5-$g_3(x)$	142 (18.33%)	4.499 (11.25%)	0.466 ($\simeq$ 0%)

Configurable logic blocks (CLBs), which are the main resources for implementing general-purpose combinational and sequential circuits, are read in the Vivado's place utilization report to obtain the area. To determine the delay, the Timing Constraints Wizard function in Vivado is used, setting a primary clock period constraint of 20 ns, which equals to a frequency of 50 MHz. The total on-chip power is also reported, which is the power consumed internally within the FPGA and it is obtained by adding device static power and design power. As seen in this table, acceptable overheads are obtained with efficiency degradations of at most 19%. The error detection architecture that uses the primitive generator polynomial $g_2(x)$ has the least amount of area overhead with 9.17%; however, the error detection implementation using $g_0(x)$, or the standardized generator polynomial for CRC-*5*, performs the fastest, obtaining the least amount of delay overhead with 3.71%. These degradations are acceptable for providing error detection to the original architectures which lack such capability to thwart natural or malicious faults.

Chapter 6: CRC-Oriented Error Detection Schemes for Fast Inversions in $GF(2^m)$ Normal Basis

6.1 Finite Field Inversions

In the previous chapters, finite field arithmetic with polynomial basis have been studied. This chapter explores finite field operations over $GF(2^m)$ with normal basis. Normal basis has been used in many works such as [81], [82], [83], [84], and [85]. The importance of finite field inversions with normal basis has attracted researchers to investigate many different approaches in an effort to reduce the size, delay, and power of such designs [86], [87], [88]. In this chapter, new error detection approaches for the well-known ITA algorithm, based in the FLT algorithm, are proposed.

Binary fields with normal basis are of special interest for hardware constructions since the squaring is done by simply performing a cyclic right shift. However, some works have been focusing on either proposing faster finite field inversion constructions or in modifying the finite field multipliers used in such architectures [74], [89], [90], [91]. In [89], the authors present new architectures for digit-level single, hybrid-double, and hybrid-triple multiplication over $GF(2^m)$ elements based on the Gaussian normal basis (GNB) representation. In a more recent work, the authors of [90] propose two new inversion architectures, an improved architecture for classic inversion scheme using a single multiplier and a novel fully-serial-in square-multiply processor. In [74] and [91], authors perform fast inversion over $GF(2^m)$ by using single and hybrid-double multipliers. Lastly, other works such as [92], [93], and [94] explore different modifications of the ITA. To the best of the author's knowledge, this is the first work that uses cyclic redundancy check (CRC) signatures to obtain secure finite field inversions over $GF(2^m)$ with normal basis using the well-known ITA. Some previous works

perform concurrent error detection in classical and post-quantum cryptography, e.g., [55], [95], [76] [96], and [97], and some in bit-serial and digit-normal basis multiplication using parity prediction such as those in [98] and [99], respectively. However, as it has been mentioned previously, the main concern with parity prediction schemes is that the error coverage percentage is at most 50%, since if the natural or injected faults are even, the predicted parity bit will be the same as the original parity bit. The contributions in this chapter can be summarized as follows:

1. Error detection schemes for finite field inversion in $GF(2^m)$ with normal basis are proposed, used in many different traditional and post-quantum cryptographic algorithms. Even though these error detection schemes are based on CRC-*3*, larger CRCs can be used by applying the similar derivations as presented in this chapter.

2. Formulations for error detection in finite field inversion employing FLT and ITA are derived and verified by performing software implementations. m=*7* is used for the sake of brevity as a case example. Nonetheless, the error detection schemes presented in this chapter can be applied to any other NIST field [100]. This will allow generalizing the scheme to be used in classical and post-quantum cryptographic algorithms and implementations.

3. The proposed fault-detection architectures are embedded into the original finite field inversion to benchmark the overhead/degradation tolerance using Xilinx FPGA family Virtex-7 for device xc7vx1140tflg1930-i. The results of the presented work show acceptable overhead and very high error coverage suitable for deeply-embedded constructions.

In this chapter, finite fields $GF(2^m)$ are constructed by using a normal basis $N = \{\alpha, \alpha^2, \alpha^{2^2}, ..., \alpha^{2^{m-1}}\}$, where each α is a normal element of $GF(2^m)$. Any element A in $GF(2^m)$ with normal basis is represented as:

$$A = \sum_{i=0}^{m-1} a_i \alpha^{2^i},$$

where $a_i \, \epsilon \, GF(2)$.

Gaussian normal basis (GNB) is a specific class of normal basis where $m > 1$ and it is not divisible by 8. A more detailed definition presented in [60] states that m and t are positive integers such that *p=mt+1* is a prime number. A Gauss period of type *(m,t)* over *GF(2)* is represented as:

$$\alpha = \sum_{i=0}^{t-1} \beta^{\tau^i},$$

where β is the primitive *(mt+1)*th root of unity in *GF(mt+1)*. To calculate τ, which is the primitive tth root of unity, the following property is applied: $\tau^t = 1 \bmod p$. For example, the GNB with type-*4* over $GF(2^7)$ has $\tau = 12$ since $12^4 = 1 \bmod 29$. Moreover, α is calculated as follows:

$$\alpha = \sum_{i=0}^{t-1} \beta^{12^i} = \beta + \beta^{12} + \beta^{17} + \beta^{28}.$$

To perform addition of elements A and B over $GF(2^m)$, the coefficients of each element are added such as $\sum_{i=0}^{m-1}(a_i + b_i)\alpha^{2^i}$ using XOR gates. As stated earlier, one of the main advantages of normal basis is that squaring has no cost in hardware, it is performed by just applying cyclic right shifts. Squaring an element A over $GF(2^m)$ in normal basis can be expressed as:

$$A^{2^i} = \sum_{j=0}^{m-1} a_{<j-i>}\alpha^{2^i}.$$

Lastly, to perform multiplication of elements A and B and obtain C, the scheme from [99] stating that

$$C = (((a_{m-1}\alpha B^{2^{-(m-1)}})^2 + a_{m-2}\alpha B^{2^{-(m-2)}})^2 + \\ +...)^2 + a_0\alpha B,$$

is used in this work.

6.2 Proposed Fault Detection Schemes

The inverse of an element $A \, \epsilon \, GF(2^m)$ is expressed as $A^{-1} \, \epsilon \, GF(2^m)$ as $A \times A^{-1} = 1$. As previously mentioned, FLT has been used by many hardware implementations since it obtains less logic overhead by reusing finite field multipliers and squarers. Nevertheless, finite

Table 17: Steps to perform the inverse of $A \epsilon GF(2^7)$ using addition chains.

Step	$\gamma_{V_i}(x)$	$\gamma_{V_j+Y_k}(x)$	Exponentiation
1	$\gamma_1(x)$	-	A
2	$\gamma_2(x)$	$\gamma_{1+1}(x)$	$(\gamma_1)^{2^1}\gamma_1 = A^{2^2-1}$
3	$\gamma_3(x)$	$\gamma_{2+1}(x)$	$(\gamma_2)^{2^1}\gamma_1 = A^{2^3-1}$
4	$\gamma_6(x)$	$\gamma_{3+3}(x)$	$(\gamma_3)^{2^3}\gamma_3 = A^{2^6-1}$

field inversion using FLT requires a total of *m-1* finite field squarings and *m-2* finite field multiplications. This might not be practical for some resource-constrained deeply-embedded systems, where low complexity and high performance is a requirement.

ITA has gained attention since it performs not as many finite fields multiplications. ITA is capable of reducing the number of multiplications to $log_2(m-1)+H_2(m-1)-1$, where $H_2(m-1)$ is the Hamming weight, by renaming $2^0+2^1+2^2+...+2^{m-2}$ into $1+2^n+2^{2n}+...+2^{(k-2)n}$ and decomposing it as follows: If $k-1 \equiv 0 \bmod 2$, then $(1+2^n)\times(1\times 2^{2n}+2^{4n}+...+2^{(k-3)n})$ and if $k-1 \equiv 1 \bmod 2$, then $1+2^n \times (1+2^n) \times (1 \times 2^{2n}+2^{4n}+...+2^{(k-4)n})$.

Following ITA, the field $GF(2^7)$ has the following decomposition: $1+2+2^2+...+2^5 = (1+2)\times(1+2^2\times(1+2^2)\times(1+2^4))$. Other approaches perform ITA with the effective use of addition chains. As in [101], the inverse of element A can be expressed as $A^{-1} = [\gamma_{m-1}(A)]^2$, where $\gamma_k(A) = A^{2^k-1}$ and $k \epsilon N$. To find the addition chain $U = \{u_1, u_2, ..., u_t\}$, $u_0 = 0$ and $u_t = m-1$ are assigned, and if u_i is even, $u_{t-1} = u_t/2$, and if u_i is odd, $u_{i-1} = u_i - 1$. For *m=7*, the addition chain is $U = \{1, 2, 3, 6\}$. Once the addition chain is calculated, the steps to perform finite field inversion over $GF(2^7)$ are shown in Table 17, where V_i's correspond to the integers of the addition chain U, $V_j = V_{i-1}$, and $Y_k = V_i - V_j$. Unfortunately, addition chains do not reduce the number of multiplications for the case of $GF(2^7)$. Along this work, the original ITA is used; however, the same error detection schemes are applicable for ITA using addition chains.

By employing ITA, only squarings that have no cost and multiplications of elements over $GF(2^m)$ with normal basis are needed. To perform multiplication, (1) is computed as shown in Algorithm 1.

Algorithm 1 Multiplication of elements in $GF(2^m)$ with normal basis

1: **Input:** $A \epsilon GF(2^m)$, $B \epsilon GF(2^m)$
2: $C_0 \leftarrow 0$
3: **for** $i \leftarrow 1$ to m **do**
4: $\quad C_i = C_{i-1}^2 + a_{m-i}\alpha B^{2^{-(m-i)}}$
5: **end for**
6: $C = C_m$
7: **return** C

The most complex step in Algorithm 1 is done by the αB multiplier, which multiplies α by the field element B. To perform αB, $\alpha = \gamma + \gamma^{2^m} + ... + \gamma^{2^{m(t-1)}}$ is used to produce the normal basis and B is expressed as:

$$B = b_{F(0)} + b_{F(1)}\gamma + ... + b_{F(p-1)}\gamma^{p-1},$$

where $F(2^i 2^{mj} \bmod p) = i$, $0 \leq i \leq m-1$, $0 \leq j \leq t-1$. For the case of $GF(2^7)$, the F values of type-4 GNB in $GF(2^7)$ are shown in Table 18.

Moreover, the GNB form of αB is computed as

$$\alpha B = B^{(1)} + B^{(2^m \bmod p)} + ... + B^{(2^{m(t-1)} \bmod p)},$$

where $B^{(i)} = \sum_{i=0}^{p-1} b_{F(j-i)}\gamma^j$. The normal basis form is then expressed as

$$\alpha B = \sum_{i=0}^{m-1} \bar{b_i}\alpha^{2^i},$$

where $\bar{b_i} = \sum_{i=0}^{t-1} b_{F(2^i - 2^{mj})} + b_{F(0)}$. If t is even, $b_{F(0)}$ is omitted.

In this chapter, error detection schemes based on CRC signatures are proposed. CRC-X divides the output of an specific block/module into X groups, called actual CRC-X signatures

Table 18: F values of type-*4* GNB in $GF(2^7)$.

n	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	5	2	1	6	5	3	3	2
n	11	12	13	14	15	16	17	18	19	20
F(n)	4	0	4	6	6	4	0	4	2	3
n	21	22	23	24	25	26	27	28	-	-
F(n)	3	5	6	1	2	5	1	0	-	-

or $ACRC_x$, and compares them with predicted CRC-X signatures or $PCRC_x$, which are pre-calculated by a set of derivations. $ACRC_x$ and $PCRC_x$ are compared by using X two-input XOR gates. If any of the outputs is a '*1*', there is a faulty bit in the block/module. For the case study of $GF(2^7)$, CRC-*3* is used to provide a low overhead needed for deeply-embedded systems such as implantable or wearable medical devices; however, for not so resource-constrained systems, larger CRC signatures can be applied following the formulations derived next.

For $GF(2^7)$, $B = (b_0, b_1, ..., b_7)$ is the normal basis element and $\alpha = \gamma + \gamma^{12} + \gamma^{17} + \gamma^{28}$ is used to produce the normal basis. The redundant basis of B is expressed using the results form Table 18 and applying them as

$$B = b_0\gamma + b_1\gamma^2 + b_5\gamma^3 + b_2\gamma^4 + b_1\gamma^5 + b_6\gamma^6 +$$
$$b_5\gamma^7 + b_3\gamma^8 + b_3\gamma^9 + b_2\gamma^{10} + b_4\gamma^{11} + b_0\gamma^{12} +$$
$$b_4\gamma^{13} + b_6\gamma^{14} + b_6\gamma^{15} + b_4\gamma^{16} + b_0\gamma^{17} + b_4\gamma^{18} +$$
$$b_2\gamma^{19} + b_3\gamma^{20} + b_3\gamma^{21} + b_5\gamma^{22} + b_6\gamma^{23} + b_1\gamma^{24} +$$
$$b_2\gamma^{25} + b_5\gamma^{26} + b_1\gamma^{27} + b_0\gamma^{28}.$$

To calculate αB, the previous derivations are used, obtaining $\alpha B = (\gamma + \gamma^{12} + \gamma^{17} + \gamma^{28})B$, or $\alpha B = B^{(1)} + B^{(12)} + B^{(17)} + B^{(28)}$, where

$$B^{(1)} = b_0 + b_0\gamma^2 + b_1\gamma^3 + b_5\gamma^4 + b_2\gamma^5 + b_1\gamma^6 +$$
$$b_6\gamma^7 + b_5\gamma^8 + b_3\gamma^9 + b_3\gamma^{10} + b_2\gamma^{11} + b_4\gamma^{12} +$$
$$b_0\gamma^{13} + b_4\gamma^{14} + b_6\gamma^{15} + b_6\gamma^{16} + b_4\gamma^{17} + b_0\gamma^{18} +$$
$$b_4\gamma^{19} + b_2\gamma^{20} + b_3\gamma^{21} + b_3\gamma^{22} + b_5\gamma^{23} + b_6\gamma^{24} +$$
$$b_1\gamma^{25} + b_2\gamma^{26} + b_5\gamma^{27} + b_1\gamma^{28},$$

$$B^{(12)} = b_0 + b_4\gamma + b_2\gamma^2 + b_3\gamma^3 + b_3\gamma^4 + b_5\gamma^5 +$$
$$b_6\gamma^6 + b_1\gamma^7 + b_2\gamma^8 + b_5\gamma^9 + b_1\gamma^{10} + b_0\gamma^{11} +$$
$$b_0\gamma^{13} + b_1\gamma^{14} + b_5\gamma^{15} + b_2\gamma^{16} + b_1\gamma^{17} + b_6\gamma^{18} +$$
$$b_5\gamma^{19} + b_3\gamma^{20} + b_3\gamma^{21} + b_2\gamma^{22} + b_4\gamma^{23} + b_0\gamma^{24} +$$
$$b_4\gamma^{25} + b_6\gamma^{26} + b_6\gamma^{27} + b_4\gamma^{28},$$

$$B^{(17)} = b_0 + b_4\gamma + b_6\gamma^2 + b_6\gamma^3 + b_4\gamma^4 + b_0\gamma^5 +$$
$$b_4\gamma^6 + b_2\gamma^7 + b_3\gamma^8 + b_3\gamma^9 + b_5\gamma^{10} + b_6\gamma^{11} +$$
$$b_1\gamma^{12} + b_2\gamma^{13} + b_5\gamma^{14} + b_1\gamma^{15} + b_0\gamma^{16} +$$
$$b_0\gamma^{18} + b_1\gamma^{19} + b_5\gamma^{20} + b_2\gamma^{21} + b_1\gamma^{22} + b_6\gamma^{23} +$$
$$b_5\gamma^{24} + b_3\gamma^{25} + b_3\gamma^{26} + b_2\gamma^{27} + b_4\gamma^{28},$$

and

$$B^{(28)} = b_0 + b_1\gamma^1 + b_5\gamma^2 + b_2\gamma^3 + b_1\gamma^4 + b_6\gamma^5 +$$
$$b_5\gamma^6 + b_3\gamma^7 + b_3\gamma^8 + b_2\gamma^9 + b_4\gamma^{10} + b_0\gamma^{11} +$$
$$b_4\gamma^{12} + b_6\gamma^{13} + b_6\gamma^{14} + b_4\gamma^{15} + b_0\gamma^{16} + b_4\gamma^{17} +$$
$$b_2\gamma^{18} + b_3\gamma^{19} + b_3\gamma^{20} + b_5\gamma^{21} + b_6\gamma^{22} + b_1\gamma^{23} +$$
$$b_2\gamma^{24} + b_5\gamma^{25} + b_1\gamma^{26} + b_0\gamma^{27}.$$

Table 19: $\alpha B^{2^{-i}}$ predicted CRC-*3* signatures.

Step	Predicted CRC-3 signatures
$\alpha B^{2^{-6}}$	$(b_3+b_6)+(b_1+b_2+b_3+b_5+$ $b_6)x+(b_1+b_3+b_4)x^2$
$\alpha B^{2^{-5}}$	$(b_1+b_2+b_4+b_5)+(b_1+b_3+$ $b_5+b_6)x+x^2$
$\alpha B^{2^{-4}}$	$(b_0+b_1+b_2+b_5+b_6)+(b_1+b_2+$ $b_3+b_5+b_6)x+(b_1+b_4+b_6)x^2$
$\alpha B^{2^{-3}}$	$(b_1+b_3)+(b_0+b_1+b_3+b_6)x+$ $(b_3+b_5)x^2$
$\alpha B^{2^{-2}}$	$(b_1+b_6)+(b_1+b_2+b_4+b_6)x+$ $(b_0+b_1+b_2+b_3+b_4+b_5)x^2$
$\alpha B^{2^{-1}}$	$(b_1+b_2+b_5)+(b_0+b_6)x+$ $(b_1+b_4+b_6)x^2$

Using these derivations, it can be concluded that

$$\alpha B = (b_4+b_4+b_1)\alpha+(b_0+b_2+b_6+b_5)\alpha^2+$$
$$(b_1+b_3+b_6+b_2)\alpha^{2^2}+(b_5+b_3+b_4+b_1)\alpha^{2^3}+$$
$$(b_1+b_6+b_4+b_5)\alpha^{2^4}+(b_5+b_2+b_3+b_3)\alpha^{2^5}+$$
$$(b_6+b_2+b_0+b_0)\alpha^{2^6}.$$

It is noted that the previous equations are derived using i=*7* in Algorithm 1, which makes $\alpha B^{2^{-(7-7)}}$ or αB. To derive the rest of formulations for other i's, the formulation

$$B^{2^{-i}} = \sum_{j=0}^{m-1} b_{<j-i>}\alpha^{2^i}$$

is used. The formulations and the error detection schemes for the other i's in Algorithm 1 are not derived in this chapter for the sake of brevity, but they are presented in Table 19.

To produce CRC-*3* error detection schemes in the calculation of αB, a generator polynomial $p(x) = x^3 + x + 1$ for CRC-*3* is chosen. A set of equations are then derived

$$x^4 \equiv x^2 + x \bmod p(x),$$
$$x^5 \equiv x^3 + x^2 \equiv x^2 + x + 1 \bmod p(x),$$
$$x^6 \equiv x^3 + x^2 + x \equiv x^2 + 1 \bmod p(x),$$

and the predicted CRC-*3* signatures are obtained

$$PCRC_3 = (b_4 + b_4 + b_1) + (b_0 + b_2 + b_6 + b_5)x +$$
$$(b_1 + b_3 + b_6 + b_2)x^2 + (b_5 + b_3 + b_4 + b_1)(x +$$
$$1) + (b_1 + b_6 + b_4 + b_5)(x^2 + x) + (b_5 + b_2 + b_3 +$$
$$b_3)(x^2 + x + 1) + (b_6 + b_2 + b_0 + b_0)(x^2 + 1)$$

or

$$PCRC_3 = (b_3 + b_4 + b_6) + (b_0 + b_3)x +$$
$$(b_3 + b_4 + b_6 + b_2)x^2.$$

To calculate the actual CRC-*3* signatures, the coefficients are renamed: b_0 as $\delta_0, ..., b_6$ as δ_6 to obtain

$$ACRC_3 = (\delta_0) + (\delta_1)x + (\delta_2)x^2 + (\delta_3)(x + 1) +$$
$$(\delta_4)(x^2 + x) + (\delta_5)(x^2 + x + 1) + (\delta_6)(x^2 + 1),$$

or

$$ACRC_3 = (\delta_0 + \delta_3 + \delta_5 + \delta_6) + (\delta_1 + \delta_3 +$$
$$\delta_4 + \delta_5)x + (\delta_2 + \delta_4 + \delta_5 + \delta_6)x^2.$$

The actual CRC-*3* signatures are the same for all i's. In Fig. 6.1, the error detection schemes proposed in this chapter for the multiplication of αB are shown. Since CRC-*3* is used, three different error flags or E_i are obtained by XORing the predicted CRC-*3* signatures with the actual CRC-*3* signatures. E_1 is obtained by XORing the predicted CRC_3 signature $(b_3 + b_4 + b_6)$ with the actual CRC_3 signature $(\delta_0 + \delta_3 + \delta_5 + \delta_6)$ that is obtained after α is multiplied with input B. In the same way, $(b_0 + b_3)$ is compared with $(\delta_1 + \delta_3 + \delta_4 + \delta_5)$ to obtain E_2, and $(b_3 + b_4 + b_6 + b_2)$ is compared with $(\delta_2 + \delta_4 + \delta_5 + \delta_6)$ to obtain E_3.

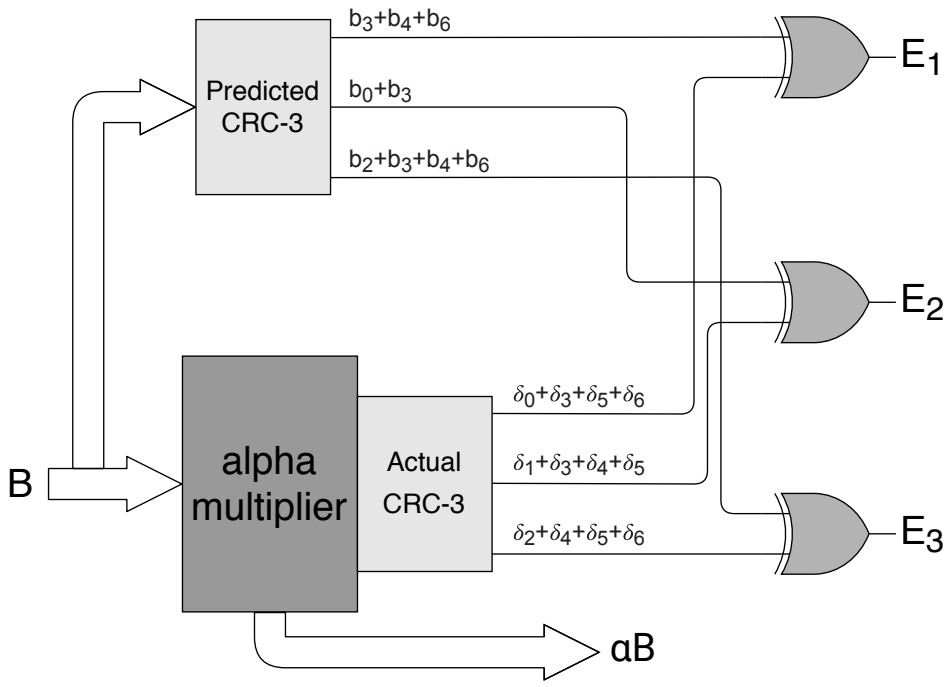


Figure 6.1: αB multiplication with the proposed error detection scheme based on CRC signatures.

6.3 Error Coverage and FPGA Implementations

The error coverage provided by the different error detection schemes presented in this chapter is calculated by finding the number of signatures and by performing $100 \cdot (1-(\frac{1}{2})^s)\%$, where s stands for the number of signatures. To perform the finite field inversion of an element in $GF(2^7)$ with normal basis, a total of $3_{mult} \cdot (6_{\alpha B} \cdot (3_{EF}))$ or 54 signatures are needed, where *mult* denotes the number of finite field multiplications, αB stands for the number of αB operations in each multiplication, and EF is the number of error flags in each αB operation. These 54 signatures translate into an error coverage of $100 \cdot (1 - (\frac{1}{2})^{54})\%$ or close to 100%.

To demonstrate that the error detection schemes proposed in this work provide high error coverage with an acceptable overhead, such schemes are embedded into the original αB multipliers to perform finite field inversion using ITA. The implementations are performed on Xilinx FPGA family Virtex-7 device xc7vx1140tflg1930-i using the Vivado tool and Verilog

Table 20: Overheads of the proposed error detection schemes for finite field inversion using ITA and elements in $GF(2^7)$ with normal basis.

Architecture	Area (CLBs)	Delay (ns)	Power (mW) @50 MHz
Original ITA	294	1.452	0.599
ITA with CRC-3	408 (38.78%)	1.609 (10.81%)	0.610 (1.84%)

as the hardware design entry. The respective overheads are presented in Table 20 in terms of area (occupied slices), delay (ns), and power (mW) with the clock frequency of 50 MHz. As shown in Table 20, CRC-*3* obtains an area overhead of less than 39% and a delay overhead of close to 11%.

To the best of the author's knowledge, there is no previous research on this type of error detection schemes for finite field inversions using ITA for elements in $GF(2^m)$ with normal basis. In [99], authors performed error detection based on parity prediction for normal basis multiplication obtaining a combined worst-case area and delay overhead of 58.1%. Additionally, parity prediction signatures provide an error detection of up to 50%, i.e., if the number of faults is even, the approach would not be able to detect the faults. This highly predictable countermeasure can be circumvented by intelligent fault injection. In this chapter, CRC-*3* is proposed as the case study for this chapter, which overcomes this problem and is intended for deeply-embedded systems where high-performance, low-overhead, and low-energy are preferred; however, larger CRC signatures can be employed for devices where power and area are not critical.

In Table 21, worst-case complexity comparisons in terms of XOR gates, finite field multiplications, error flags, and error coverage for larger $GF(2^m)$ finite field inversions with normal basis using ITA are presented. The number of XOR gates is calculated by performing $r \cdot (m-1) \cdot (m \cdot (t-1)+1)$ for constructions without any error detection scheme and $r \cdot (m-1) \cdot ((m \cdot (t-1)+1)+(2 \cdot m \cdot (x-1)+(x-1) \cdot x))$ for constructions with CRC-*3* signatures. In such formulations, x corresponds to the CRC-X used and r stands for the number of finite field multiplications needed using ITA. Following ITA, the field $GF(2^{163})$ has the decomposi-

Table 21: Comparisons for different $GF(2^m)$ inversions using ITA.

Finite field	t-type	Sign.	XOR gates	Mult.	Error Flags	Error Coverage
$GF(2^{163})$	4	None	635,040	8	-	-
		CRC-3	1,487,808		3,888	$100 \cdot (1 - (\frac{1}{2})^{3,888})\%$
$GF(2^{233})$	2	None	635,040	9	-	-
		CRC-3	1,487,808		6,264	$100 \cdot (1 - (\frac{1}{2})^{6,264})\%$
$GF(2^{283})$	6	None	635,040	10	-	-
		CRC-3	1,487,808		8,460	$100 \cdot (1 - (\frac{1}{2})^{8,460})\%$
$GF(2^{409})$	4	None	635,040	10	-	-
		CRC-3	1,487,808		12,240	$100 \cdot (1 - (\frac{1}{2})^{12,240})\%$
$GF(2^{571})$	10	None	635,040	12	-	-
		CRC-3	1,487,808		20,520	$100 \cdot (1 - (\frac{1}{2})^{20,520})\%$

tion: $(1+2)\times(1+2^2\times(1+2^2)\times(1+2^4)\times(1+2^8)\times(1+2^{16})\times(1+2^{32}\times(1+2^{32})\times(1+2^{64})))$; the field $GF(2^{233})$ has the decomposition: $(1+2)\times(1+2^2)\times(1+2^4)\times(1+2^8\times(1+2^8)\times(1+2^{16})\times(1+2^{32}\times(1+2^{32})\times(1+2^{64}\times(1+2^{64}))))$; the field $GF(2^{283})$ has the decomposition: $(1+2)\times(1+2^2\times(1+2^2)\times(1+2^4)\times(1+2^8\times(1+2^8)\times(1+2^{16}\times(1+2^{16})\times(1+2^{32})\times(1+2^{64})\times(1+2^{128}))))$; the field $GF(2^{409})$ has the decomposition: $(1+2)\times(1+2^2)\times(1+2^4)\times(1+2^8\times(1+2^8)\times(1+2^{16}\times(1+2^{16})\times(1+2^{32})\times(1+2^{64})\times(1+2^{128}\times(1+2^{128}))))$; and the field $GF(2^{571})$ has the decomposition: $(1+2)\times(1+2^2\times(1+2^2)\times(1+2^4)\times(1+2^8\times(1+2^8)\times(1+2^{16}\times(1+2^{16})\times(1+2^{32}\times(1+2^{32})\times(1+2^{64})\times(1+2^{128})\times(1+2^{256})))))$. The column $GF(2^m)$ multiplications is determined by such decompositions and the error flags are calculated by performing $r\cdot(m-1)\cdot x$. As it is shown in Table 21, the larger the NIST field is, the more signatures and the more error coverage the architecture will have.

Chapter 7: Conclusion

In this dissertation, many different error detection schemes based on normal, interleaved, two-part, three-part, and CRC signatures have been proposed. Chapters 2, 3, and 4, present fault diagnosis approaches for code-based cryptosystems, i.e., McEliece and Niederreiter cryptosystems. Chapter 5 applies error detection techniques to the multivariate-based Luov's cryptosystem. Lastly, Chapter 6 presents countermeasures against side-channel attacks for the algorithms FLT and ITA, which are used to perform finite field inversions in a more efficient way. The latter one is not confined to specific cryptosystem, e.g., any cryptographic application that uses finite field inversions within its design can apply such countermeasures to create a more reliable system. Many error detection schemes are derived along this dissertation to provide reliability to a broad type of applications. For applications such as where performance is critical (and power consumption is not because these are plugged in), larger signatures can be applied. However, for deeply-embedded systems such as implantable and wearable medical devices, smaller signatures are preferred.

Such schemes have been implemented on different Xilinx FPGA families to benchmark the overhead of the proposed schemes with the original designs. In most of the implementations, the target platform does not necessarily affect the results because the chosen FPGAs belong to the same series (Xilinx series 7), being very similar from a technological point of view. As there are not any specific FPGA-related sub-blocks in most of them, such as large multipliers or inner multiplexers, the choice for hardware platform does not directly affect the derived overhead. The error coverage percentage for the different designs are all close to 100% with an acceptable overhead.

As long-term extensions to this dissertation, countermeasures for side-channel attacks against lattice-based cryptography will be studied. Lattice-based cryptosystems are not new, they have been used for many years in traditional known public-key schemes such as the RSA, Diffie-Hellman or elliptic-curve cryptosystems. Lattice-based cryptography involves all cryptographic primitives that include lattices, either in the construction itself or in the security proof. Their high efficiency, strong security guarantees from worst-case hardness, and simplicity to implement, make lattice-based cryptography a promising quantum-resistant class. Another type of concurrent error detection technique based on recomputing with shifted, negated, and scaled operands will be applied to their original implementation on FPGA to provide reliability to lattice-based cryptography systems.

References

[1] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. 1999.

[2] Don Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM journal of research and development*, 38(3):243–250, 1994.

[3] Qiuxia Zhang, Zhan Li, and Chao Song. The improvement of digital signature algorithm based on elliptic curve cryptography. In *2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)*, pages 1689–1691. IEEE, 2011.

[4] Nan Li. Research on Diffie-Hellman key exchange protocol. In *2010 2nd International Conference on Computer Engineering and Technology*, volume 4, pages V4–634. IEEE, 2010.

[5] Xin Zhou and Xiaofei Tang. Research and implementation of RSA algorithm for encryption and decryption. In *Proceedings of 2011 6th international forum on strategic technology*, volume 2, pages 1118–1121. IEEE, 2011.

[6] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[7] Dustin Moody. Post-quantum cryptography: NIST's plan for the future. In *The Seventh International Conference on Post-Quntum Cryptography, Japan*, 2016.

[8] Daniel J Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, et al. Classic McEliece: conservative code-based cryptography. *NIST submissions*, 2017.

[9] Upendra Kapshikar and Ayan Mahalanobis. A quantum-secure Niederreiter cryptosystem using quasi-cyclic codes. *arXiv preprint arXiv:1803.07827*, 2018.

[10] Sabah Suhail, Rasheed Hussain, Abid Khan, and Choong Seon Hong. On the role of hash-based signatures in quantum-safe internet of things: Current solutions and future directions. *IEEE Internet of Things Journal*, 2020.

[11] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS+ signature framework. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2129–2146, 2019.

[12] Brian Koziel, Reza Azarderakhsh, and Mehran Mozaffari Kermani. A high-performance and scalable hardware architecture for isogeny-based cryptography. *IEEE Transactions on Computers*, 67(11):1594–1609, 2018.

[13] Brian Koziel, Amir Jalali, Reza Azarderakhsh, David Jao, and Mehran Mozaffari-Kermani. NEON-SIDH: Efficient implementation of supersingular isogeny Diffie-Hellman key exchange protocol on ARM. In *International Conference on Cryptology and Network Security*, pages 88–103. Springer, 2016.

[14] Youngho Yoo, Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. A post-quantum digital signature scheme based on supersingular isogenies. In *International Conference on Financial Cryptography and Data Security*, pages 163–181. Springer, 2017.

[15] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.

[16] Naina Gupta, Arpan Jati, Amit Kumar Chauhan, and Anupam Chattopadhyay. PQC acceleration using GPUs: FrodoKEM, NewHope, and Kyber. *IEEE Transactions on Parallel and Distributed Systems*, 32(3):575–586, 2020.

[17] Viet B Dang, Farnoud Farahmand, Michal Andrzejczak, and Kris Gaj. Implementing and benchmarking three lattice-based post-quantum cryptography algorithms using software/hardware codesign. In *2019 International Conference on Field-Programmable Technology (ICFPT)*, pages 206–214. IEEE, 2019.

[18] W Beullens, A Szepieniec, F Vercauteren, and B Preneel. Luov: Signature scheme proposal for NIST PQC project (round 2 version), 2018.

[19] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *International Conference on Applied Cryptography and Network Security*, pages 164–175. Springer, 2005.

[20] Mehran Mozaffari Kermani, Meng Zhang, Anand Raghunathan, and Niraj K Jha. Emerging frontiers in embedded security. In *2013 26th international conference on VLSI design and 2013 12th international conference on embedded systems*, pages 203–208. IEEE, 2013.

[21] Mehran Mozaffari Kermani, Erkay Savas, and Shambhu J Upadhyaya. Guest editorial: Introduction to the special issue on emerging security trends for deeply-embedded computing systems. *IEEE Transactions on Emerging Topics in Computing*, 4(3):318–320, 2016.

[22] Xiaofei Guo and Ramesh Karri. Recomputing with permuted operands: A concurrent error detection approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(10):1595–1608, 2013.

[23] Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh. Concurrent structure-independent fault detection schemes for the Advanced Encryption Standard. *IEEE Transactions on Computers*, 59(5):608–622, 2010.

[24] Mehran Mozaffari-Kermani and Reza Azarderakhsh. Efficient fault diagnosis schemes for reliable lightweight cryptographic ISO/IEC standard CLEFIA benchmarked on ASIC and FPGA. *IEEE Transactions on Industrial Electronics*, 60(12):5925–5932, 2012.

[25] Paolo Maistri and Régis Leveugle. Double-data-rate computation as a countermeasure against fault analysis. *IEEE Transactions on Computers*, 57(11):1528–1539, 2008.

[26] Duško Karaklajić, Jörn-Marc Schmidt, and Ingrid Verbauwhede. Hardware designer's guide to fault attacks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(12):2295–2306, 2013.

[27] Nahid Farhady Ghalaty, Bilgiday Yuce, and Patrick Schaumont. Analyzing the efficiency of biased-fault based attacks. *IEEE Embedded Systems Letters*, 8(2):33–36, 2016.

[28] Nahid Farhady Ghalaty, Bilgiday Yuce, Mostafa Taha, and Patrick Schaumont. Differential fault intensity analysis. In *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 49–58. IEEE, 2014.

[29] ©2020 IEEE. Reprinted with permission from Alvaro Cintas Canto, Mehran Mozaffari Kermani, and Reza Azarderakhsh. Reliable architectures for composite-field-oriented constructions of McEliece post-quantum cryptography on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[30] Daniel J Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the McEliece cryptosystem. In *International Workshop on Post-Quantum Cryptography*, pages 31–46. Springer, 2008.

[31] Rafael Misoczki and Paulo SLM Barreto. Compact McEliece keys from Goppa codes. In *International Workshop on Selected Areas in Cryptography*, pages 376–392. Springer, 2009.

[32] Paulo SLM Barreto, Richard Lindner, and Rafael Misoczki. Monoidic codes in cryptography. In *International Workshop on Post-Quantum Cryptography*, pages 179–199. Springer, 2011.

[33] V Gauthier Umana and Gregor Leander. Practical key recovery attacks on two McEliece variants. In *Proceedings of the Second International Conference on Symbolic Computation and Cryptography*, pages 27–44. Citeseer, 2010.

[34] Abdulhadi Shoufan, Thorsten Wink, H Gregor Molter, Sorin A Huss, and Eike Kohnert. A novel cryptoprocessor architecture for the McEliece public-key cryptosystem. *IEEE Transactions on Computers*, 59(11):1533–1546, 2010.

[35] Falko Strenzke, Erik Tews, H Gregor Molter, Raphael Overbeck, and Abdulhadi Shoufan. Side channels in the McEliece PKC. In *International Workshop on Post-Quantum Cryptography*, pages 216–229. Springer, 2008.

[36] Cong Chen, Thomas Eisenbarth, Ingo Von Maurich, and Rainer Steinwandt. Differential power analysis of a McEliece cryptosystem. In *International Conference on Applied Cryptography and Network Security*, pages 538–556. Springer, 2015.

[37] Ingo Von Maurich and Tim Güneysu. Towards side-channel resistant implementations of QC-MDPC McEliece encryption on constrained devices. In *International Workshop on Post-Quantum Cryptography*, pages 266–282. Springer, 2014.

[38] Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh. A low-cost S-box for the Advanced Encryption Standard using normal basis. In *2009 IEEE International Conference on Electro/Information Technology*, pages 52–55. IEEE, 2009.

[39] Xiaofei Guo, Debdeep Mukhopadhyay, Chenglu Jin, and Ramesh Karri. Security analysis of concurrent error detection against differential fault analysis. *Journal of Cryptographic Engineering*, 5(3):153–169, 2015.

[40] Mehran Mozaffari-Kermani, Reza Azarderakhsh, and Anita Aghaie. Fault detection architectures for post-quantum cryptographic stateless hash-based secure signatures benchmarked on ASIC. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(2):1–19, 2016.

[41] Mehran Mozaffari-Kermani, Niranjan Manoharan, and Reza Azarderakhsh. Reliable radix-4 complex division for fault-sensitive applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(4):656–667, 2015.

[42] Mehran Mozaffari Kermani, Rajkumar Ramadoss, and Reza Azarderakhsh. Efficient error detection architectures for CORDIC through recomputing with encoded operands. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2154–2157. IEEE, 2016.

[43] Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh. A high-performance fault diagnosis approach for the AES SubBytes utilizing mixed bases. In *2011 Workshop On Fault Diagnosis And Tolerance In Cryptography*, pages 80–87. IEEE, 2011.

[44] Arash Reyhani-Masoleh and M Anwarul Hasan. Error detection in polynomial basis multipliers over binary extension fields. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 515–528. Springer, 2002.

[45] Mariano López-García and Enrique Cantó-Navarro. Hardware-software implementation of a McEliece cryptosystem for post-quantum cryptography. In *Future of Information and Communication Conference*, pages 814–825. Springer, 2020.

[46] Lake Bu, Rashmi Agrawal, Hai Cheng, and Michel A Kinsy. A lightweight McEliece cryptosystem co-processor design. *arXiv preprint arXiv:1903.03733*, 2019.

[47] Srivatsan Subramanian, Mehran Mozaffari-Kermani, Reza Azarderakhsh, and Mehrdad Nojoumian. Reliable hardware architectures for cryptographic block ciphers LED and HIGHT. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(10):1750–1758, 2017.

[48] Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in $gf(2^m)$ using normal bases. *Information and computation*, 78(3):171–177, 1988.

[49] Jorge Guajardo and Christof Paar. Itoh-Tsujii inversion in standard basis and its application in cryptography and codes. *Designs, Codes and Cryptography*, 25(2):207–216, 2002.

[50] Francisco Rodríguez-Henríquez, N Cruz-Cortes, and NA Saqib. A fast implementation of multiplicative inversion over $gf(2^m)$. In *International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II*, volume 1, pages 574–579. IEEE, 2005.

[51] Jean-Luc Danger, Youssef El Housni, Adrien Facon, Cheikh T Gueye, Sylvain Guilley, Sylvie Herbel, Ousmane Ndiaye, Edoardo Persichetti, and Alexander Schaub. On the performance and security of multiplication in $gf(2^n)$. *Cryptography*, 2(3):25, 2018.

[52] Bao Liu and Ravi Sandhu. Fingerprint-based detection and diagnosis of malicious programs in hardware. *IEEE Transactions on Reliability*, 64(3):1068–1077, 2015.

[53] Dariush Abbasinezhad-Mood and Morteza Nikooghadam. Efficient design of a novel ECC-based public key scheme for medical data protection by utilization of NanoPi fire. *IEEE Transactions on Reliability*, 67(3):1328–1339, 2018.

[54] Muhammad Yasin, Bodhisatwa Mazumdar, Sk Subidh Ali, and Ozgur Sinanoglu. Security analysis of logic encryption against the most effective side-channel attack: DPA. In *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 97–102. IEEE, 2015.

[55] Mehran Mozaffari-Kermani and Reza Azarderakhsh. Reliable hash trees for post-quantum stateless cryptographic hash-based signatures. In *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 103–108. IEEE, 2015.

[56] Mehran Mozaffari Kermani and Reza Azarderakhsh. Reliable architecture-oblivious error detection schemes for secure cryptographic GCM structures. *IEEE Transactions on Reliability*, 68(4):1347–1355, 2018.

[57] Sayandeep Saha, Dirmanto Jap, Debapriya Basu Roy, Avik Chakraborty, Shivam Bhasin, and Debdeep Mukhopadhyay. A framework to counter statistical ineffective fault analysis of block ciphers using domain transformation and error correction. *IEEE Transactions on Information Forensics and Security*, 15:1905–1919, 2019.

[58] Sikhar Patranabis, Debapriya Basu Roy, Anirban Chakraborty, Naveen Nagar, Astikey Singh, Debdeep Mukhopadhyay, and Santosh Ghosh. Lightweight design-for-security strategies for combined countermeasures against side channel and fault analysis in IoT applications. *Journal of Hardware and Systems Security*, 3(2):103–131, 2019.

[59] Wen Wang, Jakub Szefer, and Ruben Niederhagen. FPGA-based key generator for the Niederreiter cryptosystem using binary Goppa codes. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 253–274. Springer, 2017.

[60] Alfred J Menezes, Ian F Blake, XuHong Gao, Ronald C Mullin, Scott A Vanstone, and Tomik Yaghoobian. *Applications of finite fields*, volume 199. Springer Science & Business Media, 2013.

[61] Burton S Kaliski. The Montgomery inverse and its applications. *IEEE transactions on computers*, 44(8):1064–1065, 1995.

[62] Tenkasi V Ramabadran and Sunil S Gaitonde. A tutorial on CRC computations. *IEEE micro*, 8(4):62–75, 1988.

[63] Vladimir M Sidelnikov and Sergey O Shestakov. On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications*, 2(4):439–444, 1992.

[64] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, Frédéric De Portzamparc, and Jean-Pierre Tillich. Structural cryptanalysis of McEliece schemes with compact keys. *Designs, Codes and Cryptography*, 79(1):87–112, 2016.

[65] Gustavo Banegas, Paulo SLM Barreto, Brice Odilon Boidje, Pierre-Louis Cayrel, Gilbert Ndollane Dione, Kris Gaj, Cheikh Thiécoumba Gueye, Richard Haeussler, Jean Belo Klamti, Ousmane N'diaye, et al. Dags: Reloaded revisiting dyadic key encapsulation. In *Code-Based Cryptography Workshop*, pages 69–85. Springer, 2019.

[66] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. LEDAkem: A post-quantum key encapsulation mechanism based on QC-LDPC codes. In *International Conference on Post-Quantum Cryptography*, pages 3–24. Springer, 2018.

[67] Pierre-Louis Cayrel and Pierre Dusart. McEliece/Niederreiter PKC: Sensitivity to fault injection. In *2010 5th International Conference on Future Information Technology*, pages 1–6. IEEE, 2010.

[68] Bhaskar Biswas and Nicolas Sendrier. Mceliece cryptosystem implementation: Theory and practice. In *International Workshop on Post-Quantum Cryptography*, pages 47–62. Springer, 2008.

[69] Stefan Heyse and Tim Güneysu. Towards one cycle per bit asymmetric encryption: Code-based cryptography on reconfigurable hardware. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 340–355. Springer, 2012.

[70] Ausmita Sarker, Mehran Mozaffari-Kermani, and Reza Azarderakhsh. Hardware constructions for error detection of number-theoretic transform utilized in secure cryptographic architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(3):738–741, 2018.

[71] Abdel Alim Kamal and Amr M Youssef. Strengthening hardware implementations of NTRUEncrypt against fault analysis attacks. *Journal of Cryptographic Engineering*, 3(4):227–240, 2013.

[72] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Prob. Contr. Inform. Theory*, 15(2):157–166, 1986.

[73] Luca Breveglieri, Israel Koren, and Paolo Maistri. An operation-centered approach to fault detection in symmetric cryptography ciphers. *IEEE Transactions on Computers*, 56(5):635–649, 2007.

[74] Reza Azarderakhsh, Kimmo Järvinen, and Vassil Dimitrov. Fast inversion in $gf(2^m)$ with normal basis using hybrid-double multipliers. *IEEE Transactions on Computers*, 63(4):1041–1047, 2012.

[75] ©2020 IEEE. Reprinted with permission from Alvaro Cintas Canto, Mehran Mozaffari-Kermani, and Reza Azarderakhsh. Reliable CRC-based error detection constructions for finite field multipliers with applications in cryptography. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(1):232–236, 2020.

[76] Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh. Reliable hardware architectures for the third-round SHA-3 finalist Grostl benchmarked on FPGA platform. In *2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 325–331. IEEE, 2011.

[77] Mehran Mozaffari-Kermani, Reza Azarderakhsh, Ausmita Sarker, and Amir Jalali. Efficient and reliable error detection architectures of Hash-Counter-Hash tweakable enciphering schemes. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(2):1–19, 2018.

[78] Mehran Mozaffari-Kermani and Reza Azarderakhsh. Reliable hash trees for post-quantum stateless cryptographic hash-based signatures. In *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 103–108. IEEE, 2015.

[79] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 206–222. Springer, 1999.

[80] EPC Global. Epc radio-frequency identity protocols class-1 generation-2 UHF RFID protocol for communications at 860 mhz–960 mhz. *Version*, 1(0):23, 2008.

[81] Brian Koziel, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. Low-resource and fast binary edwards curves cryptography. In *International Conference on Cryptology in India*, pages 347–369. Springer, 2015.

[82] Reza Azarderakhsh and Arash Reyhani-Masoleh. Efficient FPGA implementations of point multiplication on binary Edwards and generalized Hessian curves using Gaussian normal basis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(8):1453–1466, 2011.

[83] Qiliang Shao, Zhenji Hu, Shaobo Chen, Pingxiuqi Chen, and Jiafeng Xie. Low-complexity digit-level systolic Gaussian normal basis multiplier. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2817–2827, 2017.

[84] Hayssam El-Razouk, Kirthi Kotha, and Mahidhar Puligunta. Novel $gf(2^m)$ digit-serial PISO multipliers for the self-dual Gaussian normal bases. *IEEE Transactions on Computers*, 2020.

[85] Bahram Rashidi, Sayed Masoud Sayedi, and Reza Rezaeian Farashahi. Efficient and low-complexity hardware architecture of Gaussian normal basis multiplication over $gf(2^m)$ for elliptic curve cryptosystems. *IET Circuits, Devices & Systems*, 11(2):103–112, 2017.

[86] Atef Ibrahim, Turki Alsomani, and Fayez Gebali. Unified systolic array architecture for finite field multiplication and inversion. *Computers & Electrical Engineering*, 61:104–115, 2017.

[87] Haibo Yi, Shaohua Tang, and Ranga Vemuri. Fast inversions in small finite fields by using binary trees. *The Computer Journal*, 59(7):1102–1112, 2016.

[88] Jiakun Li, Zhe Li, Chengbo Xue, Jingqi Zhang, Wei Gao, and Shan Cao. A fast modular inversion FPGA implementation over $gf(2^m)$ using modified x^2n unit. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[89] Hayssam El-Razouk and Arash Reyhani-Masoleh. New architectures for digit-level single, hybrid-double, hybrid-triple field multiplications and exponentiation using Gaussian normal bases. *IEEE Transactions on Computers*, 65(8):2495–2509, 2015.

[90] Arash Reyhani-Masoleh, Hayssam El-Razouk, and Amin Monfared. New multiplicative inverse architectures using Gaussian normal basis. *IEEE Transactions on Computers*, 68(7):991–1006, 2018.

[91] Reza Azarderakhsh and Arash Reyhani-Masoleh. Low-complexity multiplier architectures for single and hybrid-double multiplications in Gaussian normal bases. *IEEE Transactions on Computers*, 62(4):744–757, 2012.

[92] Francisco Rodríguez-Henríquez, Guillermo Morales-Luna, Nazar A Saqib, and Nareli Cruz-Cortés. Parallel Itoh–Tsujii multiplicative inversion algorithm for a special class of trinomials. *Designs, Codes and Cryptography*, 45(1):19–37, 2007.

[93] Jingwei Hu, Wei Guo, Jizeng Wei, and Ray CC Cheung. Fast and generic inversion architectures over $gf(2^m)$ using modified Itoh–Tsujii algorithms. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(4):367–371, 2015.

[94] M Kalaiarasi, VR Venkatasubramani, and S Rajaram. A parallel quad Itoh-Tsujii multiplicative inversion algorithm for FPGA platforms. In *2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP)*, pages 31–35. IEEE.

[95] Sergei Bauer, Stefan Rass, and Peter Schartner. Generic parity-based concurrent error detection for lightweight ARX ciphers. *IEEE Access*, 8:142016–142025, 2020.

[96] Prashant Ahir, Mehran Mozaffari-Kermani, and Reza Azarderakhsh. Lightweight architectures for reliable and fault detection Simon and Speck cryptographic algorithms on FPGA. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(4):1–17, 2017.

[97] Anita Aghaie, Mehran Mozaffari Kermani, and Reza Azarderakhsh. Fault diagnosis schemes for low-energy block cipher Midori benchmarked on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(4):1528–1536, 2016.

[98] Chiou-Yng Lee. Concurrent error detection in digit-serial normal basis multiplication over $gf(2^m)$. In *22nd International Conference on Advanced Information Networking and Applications-Workshops (aina workshops 2008)*, pages 1499–1504. IEEE, 2008.

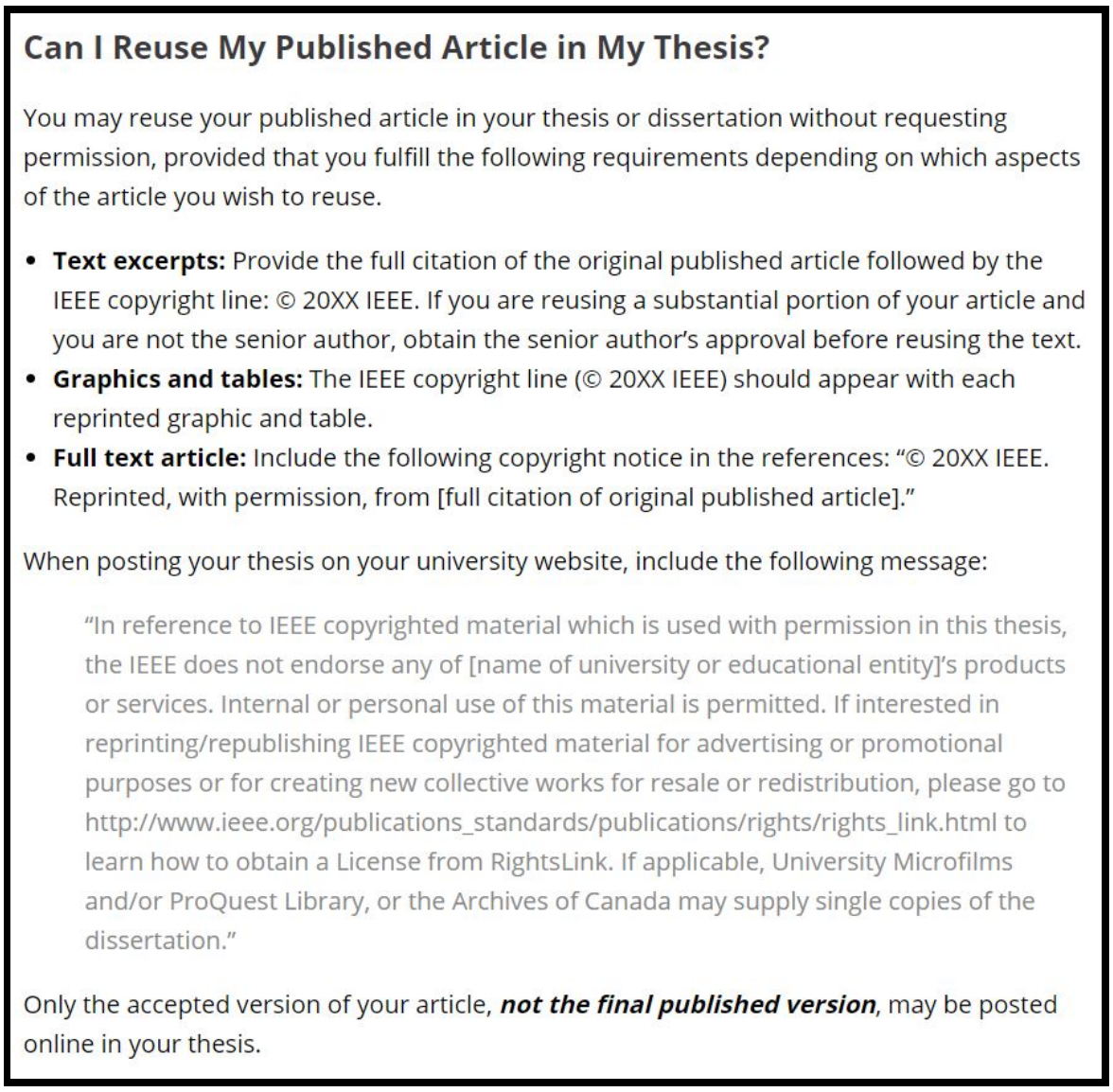
[99] C. Lee, P. K. Meher, and J. C. Patra. Concurrent error detection in bit-serial normal basis multiplication over $gf(2^m)$ using multiple parity prediction schemes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(8):1234–1238, 2010.

[100] Lily Chen, Dustin Moody, Andrew Regenscheid, and Karen Randall. Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters. Technical report, National Institute of Standards and Technology, 2019.

[101] Lijuan Li and Shuguo Li. Fast inversion in $gf(2^m)$ with polynomial basis using optimal addition chains. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

Appendix A: Copyright Permissions

The permission below is for the use of Chapter 2 and Chapter 5.



Can I Reuse My Published Article in My Thesis?

You may reuse your published article in your thesis or dissertation without requesting permission, provided that you fulfill the following requirements depending on which aspects of the article you wish to reuse.

- **Text excerpts:** Provide the full citation of the original published article followed by the IEEE copyright line: © 20XX IEEE. If you are reusing a substantial portion of your article and you are not the senior author, obtain the senior author's approval before reusing the text.
- **Graphics and tables:** The IEEE copyright line (© 20XX IEEE) should appear with each reprinted graphic and table.
- **Full text article:** Include the following copyright notice in the references: "© 20XX IEEE. Reprinted, with permission, from [full citation of original published article]."

When posting your thesis on your university website, include the following message:

> "In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [name of university or educational entity]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation."

Only the accepted version of your article, ***not the final published version***, may be posted online in your thesis.

Figure A.1: Copyright permissions for IEEE journals.