Secure Hardware Constructions for Fault Detection of Lattice-based Post-quantum Cryptosystems

Ausmita Sarker

University of South Florida

Follow this and additional works at: https://digitalcommons.usf.edu/etd

Scholar Commons Citation

This Dissertation is brought to you for free and open access by the USF Graduate Theses and Dissertations at Digital Commons @ University of South Florida. It has been accepted for inclusion in USF Tampa Graduate Theses and Dissertations by an authorized administrator of Digital Commons @ University of South Florida. For more information, please contact scholarcommons@usf.edu.
Secure Hardware Constructions for Fault Detection of Lattice-based Post-quantum Cryptosystems

by

Ausmita Sarker

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.
Srinivas Katkoori, Ph.D.
Hao Zheng, Ph.D.
Nasir Ghani, Ph.D.
Reza Azarderakhsh, Ph.D.

Date of Approval:
March 9, 2022

Keywords: Cryptography, number-theoretic transform, recomputing with encoded operands, ring learning with error, ring polynomial multiplication

Copyright © 2022, Ausmita Sarker
Dedication

To my Mom, Aparna Sarker, who fought against the world for the success of my career;

To the love of my life, Roussieu, who taught me humility and grit;

To my sister, Asma, for her unconditional support;
Acknowledgments

I would to express my sincere gratitude to my advisor, Mehran Mozaffari Kermani, Ph.D., for his relentless support and continual encouragement. His extraordinary patience, enthusiasm, along with his enlightened vision of the research, drove my Ph.D. journey. I aspire to become as kind as well as diligent as Dr. Mozaffari Kermani in my future life.

I wish to extend my special thanks to my committee members, Srinivas Katkoori, Ph.D., Hao Zheng, Ph.D., Nasir Ghani, Ph.D., Reza Azarderakhsh, Ph.D., for investing a tremendous amount of time to thoroughly analyze the dissertation. Their constructive feedback and attention to detail strengthened the dissertation many-folds. It is my honor to have such prestigious scholars on my committee.

My gratitude goes to the faculty members and staff of the department of CSE at USF, especially, Jessica Pruitt and Laura Owczarek, who dedicated their time to the smooth operations of my grad works and research. Here, I want to thank my undergrad professor, Gopa Biswas Caeser for the kindness she bestowed upon me during my tough times. To my family and friends who are supporting me from here and afar, I am grateful.

I feel the result of this work goes to my husband, Rousseau, whose inspiration and sacrifice are paramount to the fruition of my academic life. His patience and unreserved love are what kept me afloat during this tough time. I could not have done it without him.

I want to thank my mom, Aparna Sarker. Your support and love are the foundation of who I am as a person and a researcher.
Table of Contents

List of Tables ................................................................. iv
List of Figures ............................................................... v
Abstract ........................................................................ vii

Chapter 1: Introduction ......................................................... 1
  1.1 Cryptography and Internet of Things ............................... 1
  1.2 Post Quantum Cryptography ........................................ 2
  1.3 Fault Attacks and Detection ......................................... 3
  1.4 Objectives .................................................................. 4
  1.5 Dissertation Outline .................................................. 5

Chapter 2: Hardware Constructions for Error Detection of Number-Theoretic Transform Utilized in Secure Cryptographic Architectures ......................................................... 7
  2.1 Number-Theoretic Transform ....................................... 7
  2.2 Preliminaries ............................................................ 8
  2.3 Proposed Error Detection Scheme ................................. 9
    2.3.1 Efficient NTT Implementation ............................... 10
    2.3.2 Recomputing with Negated Operands .................... 11
    2.3.3 Recomputing with Scaled Operands ....................... 12
    2.3.4 Recomputing with Swapped Operands .................... 14
  2.4 ASIC Assessments and Comparisons ............................... 15
  2.5 Conclusion .................................................................. 16

Chapter 3: Error Detection Architectures for Ring Polynomial Multiplication and Modular Reduction of Ring-LWE in \( \mathbb{Z}/p\mathbb{Z}[x]/x^n+1 \) Benchmarked on ASIC ............................... 18
  3.1 Ring Polynomial Multiplication and Ring-Learning With Error 18
  3.2 Preliminaries ............................................................ 21
    3.2.1 Ring Polynomial Multiplication ............................. 21
    3.2.2 Ring-LWE Encryption Scheme ............................... 21
  3.3 Proposed Error Detection Scheme for Ring Polynomial Multiplication 22
    3.3.1 Ring Polynomial Multiplication Architecture ............ 25
    3.3.2 Proposed Error Detection Scheme through Recomputing 26
    3.3.3 Ameliorating the Throughput Overhead through Pipelining 31
  3.4 Proposed Error Detection Schemes for Ring-LWE Architecture 32
    3.4.1 Error Detection Scheme for Polynomial Multiplier and \( q=16381 \) 33
    3.4.2 Error Detection Scheme for SAMS2 Approach and \( q=12289 \) 34
### Chapter 3: Fault Detection Architectures for Inverted Binary Ring-LWE Construction Benchmarked on FPGA

#### 4.1 Inverted Binary Ring-LWE

#### 4.2 Preliminaries

#### 4.3 Proposed Fault Detection Schemes

4.3.1 Recomputing with Encoded (Shifted) Operands

4.3.1.1 Key Generation

4.3.1.2 Encryption

4.3.1.3 Decryption

4.3.2 Recomputing with Encoded (Negated) Operands

4.3.2.1 RENO on Key Generation

4.3.2.2 RENO on Decryption

#### 4.4 Error Coverage and FPGA Implementations

4.4.1 Fault Simulation

4.4.2 FPGA Comparison for Error Detection

#### 4.5 Conclusion

### Chapter 4: Fault Detection Architectures for Post Quantum Signature Falcon’s Sampler and KEM SABER

#### 5.1 Post-Quantum KEM and Signature Schemes

#### 5.2 Preliminaries

5.2.1 Recomputing Overview

5.2.1.1 Saber Overview

5.2.2 Falcon Overview

#### 5.3 Proposed Error Detection Techniques

5.3.1 Fault Attacks and Threat Model

5.3.2 Proposed Error Detection Schemes on SABER

5.3.2.1 Error Detection on Binomial Sampler

5.3.2.2 Error Detection on Parallel Polynomial Multiplication

5.3.2.3 Error Detection on HW/SW Codesign

5.3.3 Error Detection Schemes on Falcon Sampler

5.3.3.1 Recomputing on Negation

5.3.3.2 RESwO on Multiplication

5.3.3.3 RENO on Multiplication

5.3.3.4 RENO on Multiplication-and-Accumulator (MAC)

5.3.3.5 RENO on Overall $f_{\text{fsampling}_n^*}$

5.3.4 Implementation of Constant-time Falcon Sampler

5.3.4.1 ModFalcon Implementation and Error Detection
List of Tables

Table 2.1 Implementation results for ASIC through TSMC 65-nm for three case studies, i.e., \((n, p)_1 = (64, 257), (n, p)_2 = (256, 65537),\) and \((n, p)_3 = (512, 4294967297),\) and two proposed architectures, i.e., recomputing with swapped operands-RESwO and its modified variant RESwO-modified (RESwO-m) ........................................................................ 16

Table 3.1 Implementation results for ASIC TSMC 65-nm of RPM architecture (Prop. 1: Negating both operands, Prop. 2: Negating one operand) .......................................................................................................................... 40

Table 4.1 Implementation results for FPGA through Kinex-UltraScale+ and Virtex-7 for encryption (Enc_{Kin} and Enc_{Vir}, respectively) and key generation/decryption (Gen_{Kin} and Gen_{Vir}, respectively). .............. 55

Table 5.1 Implementation results for FPGA through Xilinx Zynq-UltraScale+ ZCU102 (xczu9eg-ffvb1156-2-e) for binomial sampling, polynomial multiplication and hardware/software codesign. .................. 78
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Proposed butterfly construction for NTT through recomputing with negated tri operands (RENTO)</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Proposed butterfly construction for NTT through recomputing with scaled dual operands (RESdo)</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>Proposed butterfly construction for NTT through recomputing with Swapped Operands (RESwo)</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>Hardware architecture of ring polynomial multiplication in $\mathbb{R} = \mathbb{Z}/p\mathbb{Z}[x]/x^n+1$.</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Hardware architecture of proposed recomputing with scaled operand (RESco)</td>
<td>27</td>
</tr>
<tr>
<td>3.3</td>
<td>Hardware architecture of proposed recomputing with negated operand (RENO)</td>
<td>28</td>
</tr>
<tr>
<td>3.4</td>
<td>Hardware architecture of modified RENO.</td>
<td>30</td>
</tr>
<tr>
<td>3.5</td>
<td>Pipelined scheduling for data path of the proposed schemes.</td>
<td>32</td>
</tr>
<tr>
<td>3.6</td>
<td>Proposed construction of schoolbook $\log_2 q \times \log_2 q$ bit multiplier for $q = 16381$.</td>
<td>34</td>
</tr>
<tr>
<td>3.7</td>
<td>Our proposed SAMS2 construction for error detection in modular reduction.</td>
<td>35</td>
</tr>
<tr>
<td>3.8</td>
<td>SAMS2 construction of Shift-add block and Multq. block.</td>
<td>36</td>
</tr>
<tr>
<td>3.9</td>
<td>SAMS2 construction Subt. block.</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>Hardware construction of recomputing with shifted operands for key generation of InvRBLWE.</td>
<td>47</td>
</tr>
<tr>
<td>4.2</td>
<td>Hardware construction of recomputing with shifted operands for encryption of InvRBLWE.</td>
<td>48</td>
</tr>
<tr>
<td>4.3</td>
<td>Hardware construction of recomputing with shifted operands for decryption of InvRBLWE.</td>
<td>49</td>
</tr>
</tbody>
</table>
Figure 4.4 Hardware construction of recomputing with negated operands (RENO) for (a) key generation (b) encryption of InvRBLWE (the gray-colored box denotes the module on the corresponding scheme in the RESO figures.) ........................................... 51

Figure 4.5 Hardware construction of recomputing with negated operands (RENO) for decryption. .................................................. 52

Figure 5.1 Error detection architecture on binomial sampler. ............... 63

Figure 5.2 Proposed error detection architecture on polynomial multiplication with multiply-and-accumulate (MAC) unit construction. ......... 65

Figure 5.3 Proposed error detection architecture on hardware accelerator. .... 66

Figure 5.4 Proposed recomputing with negated operands (RENO) on negation of key generation in Falcon ................................. 69

Figure 5.5 Proposed recomputing with swapped operands (RESwO) on multiplication of key generation in Falcon ............................. 70

Figure 5.6 RENO on multiplication of key generation in Falcon ............... 71

Figure 5.7 RENO on multiplication-and-accumulator (MAC) module of Falcon . 72

Figure 5.8 RENO on the overall ffsampling∗ ...................................... 73

Figure 5.9 Signature scheme of ModFalcon architecture. .................... 74
Abstract

The advent of quantum computers and the exponential speed-up of quantum computation will render classical cryptosystems insecure, as that can solve current encryptions in minutes, resulting in a catastrophic failure of privacy preservation and data security. Through the standardizing of quantum-resistant public-key cryptography algorithms, the National Institute of Standards and Technology (NIST) is evaluating potential candidates to thwart such quantum attacks. In this dissertation, countermeasures against fault attacks are proposed to secure various lattice-based cryptosystems, one of the most promising post-quantum cryptosystems. Fault detection architectures for crucial building blocks of lattice-based cryptosystems, i.e., number-theoretic transform, ring polynomial multiplication, and ring learning with error are introduced. Moreover, the secure hardware architecture of post-quantum key encapsulation mechanism SABER and the signature scheme Falcon are explored. The proposed architectures can also detect natural faults, caused by device malfunctions, which are crucial to proper functionalities of sensitive and secure deeply-embedded systems with stringent constraints.
Chapter 1: Introduction

1.1 Cryptography and Internet of Things

The emergence of the Internet of Things (IoT) broadens the traditional Internet by including smart devices in computing systems. Ensuring secure communication for IoT devices is crucial to thwart privacy attacks and prevent the exploitation of security vulnerabilities, e.g., smart plug [1], thermostat [2], and smart light [3] to name a few. Although symmetric key cryptography, where the sender and receiver share the same key to encrypt and decrypt, has proven to be extremely efficient in terms of performance [4], the security vulnerability of pre-shared keys as well as key distribution problems raise concerns for real-world applications [5]. Public-key cryptography (PKE), on the other hand, uses a pair of keys, i.e., public and private, preserving the integrity and confidentiality of two-party communication systems. While PKE schemes are the most prominent protocol for secure key-exchange and communication establishment, classic PKEs, i.e., RSA [6] and elliptic curve cryptography (ECC) [7] could be impractical for resource constrained architecture of IoT, because of high complexity and expensive performance metrics, e.g., device area, run-time, or energy consumption. Secure communication and establishing temporary session keys are two crucial aspects of IoT security, with a goal of fast key generation and low resource utilization. Several works have explored the efficient implementation of PKE for resource constrained application [8, 9] for a specific target platform or code-based cryptography [10]. However, costly key generation and the large key size of the aforementioned approaches have prevented the practical application so far. Moreover, the advent of quantum computers pose imminent threats to traditional cryptosystems as the classic PKEs are vulnerable to quantum attacks based on
Shor’s algorithm [11], leading to active research on alternatives to PKEs for post-quantum era.

1.2 Post Quantum Cryptography

The security of currently employed PKEs depends on the hardness of factoring (RSA) or the elliptic curve discrete logarithm problem (ECC). However, classical PKEs are not sufficient to protect our cryptosystems in the long term [12], as Shor demonstrated that both classes of problems will be efficiently solved in polynomial time by quantum computers [11]. The fast development in the field of quantum computers and their computational power as well as the progress in cryptanalysis urge the research on post-quantum secure yet practical cryptosystems, namely post-quantum cryptography (PQC). In late 2017, the National Institute of Standards and Technology (NIST) has announced the soliciting and standardizing of one or more quantum-resistant public-key cryptography algorithms [13] to be finalized in 2024.

The PQC research is focused on five classes of algorithms: Lattice-based, hash-based [14], multivariate-based [15], code-based [16], and isogeny-based [17] cryptography. Among them, lattice-based cryptography has long been considered secure yet inefficient, for having large parameters beyond practicality. The introduction of cyclic and ideal lattices [18] changed this perception through theoretically elegant and efficient cryptographic primitives. Based on hard and quantum-resistant problems of finding solution of linear equation, the lattice-based cryptography is one of the most resourceful approaches which can be employed on many aspects of a cryptosystem, e.g., encryption [18], digital signature [19] and identification [20]. The hardness of learning with error (LWE), a variant of lattice-based cryptosystem, has received much attention due to its lower complexity, high efficiency, and scalability, which are suitable for resource constraint IoT applications as well as its robustness against quantum computations [21].
1.3 Fault Attacks and Detection

The side-channel security analysis of NIST PQC standardization is an emerging research topic demanding extensive study for practical deployment. Every implementation should be evaluated against side-channel analysis (SCA) attacks. Such physical attacks exploit the information externally available information (e.g., power consumption, run-time), rather than the vulnerabilities of a cryptographic algorithm. The adversaries exploit the inadvertent information leakage of a device, e.g., timing information, power consumption, or electromagnetic radiation. As these attacks are non-invasive and require cheap equipment, SCA pose serious security threats to most cryptographic hardware device, ranging from smart card to computers [22].

One popular variety of SCA is active fault analysis (FA) [23], where the adversary introduces faults into cryptographic systems and observes the difference. When the attackers simply observe the device’s behavior without disturbing the proper functioning or attempting to access the inside functionalities, these are called passive SCA. On the contrary, in case of active SCAs, such as differential fault analysis (DFA), the adversary injects faults into the systems and compares the faulty output with the non-faulty operation, with little influence on the actual fault value. Laser injection is one of the most precise fault injection techniques, where the adversary has control over the timing and location of the fault. Other cheaper yet effective fault injection methods are clock glitches and power supply drop. Based on the set of incorrect responses, the attacker can decipher the secret information of the device. Exploiting the presence of transient faults (lasting one or few clock cycles) or permanent faults, these attacks pose threat to majority of cryptosystems [24], even lattice-based cryptosystems [25, 26].

Fault detection schemes can determine if a system is tampered with and fault analysis has taken place. Previous works on fault detection [27–31] have explored fault detection on classical cryptosystems. Among the three fault detection techniques, concurrent error detection (CED), off-line detection and Roving fault detection, this dissertation studies the
concurrent error detection technique. CED can be classified into four types redundancy, hardware, time, information, and hybrid redundancy. Hardware redundancy duplicates the function and detects faults by comparing the output of two operations. Time redundancy deals with performing the same function twice and may detect both permanent and transient faults [32]. Information redundancy involves adding check bits (e.g., parity) to determine fault attacks. Hybrid redundancy requires an operation is followed by its inverse operation. In this dissertation, we explore the fault attacks of various lattice-based encryption as well as signature schemes and propose fault detection methods which thwart permanent and transient fault attacks.

1.4 Objectives

The introduced error detection schemes of different promising post-quantum cryptosystems are explored for different performance and implementation metrics and efficiency. The proposed architectures are benchmarked to assess their ability to detect transient and permanent faults. With high error coverage, the presented approaches achieve acceptable overhead and can be tailored towards the objectives in terms of error detection and reliability. These approaches add very little hardware overheads, which is advantageous to incorporate in deeply-embedded systems.

The objectives of this dissertation are as follows:

- We devise fault detection schemes error detection architecture in key generation, encryption and decryption stages of multiple state-of-the-art lattice based cryptosystems. The fault detection explored in this dissertation emphasizes on the performance bottlenecks and the most computationally exhaustive stage each crypto-algorithm, whose security is crucial for the proper operation of that entire cryptosystem. Our proposed schemes are not confined to certain cryptographic constructions.
• We also explore recomputing schemes in the signature algorithms of post-quantum signature scheme Falcon. We apply recomputing schemes to achieve high fault coverage.

• We have simulated the error coverage of our proposed work with HDL as design entry, by injecting stuck-at faults. We observed high error detection rates for both permanent and transient faults incorporating our schemes.

• We implement our schemes on application-specific integrated circuit (ASIC) using Synopsys Design Compiler or field-programmable gate array (FPGA) to derive the implementation and performance metrics. The proposed error detection schemes add acceptable overheads, compared to the original implementation.

1.5 Dissertation Outline

The error detection of lattice-based post-quantum cryptosystems are investigated in this dissertation. The chapter outline is as follows:

• Chapter 2: This chapter introduces efficient error detection schemes for number-theoretic transform, a crucial as well as efficient Fourier transform over ring, for the state-of-the-art lattice-based cryptosystems.

• Chapter 3: Error detection schemes of both RPM and modular reduction blocks, as different ring-LWE architectures use different moduli, depending on the security level and application, are proposed in this chapter.

• Chapter 4: This chapter presents fault detection constructions on Ring-BinLWE architecture, which can be tailored based on the needs in terms of reliability and the restrictions in terms of the added overhead in constrained applications.

• Chapter 5: Fault detection schemes for SABER on the performance bottleneck, the PRNG generator involving a binomial sampler, the polynomial multiplier architecture as well as high-level architecture of the HW/SW codesign approach of SABER
are introduced in this chapter. Moreover, this chapter also proposes error detection schemes for the hardware construction of Falcon’s sampler, specifically, in the signature algorithm of ModFalcon and the Gaussian sampler.

• Chapter 6: The dissertation is concluded in this chapter.
Chapter 2: Hardware Constructions for Error Detection of Number-Theoretic Transform Utilized in Secure Cryptographic Architectures

2.1 Number-Theoretic Transform

Number theoretic transform (NTT) [34] is a discrete Fourier transform defined over a finite ring or field. Being an elegant polynomial multiplication technique, NTT is essential to post-quantum cryptosystems, e.g., lattice-based cryptosystems. Such cryptosystems rely on well-studied, hard problems, the merit of which is that quantum algorithms to solve these problems efficiently are yet unknown. One of the most common average-case lattice problems are learning with errors (LWE) problem [18], which assures the hardness of solving other lattice problems in the worst case [21]. However, this very appealing technique gives an impractical key-size of quadratic, i.e., $O(n^2)$ complexity, for security parameter $n$ [35]. To reduce the complexity, cyclic [36] and ideal lattices [37] are introduced. Using computation based on fast Fourier transform (FFT), these structures can enable construction of theoretically robust and efficient cryptosystems with quasi-linear, i.e., $O(n.lgn)$, key lengths.

Ideal lattices are also employed in fully homomorphic encryption (FHE) [38] or somewhat homomorphic encryption (SHE) [39], two new primitives with strong potential for securing cloud computing. Polynomial multiplication is the most computationally-exhaustive operation of ideal lattices. Applying number theoretic constructions provides speed advantage, because the polynomial multiplication can be efficiently computed in quasi-linear time $O(n.lgn)$ using FFT [40].

\footnote{This chapter was published in the IEEE Transactions on Very Large Scale Integration Systems (TVLSI) [33] ©2019 IEEE}
Besides post-quantum cryptography, NTT can radically improve currently-used schemes by increasing their security parameters. For example, NTT proves to be a valuable tool to signature schemes [41], collision resistant hash functions [42], as well as identification schemes [20]. As a result, efficient error detection schemes of NTT in polynomial multiplication will boost the security and reliability of post-quantum cryptography as well as existing cryptosystems.

Previous studies of NTT-based polynomial multiplication have dealt with reconfigurable hardware [43] and efficient architecture to achieve high speed [44]. Examples of other interesting recent works related to the respective implementations include [45], [46]. However, no work is yet proposed in open literature focusing on error detection of NTT polynomial multiplier.

Error detection in cryptography has been center of attention in previous work [29, 47–54]. In this chapter, we propose error detection schemes of NTT polynomial multiplier. The Main contributions of this chapter are summarized as follows:

- We introduce a number of categories for error detection in NTT of the ring \( \mathbb{R} = \mathbb{Z}/p\mathbb{Z}[x]/x^n+1 \). Our proposed schemes are not confined to certain cryptographic constructions.
- The first category of the proposed error detection schemes involves recomputing with negated operands. Moreover, we present recomputing with scaled operands. The last category constitutes recomputing with swapped operands. Our target is low hardware overhead, which is favorable to compact and deeply-embedded architectures.
- We implement the proposed error detection architectures on application-specific integrated circuit (ASIC) for a 65nm library to assess the implementation and performance metrics.

2.2 Preliminaries

In this chapter, we have considered ideal lattices, defined by \( \mathbb{R} = \mathbb{Z}/p\mathbb{Z}[x]/x^n+1 \). Here, \( f(x) \) is an irreducible polynomial of degree \( n \), which can be represented as \( f(x) = f_0 + f_1x + \)
\( f_2x^2 + \ldots + f_{n-1}x^{n-1} \). Also, \( n \) is a power of 2, and \( p \) is a prime number where \( p \equiv 1 \mod 2n \).

Multiplication of two polynomials \( a(x), b(x) \in \mathbb{Z}_p \), can be represented as:

\[
a(x) \cdot b(x) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j x^{i+j} \mod f(x),
\]

(2.1)

taking quadratic complexity of \( O(n^2) \) utilizing school book algorithm.

On the contrary, number theoretic transform is a discrete Fourier transform, defined in a finite field, \( \mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}[x] \) [1]. For a given primitive \( n \)-th root of unity in \( \mathbb{Z}_p \), \( A(x) \) and \( B(x) \) are polynomials under \( \mathbb{Z}_p \), where both are generic forward \( NTT\omega(a) \) and \( NTT\omega(b) \), respectively:

\[
A_i = NTT^n_\omega(a(x))_i = \sum_{j=0}^{n-1} a_j \omega^{ij} \mod p, \ i = 0, 1, \ldots, n - 1
\]

(2.2)

The NTT exists if and only if the block length \( n \) divides \( q - 1 \) for every prime factor \( q \) of \( p \), where \( p \) is a prime and \( n \) is a power of 2. Computing Inverse NTT (INTT) is similar to computing NTT, while replacing \( \omega \) with \( \omega^{-1} \) and introducing \( n^{-1} \), i.e.,

\[
a_i = INTT^n_\omega(A(x))_i = n^{-1} \sum_{j=0}^{n-1} A_j \omega^{-ij} \mod p, \ i = 0, 1, \ldots, n - 1
\]

(2.3)

As \( p \) is a prime, the inverse of \( n, n^{-1} \) can be computed in modulo \( p \), where \( n.n^{-1} \equiv 1 \mod p \). Applying NTT and INTT to compute polynomial multiplication reduces the time complexity from \( O(n^2) \) to \( O(n.lgn) \).

### 2.3 Proposed Error Detection Scheme

For high-performance lattice-based cryptography, a flexible NTT-based polynomial multiplier is required. In this section, we present our schemes to provide error detection hardware architectures with low complexity. The proposed approaches constitute three categories, i.e., recomputing with encoded operands through negated, scaled, and swapped operands.
2.3.1 Efficient NTT Implementation

In Algorithm 2.1 [55], the iterative FFT implementation computes the NTT of a given polynomial \( a(x) \in \mathbb{Z}_p \). The Bit-Reverse(a) operation (line 1) reorders the input vector \( a \), in which, the new position of the elements in position \( k \) can be found by reversing the binary representation of \( k \). This algorithm utilizes the “butterfly operation” [21] (lines 8 and 9), which is the multiplication of the factor \( \omega^N \mod n \) with \( d \), and addition with or subtraction of the result from \( c \). Lines 5-10 divide the input polynomial into two smaller polynomials, each with length \( n/2 \) and perform NTT on each polynomial simultaneously. Instead of transforming the entire polynomial of degree \( n \), decomposing \( a \) in two halves and computing the NTT in parallel improves the time complexity from quadratic \( (O(n^2)) \) to quasi-linear \( (O(n \cdot \log n)) \).

\begin{algorithm}
\textbf{Algorithm 2.1 Iterative-NTT}
\begin{algorithmic}[1]
\State \textbf{Input:} \( a \in \mathbb{Z}_p[x] \) of length \( n = 2^k \) with \( k \in \mathbb{N} \) and a primitive \( n \)-th root of unity \( \omega \in \mathbb{Z}_p \)
\State \textbf{Output:} \( y = \text{NTT}_\omega(a) \)
\State 1: \( A \leftarrow \text{Bit-reverse}(a); \ m \leftarrow 2 \)
\State 2: \textbf{while} \( m \leq N \) \textbf{do}
\State 3: \( s \leftarrow 0 \)
\State 4: \textbf{while} \( s < N \) \textbf{do}
\State 5: \textbf{for} \( i \) \textbf{to} \( m/2 - 1 \) \textbf{do}
\State 6: \( N \leftarrow i.\frac{n}{m}; \ a \leftarrow s + i; \ b \leftarrow s + i + m/2 \)
\State 7: \( c \leftarrow A[a]; \ d \leftarrow A[b] \)
\State 8: \( A[a] \leftarrow c + \omega^N \mod n \ mod \ p \)
\State 9: \( A[b] \leftarrow c - \omega^N \mod n \ mod \ p \)
\State 10: \textbf{end for}
\State 11: \( s \leftarrow s + m \)
\State 12: \textbf{end while}
\State 13: \( m \leftarrow m.2 \)
\State 14: \textbf{end while}
\State 15: \textbf{return} \( A \)
\end{algorithmic}
\end{algorithm}
2.3.2 Recomputing with Negated Operands

In proposing the error detection approaches, we make sure that augmenting the original constructions with the proposed schemes leads to low-complexity architectures. As a result, we have applied a number of recomputing with negated operands schemes.

The architecture for NTT consists of the common butterfly structure (lines 8 and 9 of Algorithm 2.1). This well-known structure performs the core operation of NTT implementation, multiplying elements of the polynomial by powers of $\omega$. Each cycle computes one node of NTT flow, where a multiplier, followed by a modular reduction (mod $p$ block in Figure 2.1) circuit performs polynomial multiplication by reiterating the butterfly operation. For this most rigorous operation within such constructions, we propose two variants of our scheme. The first one is through recomputing with negated triple operands (RENtO) in

![Figure 2.1: Proposed butterfly construction for NTT through recomputing with negated triple operands (RENtO).](image-url)
which, as the name suggests, two operands are negated. The second one, shown in Figure 2.1, is recomputing with negated tri operands (REntO), in which all three operands, i.e., \(c\), \(\omega\), and \(d\), are negated. In these approaches, encoding/decoding are the most prominent operations (and carefully-thought operations to implement). In the latter, i.e., RENtO, for a modified architecture of NTT-butterfly, we insert a negation unit for modulo \(p\) negation, multiplexer, and comparator circuits. The select of multiplexer Norm/REntO, determines original NTT or RENtO operation. In accordance with lines 8 and 9 of Algorithm 2.1, at the original NTT operation, the outputs are \(A\) and \(B\) where, \(A = c + \omega d\) and \(B = c - \omega d\). During the encoding stage, which is active at RENtO only, we negate all inputs, i.e., \(c\), \(\omega\) and \(d\), and they eventually become \(p - c\), \(p - \omega\) and \(p - d\), respectively. Thus, the encoded operands are \(A'\) and \(B'\), where \(A' = -c + \omega d\) and \(B' = -c - \omega d\). The decoding operation is as follows: We negate \(A'\) and \(B'\), and the decoded outputs are compared with their alternate pre-recomputed outputs. At the input of the decoder, depending on the multiplexer select, the data bus flows either \(A\) or \(A'\), which is represented as \(A/A'\) in Figure 2.1. In addition, for the former approach, i.e., RENdO, encoding and decoding blocks are identical to RENtO. However, in the comparator circuit, we compare the decoded output with their respective original output.

2.3.3 Recomputing with Scaled Operands

A second variant of the proposed error detection schemes involves scaling the operands, e.g., doubling, quadrupling, or multiplying with a factor. Let us present an example to explain the scheme. A first example, i.e., recomputing with doubled and quadrupled operands (REdqO), involves doubling \(\omega\) and \(d\), and deriving the quadruple of \(c\). The encoded operands would be \(A' = 4c + (2\omega \cdot 2d)\) and \(B' = 4c - (2\omega \cdot 2d)\). The decoding is performed by dividing the outputs by 4. In binary, dividing by 4 is right shift two places, making decoding a relatively-inexpensive operation. A second example would be, instead of doubling all the operands as REdqO, doubling only \(\omega\) and \(c\), i.e., recomputing with doubled operands (REdO). The
encoding and decoding of REdO is much similar to REdqO, requiring only one doubler and one divider, i.e., one left and one right shift operation, resulting in low hardware overhead and time delay.

In a more general variant of recomputing with scaled operands, namely, recomputing with scaled dual operand (REScdO), we scale both $\omega$ and $c$ by the factor $k$. This is shown in Figure 2.2. Encoding operations would give $A' = kc + (k\omega) \ast d = k(c + \omega d)$ and $B' = kc - (k\omega) \ast d = k(c - \omega d)$. Decoding is performed by dividing both operands with $k$ Figure 2.2. As $p$ is a prime number, $gcd(k, p) \equiv 1 \mod p$, for all values of $k$. 

Figure 2.2: Proposed butterfly construction for NTT through recomputing with scaled dual operands (REScdO).
2.3.4 Recomputing with Swapped Operands

If we swap $\omega$ and $d$, while negating $c$, we can perform recomputing with swapped operands (RESwO). The recomputed operands are $A' = -c + \omega d$ and $B' = -c - \omega d$. As shown in Figure 2.3, there is no necessity for decoding, and RESwO just requires comparison with alternate pre-recomputed values. The only negation unit in the scheme makes it inexpensive and efficient. We also present a modified variant of RESwO, i.e., RESwO-m in Figure 2.3, in which we lower the overhead by swapping just $\omega$ and $d$, having $c$ intact. This would result in even lower overhead as decoding would be free in hardware.
### 2.4 ASIC Assessments and Comparisons

The proposed error detection schemes are able to detect transient and permanent faults (intelligent attackers for intentional/malicious faults as well as natural defects). In this section, we present the results of our ASIC assessments using Synopsys Design Compiler and VHDL with TSMC 65-nm for three pairs of \((n, p)\) and two of our architectures to assess the overhead in Table 2.1. We have used Fermat primes in the form of \(1 + 2^i\) for \(i = 8, 16, 32\) which result in having \(\omega = 2\). Using 65-nm ASIC synthesis, and for three cases \((n, p)_1 = (64, 257), (n, p)_2 = (256, 65537),\) and \((n, p)_3 = (512, 4294967297)\), we also present the overhead of the presented constructions for the case studies of the proposed RESwO and RESwO-modified in this chapter. The benchmarking is performed for the error detection architectures (for two proposed schemes) and also for the original constructions, and overheads are shown in parentheses in Table 2.1. As shown in Table 2.1, the area [in terms of \(\mu m^2\)], delay (which is indication of maximum working frequency), and power consumption at the frequency of 50MHz are tabulated. The proposed schemes achieve acceptable overhead with very high error coverage. One would use RESwO if both permanent and transient faults in the entire architectures are to be detected. RESwO-modified has slightly less overhead and can detect transient faults in the structures.

We have performed simulations for (a) single, (b) two-bit, and (c) multiple-bit stuck-at-faults. For each experiment, more than 65,000 cases have been considered. From the results, we achieved that our schemes can detect these three cases with 100 percent error coverage. Further analysis shows that if the comparison units (i.e., voters) are compromised, the error detection scheme will degrade. Hardening the comparators, using triple modular redundancy and other fault tolerant techniques, can solve this faulty comparator status situation.

We would like to finalize this section by noting that the proposed architectures are oblivious of the standard-cell library and hardware platform. Therefore, we expect similar results on field-programmable gate array (FPGA) and ASIC libraries. We also note that the
Table 2.1: Implementation results for ASIC through TSMC 65-nm for three case studies, i.e., \((n, p)_1 = (64, 257), (n, p)_2 = (256, 65537),\) and \((n, p)_3 = (512, 4294967297),\) and two proposed architectures, i.e., recomputing with swapped operands-RESwO and its modified variant RESwO-modified (RESwO-m)

<table>
<thead>
<tr>
<th></th>
<th>Area ((\mu m^2))</th>
<th>Delay ((ns))</th>
<th>Power ((mW))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original ((n, p)_1)</td>
<td>2.942</td>
<td>12.24</td>
<td>0.047</td>
</tr>
<tr>
<td>RESwO ((n, p)_1)</td>
<td>3.674 (24%)</td>
<td>13.37 (9%)</td>
<td>0.054 (16%)</td>
</tr>
<tr>
<td>RESwO-m ((n, p)_1)</td>
<td>3.544 (20%)</td>
<td>13.19 (8%)</td>
<td>0.052 (12%)</td>
</tr>
<tr>
<td>Original ((n, p)_2)</td>
<td>8.995</td>
<td>13.80</td>
<td>0.093</td>
</tr>
<tr>
<td>RESwO ((n, p)_2)</td>
<td>11.170 (24%)</td>
<td>14.41 (4%)</td>
<td>0.111 (18%)</td>
</tr>
<tr>
<td>RESwO-m ((n, p)_2)</td>
<td>11.001 (22%)</td>
<td>14.23 (3%)</td>
<td>0.108 (16%)</td>
</tr>
<tr>
<td>Original ((n, p)_3)</td>
<td>30.829</td>
<td>14.76</td>
<td>0.207</td>
</tr>
<tr>
<td>RESwO ((n, p)_3)</td>
<td>37.476 (22%)</td>
<td>15.90 (8%)</td>
<td>0.231 (15%)</td>
</tr>
<tr>
<td>RESwO-m ((n, p)_3)</td>
<td>35.972 (17%)</td>
<td>15.45 (5%)</td>
<td>0.228 (11%)</td>
</tr>
</tbody>
</table>

throughput and frequency overhead can be alleviated through pipelining at the expense of added hardware overhead.

2.5 Conclusion

In this chapter, we have presented a number of categories for error detection schemes of NTT in the ring \(\mathbb{R} = \frac{\mathbb{Z}/p^q[x]}{x^n+1}\), which are also platform-oblivious. The proposed schemes constitute error detection architectures on hardware based on recomputing with encoded operands. Our target has been low hardware overhead, which is favorable to compact and deeply-embedded architectures. We have implemented the proposed error detection techniques on ASIC for a 65nm library to assess the implementation and performance metrics. With high error coverage, the presented approaches achieve acceptable overhead (at most
24% area, 18% power consumption, and 9% delay for the synthesized case studies) and can be tailored towards the objectives in terms of error detection and reliability.
Chapter 3: Error Detection Architectures for Ring Polynomial Multiplication
and Modular Reduction of Ring-LWE in \( \mathbb{Z}/p\mathbb{Z}[x]/x^n+1 \) Benchmarked on ASIC

3.1 Ring Polynomial Multiplication and Ring-Learning With Error

Lattice-based cryptography is popular for resistance against known quantum algorithms, as its security incorporates worst-case hardness of lattice problems [35]. Ideal lattices have revolutionized post-quantum cryptography by providing realizable execution, higher efficiency, and low parameter size. Learning with error (LWE) [21] is one of the most versatile worst-case lattice problems and allows us to completely pull out the lattice interpretation, resulting in an extremely-simple scheme. Ring learning with error (ring-LWE) [18] is one of the most explored and studied lattice-based cryptographic schemes, introducing even more efficient encryption scheme than the standard lattice problems [57], practically realizable and efficient for hardware implementation [58, 59], among post-quantum cryptosystems.

Ring-LWE emerges as a promising post-quantum cryptosystem to employ at limited-resource environments. Besides encryption and key generation, fully homomorphic encryption (FHE) [38] and somewhat homomorphic encryption (SHE) [39], two emerging groundbreaking techniques to secure cloud data, rely on ring-LWE for efficient and advanced operations.

Ring polynomial multiplication (RPM) is an integral part of a number of emerging post-quantum cryptographic algorithms and various non-cryptographic applications. RPM is the most rigorous computation for Ring-LWE, FHE, SHE, and a number of other cryptographic architectures. Thus, designing an efficient RPM architecture will certainly improve the performance of these state-of-the-art cryptosystems. RPM has versatile applications outside

---

2This chapter was published in the IEEE Transactions on Reliability [56] ©2020 IEEE
the cryptographic area. Erasure coding [60], a strategy to reconstruct corrupt data, uses RPM to ensure cost effectiveness and less complexity. Ensuring the privacy of electronic medical records [61] or multi-party communication [62], along with other applications [63–66], apply efficient realization of RPM. Consequently, a robust and efficient RPM will be much beneficial in terms of time and hardware complexities.

Ring-LWE involves addition and multiplication over a polynomial ring, where multiplication is the most rigorous operation and is computed using number theoretic transformation (NTT) [34], a robust and efficient construction [44–46], with smaller key lengths. Thus, efficient and fault-free modular multiplication of NTT is crucial to both high-speed and secure operation. Error detection architectures for both multiplication and modular reduction operations of NTT will enhance the security of current ring-LWE cryptosystems to a great scale.

Previous works have been performed on error detection schemes on several cryptosystems, see, for instance, [47, 48, 52, 67, 68]. The research in [52] focused on different aspects of tweakable enciphering schemes (TES), including implementations on hardware and software platforms, algorithmic security, and applicability to sensitive, security-constrained usage models on TES. The work in [47] challenged the traditional use of fault coverage for uniformly-distributed faults as a metric for evaluating the security of concurrent error detection (CED) against differential fault analysis (DFA). In [48], the security of logic encryption against side-channel attacks has been evaluated. The problem of exploitable fault characterization in the context of DFA attacks on block ciphers was addressed in [67]. The research work in [68] identified the weaknesses in the infection mechanism of the countermeasure that could be exploited by attacks which change the flow sequence. This research work proposes suitable randomization to reduce the success probabilities of attacks which change the flow sequence and develop a fault tolerant implementation of the countermeasure. While these works are based on classical cryptosystems, there exist some limited work on error detection for post-quantum cryptosystems. The major contribution of our work is that we apply error
detection schemes on post-quantum cryptosystems, unlike these previous works based on classical cryptosystems. Our error detection schemes are applied on ring polynomial multiplication and modular reduction. While the previous works have explored error detection on hash-based secure signature [69] and number-theoretic transformation of lattice-based cryptosystems [33], both of which are post-quantum cryptosystems, this work, for the first time, explores error detection schemes on RPM and modular reduction architectures, both integral to any lattice-based cryptosystems.

In this chapter, we propose error detection schemes of both RPM and modular reduction blocks, as different ring-LWE architectures use different moduli, depending on the security level and application. The main contributions of the chapter are as follows:

- We introduce error detection schemes for RPM with several “modulo $q$” architectures within the ring $R = \mathbb{Z}_q[x]/x^n+1$. Among the merits of proposed schemes is that they are platform-oblivious.
- The proposed error detection schemes are recomputing with shifted (RESO) and recomputing with swapped operands (RESwO). We apply both these schemes to different modulo $q$ architectures, where they could detect the faults injected with high error coverage.
- We also introduce error detection schemes for RPM architecture, recomputing with negated operands (RENO), a subset of RESo with different performance and implementation metrics and efficiency. These approaches add very little hardware overhead, which is advantageous to incorporate in deeply-embedded systems.
- The proposed error detection schemes are assessed and the results show acceptable error coverage. We implement our schemes on application-specific integrated circuit (ASIC), using Synopsys Design Compiler and a 65-nm standard-cell library, to derive the implementation and performance metrics.

The rest of the chapter is organized as follows. The next section recaps the theoretical background of ring polynomial multiplication technique and ring-LWE encryption. Section 3.3 and 3.4 discusses the proposed error detection schemes for ring polynomial multiplica-
tion and ring-LWE architectures, respectively. We summarize our hardware implementation results in Section 3.5. Section 3.6 draws conclusions to the chapter.

3.2 Preliminaries

3.2.1 Ring Polynomial Multiplication

In this chapter, we have considered polynomial in the ring \( \mathbb{R} = \frac{\mathbb{Z}[x]}{x^n+1} \). The irreducible polynomial inside this ring is represented as \( f(x) \) with degree of \( n \). Let two polynomials in this ring be \( a(x) \) and \( b(x) \). The multiplication of \( a(x) \) and \( b(x) \) is derived as:

\[
 a(x) \cdot b(x) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j x^{i+j} \mod f(x). \tag{3.1}
\]

Here, we use the case presented in [18]. \( f(x) \) is an irreducible polynomial where, \( f(x) = x^n + 1 \). Here, \( n \) is a power of 2, \( p \) is a prime number, and \( p \equiv 1 \mod 2n \). From the properties of irreducible polynomial, we can write \( x^n \equiv -1 \mod f(x) \). Using this value of \( x^n \) in (3.1), we derive the polynomial multiplication as:

\[
 c(x) = a(x) \cdot b(x)
 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (-1)^{\left\lfloor \frac{i+j}{n} \right\rfloor} a_i b_j x^{i+j \mod n} \mod f(x). \tag{3.2}
\]

3.2.2 Ring-LWE Encryption Scheme

Public-key encryption and signatures are essential for constructing lattice-based cryptosystems. Difficulty of Ring-LWE problems is the measure of their security, comparable to the worst case lattice problems [18]. Ring-LWE provides both encryption and portions of signature scheme of ideal lattices, within a short key space, resulting in faster algebraic operations. The cryptographic schemes of Ring-LWE problem perform addition and multiplication over \( R = \frac{\mathbb{Z}[x]}{x^n+1} \), and \( R_q = \frac{\mathbb{Z}_q[x]}{x^n+1} \), where \( q \) is a prime number and \( n \) is power of 2.
Such problems need one to decide whether the samples \((a_1, t_1), \ldots, (a_m, t_m) \in R_q \times R_q\) are chosen uniformly random, or each \(t_i = a_i s + e_i, \text{where}\) \(e_1, \ldots, e_m\) have small coefficients from the (one-dimensional) discrete Gaussian distribution \(D_\sigma\), with standard deviation \(\sigma\) and mean 0, to attain best entropy/standard deviation ratio \([59]\).

In the following, we describe the steps of the encryption scheme. The NTT of polynomial \(a\) is denoted as \(\tilde{a}\).

- **Key generation stage GEN(\(a\)):** Two error polynomials \(\tilde{r}_1\) and \(\tilde{r}_2\) are sampled from \(D_\sigma\) and let \(\tilde{p} = \tilde{r}_1 - \tilde{a} \tilde{r}_2 \in R_q\). The public key is the polynomial pair \((\tilde{a}, \tilde{p})\) and the secret key is \(\tilde{r}_2\).

- **Encryption stage ENC(\(\tilde{a}, \tilde{p}, M\)):** The input message \(M \in \{0, 1\}^n\), is encoded into a polynomial \(\tilde{M} = \text{encode}(M) \in R\), by multiplying each message bit by \(\lfloor (q/2) \rfloor\). The ciphertext can be obtained as \(\tilde{c}_1 = \tilde{a} \tilde{e}_1 + \tilde{e}_2\) and \(\tilde{c}_2 = \tilde{p} \tilde{e}_1 + \tilde{e}_3 + \tilde{M}\), where \(\tilde{e}_1, \tilde{e}_2\) and \(\tilde{e}_3 \in R\) are three error polynomials, sampled from \(D_\sigma\).

- **Decryption stage DEC(\(\tilde{c}_1, \tilde{c}_2, \tilde{r}_2\)):** Inverse NTT will recover \(\tilde{M}\) using \(\tilde{M} = \text{INTT}(\tilde{r}_2 \tilde{c}_1 + \tilde{c}_2)\). Decoding of \(M\) from \(\tilde{M}\) can be found elementwise, using following rule: if \(\tilde{M}[i] \in (-\lfloor(q/4), \lfloor(q/4)\rfloor)\), then \(M[i] = 0\), else \(M[i] = 1\), for \(0 < i < n - 1\).

A number of combinations of \((n, q, \sigma)\) have been explored in previous work. The research works in \([35]\) and \([18]\) have proposed \((256, 4093, 8.35)\) and \((214, 16381, 7.37)\) as medium and high-security parameter sets. Here, medium and high security correspond to the hardness of breaking an AES-128 and AES-256 bit block cipher, respectively. The works in \([59]\) and \([45]\) adopt the parameter sets to \((256, 7861, 11.31/\sqrt{2\pi})\) and \((512, 12289, 12.18/\sqrt{2\pi})\) as medium and high security parameters, compared to AES-128 and AES-256, respectively.

### 3.3 Proposed Error Detection Scheme for Ring Polynomial Multiplication

In this chapter, we present efficient error detection architectures for polynomial ring multiplication within ring \(\mathbb{R} = \mathbb{Z}[x]/p^2[x]/x^n + 1\). The proposed schemes can be applied to general
polynomials or operands, not confined to a subset or special cases of polynomials. Previous work in [70] has presented shift operation for the coefficients of one of the operands, as a countermeasure. This method perfectly worked for their model, where one of the operands of the RPM was ternary polynomial.

For general polynomial case, if we rewrite (3.2) in matrix form, the multiplication within $\mathbb{R} = \frac{\mathbb{Z}[x]}{x^n+1}$ can be expressed as:

$$
\begin{bmatrix}
    c_0 \\
    c_1 \\
    c_2 \\
    \vdots \\
    c_{n-1}
\end{bmatrix}
= 
\begin{bmatrix}
    a_0 & -a_{n-1} & \ldots & -a_1 \\
    a_1 & a_0 & \ldots & -a_2 \\
    a_2 & a_1 & \ldots & -a_3 \\
    \vdots & \vdots & \vdots & \vdots \\
    a_{n-1} & a_{n-2} & \ldots & a_0
\end{bmatrix}
\begin{bmatrix}
    b_0 \\
    b_1 \\
    b_2 \\
    \vdots \\
    b_{n-1}
\end{bmatrix}
$$

(3.3)

Shifting the coefficients of $a(x)$ produces a very complex circuitry, and decoding the shifted message is practically impossible with low overhead. As a result, we do not utilize shifting operation for general polynomial within $\mathbb{R} = \frac{\mathbb{Z}[x]}{x^n+1}$, although it worked smoothly for the case of [70].

Besides shifting, research in [70] has also applied checksum method as fault detection technique. Actual and predicted checksums are compared to verify if the data are intact. As one of the polynomials for ring multiplication in this research work is ternary polynomial, the checksum of one of the multiplication operands and that of an intermediate computation are theoretically equal. Nonetheless, for the proposed ring multiplication here, the following is derived for checksum $C_s$:

$$
C_s = \sum_{k=0}^{n-1} c_k = (a_0b_0 - a_{n-1}b_1 - a_{n-2}b_2 - \ldots - a_1b_{n-1}) + (a_1b_0 + a_0b_1 - a_{n-1}b_2 - \ldots - a_2b_{n-1}) + \ldots + (a_{n-1}b_0 + a_{n-2}b_1 + a_{n-3}b_2 + \ldots + a_0b_{n-1}) = a_0(b_0 + b_1 + b_2 + \ldots + b_{n-1}) + a_1(b_0 + b_1 + b_2 + \ldots + b_{n-1}) + a_2(b_0 + b_1 + \ldots - b_{n-2} - b_{n-1}) + \ldots + a_{n-1}(b_0 - b_1 - \ldots - b_{n-1}).
$$

23
Additionally, we have derived the interleaved checksum, where we add the even and odd coefficients of the product of multiplication. The results are given below, where \( \text{Int}_e \) and \( \text{Int}_o \) are even and odd interleaved checksums, respectively:

\[
\text{Int}_e = \sum_{k=0,2,4,...}^{n-1} c_k = (a_0 b_0 - a_{n-1} b_1 - a_{n-2} b_2 - ... - a_1 b_{n-1}) + (a_2 b_0 + a_1 b_1 + a_0 b_2 - ... - a_3 b_{n-1}) + ... + (a_{n-1} b_0 + a_{n-2} b_1 + a_{n-3} b_2 + ... + a_0 b_{n-1}) = a_0 (b_0 + b_2 + ... + b_{n-1}) + a_1 (b_1 + ... + b_{n-2} - b_{n-1}) + a_2 (b_0 + b_2 + ... - b_{n-2}) + ... + a_{n-1} (b_0 - b_1 - b_3 ... - b_{n-2})
\]

\[
\text{Int}_o = \sum_{k=1,3,5,...}^{n-1} c_k = (a_1 b_0 + a_0 b_1 - a_{n-1} b_2 - ... - a_2 b_{n-1}) + (a_3 b_0 + a_2 b_1 + a_1 b_2 - ... - a_4 b_{n-1}) + ... + (a_{n-2} b_0 + a_{n-3} b_1 + a_{n-4} b_2 + ... - a_{n-1} b_{n-1}) = a_0 (b_1 + b_3 + b_5 + ... + b_{n-2}) + a_1 (b_0 + b_2 + ... - b_{n-3}) + a_2 (b_1 + b_3 + ... - b_{n-1}) + ... + a_{n-1} (-b_2 - b_4 ... - b_{n-1})
\]

Both checksum and interleaved checksum will incur high area overhead, as there is no efficient approach that can minimize the cost of the circuit. The checksum presented in [70] can be applied to ring \( \mathbb{R} = \mathbb{Z}/p\mathbb{Z}[\alpha] \) however, it is not efficient for our ring \( \mathbb{R} = \mathbb{Z}/p\mathbb{Z}[\alpha] / \alpha^n + 1 \). Moreover, the checksum operation of convolution multiplication block in [70] requires no multiplication operation, whereas the checksum in our RPM architecture requires \( n \) modular multiplication units. Multiplication is an expensive operation that incurs high area overhead, which makes checksum an unsuitable scheme for RPM. Checksum will be an acceptable approach for high performance applications, where area overhead is not vital but delay is.

However, as our work is focused on embedded systems, area overhead is crucial. As a result, we introduce recomputing schemes which will provide us error detection with low cost.

\[
\begin{array}{cccccccc}
+ & a_0 b_0 & a_1 b_0 & a_2 b_0 & ... & a_{n-1} b_0 \\
+ & -a_{n-1} b_1 & a_0 b_1 & a_1 b_1 & ... & a_{n-2} b_1 \\
+ & -a_{n-2} b_2 & -a_{n-1} b_2 & a_0 b_2 & ... & a_{n-3} b_2 \\
& \vdots & \vdots & \vdots & \ddots & \vdots \\
+ & -a_1 b_{n-1} & -a_2 b_{n-1} & -a_3 b_{n-1} & ... & a_0 b_{n-1} \\
c_0 & c_1 & c_2 & ... & c_{n-1}
\end{array}
\]
3.3.1 Ring Polynomial Multiplication Architecture

In this chapter, we propose error detection schemes for the RPM within \( \mathbb{R} = \mathbb{Z}/p\mathbb{Z}[x]/x^n+1 \). However, our scheme is applicable to another polynomial ring multiplication construction, \( \mathbb{R} = \mathbb{Z}/p\mathbb{Z}[x]/x^n-1 \). The aforementioned multiplication can be expressed as following:

We utilize a multiplication (modulo \( p \)) circuit to compute \( a(x) \cdot b(x) \). The coefficients of element-wise multiplications are either positive and negative as shown through preceding partial products. We can explain this using (3.2), where the term \((-1)^{\lfloor \frac{i+j}{n} \rfloor} \) decides whether the coefficients are positive or negative. When \( i+j < n \), \( \lfloor \frac{i+j}{n} \rfloor = 0 \), then \((-1)^0 = 1\), making the coefficients positive. On the contrary, when \( i+j \geq n \), \( \lfloor \frac{i+j}{n} \rfloor = 1 \), then \((-1)^1 = -1\), and the coefficients are negative. The function is given as follows:

\[
(-1)^{\lfloor \frac{i+j}{n} \rfloor} = \begin{cases} 
1 & \text{when } i+j < n \text{ or } \lfloor \frac{i+j}{n} \rfloor = 0, \\
-1 & \text{when } i+j \geq n \text{ or } \lfloor \frac{i+j}{n} \rfloor = 1.
\end{cases} \tag{3.4}
\]

We require a module capable of performing both the addition and subtraction of two operands. To achieve that, we use a multiplexing adder/subtractor unit. The selector of multiplexer, \( Sel \), is basically the term \( \lfloor \frac{i+j}{n} \rfloor \). According to (3.4), the module acts as a mod \( p \) adder and as a mod \( p \) subtractor when \( Sel \), i.e., \( \lfloor \frac{i+j}{n} \rfloor \), is 0 and 1, respectively. We decide from Figure 3.1, each box computes one coefficient of \( c(x) \), which takes \( n \) cycles. Therefore, we get the final result of the computation in \( n \) cycles, as each coefficient is computed in parallel.

In our schemes, we utilize the aforementioned multiplication module to ensure smooth operation. At first, we apply recomputing with scaled operands and compare the decoded product of multiplication with the output.

Afterwards, as an economical and low-power subset of scaling, we recompute the multiplication by negating one or both of the operands and compare the decoded message with the output. The latter approach adds very little area overhead, utilizing RPM for both the
rings efficiently. Depending on objectives for error coverage and overhead, our schemes can be tailored to negate as well as scale one or both of the operands of multiplication, for a high error coverage error detection scheme.

3.3.2 Proposed Error Detection Scheme through Recomputing

Error detection codes might be generally inefficient and expensive for general polynomial ring multiplication. With a view to making such detection schemes faster and cheaper, we have utilized a recomputing method that scales one or both operands of multiplication as encoding operation. Figure 3.2 describes REScO, which is a modified architecture of RPM, where we insert a multiplexer, a multiplier, and dividers. The selector of the multiplexer, Normal/REScO, determines whether it performs original RPM or REScO, respectively. In the latter case, one of the operands, e.g., $b$, is scaled with a factor $k$ and we get the encoded operand, $e(x) = k \cdot a(x) \cdot b(x)$. For the decoding process, we have to apply multiplicative inversion mod $p$ of the factor, $k$. Thus, we have to select $k$, carefully, to avoid cases of the non-existing multiplicative inverse. To achieve that goal, $k$ has to be a non-zero integer where $gcd(k, p) = 1$. For example, if $p = 128$, we cannot use $k = 2$, as the $gcd(2, 128) \neq 1$. On the contrary, $k = 3$ can be easily inverted in mod $p$, as $gcd(3, 128) = 1$. For prime $p,$
we do not have this restriction. As shown in Figure 3.2, we apply modular division to each coefficient of $e(x)$ by $k$, and get the decoded output, $d(x) \equiv k^{-1} \cdot e(x) \mod p$.

In terms of error coverage, REScO is effective in countering faults. However, it incurs substantial hardware overhead from the multiplier and divider modules, required during encoding and decoding stages. Specifically, we need to provide $n$ number of dividers while decoding. Division is a costly arithmetic operation, and inserting many of dividers in the implementation makes it expensive, power consuming, and slow. To solve this situation, we explore a more efficient and low power subset of REScO, i.e., RENO. In RENO, we recompute by negating one or both of the operands of multiplication. Negating can be inferred as multiplying with $-1$; hence, RENO is a special case of REScO. The hardware architecture of this scheme (Figure 3.3) is very similar to the original RPM design. During encoding, the select of the additional multiplexer chooses between normal or RENO operation. In
the case of RENO, the coefficients of the operands, given by (3.4), are swapped, using an inverter. Mathematically, \( i + j < n, \left\lfloor \frac{i + j}{n} \right\rfloor = 0 \) becomes 1 after inversion, \((-1)^1 = -1\), making the coefficients negative and vice versa. Therefore, we get the encoded operand \( e(x) \) as, \( e(x) = -a(x) \cdot b(x) \). During the decode stage, we need to find additive inverse of each coefficient of \( e(x) \) mod \( p \). Let any of the coefficients of decoded output \( d(x) \) be \( d_i \), where, \( d_i \equiv (p - e_i) \mod p \equiv -e_i \mod p \). We use \( n \) number of subtractors in this stage of RENO.

As adder/subtractor modules are inexpensive, compared to multiplier and divider modules of REScO, the structure in Figure 3.3 which derived the decoded results, i.e., \( (p - e_i) \mod p \), is not as costly as general subtractors. This is because one of the inputs is always fixed, i.e., \( p \), which simplifies the architecture. Thus, RENO provides an efficient and low overhead error detection method. We would like to emphasize that, REScO and RENO are not two
different techniques. RENO is a subset of REScO, where we are scaling the operands with $-1$.

The architecture of RENO in Figure 3.3 can detect transient faults correctly. However, this architecture can only detect permanent faults present in the multiplexing adder/subtractor module, while failing to detect such faults in the operands or in any other section of the architecture. To resolve this issue, we introduce modification Figure 3.4 and negate one of the operands since the beginning of the computation. The Normal/RENO multiplexer will now select normal operand $b_j$ or negated operand $(p - b_j)$. As a result, the operands are negated at the input stage, and that enables this scheme to detect both permanent and transient faults in operands as well as entire architecture satisfactorily. Moreover, the structure in Figure 3.4 reduces the hardware overhead by removing the $(p - e_i)$ mod $p$ box and adding two multiplexers before the feedback structure. Using the select Enc/Dec, the multiplexers either perform encoding by element-wise multiplication of $a$ and negated $b$, i.e., $(p - b_j)$, or subtract $e_i$ from $p$, giving the decoded output $d_i \equiv (p - e_i) \mod p$. We perform computations on the operands in two runs: First run, i.e., $run_1$, deals with normal computation, and second run, i.e., $run_2$, deals with RENO. In both Figures 3.4a and 3.4b, thickened lines and bold texts represent the multiplexer paths for $run_1$ and $run_2$, accordingly. We use the selector Enc throughout $run_1$ and the first $n$ clock cycles of $run_2$. On the other hand, Dec is selected only in the $(n + 1)$th cycle of RENO, in order to complete the decoding of negated operands. In such manner, we eliminate $n$ number of subtractors by performing the $(p - e_i)$ operation through the already existing adder/subtractor modules. As the hardware overheads of multiplexers are considerably lower than these modules, modified RENO costs even less than regular RENO, while ensuring higher error coverage. One can modify RENO by negating both of the operands. In this case, the input operands are $(p - a_i)$ and $(p - b_i)$, instead of $a$ and $b$. As multiplication of two negative terms gives a positive result, there is no need for decoding. Negating both input operands requires more involved encoding and
(a) run1, the select of top multiplexer is Normal

(b) run2, the select of top multiplexer is RENO

Figure 3.4: Hardware architecture of modified RENO.
hardware overhead (as seen in the next section on ASIC). Nevertheless, in some platforms, absence of decoding stage might compensate for this excess circuitry.

Our proposed schemes can detect transient faults close to 100% error coverage (one needs to harden the comparison logic to achieve higher error coverage). RENO also provides close to 100% error coverage for permanent and long transient faults. We utilize an error detection flag, which is logic OR operation of comparisons for every column. Even if only one of the columns of Figure 3.4 has erroneous output, the flag will be set to 1 and we can detect the error.

Two unlikely cases may appear during assertion of permanent or long transient faults. One event can be “masking”, in which the output is not erroneous, even if a fault exists in the intermediate logic. Such cases are excluded because the circuit masks the faults and these are not translated to errors. The second instance is a rare case where all the entries of operands $a_i$ and $b_i$ are zero. RENO cannot detect these errors, because negating any zero value will keep it unaltered. However, applying all the input bits to a logic OR gate can be secondary measure to detect such case. We would like to emphasize that this would be equivalent to multiplying two zero polynomials which is an unlikely case.

3.3.3 Ameliorating the Throughput Overhead through Pipelining

The delay overhead we took into account is the critical path delay, where critical path is the path that incurs the highest delay. As our error detection is a time redundancy technique, the total time of a recomputed architecture will be twice of an original architecture deteriorating the throughput, if no measure is in place to compensate such shortcoming. Such absence of pipelining will degrade the throughput drastically, which can be improved by applying subpipelining. Subpipelining will increase the frequency to make sure the design throughput is close to that of the original architecture. This will incur slightly higher area overhead, which can be overlooked as we are achieving low throughput degradation of the error detection approach. We insert registers in locations which will in turn break the timing
paths into approximately equal halves. We denote the two halves of the pipelined stages as $H_1$ and $H_2$. According to Figure 3.5, our scheduling order of normal ($N_i$) and recomputed ($R_i$) operations are shown, where $1 \leq i \leq n$, $n$ being the number of cycles in original non-pipelined approach. We compute $R_i$ and $N_i$ at the same cycle but in different pipelined stages, whereas in the next cycle, $N_{i+1}$ and $R_i$ are computed.

3.4 Proposed Error Detection Schemes for Ring-LWE Architecture

To construct the ring-LWE encryption architecture, based on preliminaries presented in this chapter, we utilize DSP-enabled schoolbook polynomial multiplier, along with modular reduction block. Here, we emphasize on two sets of parameters, i.e., $(n, q, \sigma) = (214, 16381, 7.37)$ and $(512, 12289, 12.18/\sqrt{2\pi})$, both being high security parameters. Resemblance between the reduction method of $(n, q, \sigma) = (214, 16381, 7.37)$ and $(256, 4093, 8.35)$, makes our scheme easily modifiable to apply to the other parameter sets [71]. On the other hand, $(n, q, \sigma) = (256, 7681, 11.31/\sqrt{2\pi})$ and $(512, 12289, 12.18/\sqrt{2\pi})$ both use SAMS2 technique for modular reduction in the research works of [45] and [72]. Thus, our error detection scheme presented through such parameter sets, is also applicable to the former.

Choosing the proper value of $q$ varies upon level of security, efficient modular reduction and based on the property of the modulus, e.g., Fermat number or a large prime number. This work, for the first time, explores error detection schemes within modular operations. Subsection 3.3.2 introduced error detection schemes using recomputing for multiplication operation, i.e., RPM. In the following, i.e., Subsections 3.4.1 and 3.4.2, we explore error
detection schemes for modular reduction operations. In Figure 3.3, we have seen mod \( p \) block, where we can apply our modular reduction operations, based on the value of \( p \). Our error detection schemes on modular reduction can be used in any compatible architecture, not being limited to RPM only.

3.4.1 Error Detection Scheme for Polynomial Multiplier and \( q=16381 \)

In this construction, we use a DSP-based schoolbook polynomial multiplication scheme, followed by the modulo \( q \) operation. For \( q = 16381 \), it is found that \( 2^{14} \mod 16381 = 3 \). As a result, the inputs of the DSP blocks are 14 bits in length, and the product can be written as: \( x_{27...0} = 2^{14}x_{27...14} + x_{13...0} = 3x_{27...14} + x_{13...0} = (x_{27...14} << 1) + (x_{27...14}) + x_{13...0} \), where left shift is denoted by \( << \). The modular operation reduces the result within \([0, 16380]\), requiring two modulo \( q \) operations at most, which is performed by the modulo \( q \) reducer block of Figure 3.6. On the other hand, the DSP block computes the unsigned multiplication through \((AB + C)\). In the case of signed multiplication, i.e., multiplication with a negative number, \((D - A)B + C\) is performed, where \( D = q \).

In this section, we propose two variants of recomputing schemes, which we apply to the most rigorous computation of ring-LWE encryption operation, i.e., the entire DSP as well as modular \( q \) reducer block. Figure 3.6 shows recomputing with shifted operands (RESO), in which two of the input operands are shifted to left by 1 bit, which is multiplication by 2 in binary operation. Another approach is recomputing with swapped operands (RESwO), where two of the input operands are swapped. In the former approach, we insert a multiplexer that controls either normal mode (Norm) or RESO mode (RESO) of operation through the select pin Norm/RESO. In a Norm operation, we get the usual multiplier output with mod \( q \) reduction. Whereas, RESO mode performs left shift of both operands \( A \) and \( C \), giving the multiplier output as \( 2AB + 2C = 2(AB + C) \). The output of the modulo \( q \) block is shifted to the right by 1 bit, which will provide \( AB + C \), in fault-free scenario. The outputs of both the rounds are compared and any discrepancy between the results detect the presence of faults.
in the architecture. RESwO can also be applied in a similar manner, which will detect both permanent and transient faults with less overhead.

### 3.4.2 Error Detection Scheme for SAMS2 Approach and \( q=12289 \)

Modular reduction operations for a number of values are computationally less efficient than the former values we explored, yet such values of \( q \) are famous and widely used in SHE and other cryptographic applications. The works of [45, 72] apply the values of \( q = 7681 \) and 12289, and use shift-addition-multiplication-subtraction-subtraction (SAMS2) for faster modular reduction operation.
In Figure 3.7, we explain the Norm mode for $q = 12289$, where the input contains 14 bits, as $2^{14} = 2^{12} - 1 \mod 12289$. In Shift-Add block (Figure 3.8a), we approximate quotient $t$ of $x_{out} = x - tq$ as $x \gg 14 + x \gg 16 + x \gg 18 + x \gg 20 + x \gg 22 + x \gg 24 + x \gg 26$, which is a combination of shift and addition operation, based on [72]. From this value of $t$, we use $Multq$ block (Figure 3.8b) to find the product $tq$. However, the $Multq$ block is a combination of left-shifts and addition, resulting in a much efficient scheme, compared to a multiplier. In the last block, i.e., $Subt.$, the subtraction between $xin$ and multiples of $q$ from $q$ to $7q$ are performed in parallel (Figure 3.9), providing a much faster reduction due to simultaneous calculations. Taking the least positive number between $xin - tq$ and the results of the above subtractions, the $Subt.$ block works in a loop until the output is not lower than $q$.

In this chapter, we present the error detection architecture of SAMS2 operation, through the RESO mode of the multiplier (Figure 3.7). We apply recomputing with shifted operands as encoding and decoding operations of the input and output, respectively. As the entire SAMS2 operation is linear, applying a left shift at the input stage, and a right shift at the output stage should retain the same $x_{out}$ as Norm mode. Thus, comparing the values of both rounds of operations provides us the error detection if both values of $x_{out}$ fail to coincide.
3.5 Error Coverage and ASIC

3.5.1 Fault Model

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 subkey 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). We note that 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.
We note that such fault models consider both malicious faults and also natural faults based on stuck-at faults considered in this chapter.

### 3.5.2 Assessments

In this section, we present the results of our error simulations and ASIC assessments using Synopsys Design Compiler and VHDL with TSMC 65-nm for two security levels and two of our architectures to assess the overhead. Using 65-nm ASIC synthesis, we also present the overhead of the presented constructions for the case studies of moderate and high security levels, i.e., for \((n = 256, \ p = 1049089)\), and \((n = 512, \ p = 4206593)\), respectively. We have also chosen third set of parameters \((n = 1024, \ p = 536903681)\), based on SHE [39]. The benchmarking is performed for the error detection architectures (for two proposed schemes, i.e., Prop. 1: Negating both operands and Prop. 2: Negating one operand, respectively) and also for the original construction.
3.5.3 Fault Simulations

We evaluated the error detection capability of the proposed work based on fault-injection simulation coded in VHDL. We injected three types of stuck-at faults, i.e., (a) single, (b) two-bit, and (c) multiple-bit faults for over $10^{12}$ cases, all injected at the input state of the algorithm. An attacker may not be successful at flipping exactly one bit to collect sensitive information due to technological constraints, which led us to consider multiple stuck-at faults. The faults that we consider are stuck at 0 and stuck at 1. The schemes provide high error coverage (reservation on the comparator is explained below) for these three cases. The simulation results are confirming that the schemes can detect both transient and permanent faults satisfactorily. We assume the comparators are hardened, i.e., the comparators are fault free and not compromised.

In cryptographic engineering, reliability of a cryptosystem is the measure of its ability to thwart malicious fault and, in this work, natural stuck-at faults. The higher the accuracy an error detection scheme can provide, the higher is its reliability. As our schemes provide high error coverage, our schemes are relatively-reliable. To further elaborate, let us explore the rate of missed detection, also known as false negative rate ($FNR$), which is 0.001%. This can be explained by masking of errors, where, the presence of one defect hides the presence of another defect. Our schemes have no false positives (false alarms), as such cases are relevant where error detection is done in mid-stages of error detection constructions, where detected faults are masked. All faults are correctly detected and there has not been a case where a fault-free condition was flagged as a faulty one, resulting in zero false positive ($FP$) detection and zero percent false positive rate ($FPR$) of our schemes. We can deduce the true positive rate ($TPR$) of our schemes from $TPR = 1 - FNR$, to be 99.999%. Moreover, our schemes have full coverage for the ratio of the number of true positives to the number of all positives, i.e., $precision = \frac{TP}{TP+FP}$, where $FP$ is explained before and $TP$ is the total number of true positive detection.
Sensitivity is the measure to correctly detect faults where faults are actually injected, i.e., \( \text{sensitivity} = \frac{TPR}{TPR+FNR} \). Our proposed schemes are highly sensitive, with close to 100% sensitivity, using the above-mentioned values of \( TPR \) and \( FNR \). To find the receiver operating characteristics (ROC), one has to plot \( TPR \) against \( FPR \) with respect to varying threshold values. As the output of comparators shows either a high or low flag for fault and fault-free output, respectively, we do not require various values of threshold in our scenario. Considering the cases in which the comparison units are hardened, the resulting ROC curve is almost a vertical graph as the horizontal axis denotes \( FPR \) and vertical axis denotes \( TPR \), the values of which are presented before.

3.5.4 ASIC Comparison for Error Detection in RPM Module

As shown in Table 3.1, the area [in terms of \( \mu m^2 \) which can be converted to kilo gate equivalent (kGE) which is the normalized area for 2-input NAND gate by dividing the column numbers by \( 1.41 \times 10^3 \)], delay (which is indication of maximum working frequency), and power consumption at the frequency of 20 MHz are tabulated. The original architecture denotes where no error detection schemes were applied. The proposed schemes achieve acceptable overhead compared to the original architecture, with very high error coverage. We also note that negating one of the input operands requires more involved encoding and hardware overhead, compared to negation both input operation.

Here, we note that the RENO operation in Subsection 3.3.2 and RESO operation in Subsections 3.4.1 and 3.4.2 are compatible. We explored the efficient schemes in each of the architectures. RENO is computationally more efficient than RESO, in case of RPM architecture, as only one mod \( p \) negation block suffices the RENO operation. On the other hand, to apply RESO on RPM, we require to add left shift blocks to all of the input operands, which will require \( (n+1) \) left shift blocks at the input as we are feeding the coefficients of \( a \) in parallel, and \( n \) right shift block at the output. Thus, RESO incurs much higher area overhead than RENO, making RENO a better recomputing scheme for RPM. However,
Table 3.1: Implementation results for ASIC TSMC 65-nm of RPM architecture (Prop. 1: Negating both operands, Prop. 2: Negating one operand)

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Area ($\mu m^2)$</th>
<th>Delay(ns)/ Frequency</th>
<th>Power (mW) at 20MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original ($n = 256, p = 1049089$)</td>
<td>260,055 (184 kGE)</td>
<td>25.2 (39.6 MHz)</td>
<td>5.3</td>
</tr>
<tr>
<td>Original ($n = 512, p = 4206593$)</td>
<td>539,766 (382 kGE)</td>
<td>26.5 (37.7 MHz)</td>
<td>10.7</td>
</tr>
<tr>
<td>Original ($n = 1024, p = 536903681$)</td>
<td>1,097,602 (778 kGE)</td>
<td>28.0 (35.7 MHz)</td>
<td>23.9</td>
</tr>
<tr>
<td>Prop. 1 ($n = 256, p = 1049089$)</td>
<td>290,846 (11.5%) (206 kGE)</td>
<td>33.9 (34.5%) (29.5 MHz)</td>
<td>5.9 (11.3%)</td>
</tr>
<tr>
<td>Prop. 1 ($n = 512, p = 4206593$)</td>
<td>589,111 (11.1%) (417 kGE)</td>
<td>34.7 (32.8%) (28.8 MHz)</td>
<td>12.8 (19.6%)</td>
</tr>
<tr>
<td>Prop. 1 ($n = 1024, p = 536903681$)</td>
<td>1,254,009 (14.3%) (889 kGE)</td>
<td>36.3 ns (28.6%) (27.5 MHz)</td>
<td>26.3 mW (10.1%)</td>
</tr>
<tr>
<td>Prop. 2 ($n = 256, p = 1049089$)</td>
<td>311,056 (19.6%) (220 kGE)</td>
<td>28.6 ns (13.5%) (34.9 MHz)</td>
<td>6.1 (15.1%)</td>
</tr>
<tr>
<td>Prop. 2 ($n = 512, p = 4206593$)</td>
<td>608,223 (12.7%) (431 kGE)</td>
<td>29.5 ns (11.3%) (33.8 MHz)</td>
<td>12.4 mW (15.8%)</td>
</tr>
<tr>
<td>Prop. 2 ($n = 1024, p = 536903681$)</td>
<td>1,262,960 (19.6%) (895 kGE)</td>
<td>33.8 (20.7%) (29.6 MHz)</td>
<td>26.7 (11.7%)</td>
</tr>
</tbody>
</table>

applying RENO in the modular reduction operation further complicate the mod $q$ negation inside the modulo $q$ reducer block for Figure 3.6 and Subt. block for Figure 3.7, as they are already a series of negation operation. Consequently, adding another mod $q$ negation will result in a discrepancy in the reduction operation. Moreover, to apply RESO, we only use two left shift blocks and one right shift block in Figure 3.6, which is much cheaper than mod $q$ negation units. We note that the results of our ASIC analysis can be compared with theoretical analysis of overhead which leads to justifications on such results. The area overhead would include those of comparators and added registers; thus, the overhead is not unacceptable as seen in the table (this holds for power overhead as well). The delay overhead is not high but as we have explained the throughput degradation is not negligible and pipelining would ameliorate such degradation.
We would like to finalize this section by noting that the proposed architectures are oblivious of the standard-cell library and hardware platform. Therefore, we expect similar results on field-programmable gate array (FPGA) and ASIC libraries. We also note that the throughput and frequency overhead can be alleviated through pipelining at the expense of added hardware overhead. Differential fault intensity analysis (DFIA) which is a combination of differential power analysis and fault injection concepts has gained much attention in the recent past. The biased fault models range from low intensity to higher ones in previous works. Our aforementioned proposed fault detection schemes have capabilities to detect these biased faults. Finally, previous studies of [70] introduced efficient countermeasures against fault attacks on NTRU-Encrypt. However, RPM in NTRU-Encrypt [70] is a special case and it is not applicable for encrypting other systems, for example, the ring-LWE in [44]. Moreover, we do not utilize shifting operation for general polynomial within $\mathbb{R} = \mathbb{Z}/p\mathbb{Z}[x]/x^n + 1$, although it worked smoothly for the case of [70]. We also note that the presented error detection schemes in this chapter for RPM in the ring $\mathbb{R} = \mathbb{Z}/p\mathbb{Z}[x]/x^n + 1$ are not confined to this ring and can be incorporated into a number of other constructions, such as the ring $\mathbb{R} = \mathbb{Z}/p\mathbb{Z}[x]/x^n - 1$.

3.6 Conclusion

In this chapter, we have proposed efficient error detection schemes, i.e., REScO and RENO a subset of REScO with different performance and implementation metrics and efficiency. Additionally, we employ RESO and RESwO, to a number of ring-LWE architectures and modular reduction stages, which can be applied to most well-known modulo operations. These approaches add very little hardware overheads, which is advantageous to incorporate in deeply-embedded systems. We have benchmarked the proposed architectures to assess their ability to detect transient and permanent faults. Moreover, we have implemented the proposed error detection architectures on ASIC and our results show that the proposed efficient error detection architectures can be feasibly utilized for RPM in the rings $\mathbb{R} = \mathbb{Z}/p\mathbb{Z}[x]/x^n + 1$ and $\mathbb{R} = \mathbb{Z}/p\mathbb{Z}[x]/x^n - 1$.
\[ R = \frac{z/pz[p]}{x^n - 1} \]. We note that our scheme is suitable for the required performance, reliability, and implementation metrics for constrained applications.
Chapter 4: Fault Detection Architectures for Inverted Binary Ring-LWE Construction Benchmarked on FPGA

4.1 Inverted Binary Ring-LWE

Lattice-based cryptography has revolutionized post-quantum cryptography (PQC) through realizable execution, efficiency, and low parameter size. Learning with errors (LWE) is a highly-explored worst-case lattice problem and provides an efficient scheme. Ring learning with errors (RLWE) is a family of assumptions which lead to one of the most versatile encryption schemes, compared to the standard lattice problems. A new variant of RLWE is proposed in the research presented in [75], involving a binary distribution to choose binary coefficients instead of Gaussian, namely, Ring-BinLWE. A hardware-optimized scheme of Ring-BinLWE proposed in [76] utilizes an inverted ring of Ring-BinLWE (InvRBLWE) and 2’s-complement notation range.

In this chapter, we introduce fault detection constructions on Ring-BinLWE architecture, which can be tailored based on the needs in terms of reliability and the restrictions in terms of the added overhead in constrained applications. Past research works have been performed for fault detection schemes on several cryptosystems [25, 68, 77–81]. These include research works on different public and symmetric-key cryptosystems, and are mainly based on error-detecting codes on classical cryptosystems. Very few works exist on fault detection of PQC, e.g., hash-based secure signature [50], the number-theoretic transformation of lattice-based cryptosystems [33], and ring polynomial multiplication of RLWE [56]. Some examples for error detection in general computations and classical cryptography exist as well [82, 83].

The main contributions of this work are as follows:

3This chapter was published in the IEEE Transactions on Circuits and Systems II [74] ©2021 IEEE
• We devise architectures for key-generation and encryption of Ring-BinLWE problem. The construction clarifies the gate-level architectures of these two stages and supports the validity of the augmented fault detection modules.

• We introduce fault detection schemes for Ring-BinLWE within the ring \( R = \frac{\mathbb{Z}[x]}{x^n+1} \), for all three phases, i.e., key generation, encryption, and decryption. The proposed fault detection schemes are based on encoding, recomputing, and decoding the operands. We apply these schemes to three stages of InvRBLWE architecture, which can be tailored to apply on other RLWE architectures as well.

• The assessed results of the proposed schemes show acceptable error coverage. To assess the overhead, we implement the proposed schemes on a Xilinx field-programmable gate array (FPGA) family.

### 4.2 Preliminaries

RLWE provides both encryption and portions of the signature scheme of ideal lattices, within a short keyspace, resulting in faster algebraic operations. The cryptographic schemes of RLWE problem perform addition and multiplication over \( R = \frac{\mathbb{Z}[x]}{x^n+1} \), and \( R_q = \frac{\mathbb{Z}[x]}{x^n+1} \), where \( q \) is a prime number and \( n \) is power of 2. Using \( x^n + 1 \) as modulus leverages the efficiency during implementation of anti-circular rotation through shift operation. Among multiple variants of RLWE, the work in [84] proposes binary error distribution instead of the Gaussian, namely, Ring-BinLWE, which led to smaller key and ciphertext sizes and no expensive computations of Gaussian distributions. Moreover, another improvement on Ring-BinLWE was achieved in [76] using 2’s complement notation of the coefficients, namely, InvRBLWE, by selecting the range of \( R_q = \frac{\mathbb{Z}[x]}{x^n+1} = (-\left \lfloor \frac{q}{2} \right \rfloor, \left \lfloor \frac{q}{2} \right \rfloor - 1) \) and eliminating the need for modular reduction. In the following, we describe the steps for InvRBLWE problem.

• Key Generation stage GEN\((a)\): Let us assume two error polynomials \( r_1, r_2 \in \{0, 1\}^n \) and let \( p = r_1 - ar_2 \in R_q \). The public key is the polynomial pair \((a, p)\in R_q\) and the secret key is \( r_2 \).
• Encryption stage ENC\((a, p, m)\): The input message \(m \in \{0, 1\}^n\) is encoded into a polynomial \(\tilde{m} = \text{encode}(m) \in R_q\), where encode is defined as follows:

\[
(m_0, m_1, \ldots, m_{n-1}) \rightarrow \sum_{i=0}^{n-1} m_i(-\frac{q}{2})x^i
\]  

The ciphertext can be obtained as \(c_1 = ae_1 + e_2\) and \(c_2 = pe_1 + e_3 + \tilde{m}\), where \(e_1, e_2\) and \(e_3 \in R_q\) are three error polynomials, sampled from \(\{0, 1\}^n\).

• Decryption stage DEC\((c_1, c_2, r_2)\): To recover \(m\) from \(\tilde{m}\), first \(\tilde{m} = c_1r_2 + c_2\) is computed. Decoding of \(m\) from \(\tilde{m}\) can be performed using the following decode function:

\[
\text{DECODE} : R_q \rightarrow \{0, 1\}^n
\]

\[
\sum_{i=0}^{n-1} a_i x^i \rightarrow (m_0, m_1, \ldots, m_{n-1})
\]

\[
m_i = \begin{cases} 
0 & \text{when } |a_i - i - \lfloor \frac{n-3}{2} \rfloor| > \frac{q}{4} \\
1 & \text{else.}
\end{cases}
\]  

4.3 Proposed Fault Detection Schemes

From most recent attack [81], we get 73/84 bits and 140/190 bits of quantum/classical security from the parameter sets of \((n, q) = (256, 256)\) and \((512, 256)\), respectively. Our schemes are applicable to both security levels and we apply recomputing schemes on three stages of InvRBLWE. Our motivation is to achieve low-complexity schemes; thus, we ensure that the augmented fault detection schemes lead to acceptable overhead, compared to the original architecture.
4.3.1 Recomputing with Encoded (Shifted) Operands

In this chapter, we adopt shifting the operands by doubling the inputs and dividing the outputs by 2, which can be interpreted as shifting the input to the left and right one place in binary, respectively.

4.3.1.1 Key Generation

The multiplexer select input, Norm/RESO, shown in Figure 4.1, determines whether the original or the recomputed operation (denoted as recomputing with shifted operands (RESO)) will be performed. During Norm/RESO=0, i.e., the original operation, the NAND gate produces $\overline{a.r_2}$, while the left adder of the top block, completes the 2’s complement of $a.r_2$ by adding 1 and produces $-a.r_2$. The right adder input is either $-a.r_2$ or $r_1$ during multiplexer select $S1=0$ and 1, respectively. The anti-circular rotation is implemented in hardware by adding the registers $Res[i]$ to the next adder, and the negative of $Res[n-1]$ to the right adder of the top block. The architecture performs multiplication when the control signal $S1$ is set to zero, through the shift-and-add method, requiring $n$ parallel adders of 8 bits. In such a cycle, all the adders, except the top one, performs add operation to find the product of $a$ and $r_2$. A shift register feeds each bit of $r_1$, $r_2$, namely, $r_1[i]$, $r_2[i]$, during each clock cycle of multiplication, while $r_1, r_2 \in \{0, 1\}^n$. Each bit of n-bit length vector, $r_1$ and $r_2$ is extended as 8-bit ($\log_2 q$) as the results are stored in registers of 8-bit length. Such notation, using the index i, e.g., $r_2[i]$, has been used throughout the chapter, representing each bit of binary vector being stretched to 8-bit using a shift register to maintain consistency. During $run_2$, i.e., the recomputed operation, we multiply $a$ and $r_1$ with 2, which can be represented as each being left shifted one place and the output being $Sub_{run2} = 2(r_1 - ar_2)$. The left shift explains the size of the $a$ and $r_2$ becoming 9 bits in RESO operation, instead of 8 in the Norm cycle. Afterward, to compute the decoded operands, we discard the least significant bit of the output. In Figure 4.1 and subsequent figures, the gray-colored box represents the original architecture, whereas the components outside the box, represent the fault detection
modules. For example, the multiplexers, the shifters, and the comparator modules outside the gray-colored box in Figure 4.1 are our added circuitry for fault detection.

4.3.1.2 Encryption

The encryption operations provide two outputs, $c_1$ and $c_2$. Based on Figure 4.2a, the output $c_1$ can be computed using logic circuitry similar to that of key generation. The original architecture requires multiplication of $a$ and $e_1$, which is performed during the $S1=0$ cycle of the multiplexer. The addition is complete through multiplexer when $S1=1$. The anti-circular rotation is performed as described above. In order to perform recomputing on $c_1$, we set multiplexer select Norm/RESO to 1 for RESO operation. During the encoding, the output of Figure 4.2a adders provide $Add_{run2} = 2(ae_1 + e_2)$. We extract the most significant 8
Figure 4.2: Hardware construction of recomputing with shifted operands for encryption of InvRBLWE.
Figure 4.3: Hardware construction of recomputing with shifted operands for decryption of InvRBLWE.

bits of the output and compare it with the Norm cycle output. To construct the architecture computing \( c_2 = pe_1 + e_3 + \overline{m} \), we assume the \( \overline{m} \) is pre-computed from (4.1). According to Figure 4.2b, during multiplexer select \( S1=0 \), we multiply the \( p \) and \( e_1 \), then during \( S1=1 \), the addition of \( e_3 \) and \( \overline{m} \) is performed. During RESO run, we encode twice of \( c_2 \) by shifting \( p \), \( e_3 \), and \( \overline{m} \) one place to left each, which gives us the encoded output, \( Add_{run2} = 2(pe_1 + e_3 + \overline{m}) \).

The decoding operation halves the output, which is then compared with the Norm cycle output to detect the presence of any faults.

4.3.1.3 Decryption

The decryption computes \( \overline{m} = c_1r_2 + c_2 \), which we deduce by applying the same architecture of computing \( c_1 \), as shown in Figure 4.3. During the Norm run, we compute the original \( \overline{m} \) and compare it with the RESO cycle output. The latter uses shifting one place
to the right, that gives us $2 \cdot \overline{m}$, and the decoding takes the most significant 8 bits, in order to find the half of the encoded output.

### 4.3.2 Recomputing with Encoded (Negated) Operands

While RESO has a high rate of fault detection, the increase in bus size makes RESO relatively expensive to perform the rigorous multiplication operation. Moreover, the comparator unit requires the selective 8 bits ranging from LSB to (MSB-1), further complicating the process. Hence, we explore a less extensive alternative, namely, recomputing with negated operands (RENO). The operands in InvRBLWE are already in 2’s complement; thus, we can avoid the cost of performing 2’s complement externally, which eventually makes RENO a highly-efficient fault detection scheme while maintaining high error coverage.

#### 4.3.2.1 RENO on Key Generation

To perform recomputing with negated operands in the key generation stage, we insert a multiplexer that controls the regular operation without error-detection (NORM) and the RENO operations. While the NORM operation computes the $p = r_1 - ar_2$, we negate both operands $a$ and $r_2$, which provides $p' = r_1 - (-a)(-r_2)$. $-a$ and $-r_2$ are denoted as $a'$ and $r'_2$ in Figure 4.4a. In a fault-free scenario, the recomputed output of the adder, i.e., $Sub'$ will be equal to the original output, i.e., $Sub$. RENO benefits in terms of overhead in two ways: 1) there is no need for decoding, as the negating two operands is self-decoding and 2) the representation of the operands in 2’s complement discards the need to compute negation with external circuitry.

As encryption provides two outputs, we have to enforce fault detection schemes in both computations. We compute RENO outputs of $c_1$ as $c_{1_{reno}} = (-a)(-e_1) + e_2$, whose operation is identical to the decryption as described below, and $c_2$ as $c_{2_{reno}} = (-p)(-e_1) + e_3 + \overline{m}$, as shown in Figure 4.4b. Eventually, we compare the RENO outputs ($Add'$) with their corresponding original round outputs ($Add$) and any discrepancy will be detected.
Figure 4.4: Hardware construction of recomputing with negated operands (RENO) for (a) key generation (b) encryption of InvRBLWE (the gray-colored box denotes the module on the corresponding scheme in the RESO figures.)
Figure 4.5: Hardware construction of recomputing with negated operands (RENO) for decryption.

4.3.2.2 RENO on Decryption

Here, we compare the non-recomputed round output of decryption, \( m \) with the recomputed output \( m_{\text{reno}} = (-c_1)(-r_2) + c_2 \), as shown in Figure 4.5. The Norm round output of each byte Add is compared with the RESO round output of the same byte, Add'.

4.4 Error Coverage and FPGA Implementations

4.4.1 Fault Simulation

Our proposed fault detection schemes can detect both permanent and transient faults. An attacker may not be successful in flipping exactly one bit to collect sensitive information due to technological constraints, which leads to considering schemes that can detect multiple stuck-at faults (stuck-at 0 and stuck-at 1), in addition to single faults. Our fault model considers stuck-at faults, whose effect time can range from multiple clock cycles (transient faults) throughout a full operation (permanent faults). We consider the cases of faulty wires, even the cases where such a wire does not affect the other connected wires. Hence, our fault model encompasses the events which are excluded by the assumptions of the multivariate fault model of the work in [79]. Our redundancy based schemes can thwart the fault injections
presented in the work of [25], which includes zeroing ciphertext and zeroing secret key. Such fault attacks can be counted as CCA2 (adaptive chosen-ciphertext attacks), where redundancy can protect against skipping faults in the context of RLWE. In the same line of logic, our schemes can thwart the faults presented in [80] which assumes injection of a single random fault, ranging from skipping faults to glitches in storage, which is evident from our simulation results of permanent and transient faults. A software-based fault resilient approach was presented in the work of [81], whose fault model states zeroing, skipping, and randomization faults, which can be thwarted based on the above discussion.

We evaluated the fault detection capability of the proposed work based on fault-injection simulation coded in VHDL. We injected three types of stuck-at faults, i.e., a) single-bit upset (SBU), b) single-byte double-bit upset (SBDBU), and c) multiple bit (MB) faults for over 65,000 cases, all injected at the input state of the decryption algorithm. The faults that we consider are stuck-at 0 and stuck-at 1. In each case, we attained that our schemes can achieve high fault detection rates (worst case error coverage 99.9991%), for both permanent and transient faults. Moreover, the comparator circuits can be compromised, which can be resolved by hardening them using triple modular redundancy (TMR) and other fault-tolerant techniques as a solution to faulty voter conditions. We incorporated the TMR circuit, where a module is replicated three times, and a majority voter, which is immune to faults, extracts the output. To further enhance the simulation, we injected faults in three locations, a) the inputs, b) the adder outputs, and c) one of the TMR voter inputs of the key generation scheme. Our schemes show worst-case error coverage of 99.9968% for such cases, confirming that they can detect faults with high error coverage even when the comparators are compromised. Our simulations show that recomputing can detect cryptographically impactful faults which can break the security of unprotected implementations, detecting faults with different multiplicities.

Our fault detection schemes are algorithm-oblivious, hence, the faults injected and the errors introduced in the algorithm of RLWE do not coincide. The errors added during the
three stages of RLWE do not tolerate the malicious or natural faults of our fault model, because the faults cause malfunction in the site of injection, i.e., the module or the wire. On the contrary, the errors are injected to ensure that the RLWE problem is the worst-case lattice problem. It is evident from our simulation results that our schemes strengthen the security of the RLWE architecture, as they are prone to hardware fault injection.

A subset of fault attacks that can obtain biased fault models is presented in the work of [77], with the idea of a higher probability for fault injected in both original and redundant architectures. The fault categories presented in [77] are single-bit upset (SBU), single-byte double-bit upset, single-byte triple-bit upset (SBTBU), single-byte quadruple-bit upset (SBQBU), other single byte (OSB) faults, and multiple byte (MB) faults. The redundancy based fault detection schemes presented in this chapter, along with other parity-based approaches, e.g., signatures and interleaved parity, can prevent the aforementioned faults fully [85]. While the presented redundancy based fault detection schemes may fail to detect attacks where the adversary can inject the same fault in both the input and output, i.e., bypassing the fault detection computation, cascading the encoding schemes based on fault space transformation [78] can nullify the effect of bias and thwart the biased attacks.

4.4.2 FPGA Comparison for Error Detection

We perform the benchmark for fault detection on the RESO and RENO schemes as well as part of the original implementation from [76] on Virtex-7 and Kintex UltraScale+ FPGAs. We note that we have implemented just a subset of the work in [76] which helps us in comparisons. We note that the entire architecture is much larger as seen in [76], but in order to have fair overheads, just a subset on which error detection is applied has been implemented here. Table 4.1 represents hardware implementations for $n = 256$, performing a complete encryption/key-generation operation, as shown in Figure 4.1 and Figure 4.2. In our implementation, the key generation and decryption stages provided identical results, hence we are tabulating both in one category. Our results incorporate TMR as well as subpipelining.
Table 4.1: Implementation results for FPGA through Kinex-UltraScale+ and Virtex-7 for encryption (Enc\textsubscript{Kin} and Enc\textsubscript{Vir}, respectively) and key generation/decryption (Gen\textsubscript{Kin} and Gen\textsubscript{Vir}, respectively). We chose \((n, q) = (256, 256)\) to reflect moderate security and the overheads include the cost of TMR module.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>LUT</th>
<th>FF</th>
<th>Delay (ns)</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original (Enc\textsubscript{Kin})</td>
<td>826</td>
<td>769</td>
<td>19.13</td>
<td>1.44</td>
</tr>
<tr>
<td>RESO(Enc\textsubscript{Kin})</td>
<td>1133</td>
<td>1045</td>
<td>20.58</td>
<td>1.64</td>
</tr>
<tr>
<td>RENO(Enc\textsubscript{Kin})</td>
<td>888</td>
<td>809</td>
<td>19.74</td>
<td>1.61</td>
</tr>
<tr>
<td>Original (Gen\textsubscript{Kin})</td>
<td>108</td>
<td>256</td>
<td>14.45</td>
<td>1.38</td>
</tr>
<tr>
<td>RESO(Gen\textsubscript{Kin})</td>
<td>152</td>
<td>359</td>
<td>17.51</td>
<td>1.69</td>
</tr>
<tr>
<td>RENO(Gen\textsubscript{Kin})</td>
<td>129</td>
<td>297</td>
<td>16.71</td>
<td>1.44</td>
</tr>
<tr>
<td>Original (Enc\textsubscript{Vir})</td>
<td>930</td>
<td>577</td>
<td>19.13</td>
<td>0.54</td>
</tr>
<tr>
<td>RESO(Enc\textsubscript{Vir})</td>
<td>1234</td>
<td>792</td>
<td>21.67</td>
<td>0.593</td>
</tr>
<tr>
<td>RENO(Enc\textsubscript{Vir})</td>
<td>1007</td>
<td>611</td>
<td>19.9</td>
<td>0.577</td>
</tr>
<tr>
<td>Original (Gen\textsubscript{Vir})</td>
<td>108</td>
<td>256</td>
<td>18.98</td>
<td>0.186</td>
</tr>
<tr>
<td>RESO(Gen\textsubscript{Vir})</td>
<td>151</td>
<td>378</td>
<td>21.93</td>
<td>0.274</td>
</tr>
<tr>
<td>RENO(Gen\textsubscript{Vir})</td>
<td>125</td>
<td>291</td>
<td>20.45</td>
<td>0.223</td>
</tr>
</tbody>
</table>

for throughput degradation alleviation. Subpipelining does reduce the data path delay by doubling the frequency, with the expense of higher area overhead. For fair comparison, we have utilized medium area and performance efforts for both synthesis and implementation phases in Vivado across the implementations. In absence of any compensation, the total time of recomputing architectures that do not embed throughput alleviation approaches will be twice the original, i.e., \(2n\) cycles. This drastic deterioration of the throughput can be improved by incorporating subpipelining. The design throughput will be close to the original architecture as subpipelining increases the frequency. While subpipelining introduces slight
area overhead, the overall low throughput degradation of the error detection approach highly compensates for the former. One can insert registers in locations that will eventually break the timing paths into approximately equal halves.

From Table 4.1, for both cases, the area overhead, i.e., lookup table (LUT) and flip-flop (FF), delay and power overheads are significantly lower for RENO, compared to RESO, proving that the lack of decoding stage and implementing the inputs as 2’s complement form. The overheads are also acceptable, with the highest overhead in RENO being 15.74%. The source of overheads is the modules outside the gray-colored box in all the figures.

4.5 Conclusion

The chapter presents two fault detection schemes on three separate stages of InvRBLWE architectures in the ring \( \mathbb{R} = \frac{\mathbb{Z}[x]}{x^n+1} \). The schemes add low overhead with high error coverage. The low hardware overhead is beneficial to compact and deeply embedded system applications. We assess the implementation and performance metrics of our fault detection schemes by implementing the schemes on Virtex-7 and Kintex-UltraScale+ FPGA. With the high error coverage and low overhead, our schemes can be tailored in terms of fault detection and overhead to be tolerated.
Chapter 5: Efficient Error Detection Architectures for Post Quantum Signature Falcon’s Sampler and KEM SABER

5.1 Post-Quantum KEM and Signature Schemes

Lattice-based cryptography [87] is one of the most promising classes among the NIST post-quantum cryptography (PQC) submissions of the final round (announced in 2020). One category of lattice-based encryption schemes is learning with errors (LWE)-based schemes, incorporating the worst-case lattice problem. Learning with rounding (LWR) [88] is a subclass within LWE, both of their security levels relying on noise introduction. SABER is one such module-LWR [89] encryption scheme, which is resistant against Chosen-Ciphertext Attack (CCA) and has proceeded to the third round of NIST’s PQC competition in 2020.

SABER was computationally challenging for the absence of an NTT-based multiplier, because of using an unconventional set compared to the popular number-theoretic transform (NTT) with prime parameter set [37], which has been improved by proposing a fast polynomial multiplication based on the Toom-Cook algorithm [90] in the work of [91]. Software optimization techniques of SABER have been proposed by improving the Toom-Cook multiplier [91]. The hardware/software co-design approach to accelerate the SABER computation process has been explored in [92], which achieved significant speed-up compared to software-based implementations.

Among NIST PQC competition Round 3 finalists, Falcon [93], a lattice-based signature scheme, utilizes fast Fourier sampling over NTRU lattices, instantiating the theoretical framework of a hash-and-sign-based signature technique, proposed in [94], the latter being

---

4 This chapter was published in the IEEE Transactions on Very Large Scale Integration Systems (TVLSI) [86] ©2022 IEEE
provably secure and resistant against the key-recovery attack [95]. The article in [96] presented a compact and efficient instantiation of Falcon, which allows an intermediate security level. The toolchain proposed in [97] to instantiate efficient constant-time discrete Gaussian sampler, proved to be practical and secure to use as a post-quantum signature algorithm, e.g., Falcon, with insignificant performance degradation compared to a non-constant-time sampler. To summarize, Falcon ranks best in terms of efficiency and compactness, while not sacrificing security, making it an attractive signature scheme for the PQC era.

In this chapter, we propose fault detection techniques for SABER, in both the full hardware and HW/SW codesign approach. As the security concerns of Gaussian samplers have been an issue for the scheme, we propose error detection for fault attacks on Falcon hardware implementation, a highly compact variant of Falcon, i.e., ModFalcon [96], as well as the sample algorithm of a constant time Gaussian sampler [97]. This is the first work on fault detection schemes of a post-quantum cryptographic signature scheme. Such attacks can break into state-of-the-art signature schemes and derive sensitive information. Very few works such as [33, 56, 74, 98, 99] exist on error detection of PQC. Our proposed schemes can be tailored to resource-constrained applications while being flexible to different reliability levels.

The main contributions of this chapter are as follows:

- We present fault detection schemes for SABER on the performance bottleneck, the PRNG generator involving a binomial sampler, as well as the polynomial multiplier architecture for fully hardware SABER architecture.

- We also propose error detection architecture in the high-level architecture of the HW/SW codesign approach of SABER, especially, in the evaluation and the interpolation datapath of the Toom-Cook algorithm, which is the most computationally exhaustive stage of any SABER architecture.

- We propose error detection schemes for the hardware construction of Falcon’s sampler, specifically, in the signature algorithm of ModFalcon and the Gaussian sampler. We apply
recomputing schemes to achieve high fault coverage. The schemes are flexible and can be applied to other signature schemes as well.

- We simulate the proposed scheme by injecting faults in a Xilinx FPGA family. The assessment of our proposed schemes shows high error coverage.
- We implement the proposed architecture on FPGA family to evaluate the implementation and performance metrics of SABER. The proposed error detection schemes add acceptable overheads, compared to the original implementation.

5.2 Preliminaries

5.2.1 Recomputing Overview

Recomputing is a time redundancy technique, involving encoding ($c$) and decoding ($d$) operations of the function in question ($f$) (e.g., sampler, polynomial multiplication), where decoding is the functional inverse of the encoding operation. In this method, typically the transient faults within the functions result in different outputs between the non-recomputed and recomputed cycles. However, permanent faults are typically only detected if recomputation is done using encoded operands. Fault attacks, involving clock or voltage glitches, laser beam injection, electromagnetic pulses, which tamper the operation of the electric circuit and alter the input, intermediate variable, or final results, may be detected via recomputing.

5.2.1.1 Saber Overview

The security of SABER relies on the hardness of module-LWR problem, which is given by: $(\overrightarrow{a}, b = \lfloor \frac{p}{q}(\overrightarrow{a}^T \overrightarrow{s}) \rfloor) \in \mathbb{R}_q^{l \times l} \times \mathbb{R}_q$, here $\overrightarrow{a}$ is a vector of randomly generated polynomials in $\mathbb{R}_q$ and $\overrightarrow{s}$ is a secret vector of polynomials in $\mathbb{R}_q$ whose coefficients are sampled from a centered binomial distribution, and the modulus $p$ is less than $q$.

- Key generation: This process starts by randomly generating a seed that determines an $l \times l$ matrix $\overrightarrow{A}$ consisting of $l^2$ polynomials in $\mathbb{R}_q$. A secret vector $\overrightarrow{s}$ of polynomials whose entries are sampled from a centered binomial distribution is also generated. The public key
then incorporates the matrix seed and the rounded product $\vec{A}^T \vec{s}$, while the secret key consists of the secret vector $\vec{s}$.

- Encryption: Encryption consists of generating a new ‘secret’ $\vec{s}'$ and adding the message to the inner product between the public key and the new secret $\vec{s}'$. This forms the first part of the ciphertext, while the second is used to hide the encrypting secret and contains the rounded product $\vec{A} \vec{s}'$.

- Decryption: Decryption utilizes the secret key to compute $v$, which is approximately the same as the $v'$ computed during encryption. This allows extracting the message from the ciphertext.

- Parameter Selection: SABER defines three sets of parameters which match NIST security levels 1, 3 and 5, namely, LightSABER, SABER and FireSABER. All three levels use polynomial degree $N = 256$, and moduli $q = 2^{13}$ and $p = 2^{10}$. However, the binomial distribution parameter and the message space of them are the following: LightSABER, SABER, and FireSABER use module dimensions 2, 3, 4 respectively, and their secrets are sampled from $[-5, 5]$, $[-4, 4]$, and $[-3, 3]$.

5.2.2 Falcon Overview

A lattice is a discrete subgroup $L$ of some $\mathbb{R}^n$ and the lattices are full-rank. In other terms, a lattice is a set of integer linear combinations of the rows, the basis being $B \in \mathbb{R}^{n \times n}$. The Falcon signature algorithm consists of three steps, key generation, signature generation, and verification, which are described as follows:

- Key generation: In the first step of key generation, one needs to generate the polynomials, $f, g, F, G \in \mathbb{Z}[x]/\phi$, fulfilling the NTRU equation. In the next step, Falcon tree $T$ is constructed, through LDL* decomposition of the matrix $G = BB^*$. The output of key generation is a public key $pk = h = gf^{-1} \mod q$ and a secret key $sk = (\hat{B}, T)$.

- Signature generation: In the first part of the signature generation, a hash value $c \in \mathbb{Z}_q[x]/\phi$ of the message $m$ and a salt $r$ are computed. The short values $s_1, s_2$ such that
\[ s_1 + s_2 = c \mod q, \] are computed from the hash value as well as the \( sk \); the latter taking advantage of its knowledge about \( f, g, F, G \), and \( f\)Sampling algorithm. A compressed version of \( s_2 \) which also contains a random seed \( r \), is generated as the signature. Sending only \( s_2 \) as output is sufficient because \( s_2, \) hash \( c, \) and public key \( h \) can reconstruct \( s_1 \).

- **Signature verification:** The first step of signature verification repeats the hashing of \( m \) and \( r \) into the hash value \( c \). This hashing is followed by recomputing the \( s_1 \) and checking whether \( ||s_1, s_2|| \leq \beta \) is satisfied, \( \beta \) being predefined acceptance bound.

### 5.3 Proposed Error Detection Techniques

In this section, we discuss the existing side-channel attacks on SABER and Falcon as well as present recomputing-based error detection schemes, which incur low overhead for SABER and Falcon architectures.

#### 5.3.1 Fault Attacks and Threat Model

Fault injection can be defined as an active attack that aims to disrupt the cryptographic operation processing sensitive data, and in turn, results in incorrect output revealing sensitive information [100]. As precise fault injections are getting more difficult because of the shrinking geometry size of integrated chips, studies show that arbitrary injection of faults can be utilized to exploit vulnerabilities instead [101]. Such faults attacks do not tamper with the combinational circuitry of digital systems, rather alter the sampling process of the flip-flop or decreased clock period of a register, resulting in wrong output [102], [103]. This injection can exploit the sampler of any signature algorithm, e.g., the Gaussian sampler of the Falcon signature.

The ideal attack (which is not practical in general) would be to inject bit-faults in the location and at the preferred cycle to gain much information. While technological constraints may hinder an attacker to flip exactly one bit, our fault model includes single as well as multiple stuck-at faults (stuck-at 0 and stuck-at 1). We inject single event upset
(SEU) and multiple upset (MU) with a single fault adversary, where the adversary can inject stuck-at faults at one or multiple positions, in one execution of the operation. To execute that, the fault model we chose requires minimal information on faulty and fault-free computation, resembling differential fault intensity analysis (DFIA) [104]. Although the Fujisaki-Okamoto (FO) transform applied in the encryption/decryption provides redundancy through re-encryption, it fails to detect recent attacks described in [105] which gathers linear inequality of key coefficients by observing the outcome of decapsulation after inserting an instruction-skipping fault. Our error detection schemes, combined with the FO transform can prevent such attacks. Moreover, our suggested schemes, combined with masking, can protect against recent categories of fault attacks, i.e., persistent fault analysis [106], and Statistical Ineffective Fault Attack (SIFA) [101].

5.3.2 Proposed Error Detection Schemes on SABER

The binomial sampler (essential for random coefficient generation) and polynomial multiplication (both Toom-Cook multiplication and schoolbook) are essential to the operation of SABER; hence, their error detection schemes are crucial. We also explore the error detection techniques for HW/SW codesign architectures, which are accelerated design resulting in a fast cycle and high flexibility for encapsulation and decapsulation operation.

5.3.2.1 Error Detection on Binomial Sampler

The binomial sampler computes a sample from a \( \mu \)-bit pseudo-random input string, e.g., \( r[\mu - 1 : 0] \), by computing \( HW(r[\mu/2 - 1 : 0]) - HW(r[\mu - 1 : \mu/2]) \), where \( HW() \) stands for the Hamming weight Figure 5.1. In SABER, the secret coefficients are drawn from a centered binomial distribution with the parameters \( \mu = 10, 8, \) and 6 for LightSABER, SABER, and FireSABER, respectively. In Figure 5.1, a sample is represented as a 4-bit, sign and magnitude number (pair of sign and an absolute value) in the implementation. For SABER, since \( \mu = 8 \) divides the word-length of the data memory, two 64-bit pseudo-random
words are read from the memory, then they are stored in a 128-bit buffer register, then 16 samples are generated in parallel and they are stored in an output buffer register of length 64-bit, and finally, the output buffer is written to the data memory. In our architecture from Figure 5.1, we implement recomputing with swapped operands (RESwO), to detect faults in the binomial sampler. We introduce a multiplexer with the select Norm/RESwO, which runs the original operation in Norm cycle, and swaps the inputs of the subtractor in the RESwO cycle. For example, the subtractor output is \((a - b)\) in Norm cycle and \((b - a)\) in the RESwO cycle. To detect faults, we compare the Norm and RESwO cycle outputs,
which are the same in a fault-free scenario. To ensure that, we flip the sign bit of the 2’s complement so that the output is 2’s complement of \((a - b)\) in both cases. Figure 5.1 shows error detection operation for \(\mu\) bits, which is replicated 8 times for a 64-bit data memory output for SABER.

5.3.2.2 Error Detection on Parallel Polynomial Multiplication

The Toom-Cook method is proposed in the work of [91], which can be used to split a polynomial multiplication of 256-coefficient into seven polynomial multiplications of 64-coefficient. Using such Toom-Cook multiplication, the total number of calls to schoolbook multiplication is 63 for 256-coefficient multiplication, compared to 81 calls for the Karatsuba method. The polynomial multiplier architecture that implements a parallelized version of the schoolbook multiplication is described in Algorithm 5.1. To attain maximum parallelism in data read/write, and to avoid the memory-access bottlenecks, the entire secret polynomial \(s(x)\) is stored in a shift register (Figure 5.2), as all the bits of a register can be accessed simultaneously on a hardware platform. At the beginning of a polynomial multiplication, \(s(x)\) is read from the data memory (block RAM) and then loaded into the shift register. As shown in Algorithm 5.1, only one coefficient of the other polynomial \(a(x)\) is required at a time to compute the scalar multiplication \(s(x) \cdot a[i]\). Hence, it is not necessary to store the entire \(a(x)\) polynomial. The coefficient selector block in Figure 5.2 provides the required
Figure 5.2: Proposed error detection architecture on polynomial multiplication with multiply-and-accumulate (MAC) unit construction.

Coefficient of $a(x)$ during the multiplication $s(x) \cdot a[i]$ by the parallel multiply-and-accumulate (MAC) cores, from the inset of Figure 5.2. After the multiplication $s(x) \cdot a[i]$, $s(x)$ needs to be multiplied by $x$. This operation is a simple nega-cyclic left-shift operation that moves each coefficient from position $i$ to position $i + 1$ and sends the last coefficient to the first position after a modular subtraction from zero. In this implementation, such is performed easily by flipping the 256-th coefficient, taking advantage of the sign-magnitude system representation.
Figure 5.3: Proposed error detection architecture on hardware accelerator.

5.3.2.3 Error Detection on HW/SW Codesign

The hardware/software codesign approach is an extensively researched technique that aims to achieve performance targets through a shorter development cycle than is typical for hardware-only implementations. Replacing a purely-hardware benchmarking is not the intention of hardware/software benchmarking, rather, the aim is to ease the development of hardware-only implementations via researching hardware accelerators for major operations. During the encapsulation of SABER, only the accelerated operations performed during encryption are SABER.PKE.Enc. The seed of SHAKE-128, i.e., $s_0$, is used to generate elements
of the matrix $A$, with each element representing a polynomial, as shown in Figure 5.3. The sign-extended version of matrix $A$ is used to generate $b' = (As' + h) \mod q$, where $h$ is a constant of the equation. Only one row of the $A$ matrix is produced at once and the elements of $A$ are multiplied by the corresponding elements of $s_0$, with a view to shorter execution time and smaller matrix memory. The registers on the right of MAC in Figure 5.3, stores the temporary results. The MAC constructions are shown as the inset in Figure 5.2.

In our scheme, we apply RENO at both the inputs of the MAC module in Figure 5.3. The negated input operands of the multiplication detect the presence of faults in the RENO cycle of the multiplexer select when discrepancy with the Norm cycle output is flagged by the comparator. Applying RENO does not increase the bus size; thus, the inputs remain 13-bit; hence, the implementation is compatible with the existing architecture. We perform a modular negation operation by subtracting each MAC input from $q$. Our schemes can apply to any modified version of the MAC core, thus our schemes are MAC architecture oblivious. As the SABER decapsulation stage utilizes the same mechanism, RENO can be applied there as well to detect fault injection.

5.3.3 Error Detection Schemes on Falcon Sampler

We apply the schemes for the non-constant time Gaussian sampler, which is prone to fault attacks, hence requiring additional countermeasure. Our error detection approaches are also applicable to constant time Gaussian samplers.

From Section 5.2, we recall the ffsampling algorithm is the basis to generate secret key $s_k$ for signature generation. As shown in Algorithm 5.2, the Falcon tree generation, i.e., line 7 stating $LDL^*$ decomposition of matrix $G$, and the ffsampling are combined in one algorithm, namely, ffsampling$^*$. Such combination reduces the memory consumption significantly compared to the reference Falcon implementation. Here we note that the three functions of Algorithm 5.2, i.e., ffsampling, splitfft, and mergefft are linear elementary op-
Algorithm 5.2 \texttt{ffsampling}$_{n}^{*}(t, G)$

Input: $t = (t_0, t_1) \in \text{FFT}(\mathbb{Q}[x](x^n + 1))^2$ and a full-rank Gram matrix $G \in \text{FFT}(\mathbb{Q}[x](x^n + 1))^{2 \times 2}$, $\sigma \in 1.55 \sqrt{q}$

Output: $z = (z_0, z_1) \in \text{FFT}(\mathbb{Z}[x](x^n + 1))^2$

1: if $(n = 1)$ then
2: $\sigma' \leftarrow \sigma \sqrt{G_{00}}$
3: $z_0 \leftarrow D_{Z,t_0,\sigma'}$
4: $z_1 \leftarrow D_{Z,t_1,\sigma'}$
5: return $z = (z_0, z_1)$
6: end if
7: $L, D \leftarrow LDL^*(G)$
8: $d_{01}, d_{11} \leftarrow \text{splitfft}_2(D_{11})$
9: $t_1 \leftarrow \text{splitfft}_2(t'_1)$
10: $G_1 \leftarrow \begin{bmatrix} d_{10} & d_{11} \\ xd_{11} & d_{10} \end{bmatrix}$
11: $z_1 \leftarrow \text{ffsampling}_{n/2}(t_1, G_1)$
12: $z_1 \leftarrow \text{mergefft}_2(z_1)$
13: $t'_0 \leftarrow t_0 + (t_1 - z_1) \odot L_{10}$
14: $d_{00}, d_{01} \leftarrow \text{splitfft}_2(D_{00})$
15: $t_0 \leftarrow \text{splitfft}_2(t'_0)$
16: $G_0 \leftarrow \begin{bmatrix} d_{00} & d_{01} \\ xd_{01} & d_{00} \end{bmatrix}$
17: $z_0 \leftarrow \text{ffsampling}_{n/2}(t_0, G_0)$
18: $z_0 \leftarrow \text{mergefft}_2(z_0)$
19: return $z = (z_0, z_1)$

Operations: Addition, subtraction, multiplication, and division; hence, we can apply linear encoding and decoding schemes, without any loss of information.

5.3.3.1 Recomputing on Negation

Algorithm 5.2 can be partially depicted (lines 11 through 13) by Figure 5.3, multiplexer select Norm/RENO being at Norm, i.e., unmodified operation of the \texttt{ffsampling}$_{n}^{*}$. In the original operation of line 13, the output of \texttt{ffsampling}$_{n}^{*}$, $z_1$ is subtracted from $t_1$. In our encoded scheme, we perform RENO during the RENO cycle of the multiplexer, where we negate both $t_1$ and $z_1$, and perform subtraction of $-z_1$ from $-t_1$, resulting in $\text{out}_1 = (t_1 - z_1)$ in a fault-free scenario, which is consistent with the Norm cycle output. However, in a
faulty scenario, the outputs of both Norm and RENO cycles will be discrepant, which will be flagged by the comparator comparing this output with $out_1$, detecting the presence of faults. We note that decoding in this scheme is free of hardware cost; hence, a low-overhead and inexpensive fault detection approach.

5.3.3.2 RESwO on Multiplication

In line 13 of Algorithm 5.2, $(t_1 - z_1)$, is multiplied with the left child of LDL* output $L_{10}$. In our scheme, we perform this unmodified operation during the Norm cycle of the multiplexer. For the recomputed operation, we perform recomputing with swapped operands (RESwO), where the multiplication operands $L_{10}$ and $(t_1 - z_1)$ are swapped and stored in $out_2$, as shown in Figure 5.5. Any discrepancy between the Norm and RESwO rounds is flagged by the comparator comparing $L_{10} \odot (t_1 - z_1)$, and $out_2$. RESwO scheme also requires no decoding, making it a cost-effective fault detection mechanism.
5.3.3.3 RENO on Multiplication

One can also explore negation on the aforementioned multiplicands. In such a case, as depicted in Figure 5.6, the Norm cycle will perform $L_{10} \odot out_1$, where $out_1 = (t_1 - z_1)$. On the contrary, during our proposed RENO cycle, the architecture will perform negation on both operands, resulting in $out_2 = -L_{10} \odot -out_1$. In a fault-free scenario, the RENO output should match with the Norm cycle, deviation from which will be captured by the comparator comparing $out_2$ and $L_{10} \odot out_1$. Similar to the case of RENO on negation, RENO on multiplication requires no decoding as negating both operands provides the same output in the case of multiplication.

5.3.3.4 RENO on Multiplication-and-Accumulator (MAC)

Instead of applying error detection on either the multiplication or the subtraction of line 13 in Algorithm 5.2, one can perform error detection on this overall multiplication-and-
Figure 5.6: RENO on multiplication of key generation in Falcon accumulator circuitry. We propose RENO for MAC of line 13, where the Norm cycle of multiplexer results in \( t'_0 \), according to Algorithm 5.2. In our proposed RENO operation, we negate both \( L_{10} \) and \( t_0 \), as shown in Figure 5.7, resulting in the encoded output of \((-L_{10} \odot out_1) - t_o\), where \( out_1 = (t_1 - z_1) \). We decode this encoded operand by again negating the MAC output, providing \( out_2 = -(L_{10} \odot out_1 - t_0) \), which should be identical to \( t'_0 \) in a fault-free scenario and the comparator flags any inconsistency between these two. The presence of an additional decoding circuit is somewhat more expensive than the previously mentioned schemes requiring no decoding; however, if one wishes to perform overall error detection on the entire MAC, RENO is a viable choice.

5.3.3.5 **RENO on Overall ffsampling**

We finally propose an error detection scheme that operates on the inputs of the entire Algorithm 5.2 and performs RENO on its operands, as depicted in Figure 5.8. During the multiplexer select Norm, the unmodified function of ffsampling\(_n\) is performed. On the other hand, in our proposed RENO scheme to detect faults, we negate \( t_o \), the output of ffsampling\(_n\),
Figure 5.7: RENO on multiplication-and-accumulator (MAC) module of Falcon

$z_1$ as well as $t_1$. Therefore, the encoded output becomes $-t_0 + (-t_1 - (-z_1)) \odot L_{10}$, after the subtraction and MAC operations. Now, to decode the encoded output and find $out_4$, we again negate it which, in a fault-free scenario, would result in $t_0 + (z_1 - t_1) \odot L_{10}$, resembling $t'_0$. The comparator notifies of the discrepancy between Norm and RENO rounds. We would like to conclude that a non-constant time Gaussian sampler can easily fall victim to timing attacks and other fault attacks. However, such non-constant time Falcon approaches are heavily researched and popular for micro-controller based platforms. Our proposed error detection schemes are low-overhead, while ensuring high error detection for those faulty situations, and can be implemented for already compact Falcon implementations.

5.3.4 Implementation of Constant-time Falcon Sampler

Falcon being a fairly new scheme, its resilience against fault attacks has not been analyzed thoroughly. While active attacks on Falcon are yet unknown, incorporating non-
constant time Gaussian sampler can seriously affect the security of the scheme; thus, should be replaced with a constant-time Gaussian sampler.

5.3.4.1 ModFalcon Implementation and Error Detection

ModFalcon, a new variant of signature schemes based on the Falcon design, is based on module lattices. This new implementation possesses both the compactness and efficiency of Falcon. ModFalcon achieves the highly compact lattice-based signature with a 128-bit quantum level security. This variant generalizes the instantiation of the hash-and-sign algorithm to NTRU lattices for large module ranks; hence, broadening the parameter set of the Falcon design to a much wider range.

As shown in Algorithm 5.3, the pair \((r, S)\) is the signature, \(r\) being a hashing salt and \(S\) being an encoding of a short vector \(s\) such that \(s \cdot vk = H(r||msg)\). After computing \(H(r||msg)\), the secret key \(B_{F,g}\) is used to sample a proper \(s\). Algorithm 5.3 can be partially depicted (line 4) by Figure 5.9. One can compute \(z\) via the parallel computations of \(Falcon_{sig}\).
Algorithm 5.3 Signature: $(sk, msg) \rightarrow (r, S)$

Require: A standard deviation parameter $\sigma$

1: Get $r \leftarrow U(\{0, 1\})^\lambda$
2: $\mu \leftarrow H(r||msg) \in \mathbb{R}_q$ and let $c = (\mu, 0, ..., 0)$
3: Compute $t = c \cdot B_{F,g}^{-1}$
4: Compute $z \in \mathbb{R}^{n+1}$ such that $s := (t - z) \cdot B_{F,g}$
5: $S = \text{Compress}(s)$
6: return the signature $(r, S)$

Even constant-time Falcon can be vulnerable to fault attacks, hence Figure 5.9 can be modified to incorporate error detection schemes. One can select multiplexer Norm/RENO being at Norm, i.e., unmodified operation of the ModFalcon signature scheme. In the original operation of line 4, the output of signature scheme $z$ is subtracted from $t$. In our encoded scheme, we perform RENO during the RENO cycle of the multiplexer, where we negate both $t$ and $z$, and perform subtraction of $-z$ from $-t$, resulting in $out_1 = (t - z)$ in a fault-free scenario, which is consistent with the Norm cycle output. However, in a faulty scenario, the outputs of both Norm and RENO cycles will be discrepant, which will be flagged by the comparator comparing this output with $out_1$, detecting the presence of faults. We note that decoding in this scheme is free of hardware cost; hence, a low-overhead and inexpensive fault detection approach.
Algorithm 5.4 SamplerZ \((\sigma, \mu)\)

**Require:** \(\mu \in [0, 1), \sigma \leq \sigma_0\) a scaling factor \(C = C(\sigma) \in (0, 1)\)

**Ensure:** \(z \sim D_{Z, \sigma, \mu}\)

1: while true do
2: \(z_0 \leftarrow \text{BaseSampler}()\)
3: \(b \leftarrow \{0, 1\}\) uniformly
4: \(z \leftarrow (2b - 1) \cdot z_0 + b\)
5: \(x \leftarrow \frac{(z - \mu)^2}{2\sigma^2} - \frac{z_0^2}{2\sigma_0^2}\)
6: if \(\text{BerExp}_{C(\sigma)}(x)\) then
7: \(\text{return } z\)
8: end if
9: end while

5.3.4.2 Sample\(_z\) Implementation and Error Detection

The constant-time sampler, formally described in Algorithm 5.4, works by using BaseSampler to generate a sample \(z_0\). Then, it samples a random bit \(b\), and compute \(z = (2b - 1) \cdot z_0 + b\). Finally, it calls \(\text{BerExp}_{C(\sigma)}(x)\) to determine if \(z\) is returned or rejected and start again if necessary.

We explore fault detection to thwart fault attacks on the Sample\(_z\). One can explore negation in line 4 of Algorithm 5.4. In such a case, the Norm cycle will perform the aforementioned computation of \(z\). On the contrary, during our proposed RENO cycle, the architecture will perform negation on both operands, resulting in \(out_2 = \{- (2b - 1) \cdot -z_0\} + b\). In a fault-free scenario, the RENO output should match with Norm cycle, deviation from which will be captured by the comparator comparing \(out_2\) and \(z\). Similar to the case of RENO on negation, RENO on multiplication requires no decoding as negating both operands provides the same output in the case of multiplication.

5.4 Error Coverage and FPGA Implementations

This section presents the results of our FPGA assessments using Xilinx Vivado and VHDL with an FPGA family (Zynq-UltraScale+ ZCU102), using the device xczu9eg-ffvb1156-2-e, to assess the overhead of the proposed construction for the case study of proposed RESwO.
and RENO in the SABER encapsulation algorithm as well as the hardware accelerator, as shown in Table 5.1.

5.4.1 Fault Simulation

We have simulated the error coverage of our proposed work with VHDL as design entry, by injecting three types of stuck-at faults, i.e., (a) single, (b) two-bit, and (c) multiple-bit faults for 200,000 cases, all injected at the input state of the parallel polynomial multiplication algorithm, for permanent and transient faults. In each case, we observed high error detection rates (99.9975%), for both permanent and transient faults incorporating our schemes. For example, in single-bit stuck-at 0 faults, we inserted faults at the LSB of both the inputs of the polynomial multiplication architecture of Saber, using logical AND operation between that faulty bit and logical 0. We also injected two-bit and multi-bit (6-bit) faults, similarly, for a total of 200,000 instances. After the simulation, the error flags were high for 199,995 cases, demonstrating the presence of faults. We calculated the fault detection ratio as $\frac{\text{faults detected}}{\text{faults injected}}$, which in our case resulted in 99.9975%. To be very conservative in reporting the error coverage and about the faults occurring in the entire architecture, one needs to consider those affecting the comparator unit. In case a voter is faulty, a comparator using modular redundancy can be one of the solutions for a compromised comparator circuit, among different fault-tolerant techniques.

5.4.2 FPGA Implementations

We perform the benchmark for error detection on the RESwO scheme for binomial sampler and RENO schemes for both parallel polynomial multiplier and HW/SW accelerator as well as the original. For both cases, we tabulated both the lookup table (LUT) and flip-flop (FF) as area overhead as well as delay and power overheads in Table 5.1, all of which are of the acceptable range. Both error detection schemes applied to binomial sampling and polynomial multiplication incur approximately 18% area overhead, whereas the RENO in-
corporated in HW/SW accelerator adds 22.59% overhead for LUTs. On the contrary, the RESwO and RENO of the binomial sampler and HW/SW accelerator show a lower overhead (19.32% and 17.66%, respectively), compared to the 22.72% overhead for RENO of the polynomial multiplier in FFs. In terms of power, it is evident that the RESwO added the least overhead (6.88%) compared to both the RENO architectures. The delay overhead for the RENO on the polynomial multiplier was the lowest at 11.42%, although the RESwO overhead was acceptable at 15.76%. Thus, we can conclude RESwO results in lower percent overhead compared to the RENO models, due to the simplicity of the RESwO architecture. As this is the first work on implementing error detection of SABER architecture as well as HW/SW codesign, there is no previously published architecture to compare with our performance and overhead matrices. In some of the previous works on fault detection of post-quantum architectures [33, 56], recomputing has been utilized to detect faults on number-theoretic transform and ring polynomial multiplication, respectively, two integral components of lattice-based cryptosystems. The implementation overheads in the work of [33] are 20%, 6%, and 16%, on average, for the area, delay, and power, respectively. On the other hand, the performance matrices for the error detection in [56] are 19.6%, 13.5%, and 15.1% in cases of area, delay, and power overhead, respectively. The overheads of our error detection overheads align with the performance overheads of the previous works, demonstrating the efficiency and low cost of our implementations.

Implementing lattice-based signatures is difficult, based on either the high-speed or lightweight approach, which explains the lack of literature on hardware or hardware/software implementation of Falcon and other lattice-based signatures, e.g., LUOV, HQC, NTS-KEM [107]. However, recomputing being an efficient scheme, we expect similar low overhead results for Falcon as our derived results for SABER.

In absence of any compensation, the total time of recomputing architectures that do not embed throughput alleviation approaches will be twice the original. Subpipelining is the solution to alleviate this drastic decline of the throughput. Through increasing frequency,
Table 5.1: Implementation results for FPGA through Xilinx Zynq-UltraScale+ ZCU102.(xczu9eg-ffvb1156-2-e) for binomial sampling, polynomial multiplication and hardware/software codesign. All the inputs are 256 bits and the parentheses represent percent overheads compared to original architecture.5

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Scheme</th>
<th>LUT</th>
<th>FF</th>
<th>Delay (ns)</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binomial Sampling</td>
<td>Original</td>
<td>85</td>
<td>88</td>
<td>2.03</td>
<td>0.697</td>
</tr>
<tr>
<td></td>
<td>RESwO</td>
<td>100</td>
<td>105</td>
<td>2.35</td>
<td>0.745</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(17.64%)</td>
<td>(19.32%)</td>
<td>(15.76%)</td>
<td>(6.88%)</td>
</tr>
<tr>
<td>Polynomial Multiplication</td>
<td>Original</td>
<td>17,352</td>
<td>5,171</td>
<td>2.959</td>
<td>1.724</td>
</tr>
<tr>
<td></td>
<td>RENO</td>
<td>20,420</td>
<td>6,346</td>
<td>3.297</td>
<td>1.908</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(17.68%)</td>
<td>(22.72%)</td>
<td>(11.42%)</td>
<td>(10.67%)</td>
</tr>
<tr>
<td>Hardware/software codesign</td>
<td>Original</td>
<td>14,277</td>
<td>1,025</td>
<td>3.764</td>
<td>2.097</td>
</tr>
<tr>
<td></td>
<td>RENO</td>
<td>17,502</td>
<td>1,206</td>
<td>4.508</td>
<td>2.295</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(22.59%)</td>
<td>(17.66%)</td>
<td>(19.77%)</td>
<td>(9.4%)</td>
</tr>
</tbody>
</table>

Subpipelining increments the frequency, which in turn makes the recomputed architecture throughput close to the original architecture. The slight area overhead of adding subpipelining can be reasonably traded off by achieving low throughput degradation. The timing paths can be broken into approximately equal halves by inserting registers in proper locations.

In conclusion, we would like to note that the proposed architectures are platform oblivious of the FPGA fabric and hardware platform. As a result, implementing the schemes on application-specific integrated circuits (ASIC) will also provide similar results. Moreover, adding pipelines in the architectures will improve the efficiency and throughput, with the compromise of increased hardware overhead. We would like to note that the proposed architectures are platform oblivious of the FPGA fabric and hardware platform.

5SABER polynomial degree $N = 256$, moduli $q = 2^{13}$ and $p = 2^{10}$, module dimensions 3, and their secrets are sampled from $[4, -4]$.  

78
5.5 Conclusions

We present error detection schemes for SABER on fully hardware construction and hardware/software codesign accelerators. Moreover, we propose error detection schemes for post-quantum signature scheme Falcon, its compact variant ModFalcon and Gaussian sampler, a crucial element of the Falcon signature scheme. Our error detection schemes with recomputing incur low overheads with high error coverage on these two state-of-the-art NIST PQC finalists. We achieve high error coverage of 99.9975% on average, from our recomputing schemes. Moreover, the area, delay, and power overheads are 22.59%, 19.77%, and 10.67%, respectively, in the worst-case scenario. The proposed architectures are implemented on the FPGA family Zynq-UltraScale+, which shows acceptable area, power, and delay overhead.
Chapter 6: Conclusion

With the advent of quantum computers, computationally infeasible problems can be solved efficiently through their usage of physical properties of matter and energy. Mathematical problems which will take more than a human lifetime to solve are the basis of classical cryptosystems. The exponential speed-up of quantum computation will render classical cryptosystems useless, as the encryption will be solved in mere minutes, resulting in a drastic failure of privacy preservation and data security. Thus, encryption schemes need to be developed to protect us against quantum attacks, because many experts predict that within 20 years, quantum computers can break into the current encryption infrastructures. This dissertation focused on developing cryptosystems that countermeasure against side-channel attacks on these architectures, which protect cryptosystems against adversaries, ensuring secured data for all. In addition, our architectures can also detect natural faults, caused by device malfunctions, crucial to proper functionalities of sensitive medical applications, e.g., pacemakers, ring heart rate monitors, and Bluetooth-based ECG monitors.

We present error detection schemes for various lattice-based encryption and key generation schemes. Our error detection schemes with recomputing incur low overheads with high error coverage on these state-of-the-art NIST PQC finalists. We achieve high error coverage from our recomputing schemes. The proposed architectures are implemented on the FPGA family or ASIC, which show acceptable area, power, and delay overhead. These approaches add very little hardware overheads, which is advantageous to incorporate in deeply-embedded systems. We have benchmarked the proposed architectures to assess their ability to detect transient and permanent faults. The proposed architectures are oblivious of the standard-
cell library and hardware platform. Therefore, we expect similar overhead results across different FPGA families and ASIC libraries.

As the future extensions to this dissertation, implementing our proposed countermeasures for PQC can be done on deeply-embedded architectures for instance implantable and wearable medical devices to assess the deployment challenges. Preventing threats against hardware vulnerabilities and cyber-attacks based on side-channel attacks, of both classical and post-quantum cryptosystems, are indispensable to data protection as well as the correct operation of deeply-embedded architectures. Moreover, one can consider combined fault and power analysis attacks and countermeasures on PQC, a challenging extension to this dissertation which has not got considerable attention in open literature.
References


[18] V. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with


[20] V. Lyubashevsky. Lattice-based identification schemes secure under active attacks. In


[23] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryp-
tographic protocols for faults. In *Advances in Cryptology, EUROCRYPT ’97*, pages


sensitivity to fault attacks. In *Proc. IEEE Workshop Fault Diagn. Tolerance Cryptog-

in embedded security. In *2013 26th Int. Conf. on VLSI Design and 2013 12th Int. Conf.


encryption against the most effective side-channel attack: DPA. In *Proc. Defect and

[49] M. Mozaffari Kermani and A. Reyhani-Masoleh. Fault detection structures of the S-
boxes and the inverse S-boxes for the advanced encryption standard. *J. Electronic

[50] M. Mozaffari Kermani, R. Azarderakhsh, and A. Aghaie. Fault detection architectures
for post-quantum cryptographic stateless hash-based secure signatures benchmarked

[51] M. Mozaffari Kermani, R. Azarderakhsh, and A. Aghaie. Reliable and error detection
architectures of Pomaranch for false-alarm-sensitive cryptographic applications. *IEEE

[52] M. Mozaffari Kermani, R. Azarderakhsh, A. Sarker, and A. Jalali. Efficient and reliable
error detection architectures of Hash-Counter-Hash tweakable enciphering schemes.

inversion in GF$(2^8)$ with redundant arithmetic for secure error detection of crypto-
graphic architectures. *IEEE Transactions on Computer-Aided Design of Integrated

hardware architectures for cryptographic block ciphers LED and HIGHT. *IEEE
Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(10):


Appendix A: Copyright Permissions

The permission below is for the reproduction of material in Chapters 2, 3, 4, and 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.
About the Author

Ausmita Sarker received her B.Sc. degree in Electrical and Electronic Engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2016. She is currently a Ph.D. candidate at the Department of Computer Science and Engineering, University of South Florida. Her research interests include cryptographic engineering, post-quantum cryptography, and embedded systems. As of 2022, she has 5 IEEE/ACM Transactions journal papers. She is a student member of IEEE.

Ausmita is the recipient of Chih foundation research and publication award 2021, USF graduate research symposium best poster award 2019, and FICS conference best poster award 2019. She has received NSF HOST Travel Grant: HOST conference travel grant provided by NSF in 2020, CRA conference Travel Grant in 2020, 2019, 2018, and HOST/WISE Travel Grant in 2019.