Fault Detection Architectures for Inverted Binary Ring-LWE Construction Benchmarked on FPGA

Ausmita Sarker, *Student Member, IEEE*, Mehran Mozaffari Kermani, *Senior Member, IEEE*, and Reza Azarderakhsh, *Member, IEEE*

***Abstract*—Ring learning with errors (RLWE) is an efficient lattice-based cryptographic scheme that has worst-case reduction to lattice problem, conjectured to be quantum-hard. Ring-BinLWE is an optimized variant of RLWE problem using binary error distribution, resulting in highly-efficient hardware implementation. Efficient and low-complexity architectures in hardware, thwarting natural and malicious faults, are essential for lattice-based post-quantum cryptography (PQC) algorithms. In this brief, we explore efficient fault detection approaches for implementing the Ring-BinLWE problem. This brief, for the first time, investigates fault detection schemes for all three stages of RLWE encryption. Utilizing the stuck-at fault model, we employ recomputing with encoded operands schemes to achieve high error coverage. We simulate and implement our schemes on a field-programmable gate array (FPGA) platform. Our schemes provide low hardware overhead (area overhead of 15.74%, delay overhead of 7.74%, and power consumption overhead of 4.06%), with high error coverage, which can be suitable for resource-constrained as well as high-performance usage models.**

***Index Terms*—Field-programmable gate array (FPGA), ring-binary learning with errors (Ring-BinLWE), key encapsulation mechanisms (KEM), post-quantum cryptography (PQC).**

I. Introduction

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 [1], involving a binary distribution to choose binary coefficients instead of Gaussian, namely, Ring-BinLWE. A hardware-optimized scheme of Ring-BinLWE proposed in [2] utilizes an inverted ring of Ring-BinLWE (InvRBLWE) and 2's-complement notation range.

In this brief, 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 [3]–[10]. 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 [11], the number-theoretic transformation of lattice-based cryptosystems [12], and ring polynomial multiplication of RLWE [13]. Some examples for error detection in general computations and classical cryptography exist as well [14], [15].

The main contributions of this brief are as follows:

- 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 $R = \frac{\mathbb{Z}_q[x]}{x^n+1}$, 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.

II. 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 $R_q = \frac{\mathbb{Z}_q[x]}{x^n+1}$, 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 [16] 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

Manuscript received August 26, 2020; accepted September 18, 2020. Date of publication September 22, 2020; date of current version March 26, 2021. This work was supported by the U.S. National Science Foundation under Award SaTC-1801488. This brief was recommended by Associate Editor B.-H. Gwee. *(Corresponding author: Mehran Mozaffari Kermani.)*

Ausmita Sarker and Mehran Mozaffari Kermani are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620 USA (e-mail: asarker@usf.edu; mehran2@usf.edu).

Reza Azarderakhsh is with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA (e-mail: razarderakhsh@fau.edu).

Digital Object Identifier 10.1109/TCSII.2020.3025857

Ring-BinLWE was achieved in [2] using 2's complement notation of the coefficients, namely, InvRBLWE, by selecting the range of $R_q = \frac{\mathbb{Z}_q[x]}{x^n+1} = (-\lfloor\frac{q}{2}\rfloor, \lfloor\frac{q}{2}\rfloor - 1)$ 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 $\tilde{m} =$ encode $(m) \in R_q$, where encode is defined as follows: s

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

  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:

$$\text{DECODE} : R_q \rightarrow \{0, 1\}^n$$
$$\sum_{i=0}^{n-1} a_i x^i \rightarrow (m_0, m_1, \ldots, m_{n-1})$$
$$m_i = \begin{cases} 0 & \text{when } |a_i - i - \lfloor\frac{n-3}{2}\rceil| > \frac{q}{4} \\ 1 & \text{else.} \end{cases} \tag{2}$$

III. Proposed Fault Detection Schemes

From most recent attack [9], 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.

A. Recomputing With Encoded (Shifted) Operands

In this brief, 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.

1) Key Generation: The multiplexer select input, *Norm/RESO*, shown in Fig. 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

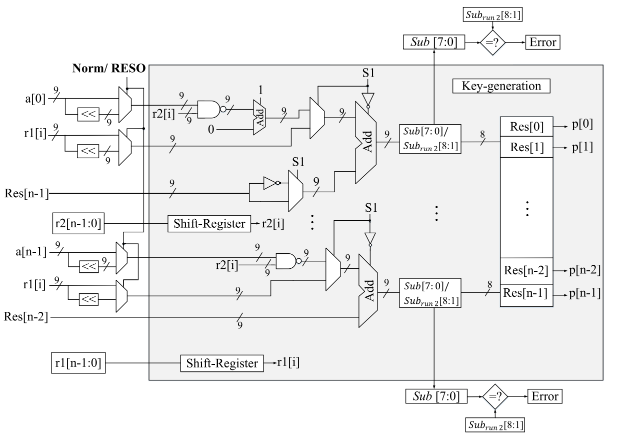

Fig. 1. Hardware construction of recomputing with shifted operands for key generation of InvRBLWE.

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_2q) 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 this brief, 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 Fig. 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 Fig. 1 are our added circuitry for fault detection.

2) Encryption: The encryption operations provide two outputs, c_1 and c_2. Based on Fig. 2(a), 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 Fig. 2(a) adders provide $Add_{run2} = 2(ae_1 + e_2)$. We extract the most significant 8 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 (1). According to Fig. 2(b), 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

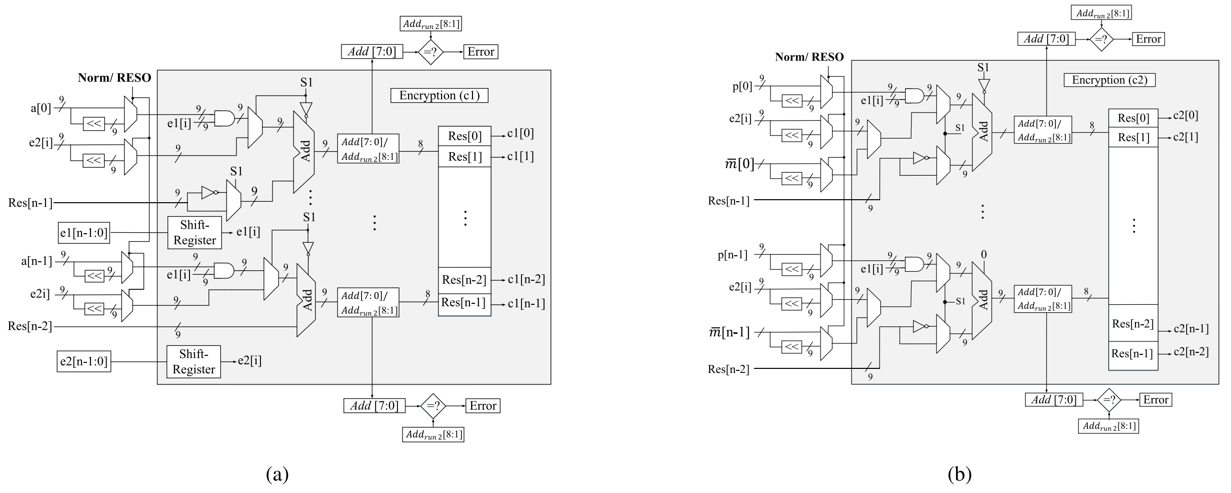

Fig. 2. Hardware construction of recomputing with shifted operands for encryption of InvRBLWE (a) fault detection for c_1 and (b) fault detection for c_2.

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

3) Decryption: The decryption computes $\overline{m} = c_1 r_2 + c_2$, which we deduce by applying the same architecture of computing c_1, as shown in Fig. 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 $2.\overline{m}$, and the decoding takes the most significant 8 bits, in order to find the half of the encoded output.

B. 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.

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 Fig. 4(a). 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.

2) RENO on Encryption: 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

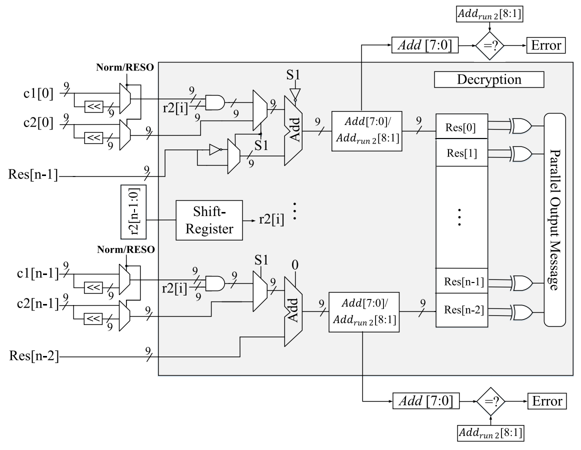

Fig. 3. Hardware construction of recomputing with shifted operands for decryption of InvRBLWE.

as described below, and c_2 as $c_{2_{reno}} = (-p)(-e_1) + e_3 + \overline{m}$, as shown in Fig. 4(b). Eventually, we compare the RENO outputs (Add') with their corresponding original round outputs (*Add*) and any discrepancy will be detected.

3) RENO on Decryption: Here, we compare the non-recomputed round output of decryption, $\overline{m}$ with the recomputed output $\overline{m}_{reno} = (-c_1)(-r_2) + c_2$, as shown in Fig. 4(c). The Norm round output of each byte *Add* is compared with the RESO round output of the same byte, Add'.

IV. Error Coverage and FPGA Implementations

A. 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

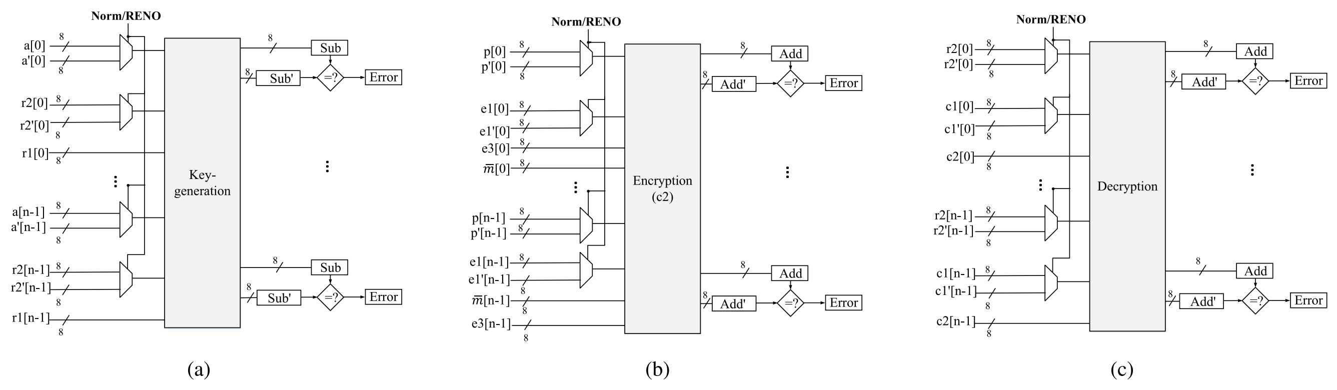

Fig. 4. Hardware construction of recomputing with negated operands (RENO) for (a) key generation (b) encryption (c_2) and (c) decryption of InvRBLWE (the gray-colored box denotes the module on the corresponding scheme in the RESO figures.)

(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 [5]. Our redundancy based schemes can thwart the fault injections presented in the work of [6], 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 [7] 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 [9], 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 [3], with the idea of a higher probability for fault injected in both original and redundant architectures. The fault categories presented in [3] 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 brief, along with other parity-based approaches, e.g., signatures and interleaved parity, can prevent the aforementioned faults fully [18]. 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 [4] can nullify the effect of bias and thwart the biased attacks.

B. 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 [2] on Virtex-7 and Kintex UltraScale+ FPGAs. We note that we have implemented just a subset of the work in [2] which helps us in comparisons. We note that the entire architecture is much larger as seen in [2], but in order to have fair overheads, just a subset on which error detection is applied has been implemented here. Table I represents hardware implementations for $n = 256$, performing a complete encryption/key-generation operation, as shown in Fig. 1 and Fig. 2. In our implementation, the key generation and decryption stages provided identical results, hence we are tabulating both in one category. Our results incorporate

TABLE I
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). WE CHOSE $(n, q) = (256, 256)$ TO REFLECT MODERATE SECURITY AND THE OVERHEADS INCLUDE THE COST OF TMR MODULE

Architecture	Area		Delay	Power
	LUT	FF	(μs)	(W)
Original (Enc_{Kin})	826	769	19.13	1.44
RESO (Enc_{Kin})	1133 (37.17%)	1045 (35.89%)	20.58 (7.56%)	1.64 (14.10%)
RENO (Enc_{Kin})	888 (7.51%)	809 (5.20%)	19.74 (3.17%)	1.61 (11.81%)
Original (Gen_{Kin})	108	256	14.45	1.38
RESO (Gen_{Kin})	152 (40.74%)	359 (40.23%)	17.51 (21.23%)	1.69 (22.41%)
RENO (Gen_{Kin})	129 (19.44%)	297 (16.02%)	16.71 (15.68%)	1.44 (4.06%)
Original (Enc_{Vir})	930	577	19.13	0.54
RESO (Enc_{Vir})	1234 (32.69%)	792 (37.26%)	21.67 (13.29%)	0.593 (9.81%)
RENO (Enc_{Vir})	1007 (8.27%)	611 (5.89%)	19.9 (4.04%)	0.577 (6.85%)
Original (Gen_{Vir})	108	256	18.98	0.186
RESO (Gen_{Vir})	151 (34.4%)	378 (41.92%)	21.93 (14.40%)	0.274 (39.46%)
RENO (Gen_{Vir})	125 (15.74%)	291 (13.67%)	20.45 (7.74%)	0.223 (19.89%)

TMR as well as subpipelining 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 I, 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.

V. CONCLUSION

This brief presents two fault detection schemes on three separate stages of InvRBLWE architectures in the ring $\mathbb{R} = \frac{\mathbb{Z}/p\mathbb{Z}[x]}{x^n+1}$. 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.

REFERENCES

[1] A. Aysu, M. Orshansky, and M. Tiwari, "Binary ring-LWE hardware with power side-channel countermeasures," in *Proc. IEEE Design Autom. Test Europe Conf. Exhibit. (DATE)*, Mar. 2018, pp. 1253–1258.

[2] S. Ebrahimi, S. Bayat-Sarmadi, and H. Mosanaei-Boorani, "Post-quantum cryptoprocessors optimized for edge and resource-constrained devices in IoT," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5500–5507, Jun. 2019.

[3] S. Patranabis, A. Chakraborty, P. H. Nguyen, and D. Mukhopadhyay, "A biased fault attack on the time redundancy countermeasure for AES," in *Proc. Int. Workshop Constructive Side Channel Anal. Secure Design (COSADE)*, 2015, pp. 189–203.

[4] S. Patranabis, A. Chakraborty, D. Mukhopadhyay, and P. P. Chakrabarti, "Fault space transformation: A generic approach to counter differential fault analysis and differential fault intensity analysis on AES-Like block ciphers," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 5, pp. 1092–1102, May 2017.

[5] A. Aghaie, A. Moradi, S. Rasoolzadeh, A. R. Shahmirzadi, F. Schellenberg, and T. Schneider, "Impeccable circuits," *IEEE Trans. Comput.*, vol. 69, no. 3, pp. 361–376, Mar. 2020.

[6] F. Valencia, T. Oder, T. Guneysu, and F. Regazzoni, "Exploring the vulnerability of R-LWE encryption to fault attacks," in *Proc. Workshop Cryptogr. Security Comput. Syst. (CS2)*, 2018, pp. 7–12.

[7] L. G. Bruinderink and P. Pessl, "Differential fault attacks on deterministic lattice signatures," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, no. 3, pp. 21–43, 2018.

[8] T. Oder, T. Schneider, T. Poppelmann, and T. Guneysu, "Practical CCA2-secure and masked ring-LWE implementation," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, no. 1, pp. 142–174, 2018.

[9] S. Ebrahimi and S. Bayat-Sarmadi, "Lightweight and fault-resilient implementations of binary Ring-LWE for IoT devices," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6970–6978, Aug. 2020.

[10] S. Patranabis, A. Chakraborty, and D. Mukhopadhyay, "Fault tolerant infective countermeasure for AES," *J. Hardw. Syst. Security*, vol. 1, no. 1, pp. 3–17, Apr. 2017.

[11] M. Mozaffari-Kermani, R. Azarderakhsh, and A. Aghaie, "Fault detection architectures for post-quantum cryptographic stateless hash-based secure signatures benchmarked on ASIC," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 2, pp. 59:1–59:19, Dec. 2016.

[12] A. Sarker, M. Mozaffari-Kermani, and R. Azarderakhsh, "Hardware constructions for error detection of number-theoretic transform utilized in secure cryptographic architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 3, pp. 738–741, Mar. 2019.

[13] A. Sarker, M. Mozaffari-Kermani, and R. Azarderakhsh, "Error detection architectures for ring polynomial multiplication and modular reduction of ring-LWE in $\frac{\mathbb{Z}/p\mathbb{Z}[x]}{x^n+1}$, benchmarked on ASIC," *IEEE Trans. Rel.*, early access, May 20, 2020, doi: 10.1109/TR.2020.2991671.

[14] M. Mozaffari-Kermani, R. Ramadoss, and R. Azarderakhsh, "Efficient error detection architectures for CORDIC through recomputing with encoded operands," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2016, pp. 2154–2157.

[15] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A low-cost S-box for the Advanced Encryption Standard using normal basis," in *Proc. IEEE Int. Conf. Electro/Inf. Technol. (EIT)*, Windsor, ON, Canada, Jun. 2009, pp. 52–55.

[16] J. Buchmann, F. Gopfert, T. Guneysu, T. Oder, and T. Poppelmann, "High-performance and lightweight lattice-based public-key encryption," in *Proc. ACM Workshop IoT Privacy Trust Security*, 2016, pp. 2–9.

[17] F. Gopfert, C. van Vredendaal, and T. Wunderer, "A hybrid lattice basis reduction and quantum search attack on LWE," in *Proc. Int. Workshop Post Quant. Cryptogr.*, 2017, pp. 184–202.

[18] A. Aghaie, M. Mozaffari-Kermani, and R. Azarderakhsh, "Fault diagnosis schemes for low-energy block cipher Midori benchmarked on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 4, pp. 1528–1536, Apr. 2017.