Efficient Error Detection Architectures for Postquantum Signature Falcon's Sampler and KEM SABER

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

***Abstract*— Among the National Institute for Standards and Technology (NIST) postquantum cryptography (PQC) standardization Round 3 finalists (announced in 2020 and anticipated to conclude in 2022–2024), SABER and Falcon are efficient key encapsulation mechanism (KEM) and compact signature scheme, respectively. SABER is a simple and flexible cryptographic scheme, highly suitable for thwarting potential attacks in the postquantum era. Implementing SABER can be performed solely in hardware (HW) or on HW/software coprocessors. On the other hand, the compact key size, efficient design, and strong reliability proof in the quantum random oracle model (QROM) make Falcon a highly suitable signature algorithm for PQC. Although Falcon is crucial as a PQC signature scheme, the utilization of the Gaussian sampler makes it vulnerable to malicious attacks, e.g., fault attacks. This is the first work to present error detection schemes embedded efficiently in SABER as well as Falcon's sampler architectures, which can detect both transient and permanent faults. Moreover, we implement HW design for the ModFalcon signature algorithm as well as the Gaussian sampler. These schemes are implemented on a formerly Xilinx field-programmable gate array (FPGA) family, for both SABER and Falcon variants, where we assess the error coverage and the performance. The proposed schemes incur low overhead (the area, delay, and power overheads being 22.59%, 19.77%, and 10.67%, respectively, in the worst case) while providing a high fault detection rate (99.9975% in the worst case scenario), making them suitable for high efficiency and compact HW implementations of constrained applications.**

***Index Terms*— Field-programmable gate array (FPGA), key encapsulation mechanisms (KEM), learning with errors (LWEs), learning with rounding (LWR), postquantum cryptography (PQC), quantum random oracle model (QROM).**

I. Introduction

LATTICE-BASED cryptography [1] is one of the most promising classes among the National Institute for Standards and Technology (NIST) postquantum cryptography (PQC) submissions of the final round (announced in 2020). One category of lattice-based encryption schemes is learning with error (LWE)-based schemes, incorporating the worst case lattice problem. Learning with rounding (LWR) [2] is a subclass within LWE, both of their security levels relying on noise introduction. SABER is one such module-LWR [3] encryption scheme, which is resistant to 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 number-theoretic transform (NTT)-based multiplier, because of using an unconventional set compared to the popular NTT with prime parameter set [4], which has been improved by proposing a fast polynomial multiplication based on the Toom–Cook algorithm [5] in the work of [6]. Software (SW) optimization techniques of SABER have been proposed by improving the Toom–Cook multiplier [6]. The hardware/SW (HW/SW) codesign approach to accelerate the SABER computation process has been explored in [7], which achieved significant speedup compared to SW-based implementations.

Among NIST PQC competition Round 3 finalists, Falcon [8], 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 [9], the latter being provably secure and resistant to the key-recovery attack [10]. The article in [11] presented a compact and efficient instantiation of Falcon, which allows an intermediate security level. The toolchain proposed in [12] to instantiate efficient constant-time discrete Gaussian sampler, proved to be practical and secure to use as a postquantum signature algorithm, e.g., Falcon, with insignificant performance degradation compared to a nonconstant-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 article, we propose fault detection techniques for SABER, in both the full 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 HW implementation, a highly compact variant of Falcon, i.e., ModFalcon [11], as well as the sample algorithm of a constant time Gaussian sampler [12]. This is the first work on fault detection schemes of a postquantum cryptographic signature scheme. Such attacks can break into state-of-the-art signature schemes and derive sensitive information. Very

Manuscript received October 28, 2021; revised February 5, 2022; accepted March 1, 2022. Date of publication March 16, 2022; date of current version May 23, 2022. This work was supported by the U.S. National Science Foundation (NSF) under Award SaTC-1801488. *(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/TVLSI.2022.3156479

few works such as [13]–[17] 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 article are as follows.

1) We present fault detection schemes for SABER on the performance bottleneck, the pseudo random number generation (PRNG) generator involving a binomial sampler, as well as the polynomial multiplier architecture for fully HW SABER architecture.
2) 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.
3) We propose error detection schemes for the HW 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.
4) We simulate the proposed scheme by injecting faults in a Xilinx field-programmable gate array (FPGA) family. The assessment of our proposed schemes shows high error coverage.
5) 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.

The rest of the article is organized as follows: Section II summarizes the SABER key encapsulation mechanism (KEM) and Falcon signature scheme. We present our proposed error detection schemes of SABER-KEM, Falcon, ModFalcon, and Gaussian sampler in Section III. We analyze the error coverage and performance metrics in Section IV. Section V concludes the article.

II. Preliminaries

A. Recomputing Overview

Recomputing is a time redundancy technique, involving encoding (c), and decoding (d) operations of the function in question (f) (e.g., sampler and 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 nonrecomputed 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, maybe detected via recomputing.

B. SABER Overview

The security of SABER relies on the hardness of module-LWR problem, which is given by: $(\overrightarrow{a}, b = \lfloor (p/q)(\overrightarrow{a}^T \overrightarrow{s}) \rceil) \in \mathbb{R}_q^{l \times l} \times \mathbb{R}_p$, 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.

1) *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 $\overrightarrow{A}^T \overrightarrow{s}$, while the secret key consists of the secret vector $\overrightarrow{s}$.
2) *Encryption:* Encryption consists of generating a new 'secret' $\overrightarrow{s'}$ and adding the message to the inner product between the public key and the new secret $\overrightarrow{s'}$. This forms the first part of the ciphertext, while the second is used to hide the encrypting secret and contains the rounded product $\overrightarrow{A}\,\overrightarrow{s'}$.
3) *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.
4) *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, and 4, respectively, and their secrets are sampled from $[-5, 5]$, $[-4, 4]$, and $[-3, 3]$. Our HW implementation supports both LightSABER and SABER. Our error detection schemes support both LightSABER and SABER operations.

C. 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 $\boldsymbol{B} \in \mathbb{R}^{n*n}$. The Falcon signature algorithm consists of three steps, key generation, signature generation, and verification, which are described as follows.

1) *Key Generation:* In the first step of key generation, one needs to generate the polynomials, $f, g, F, G \in \mathbb{Z}[x]/\varphi$, fulfilling the NTRU equation. In the next step, Falcon tree T is constructed, through LDL* decomposition of the matrix $\boldsymbol{G} = \boldsymbol{B}\boldsymbol{B}^*$. The output of key generation is a public key $pk = h = gf^{-1} \bmod q$ and a secret key $sk = (\hat{\boldsymbol{B}}, T)$.
2) *Signature Generation:* In the first part of the signature generation, a hash value $c \in \mathbb{Z}_q[x]/\varphi$ of the message m and a salt r are computed. The short values s_1, s_2 such that $s_1 + s_2 = c \bmod q$, are computed from the hash value as well as the sk, the latter taking advantage of its knowledge about f, g, F, G, and ffSampling 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.

3) *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, β being predefined acceptance bound.

III. 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.

A. 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 [18]. 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 [19]. Such faults attacks do not tamper with the combinational circuitry of digital systems, rather alter the sampling process of the flip-flop (FF) or decreased clock period of a register, resulting in wrong output [20], [21]. 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 choose requires minimal information on faulty and fault-free computation, resembling differential fault intensity analysis (DFIA) [22]. Although the Fujisaki–Okamoto (FO) transform applied in the encryption/decryption provides redundancy through reencryption, it fails to detect recent attacks described in [23] 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 [24] and statistical ineffective fault attack (SIFA) [19].

B. 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.

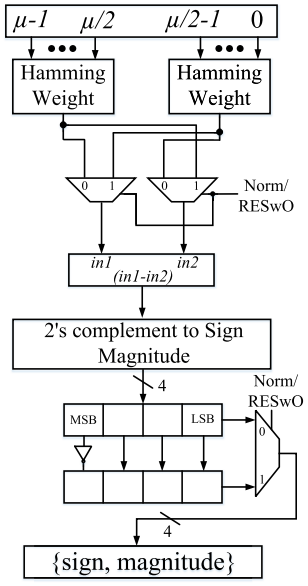


Fig. 1. Error detection architecture on binomial sampler.

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.

1) Error Detection on Binomial Sampler: The binomial sampler computes a sample from a μ-bit pseudorandom input string, e.g., $r[\mu - 1 : 0]$, by computing $\text{HW}(r[\mu/2 - 1 : 0]) - \text{HW}(r[\mu - 1 : \mu/2])$, where HW() stands for the Hamming weight (Fig. 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 Fig. 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 pseudorandom 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 Fig. 1, we implement recomputing with swapped operands (RESwOs), 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. Fig. 1 shows error detection operation for μ bits, which is replicated eight times for a 64-bit data memory output for SABER.

2) Error Detection on Parallel Polynomial Multiplication: The Toom–Cook method is proposed in the work of [6],

Algorithm 1 Schoolbook Polynomial Multiplication

Input: Two polynomials $a(x)$ and $b(x) \in \mathbb{R}_q$ of degree N
Output: The product $a(x).b(x)$ of degree N
1: $acc(x) \leftarrow 0$
2: **for** $i = 0; i < N; i = i + 1$ **do**
3: **for** $j = 0; i < N; j = j + 1$ **do**
4: $acc[j] = acc[j] + b[j].a[i] \bmod \mathbb{Z}_q$
5: **end for**
6: $b = b.x \bmod \mathbb{R}_q$;
7: **end for**
8: **return** acc

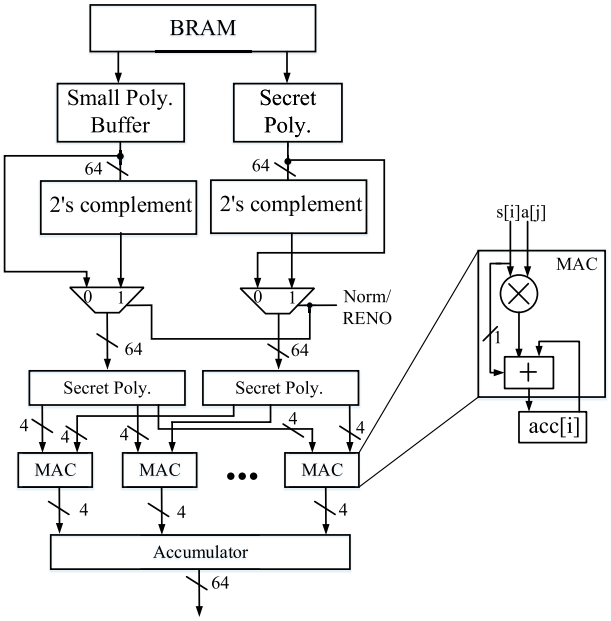


Fig. 2. Proposed error detection architecture on polynomial multiplication with MAC unit construction.

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 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 (Fig. 2), as all the bits of a register can be accessed simultaneously on a HW 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 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 Fig. 2 provides the required coefficient of $a(x)$ during the multiplication $s(x) \cdot a[i]$ by the parallel multiply-and-accumulate (MAC) cores, from the inset of Fig. 2. After the multiplication $s(x) \cdot a[i]$, $s(x)$ needs to be multiplied by x. This operation is a simple mega-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 256th coefficient, taking advantage of the sign-magnitude system representation.

The aforementioned schoolbook polynomial multiplication is one of the most exhaustive operations in this construction. Hence, we apply recomputing with negated operands (RENOs) which ensures the reliability of such architecture (Fig. 2). In the RENO cycle of the multiplexer select Norm/RENO, we perform negation of the multiplication inputs, i.e., $-s(x)$ and $-a(i)$, multiplication of which will eventually lead to the same product as Norm cycle, i.e., $s(x) \cdot a[i]$, in a fault-free scenario. In run1 and run2, the original and the recomputed polynomial multiplications are performed, respectively, and the results are then compared in the comparator unit. During permanent faults, e.g., one of the bits of MAC is stuck to 0, the output of negation operation of the faulty MAC block, after modular multiplication, will be discrepant from run1, i.e., the nonrecomputed output, as described in Section II-A. We note such discrepancies between run1 and run2 cycles, calculated in the comparator, will confirm the presence of faults. On the contrary, in the case of transient fault, the error is present in run1 and absent in run2 and vice-versa, resulting in discrepant outputs between both cycles. The 2's complement notations of the coefficients discard the need for an external negation unit, incurring low area overhead.

3) Error Detection on HW/SW Codesign: The HW/SW codesign approach is an extensively researched technique that aims to achieve performance targets through a shorter development cycle than is typical for HW-only implementations. Replacing a purely-HW benchmarking is not the intention of HW/SW benchmarking, rather, the aim is to ease the development of HW-only implementations via researching HW 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 $\boldsymbol{A}$, with each element representing a polynomial, as shown in Fig. 3. The sign-extended version of matrix $\boldsymbol{A}$ is used to generate $b' = (\boldsymbol{A}s' + h) \bmod q$, where h is a constant of the equation. Only one row of the $\boldsymbol{A}$ matrix is produced at once and the elements of $\boldsymbol{A}$ 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 Fig. 3, stores the temporary results. The MAC constructions are shown as the inset in Fig. 2.

In our scheme, we apply RENO at both the inputs of the MAC module in Fig. 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

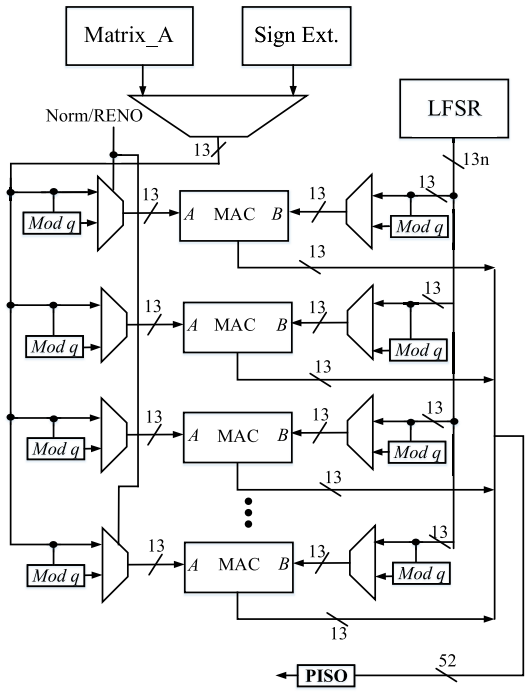


Fig. 3. Proposed error detection architecture on HW accelerator.

are MAC architecture oblivious. As the SABER decapsulation stage utilizes the same mechanism, RENO can be applied there as well to detect fault injection.

C. Error Detection Schemes on Falcon Sampler

We apply the schemes for the nonconstant 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 II, we recall the ffSampling algorithm is the basis to generate secret key s_k for signature generation. As shown in Algorithm 1, the Falcon tree generation, i.e., line 7 stating LDL* decomposition of matrix G, and the ffSampling are combined in one algorithm, namely, ffsampling$_n^*$. Such combination reduces the memory consumption significantly compared to the reference Falcon implementation. Here, we note that the three functions of Algorithm 2, i.e., ffSampling, splitfft, and mergefft are linear elementary operations: Addition, subtraction, multiplication, and division; hence, we can apply linear encoding and decoding schemes, without any loss of information.

1) Recomputing on Negation: Algorithm 2 can be partially depicted (lines 11–13) by Fig. 4(a), multiplexer select Norm/RENO being at Norm, i.e., unmodified operation of the ffsampling$_n^*$. In the original operation of line 13, the output of 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 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

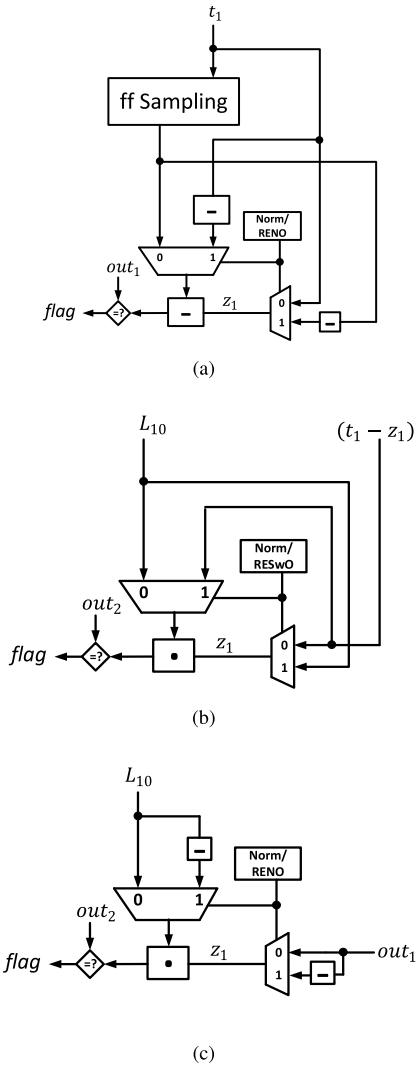


Fig. 4. Proposed recomputing-based HW architecture schemes of key generation in Falcon (a) RENOs on negation, (b) RESwOs on multiplication, and (c) RENO on multiplication.

the comparator comparing this output with out$_1$, detecting the presence of faults. We note that decoding in this scheme is free of HW cost; hence, a low-overhead and inexpensive fault detection approach.

2) RESwO on Multiplication: In line 13 of Algorithm 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 RESwOs, where the multiplication operands L_{10} and $(t_1 - z_1)$ are swapped and stored in out$_2$, as shown in Fig. 4(b). 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.

3) RENO on Multiplication: One can also explore negation on the aforementioned multiplicands. In such a case, as depicted in Fig. 4(c), 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

Algorithm 2 $\text{ffsampling}_n^*(\boldsymbol{t}, \boldsymbol{G})$

Input: $\boldsymbol{t} = (t_0, t_1) \in \text{FFT}(\mathbb{Q}[x](x^n+1))^2$ and a full-rank Gram matrix $\boldsymbol{G} \in \text{FFT}(\mathbb{Q}[x](x^n+1))^{2*2}, \sigma \in 1.55\sqrt{q}$
Output: $\boldsymbol{z} = (z_0, z_1) \in \text{FFT}(\mathbb{Z}[x](x^n+1))^2$

1: **if** $(n = 1)$ **then**
2: $\quad \sigma' \leftarrow \sigma\sqrt{G_{00}}$
3: $\quad z_0 \leftarrow D_{\mathbb{Z},t_0,\sigma'}$
4: $\quad z_1 \leftarrow D_{\mathbb{Z},t_1,\sigma'}$
5: $\quad$ **return** $\boldsymbol{z} = (z_0, z_1)$
6: **end if**
7: $\boldsymbol{L}, \boldsymbol{D} \leftarrow LDL^*(\boldsymbol{G})$
8: $d_{01}, d_{11} \leftarrow \text{splitfft}_2(D_{11})$
9: $\boldsymbol{t_1} \leftarrow \text{splitfft}_2(t_1')$
10: $\boldsymbol{G_1} \leftarrow \left[\begin{array}{c|c} d_{10} & d_{11} \\ \hline xd_{11} & d_{10} \end{array}\right]$
11: $\boldsymbol{z_1} \leftarrow \text{ffsampling}_{n/2}(\boldsymbol{t_1}, \boldsymbol{G_1})$
12: $z_1 \leftarrow \text{mergefft}_2(\boldsymbol{z_1})$
13: $t_0' \leftarrow t_0 + (t_1 - z_1) \bigodot L_{10}$
14: $d_{00}, d_{01} \leftarrow \text{splitfft}_2(D_{00})$
15: $\boldsymbol{t_0} \leftarrow \text{splitfft}_2(t_0')$
16: $\boldsymbol{G_0} \leftarrow \left[\begin{array}{c|c} d_{00} & d_{01} \\ \hline xd_{01} & d_{00} \end{array}\right]$
17: $\boldsymbol{z_0} \leftarrow \text{ffsampling}_{n/2}(\boldsymbol{t_0}, \boldsymbol{G_0})$
18: $z_0 \leftarrow \text{mergefft}_2(\boldsymbol{z_0})$
19: **return** $\boldsymbol{z} = (z_0, z_1)$

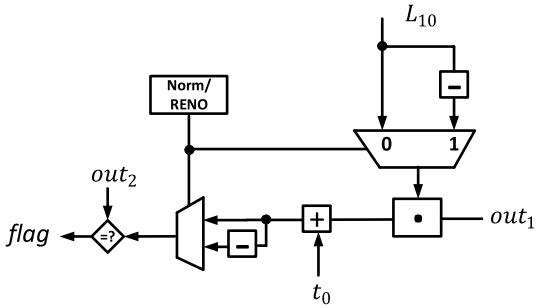


Fig. 5. RENO on multiplication-and-accumulator (MAC) module.

on both operands, resulting in $\text{out}_2 = -L_{10} \odot -\text{out}_1$. 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 $L_{10} \odot \text{out}_1$. 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.

4) RENO on Multiplication-and-Accumulator (MAC): Instead of applying error detection on either the multiplication or the subtraction of line 13 in Algorithm 2, one can perform error detection on this overall multiplication-and-accumulator circuitry. We propose RENO for MAC of line 13, where the Norm cycle of multiplexer results in t_0', according to Algorithm 2. In our proposed RENO operation, we negate both L_{10} and t_0, as shown in Fig. 5, resulting in the encoded output of $(-L_{10} \odot \text{out}_1) - t_o$, where $\text{out}_1 = (t_1 - z_1)$. We decode this encoded operand by again negating the MAC output, providing $\text{out}_2 = -(-L_{10} \odot \text{out}_1 - t_0)$, which should be identical to t_0' in a fault-free scenario and the comparator

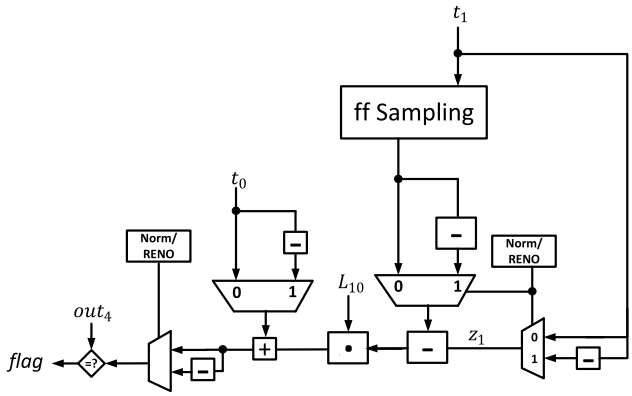


Fig. 6. RENO on the overall ffsampling_n^*.

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) RENO on Overall ffsampling$_n^*$: We finally propose an error detection scheme that operates on the inputs of the entire Algorithm 2 and performs RENO on its operands, as depicted in Fig. 6. 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^* 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 nonconstant time Gaussian sampler can easily fall victim to timing attacks and other fault attacks. However, such nonconstant time Falcon approaches are heavily researched and popular for microcontroller 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.

D. 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 nonconstant time Gaussian sampler can seriously affect the security of the scheme; thus, should be replaced with a constant-time Gaussian sampler.

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.

Algorithm 3 Signature: (sk,msg) $\rightarrow$ (r, S)

Require: A standard deviation parameter σ
1: Get $r \leftarrow U(\{0,1\})^{\lambda_r}$
2: $\mu \leftarrow H(r||msg) \in R_q$ and let $\boldsymbol{c} = (\mu, 0, ..., 0)$
3: Compute $\boldsymbol{t} = \boldsymbol{c} \cdot \boldsymbol{B}_{\boldsymbol{F,g}}^{-1}$
4: Compute $\boldsymbol{z} \in R^{n+1}$ such that $\boldsymbol{s} := (\boldsymbol{t} - \boldsymbol{z}) \cdot \boldsymbol{B}_{\boldsymbol{F,g}}$
5: $S = \text{Compress}(\boldsymbol{s})$
6: **return** the signature (r, S)

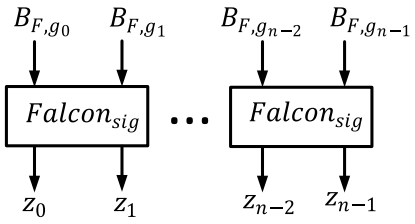


Fig. 7. Signature scheme of ModFalcon architecture.

As shown in Algorithm 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 $s \cdot vk = H(r||\text{msg})$. After computing $H(r||\text{msg})$, the secret key $\boldsymbol{B}_{\boldsymbol{F,g}}$ is used to sample a proper s. Algorithm 3 can be partially depicted (line 4) by Fig. 7. One can compute z via the parallel computations of $\text{Falcon}_{\text{sig}}$. Even constant-time Falcon can be vulnerable to fault attacks, hence Fig. 7 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 $\text{out}_1 = (t - z)$ 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 HW cost; hence, a low-overhead and inexpensive fault detection approach.

2) Sample$_z$ Implementation and Error Detection: The constant-time sampler, formally described in Algorithm 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 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 $\text{out}_2 = \{-(2b - 1) \cdot -z_0\} + b$. 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.

Algorithm 4 SamplerZ (σ, μ)

Require: $\mu \in [0, 1), \sigma \leq \sigma_0$ a scaling factor $C = C(\sigma) \in (0, 1]$
Ensure: $z \sim D_{\mathbb{Z},\sigma,\mu}$
1: **while** true **do**
2: $z_0 \leftarrow BaseSampler()$
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** $BerExp_{C(\sigma)}(x)$ **then**
7: **return** z
8: **end if**
9: **end while**

IV. 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 HW accelerator, as shown in Table I.

A. 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., 1) single; 2) two-bit; and 3) 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 multibit (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 [(faults detected)/(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.

B. 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

TABLE I

IMPLEMENTATION RESULTS FOR FPGA THROUGH XILINX ZYNQ-ULTRASCALE+ ZCU102 (XCZU9EG-FFVB1156-2-E) FOR BINOMIAL SAMPLING, POLYNOMIAL MULTIPLICATION AND HW/SW CODESIGN. ALL THE INPUTS ARE 256 bits AND THE PARENTHESES REPRESENT PERCENT OVERHEADS COMPARED TO ORIGINAL ARCHITECTURE[1]

Architecture	Scheme	Area		Delay (ns)	Power (mW)
		LUTs	FFs		
Binomial sampling	Original	85	88	2.03	0.697
	RESwO	100 (17.64%)	105 (19.32%)	2.35 (15.76%)	0.745 (6.88%)
Polynomial multiplication	Original	17,352	5,171	2.959	1.724
	RENO	20,420 (17.68%)	6,346 (22.72%)	3.297 (11.42%)	1.908 (10.67%)
Hardware/software codesign	Original	14,277	1,025	3.764	2.097
	RENO	17,502 (22.59%)	1,206 (17.66%)	4.508 (19.77%)	2.295 (9.45%)

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

as well as the original. For both cases, we tabulated both the lookup table (LUT) and FF as area overhead as well as delay and power overheads in Table I, 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 incorporated 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 postquantum architectures [13], [14], recomputing has been utilized to detect faults on NTT and ring polynomial multiplication, respectively, two integral components of lattice-based cryptosystems. The implementation overheads in the work of [13] 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 [14] 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 HW or HW/SW implementation of Falcon and other lattice-based signatures, e.g., LUOV, HQC, and NTS-KEM [25]. However, recomputing being an efficient scheme, we expect similar low overhead results for Falcon as our derived results for SABER.

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

V. CONCLUSION

We present error detection schemes for SABER on fully HW construction and HW/SW codesign accelerators. Moreover, we propose error detection schemes for postquantum 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.

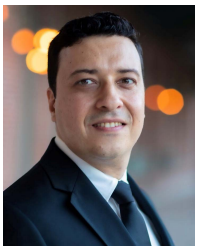
REFERENCES

[1] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann, "Practical lattice-based cryptography: A signature scheme for embedded systems," in *Proc. CHES*, Sep. 2012, pp. 530–547.

[2] H. Baan *et al.*, "Round2: KEM and PKE based on GLWR," Cryptol. ePrint Arch., Int. Assoc. Cryptol. Res., Lyon, France, Tech. Rep., 2017/1183, Feb. 2022.
[3] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom functions and lattices," in *Proc. Annu. Conf. Adv. Cryptol.*, 2012, pp. 719–737.
[4] V. Lyubashevsky and D. Micciancio, "Generalized compact knapsacks are collision resistant," in *Automata, Languages and Programming*. Cham, Switzerland: Springer, 2006, pp. 144–155.
[5] D. Knuth, *The Art of Computer Programming*, vol. 3. Boston, MA, USA: Addison-Wesley, 1997.
[6] J.-P. D'Anvers, A. K. S. S. Roy, and F. Vercauteren, "SABER: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," in *Proc. Africacrypt*, May 2018, pp. 282–305.
[7] J. M. B. Mera, F. Turan, A. Karmakar, S. S. Roy, and I. Verbauwhede, "Compact domain-specific co-processor for accelerating module lattice-based key encapsulation mechanism," Cryptol. ePrint Arch., Int. Assoc. Cryptol. Res., Lyon, France, Tech. Rep. 2020/321, Feb. 2022.
[8] T. Prest *et al.*, "Falcon," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep., Apr. 2021. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
[9] C. Gentry, C. Peikert, and V. Vaikuntanathan, "Trapdoors for hard lattices and new cryptographic constructions," in *Proc. ACM STOC*, May 2008, pp. 197–206.
[10] L. Ducas and P. Q. Nguyen, "Faster Gaussian lattice sampling using lazy floating-point arithmetic," in *Proc. ASIACRYPT*, 2012, pp. 415–432.
[11] C. Chuengsatiansup, T. Prest, D. Stehlé, A. Wallet, and K. Xagawa, "ModFalcon: Compact signatures based on module-NTRU lattices," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 853–866.
[12] A. Karmakar, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "Pushing the speed limit of constant-time discrete Gaussian sampling. A case study on the Falcon signature scheme," in *Proc. 56th Annu. Des. Autom. Conf.*, Jun. 2019, pp. 1–6.
[13] 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.
[14] 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.*, vol. 70, no. 1, pp. 362–370, Mar. 2021.
[15] A. Sarker, M. M. Kermani, and R. Azarderakhsh, "Fault detection architectures for inverted binary ring-LWE construction benchmarked on FPGA," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 4, pp. 1403–1407, Apr. 2021.
[16] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A high-performance fault diagnosis approach for the AES SubBytes utilizing mixed bases," in *Proc. Workshop Fault Diagnosis Tolerance Cryptogr.*, Sep. 2011, pp. 80–87.
[17] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "Reliable hardware architectures for the third-round SHA-3 finalist Grostl benchmarked on FPGA platform," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst.*, Oct. 2011, pp. 325–331.
[18] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Proc. Annu. Int. Cryptol. Conf.*, 1997, pp. 17–21.
[19] C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas, "SIFA: Exploiting ineffective fault inductions on symmetric cryptography," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, pp. 547–572, Aug. 2018.
[20] M. Dumont, M. Lisart, and P. Maurine, "Electromagnetic fault injection: How faults occur," in *Proc. Workshop Fault Diagnosis Tolerance Cryptogr. (FDTC)*, Aug. 2019, pp. 9–16.
[21] M. Agoyan, J. Dutertre, D. Naccache, B. Robisson, and A. Tria, "When clocks fail: On critical paths and clock faults," in *Proc. Int. Conf. Smart Card Res. Adv. Appl.*, 2010, pp. 182–193.
[22] N. F. Ghalaty, B. Yuce, M. Taha, and P. Schaumont, "Differential fault intensity analysis," in *Proc. Workshop Fault Diagnosis Tolerance Cryptogr.*, Sep. 2014, pp. 49–58.
[23] P. Pessl and L. Prokop, "Fault attacks on CCA-secure lattice KEMs," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, no. 2, pp. 37–60, Feb. 2021.
[24] F. Zhang *et al.*, "Persistent fault analysis on block ciphers," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, no. 3, pp. 150–172, Aug. 2018.
[25] V. B. Dang, F. Farahmand, M. Andrzejczak, and K. Gaj, "Implementing and benchmarking three lattice-based post-quantum cryptography algorithms using software/hardware codesign," in *Proc. Int. Conf. Field-Programmable Technol. (ICFPT)*, Dec. 2019, pp. 206–214.

Ausmita Sarker (Student Member, IEEE) received the B.Sc. degree in electrical and electronic engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2016. She is currently working toward the Ph.D. degree at the Department of Computer Science and Engineering, University of South Florida, Tampa, FL, USA.

Her research interests include cryptographic engineering, postquantum cryptography, and embedded systems.



Mehran Mozaffari Kermani (Senior Member, IEEE) received the B.Sc. degree in electrical and computer engineering from the University of Tehran, Tehran, Iran, in 2005, and the M.E.Sc. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Western Ontario, London, ON, Canada, in 2007 and 2011, respectively.

He joined Advanced Micro Devices, Santa Clara, CA, USA, as a Senior ASIC/Layout Designer, integrating sophisticated security/cryptographic capabilities into accelerated processing. In 2012, he joined the Department of Electrical Engineering, Princeton University, Princeton, NJ, USA, as an NSERC Post-Doctoral Research Fellow. From 2013 to 2017, he was an Assistant Professor with the Rochester Institute of Technology, Rochester, NY, USA, and starting 2017, he has joined the Department of Computer Science and Engineering, University of South Florida, Tampa, FL, USA, where he is currently an Associate Professor.

Dr. Mozaffari Kermani was a recipient of the prestigious Natural Sciences and Engineering Research Council of Canada Post-Doctoral Research Fellowship in 2011, the Texas Instruments Faculty Award (Douglas Harvey) in 2014, outstanding research award at College of Engineering, USF, in 2018, Nexus Initiative Global Award, in 2019, and USF university-wide Faculty Outstanding Research Achievement Award in 2021. He has been a TPC member for a number of conferences including HOST (Publications Chair), CCS (Publications Chair), DAC, DATE, RFIDSec, LightSec, WAIFI, FDTC, and DFT. He is currently serving as an Associate Editor for the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, ACM Transactions on Embedded Computing Systems, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS.

Reza Azarderakhsh (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Western University, London, ON, Canada, in 2011.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Florida Atlantic University, Boca Raton, FL, USA. His current research interests include finite field and its application, elliptic curve cryptography, pairing-based cryptography, and postquantum cryptography.

Dr. Azarderakhsh was a recipient of the NSERC Post-Doctoral Research Fellowship working in the Center for Applied Cryptographic Research and the Department of Combinatorics and Optimization, University of Waterloo. He was the Guest Editor for the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING for the special issue of Emerging Embedded and Cyber Physical System Security Challenges and Innovations (2016 and 2017). He was also the Guest Editor for the IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS for special issue on security. He is serving as an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS.