Hardware Constructions for Error Detection of Number-Theoretic Transform Utilized in Secure Cryptographic Architectures

Ausmita Sarker, Mehran Mozaffari-Kermani, and Reza Azarderakhsh

***Abstract*— Polynomial multiplication is one of the most rigorous arithmetic construction of postquantum cryptosystems. Utilizing number-theoretic transformations, the product of such multiplication can be efficiently computed in quasi-linear time $O(n.lgn)$. Error detection schemes of number-theoretic transform (NTT) architectures are essential to ensure correct mathematical operations, improved security, and thwart active side-channel attacks mounted through faults. NTT is not only significant to post-quantum cryptosystems, but the structure is also valuable to the already existing security protocols, e.g., signature schemes, hash functions, and the like. This paper, for the first time, introduces new error detection schemes of NTT architectures, successfully detecting both permanent and transient faults. Our schemes are based on recomputing with negated, scaled, and swapped operands. We have implemented the proposed schemes on the application-specific integrated circuit (ASIC). Performance and implementation metrics on this hardware platform show acceptable hardware overhead. As our schemes provide acceptable complexity and high efficiency, they can be utilized in compact hardware implementations of constrained applications, e.g., deeply embedded architectures.**

***Index Terms*— Application-specific integrated circuit (ASIC), fast Fourier transform (FFT), number-theoretic transform (NTT).**

I. Introduction

Number-theoretic transform (NTT) [1] is a discrete Fourier transform defined over a finite ring or field. Being an elegant polynomial multiplication technique, NTT is essential to postquantum 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 is learning with errors problem [2], which assures the hardness of solving other lattice problems in the worst case [3]. However, this very appealing technique gives an impractical key size of quadratic, i.e., $O(n^2)$ complexity, for security parameter n [4]. To reduce the complexity, cyclic [5] and ideal lattices [6] 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 [7] or somewhat homomorphic encryption (SHE) [8], 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 a speed advantage, because the polynomial multiplication can be efficiently computed in quasi-linear time $O(n.lgn)$ using FFT [9].

Besides postquantum cryptography, NTT can radically improve the currently used schemes by increasing their security parameters. For example, NTT proves to be a valuable tool to signature schemes [10], collision-resistant hash functions [11], as well as identification schemes [12]. As a result, efficient error detection schemes of NTT in polynomial multiplication will boost the security and reliability of postquantum cryptography as well as existing cryptosystems.

Previous studies of NTT-based polynomial multiplication have dealt with reconfigurable hardware [13] and efficient architecture to achieve high speed [14]. Examples of other interesting recent works related to the respective implementations include [15] and [16]. However, no work is yet proposed in the open literature focusing on error detection of NTT polynomial multiplier.

Error detection in cryptography has been the center of attention in previous work [17]–[25]. In this paper, we propose error-detection schemes of NTT polynomial multiplier. The main contributions of this paper are summarized as follows.

1) 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.
2) 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.
3) We implement the proposed error-detection architectures on application-specific integrated circuit (ASIC) for a 65-nm library to assess the implementation and performance metrics.

The rest of the paper is organized as follows. Section II reviews the relevant details on efficient computation of NTT. Section III presents our motivation for efficient fault detection as well as our proposed error-detection schemes. Hardware implementations on ASIC along with their overheads are given in Section IV. Finally, Section V concludes this paper.

II. Preliminaries

In this paper, 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 $f(x) = f_0 + f_1x + f_2x^2 + ... + f_{n-1}x^{n-1}$. Also, n is a power of 2, and p is a prime number where $p \equiv 1 \bmod 2n$. Multiplication of two polynomials $a(x)$, $b(x) \in \mathbb{Z}_p$, can be represented as: $a(x).b(x) = \sum_{i=0}^{n-1}\sum_{i=0}^{n-1} a_i b_j x^{i+j} \mod f(x)$, taking quadratic complexity of $O(n^2)$ utilizing school book algorithm.

On the contrary, NTT is a discrete Fourier transform, defined in a finite field, $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}[x]$ [1]. For a given primitive nth root of unity in $\mathbb{Z}_p$, $A(x)$ and $B(x)$ are the polynomials under $\mathbb{Z}_p$, where both are generic forward NTT_ω (a) and NTT_ω (b), respectively: $A_i = \mathrm{NTT}^n_\omega(a(x))_i = \sum_{j=o}^{n-1} a_j\omega^{ij} \bmod p$, $i = 0, 1, \ldots, n-1$.

Manuscript received June 27, 2018; revised October 4, 2018; accepted November 9, 2018. Date of publication December 3, 2018; date of current version February 22, 2019. This work was supported by the U.S. National Science Foundation under Award SaTC-1801488. *(Corresponding author: Mehran Mozaffari-Kermani.)*

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

R. 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).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2881097

Algorithm 1 Iterative-NTT

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 $\omega \in \mathbb{Z}_p$
Output: $y = \text{NTT}_\omega(a)$

1: $A \leftarrow$ Bit-reverse(a); $m \leftarrow 2$
2: **while** $m \leq N$ **do**
3: $s \leftarrow 0$
4: **while** $s < N$ **do**
5: **for** i to $m/2 - 1$ **do**
6: $N \leftarrow i.n/m$; $a \leftarrow s + i$; $b \leftarrow s + i + m/2$
7: $c \leftarrow A[a]$; $d \leftarrow A[b]$
8: $A[a] \leftarrow c + \omega^{N \bmod n} d \bmod p$
9: $A[b] \leftarrow c - \omega^{N \bmod n} d \bmod p$
10: **end for**
11: $s \leftarrow s + m$
12: **end while**
13: $m \leftarrow m.2$
14: **end while**
15: **return** A

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 ω with ω^{-1} and introducing n^{-1}, i.e., $a_i = \text{INTT}^n_\omega(A(x))_i = n^{-1} \sum_{j=o}^{n-1} A_j \omega^{-ij} \bmod p$, $i = 0, 1, \ldots, n-1$. As p is a prime, the inverse of n, n^{-1} can be computed in modulo p, where $n.n^{-1} \equiv 1 \bmod p$. Applying NTT and INTT to compute polynomial multiplication reduces the time complexity from $O(n^2)$ to $O(n.lgn)$.

III. 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.

A. Efficient NTT Implementation

In Algorithm 1 [26], 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 $\omega^{N \bmod n}$ 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 $(O(n.lgn))$.

B. 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.

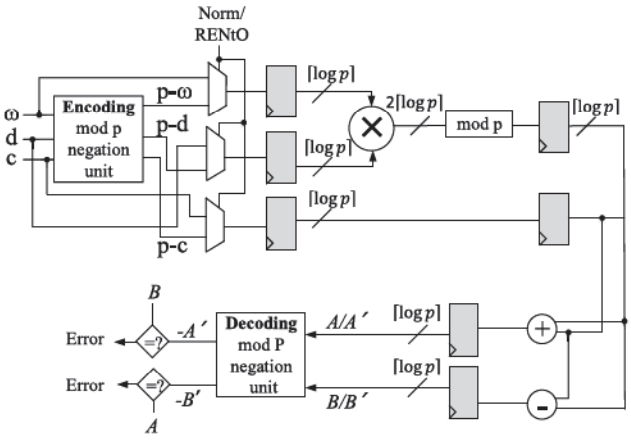

Fig. 1. Proposed butterfly construction for NTT through RENtO.

The architecture for NTT consists of the common butterfly structure (lines 8 and 9 of Algorithm 1). This well-known structure performs the core operation of NTT implementation, multiplying elements of the polynomial by powers of ω. Each cycle computes one node of NTT flow, where a multiplier, followed by a modular reduction (mod p block in Fig. 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 dual operands (RENdO) in which, as the name suggests, two operands are negated. The second one, shown in Fig. 1, is recomputing with negated tri operands (RENtO), in which all three operands, i.e., c, ω, 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 the original NTT or RENtO operation. In accordance with lines 8 and 9 of Algorithm 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, ω 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 prerecomputed 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 Fig. 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.

C. 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 ω and d, and deriving the quadruple of c. The encoded operands would be $A' = 4c + (2\omega * 2d)$ and $B' = 4c - (2\omega * 2d)$. 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,

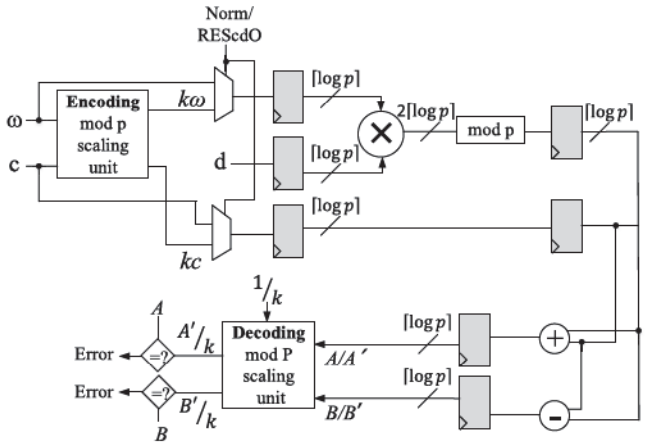

Fig. 2. Proposed butterfly construction for NTT through REScdO.

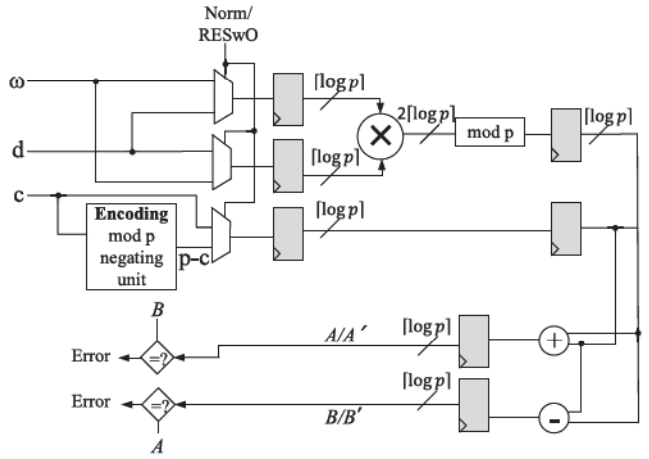

Fig. 3. Proposed butterfly construction for NTT through RESwO.

doubling only ω 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 ω and c by the factor k. This is shown in Fig. 2. The encoding operations would give $A' = kc + (k\omega) * d = k(c + \omega d)$ and $B' = kc - (k\omega) * d = k(c - \omega d)$. Decoding is performed by dividing both operands with k (Fig. 2). As p is a prime number, $\gcd(k, p) \equiv 1 \bmod p$, for all values of k.

D. Recomputing With Swapped Operands

If we swap ω 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 Fig. 3, there is no necessity for decoding, and RESwO just requires comparison with alternate precomputed 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 Fig. 3, in which we lower the overhead by swapping just ω and d, having c intact. This would result in even lower overhead as decoding would be free in hardware.

TABLE I

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)

Architecture	Area (μm^2)	Delay (ns)	Power (mW)
Original $(n,\ p)_1$	2,942	12.24	0.047
RESwO $(n,\ p)_1$	3,674 (24%)	13.37 (9%)	0.054 (16%)
RESwO-m $(n,\ p)_1$	3,544 (20%)	13.19 (8%)	0.052 (12%)
Original $(n,\ p)_2$	8,995	13.80	0.093
RESwO $(n,\ p)_2$	11,170 (24%)	14.41 (4%)	0.111 (18%)
RESwO-m $(n,\ p)_2$	11,001 (22%)	14.23 (3%)	0.108 (16%)
Original $(n,\ p)_3$	30,829	14.76	0.207
RESwO $(n,\ p)_3$	37,476 (22%)	15.90 (8%)	0.231 (15%)
RESwO-m $(n,\ p)_3$	35,972 (17%)	15.45 (5%)	0.228 (11%)

IV. 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 Very High Speed Integrated Circuit Hardware Description Language with TSMC 65 nm for three pairs of (n, p) and two of our architectures to assess the overhead in Table I. 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 paper. 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 I.

As shown in Table I, the area (in terms of μm^2), delay (which is an indication of maximum working frequency), and power consumption at the frequency of 50 MHz 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% 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 and ASIC libraries. We also note that the throughput and frequency overhead can be alleviated through pipelining at the expense of added hardware overhead.

V. Conclusion

In this paper, we have presented a number of categories for error-detection schemes of NTT in the ring $\mathbb{R} = (\mathbb{Z}/p\mathbb{Z}[x]/x^n + 1)$, which are also platform-oblivious. The proposed schemes constitute error-detection architectures on hardware based on recomputation 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 65-nm library to assess the implementation and performance metrics. With high error coverage, the presented approaches achieve an acceptable overhead (at most 24% area, 18% power consumption, and 9% delay for the synthesized case studies) and can be tailored toward the objectives in terms of error detection and reliability.

References

[1] J. M. Pollard, "The fast Fourier transform in a finite field," *Math. Comput.*, vol. 25, no. 114, pp. 365–374, 1971.

[2] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Advances in Cryptology-EUROCRYPT*. Cham, Switzerland: Springer, 2013, pp. 319–339.

[3] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *Proc. Annu. ACM Symp. Theory Comput.*, 2005, pp. 84–93.

[4] D. Micciancio and O. Regev, "Lattice-based cryptography," in *Post-Quantum Cryptography*. Berlin, Germany: Springer, 2009, pp. 147–191.

[5] D. Micciancio, "Generalized compact knapsacks, cyclic lattices, and efficient one-way functions," *Comput. Complex.*, vol. 16, no. 4, pp. 365–411, 2007.

[6] V. Lyubashevsky and D. Micciancio, "Generalized compact knapsacks are collision resistant," in *Automata, Languages and Programming*. Cham, Switzerland: Springer, 2006, pp. 144–155.

[7] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Symp. Theory Comput.*, 2009, pp. 169–178.

[8] K. Lauter, M. Naehrig, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proc. 3rd ACM Workshop Cloud Comput. Secur. Workshop*, 2011, pp. 113–124.

[9] J. W. Cooley and J. W. Turkey, "An algorithm for the machine calculation of complex Fourier serie," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, 1965.

[10] V. Lyubashevsky, "Fiat-shamir with aborts: Applications to lattice and factoring-based signatures," in *Advances in Cryptology—ASIACRYPT*. Cham, Switzerland: Springer, 2009, pp. 598–616.

[11] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen, "SWIFFT: A modest proposal for FFT hashing," in *Fast Software Encryption*. Cham, Switzerland: Springer, 2008, pp. 54–72.

[12] V. Lyubashevsky, "Lattice-based identification schemes secure under active attacks," in *Proc. Int. Workshop Public Key Cryptogr.*, 2008, pp. 162–179.

[13] T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *Progress in Cryptology—LATINCRYPT*. Cham, Switzerland: Springer, 2012, pp. 139–158.

[14] D. D. Chen *et al.*, "High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 1, pp. 157–166, Jan. 2015.

[15] C. P. Rentería-Mejía and J. Velasco-Medina, "High-throughput ring-LWE cryptoprocessors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 8, pp. 2332–2345, Aug. 2017.

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

[17] X. Guo, D. Mukhopadhyay, C. Jin, and R. Karri, "Security analysis of concurrent error detection against differential fault analysis," *J. Cryptograph. Eng.*, vol. 5, no. 3, pp. 153–169, 2015.

[18] M. Yasin, B. Mazumdar, S. Subidh Ali, and O. Sinanoglu, "Security analysis of logic encryption against the most effective side-channel attack: DPA," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFTS)*, Oct. 2015, pp. 97–102.

[19] 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. Electron. Test.*, vol. 25, nos. 4–5, pp. 225–245, Aug. 2009.

[20] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "Concurrent structure-independent fault detection schemes for the advanced encryption standard," *IEEE Trans. Comput.*, vol. 59, no. 5, pp. 608–622, May 2010.

[21] 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.

[22] M. Mozaffari-Kermani, R. Azarderakhsh, and A. Aghaie, "Reliable and error detection architectures of pomaranch for false-alarm-sensitive cryptographic applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 12, pp. 2804–2812, Dec. 2015.

[23] M. Mozaffari-Kermani, R. Azarderakhsh, A. Sarker, and A. Jalali, "Efficient and reliable error detection architectures of Hash-Counter-Hash tweakable enciphering schemes," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 2, pp. 54:1–54:19, May 2018.

[24] 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. (FDTC)*, Nara, Japan, Sep. 2011, pp. 80–87.

[25] 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. (DFT)*, Vancouver, BC, Canada, Oct. 2011, pp. 325–331.

[26] R. E. Blahut, *Fast Algorithms for Signal Processing*. Boston, MA, USA: Addison-Wesley, 1985.