Fault Detection Architectures for Post-Quantum Cryptographic Stateless Hash-Based Secure Signatures Benchmarked on ASIC

MEHRAN MOZAFFARI-KERMANI, REZA AZARDERAKHSH, and ANITA AGHAIE,
Rochester Institute of Technology

Symmetric-key cryptography can resist the potential post-quantum attacks expected with the not-so-faraway advent of quantum computing power. Hash-based, code-based, lattice-based, and multivariate-quadratic equations are all other potential candidates, the merit of which is that they are believed to resist both classical and quantum computers, and applying "Shor's algorithm"—the quantum-computer discrete-logarithm algorithm that breaks classical schemes—to them is infeasible. In this article, we propose, assess, and benchmark reliable constructions for stateless hash-based signatures. Such architectures are believed to be one of the prominent post-quantum schemes, offering security proofs relative to plausible properties of the hash function; however, it is well known that their confidentiality does not guarantee reliable architectures in the presence natural and malicious faults. We propose and benchmark fault diagnosis methods for this post-quantum cryptography variant through case studies for hash functions and present the simulations and implementations results (through application-specific integrated circuit evaluations) to show the applicability of the presented schemes. The proposed approaches make such hash-based constructions more reliable against natural faults and help protecting them against malicious faults and can be tailored based on the resources available and for different reliability objectives.

CCS Concepts: • **Security and privacy** → **Cryptanalysis and other attacks**; *Hash functions and message authentication codes*; • **Hardware** → *Application specific integrated circuits*; *Online test and diagnostics;*

Additional Key Words and Phrases: Application-specific integrated circuit (ASIC), secure hash-based signatures, reliability

ACM Reference Format:
Mehran Mozaffari-Kermani, Reza Azarderakhsh, and Anita Aghaie. 2016. Fault detection architectures for post-quantum cryptographic stateless hash-based secure signatures benchmarked on ASIC. ACM Trans. Embed. Comput. Syst. 16, 2, Article 59 (November 2016), 19 pages.
DOI: http://dx.doi.org/10.1145/2930664

1. INTRODUCTION

It is expected that the first generation of quantum computers is developed in the near future, suggesting secure and efficient post-quantum cryptography solutions. It is expected that even current efficient state-of-the-art cryptographic primitives such as those based on elliptic curve cryptography can be compromised and, thus,

This work was supported by the Texas Instruments faculty award granted to the authors.
A preliminary version of this work was presented in the Defect and Fault Tolerance in VLSI and Nanotechnology Systems Symposium (DFT'15).
Authors' addresses: M. Mozaffari-Kermani, Electrical and Microelectronic Department, Rochester Institute of Technology, Rochester, NY 14610; email: m.mozaffari@rit.edu; R. Azarderakhsh, Department of Computer and Electrical Engineering and Computer Science and is an I-SENSE Fellow, Florida Atlantic University, Boca Raton, FL, USA; email: razarderakhsh@fau.edu; A. Aghaie, Electrical and Microelectronic Department, Rochester Institute of Technology, Rochester, NY 14610; email: aa6964@rit.edu.

DOI: http://dx.doi.org/10.1145/2930664

post-quantum security mechanisms need to be well in place to thwart post-quantum attacks (based on "Shor's algorithm" [Bernstein et al. 2009; Shor 1997]. Indeed, elliptic curve cryptography would be broken in polynomial time by Shor' s algorithm, and scaling up to secure parameters seems impossible as the respective amount of time's polynomial is too small.

Shor's quantum algorithm [Bernstein et al. 2009; Shor 1997] is not applicable to post-quantum approaches such as lattice-based [Peikert 2014; Guneysu and Lyubashevsky 2012], hash-based [Bernstein et al. 2015], code-based [Overbeck and Sendrier 2009], multivariate-quadratic equations [Ding and Schmidt 2005], and symmetric-key cryptography. This, along with the fact that another quantum algorithm, "Grover' s algorithm" [Grover 1996], is partially applicable yet comparably very slow, makes the aforementioned post-quantum cryptography solutions attractive.

Hash-based signature schemes have small architectures and key sizes and are viable solutions, where every signature scheme uses a cryptographic hash function. Relativity of security proofs to the properties of hash functions and the obtained interesting results [Song 2014] (which suggest that the classical, generic construction of hash-tree-based signatures from one-way functions carry over to the quantum setting) have motivated the use of hash-based signatures security in the presence of quantum adversaries. We note that this matter is yet to be proven for many other post-quantum signature proposals, and that has been a motivating factor for this research, for example, lattice-based signature schemes are reasonably fast, yet their quantitative security levels are highly unclear.

Active side-channel analysis attacks (where maliciously injected faults make the output erroneous through which side-channel information is leaked [Boneh et al. 1997]), natural faults, and potential countermeasures against them have been studied in classical cryptography. A paramount cause leading to natural faults in very-large-scale integration (VLSI) constructions is hardware failures (caused by alpha particles from cosmic rays creating energetic neutrons, thermal neutrons, and the like), for instance, natural VLSI single event upsets, or electromagnetic waves. Previous research work on (a) specific cryptographic architectures such as the Advanced Encryption Standard (AES) [Tunstall et al. 2011; Ali et al. 2013; Mozaffari Kermani and Reyhani-Masoleh 2006; Mozaffari Kermani and Reyhani-Masoleh 2007; Mozaffari Kermani and Reyhani-Masoleh 2011; Yen and Wu 2006; Malkin et al. 2006; Di Natale et al. 2009; Mozaffari Kermani and Reyhani-Masoleh 2010; Guo et al. 2015; Mozaffari Kermani and Reyhani-Masoleh 2011; Guo and Karri 2014; Guo et al. 2014], (b) general cryptographic architectures [Maistri and Leveugle 2008; Guo and Karri 2013], (c) reliable architectures for lightweight cryptography [Mozaffari Kermani and Azarderakhsh 2013; Mozaffari-Kermani et al. 2014; Mozaffari-Kermani et al. 2015; S. Bayat-Sarmadi et al. 2014], and (d) finite field arithmetic architectures [Fenn et al. 1998; Bayat-Sarmadi and Hasan 2007] provide reliability mechanisms for crypto-systems.

Fault diagnosis for natural faults has been the center of attention in cryptography research; yet it is now well known that malicious intents of the attackers need to be taken into account in presenting fault detection schemes [Guo et al. 2015]. In other words, providing fault diagnosis mechanisms that are proven to be efficient with respect to natural faults may fail to detect intelligent fault attacks. For instance, if parity (or signatures detecting specific faults) is utilized for the architectures of cryptographic primitives, single stuck-at faults are detected, while even faults are not detected and other mechanisms such as interleaved parity schemes need to be used. In general, detection of randomly distributed natural faults may fail to alert fault attacks.

This work presents error detection schemes for the inner architectures of hash-based signatures for post-quantum resistivity. We note that our choice among other post-quantum mechanisms is due to the aforementioned security and key-size reasons;

yet it does not confine the proposed methods to be applicable to other post-quantum approaches. We refrain proposing error detection *"stateful"* signature mechanisms, that is, reading a secret key/message and generating a signature and an updated secret key, as if the secret key update fails, then security disintegrates [Bernstein et al. 2014; Bernstein et al. 2015]. Therefore, stateless post-quantum hash-based signatures (using parameters that provide 2^{128} security against quantum attacks, practical usage models, and low-cost implementations) are considered and the respective diagnosis approaches are presented. We focus on full binary hash tree constructions within such signature schemes to provide error detection approaches. Due to technical constraints, an attacker may not be able to inject a single-bit fault; therefore, in practice, multiple faults occur and this is considered in the (transient and permanent) fault model assessed throughout this article. The ratios of the errors detected differ depending on the error detection methods taken. Although we focus on VLSI defects, the high error coverage of the presented schemes would increase the difficulty for potential fault attackers. Specifically, our main contributions are presented as follows.

—In this article, we present schemes for detecting faults in the hardware implementations of hash trees in hash-based signature constructions. We start with presenting signature-based schemes for randomly distributed faults and then propose recomputing with encoded operands schemes (including swapped nodes, rotated operands, and the like) that are proven to be more robust with respect to fault attacks for the case study of the AES [Guo et al. 2015]. Our aim is to cover both natural faults and detect fault attacks for the hash-based signatures presented in this article; yet it is not always possible to cover both cases, and evaluations are needed.
—We present two different mechanisms, that is, structure-dependent and -oblivious detection schemes, and fault resilience approaches of such signatures are presented. We use two different hash functions for the algorithm-dependent schemes among the current state-of-the-art lightweight hash functions. These include two hash functions (one based on Permutation-Sponge and the other based on HAsh Iterative FrAmework (HAIFA) construction) for which fault diagnosis is performed.
—We evaluate the false-alarm resiliency of the architectures as well as simulation-based approaches to benchmark the efficiency of the proposed methods. Through error simulations, the error coverage for the proposed schemes is derived, and it is shown that, with high error coverage, reliable architectures are devised.
—Finally, we present, through application-specific integrated circuit (ASIC) evaluations, the overheads of the proposed approaches. These enable performance/implementation/reliability compromise, based on reliability requirements, overhead tolerance, and security objectives.

The remainder of this article is organized as follows. In Section 2, we present the preliminaries related to post-quantum hash-based signatures. In Sections 3 and 4, the proposed error detection approaches are discussed. The results of the fault injection simulations and false-alarm resiliency are provided in Section 5. The ASIC implementations and benchmark are presented in Section 6. Finally, in Section 7, we conclude the article.

2. PRELIMINARIES

In this section, through three subsections, we present (a) preliminaries related to hash-based signatures and their stateless variants [Bernstein et al. 2014; Bernstein et al. 2015], (b) ChaCha (Salsa's Variant) stream cipher [Bernstein 2007; Bernstein 2008] as the baseline for BLAKE hash function [Aumasson et al. 2010], and (c) the SPONGENT hash function [Bogdanov et al. 2013].

2.1. Hash-Based Signatures

Through the Lamport scheme [Lamport 1979], hash-based signatures are proposed that include public key, consisting of two hash outputs for secret inputs. Specifically, in Lamport's scheme, the public key consists of two hash outputs for secret inputs; to sign bit 0 (or 1), reveal the preimage of the first (second) output, consecutively. In Merkle's scheme (commonly referred to as the Merkle tree) [Merkle 1990], the process starts with a one-time signature scheme (OTS) and continues through authenticating 2^h key pairs using a binary hash tree of height h.

In the Merkle tree, the leaves are the hashes of the OTS public keys; nevertheless, the OTS secret keys become the secret key of the new scheme, and the root of the tree the public key. The merit of this approach is its small signatures and secret/public keys; nevertheless, the key generation and signature time are exponential in h (recent, practical solutions resolve this problem [Hlsing et al. 2013; Buchmann et al. 2011]).

Stateless hash-based signature schemes have been introduced and used for providing hash-based security mechanisms. In previous work, a binary certification tree built from one-time signature keys is used [Goldreich 2004]. For this scheme, key generation requires a single OTS key generation. Signing takes $2n$ OTS key generations and n OTS signatures (can be done in reasonable time for secure parameters). Furthermore, this approach proposes randomized leaf selection, that is, instead of applying a public hash function to the message, it selects an index randomly.

2.2. Reference Stateless Hash-Based Signature

Our reference stateless hash-based signature presented in Bernstein et al. [2015] is as follows. The functions in SPHINCS include the following: (a) Two short-input cryptographic hash functions $F : \{0,1\}^n \rightarrow \{0,1\}^n$ and $H : \{0,1\}^{2n} \rightarrow \{0,1\}^n$, (b) one arbitrary-input randomized hash function $\mathcal{H} : \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^m$ for $m = poly(n)$, (c) a family of pseudorandom generators $G_\lambda : \{0,1\}^n \rightarrow \{0,1\}^{\lambda n}$, (d) an ensemble of pseudorandom function families $\mathcal{F}_\lambda : \{0,1\}^\lambda \times \{0,1\}^n \rightarrow \{0,1\}^n$, and (e) a pseudorandom function family $\mathcal{F} : \{0,1\}^* \times \{0,1\}^n \rightarrow \{0,1\}^{2n}$ that supports arbitrary input lengths.

It is noted that full binary hash trees are utilized in the construction as well. In SPHINCS, a binary hash tree of height h has 2^h leaves that are n-bit strings $L_i, i \in [2^h - 1]$. Each node $N_{i,j}$ ($0 < j \leq h$ and $0 \leq i < 2^{h-j}$) of the tree stores an n-bit string. Moreover, root computation is performed as presented in Algorithm 1 [Bernstein et al. 2015]. In this algorithm, $Auth_i$ consists of all the sibling nodes of the nodes contained in the path from L_i to the root; moreover, throughout the article, the XOR function is denoted as $\oplus$ and concatenation as $||$. Finally, we note that we have the construction for binary tree hash so the values for internal nodes are $N_{i,j} = H((N_{2i,j-1} \,||\, N_{2i+1,j-1}) \oplus Q_j)$, where $Q_j \in \{0,1\}^{2n}$ are h masks. Our focus is hash-tree constructions; for a more

ALGORITHM 1: Root Computation.

Input: Leaf index i, leaf L_i, authentication path $Auth_i = (A_0, \ldots, A_{h-1})$ for L_i.
Output: Root node of tree that contains L_i.
Initialize: $P_0 \leftarrow L_i$.
[1] for $j = 0$ to h do
[2] If $\lfloor i/2^{j-1} \rfloor \equiv 0 \bmod 2$ then
[3] $P_j = H((P_{j-1} \,||\, A_{j-1}) \oplus Q_j)$.
[4] If $\lfloor i/2^{j-1} \rfloor \equiv 1 \bmod 2$ then
[5] $P_j = H((A_{j-1} \,||\, P_{j-1}) \oplus Q_j)$.
[6] end for.
Return P_h.

detailed introduction to SPHINCS including its parameters, message signer Hash to Obtain Random Subset with Trees (HORST), and the Winternitz one-time signature (WOTS$^+$), one can refer to Bernstein et al. [2015].

2.3. BLAKE Hash Function

In this article, we utilize the BLAKE hash algorithm [Aumasson et al. 2010] as the algorithm used in the hash-tree constructions that are constructed using the concepts of ChaCha, a variant of the Salsa20 stream cipher to increase the amount of diffusion per round. Dan Bernstein's SPHINCS construction is based on the hash ($\mathcal{H}$) BLAKE algorithm and utilizes ChaCha for deriving two short-input cryptographic hash functions (F and H) mentioned in the previous section. The core operation in BLAKE is ChaCha's quarter round (function $G(a, b, c, d)$) as follows: $a \leftarrow a + b, d \leftarrow (d \oplus a) \lll 16, c \leftarrow c + d, b \leftarrow (b \oplus c) \lll 12, a \leftarrow a + b, d \leftarrow (d \oplus a) \lll 8, c \leftarrow c + d, b \leftarrow (b \oplus c) \lll 7$, where $\lll$ denotes rotation towards the most significant bits. In addition, $+$ and $\oplus$ represent mod-2^{32} addition and 32-bit XOR operations, respectively.

ChaCha20 uses 10 iterations of the double round. Google has selected ChaCha20 along with Bernstein's Poly1305 message authentication code as a replacement for RC4 in OpenSSL. As of 2014, almost all HTTPS connections made from Android devices to Google properties have used the new cipher suite; Google plans to make it available as part of the Android platform.

2.4. SPONGENT Hash Function

SPONGENT is a sponge construction that, given a finite number of input bits, produces an n-bit hash value. There are three phases in SPONGENT: (a) an initialization phase that includes message padding, and then the message is cut into blocks of r bits; (b) an absorbing phase in which the r-bit input message blocks are XORed into the first r bits of the state, interleaved with applications of the permutation π_b (operating on a state of a fixed number b of bits); and (c) a squeezing phase where the first r bits of the state are returned as output, interleaved with applications of the permutation π_b until n bits are returned.

In its permutation stage, SPONGENT has the following building blocks:

—sBoxLayer$_b$: Includes a 4-bit to 4-bit S-box applied $b/4$ times.
—pLayer$_b$: Bit permutations are performed where they move bit j of the state to bit position $P_b(j) = j.b/4 \bmod b - 1$ for $j \in \{0, \ldots, b-2\}$ and to bit position $P_b(j) = b - 1$ if $j = b - 1$.
—lCounter$_b$: This is one of the four linear-feedback shift registers (LFSRs), clocked once every time its state has been used, and its final value is all ones.

3. ALGORITHM-OBLIVIOUS RECOMPUTING WITH SWAPPED NODES

In this article, two different error detection approaches are proposed, that is, a hash algorithm-oblivious scheme in which the fault diagnosis is performed independent of the hash function used and mechanisms for detection of errors in ChaCha stream cipher and SPONGENT. We would like to emphasize that although we use ChaCha and SPONGENT in the second scheme, the proposed methods are applicable with slight modifications to (a) similar stream ciphers and (b) hash functions that are based on these algorithms or their variants.

The error detection approaches presented in this section are based on two new schemes, that is, recomputing with swapped nodes (RESN) in the hash-tree constructions and combined signatures. We note that recomputing with encoded operands (REEO) is used in the next section through algorithm-dependent approaches. This

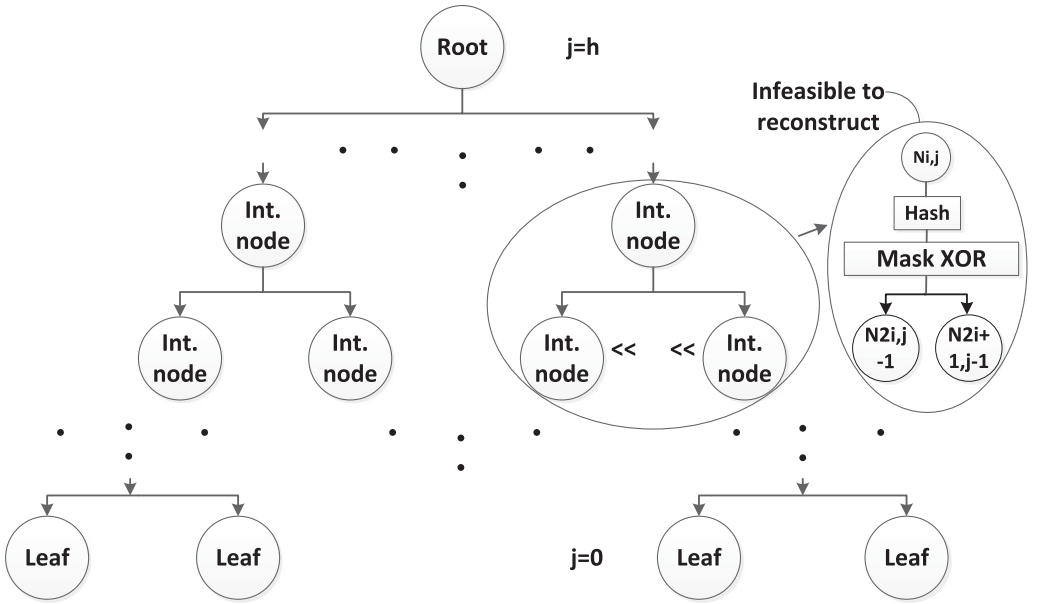

Fig. 1. Inapplicability of recomputing with rotated operands for hash tree constructions.

scheme is a subset of recomputing with permuted operands scheme [Guo et al. 2015]; yet this is a case study for implementations and the scheme is a general one.

The hash tree construction with the values for internal nodes as $N_{i,j} = H((N_{2i,j-1} \,||\, N_{2i+1,j-1}) \oplus Q_j)$, where $Q_j \in \{0,1\}^{2n}$ are h masks and those for leaf nodes as $N_{i,0} = L_i$ is used in post-quantum cryptographic hash-based signature systems to eventually compute the root, that is, $N_{0,h}$. We note that before applying the hash function to the concatenation of two child nodes to compute their parent, both child nodes are XORed with a randomly chosen mask, that is, $Q_j \in \{0,1\}^{2n}$. In what follows, full binary trees, unbalanced trees, and the relevant discussions on their error detection are presented.

3.1. Full Hash Tree Constructions

In proposing error detection schemes for hash tree constructions, we devise hash algorithm-oblivious schemes that are capable of detecting errors for any utilized hash function. A clear advantage here is that this does not confine us to a specific hash algorithm while achieving high permanent and transient error coverage. We do not use regular time-redundancy schemes because of their inability to detect permanent faults, and, instead, we investigate various recomputations applicable to hash trees.

Schemes such as recomputing with rotated/shifted operands are not applicable to hash trees. The reason is that the n-bit strings in the nodes, for example, nodes $N_{2i,j-1}$ and $N_{2i+1,j-1}$, are concatenated and modulo-2 added with the respective randomly chosen mask, that is, $Q_j \in \{0,1\}^{2n}$, and then hashed; thus, a shift or rotation in any of these nodes creates a new $2n$-bit value to be added (modulo-2) and hashed, whose output reconstruction is not readily known. Figure 1 shows a hash tree construction and details on the inapplicability of a chosen recomputation scheme, that is, recomputing with rotated operands to left. As seen in this figure, typical internal nodes are shown along with hash function and mask modulo-2 addition, where rotated operands (strings) are shown by the $\ll$ symbol adjacent to the nodes. Moreover, in Figure 2, swapping

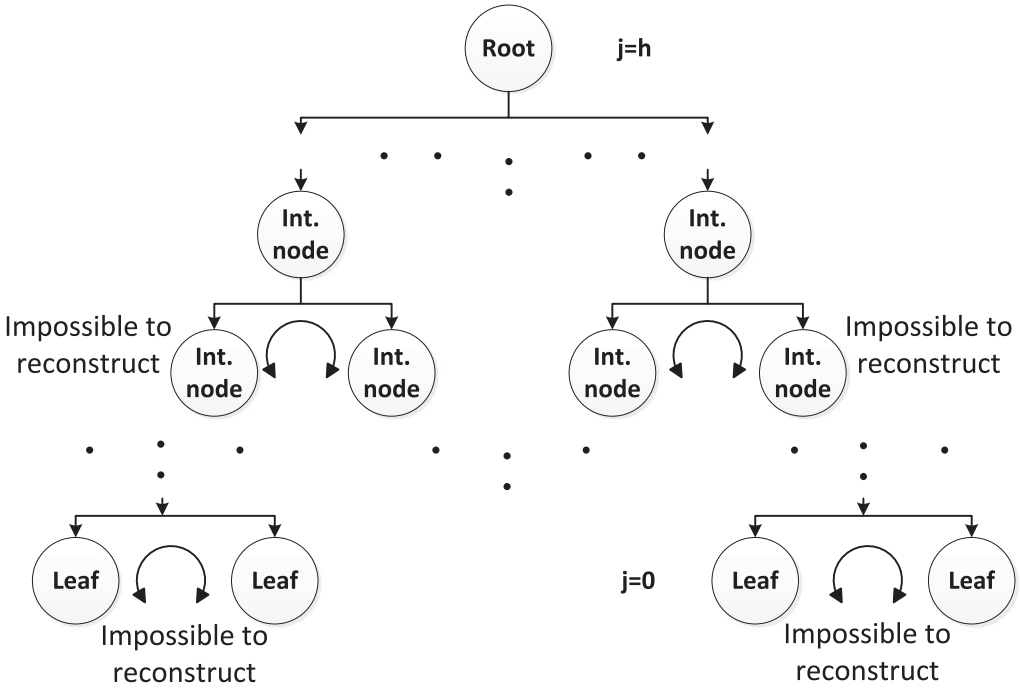

Fig. 2. Inapplicability of swapping adjacent nodes.

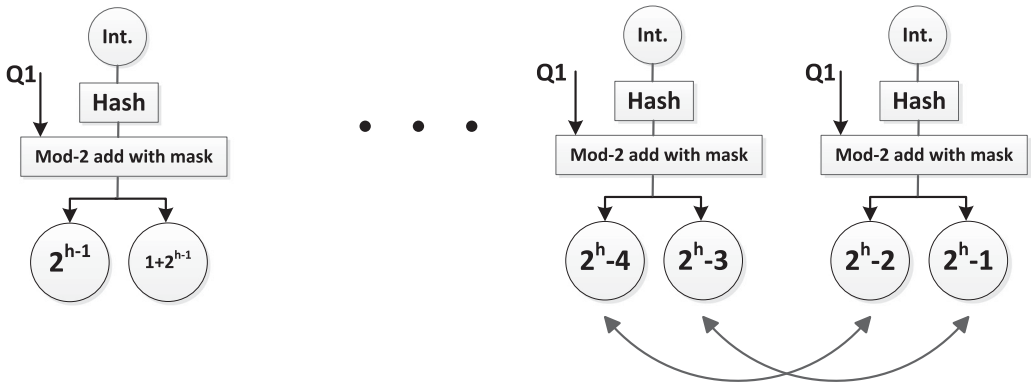

Fig. 3. The high-level structure of the proposed RESN scheme.

adjacent nodes results in swapped data blocks after concatenation that, in turn, after hashing, is not suitable to reconstruct.

Let us present our proposed scheme based on RESN by depicting the swapped strings in the leaf nodes. This is shown in Figure 3. As seen in this figure, the leaf nodes (for simplicity, we denote such leaves as $\psi_{2^{(h-1)}}$ to ψ_{2^h-1}, shown by their indices only in Figure 3) are shown, and a typical swapped structure is depicted that can be modified based on the required specifications. Let us now prove that the proposed RESN structure is a viable solution for permanent and transient error detection of hash tree constructions. The swapped nodes in Figure 3 are pairs (ψ_{2^h-2}, ψ_{2^h-1}) and (ψ_{2^h-4},ψ_{2^h-3}). As we have the same masks for deriving the internal node strings

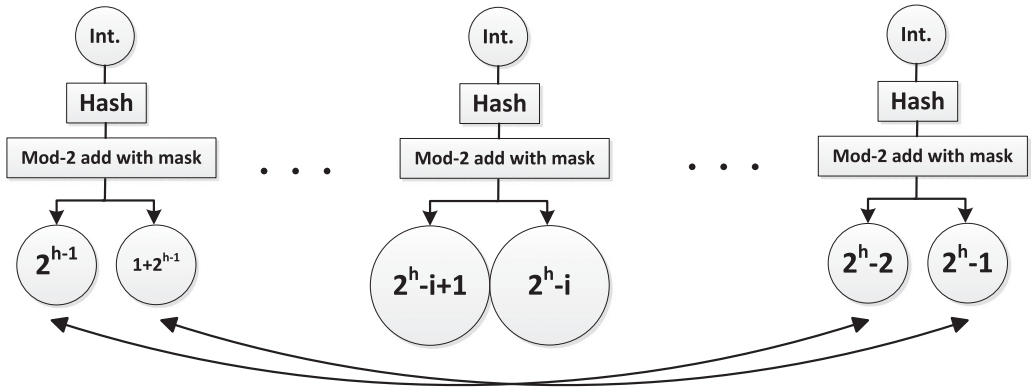

Fig. 4. An example for modified structure of the proposed RESN when fault location is known.

corresponding to $j = 1$, that is, $Q_0 \in \{0,1\}^{2n}$, and then the hash algorithm acts the same on the result, we get the same output for both of the internal nodes, that is, $N_{2^{(h-1)}-1,1} = H((\psi_{2^h-2} \,||\, \psi_{2^h-1}) \oplus Q_0)$ and $N_{2^{(h-1)}-2,1} = H((\psi_{2^h-4} \,||\, \psi_{2^h-3}) \oplus Q_0)$.

It is important to consider cases for which we might have side-channel information regarding the location of faults. For instance, let us assume that due to such information, we add the proposed countermeasure for a part of the architecture, say, first and last columns as shown in Figure 4. As swapping the nodes is free in hardware and the recomputation time is intact, no gain is acquired in terms of the induced overhead with respect to the performance. Nevertheless, comparators for RESN are reduced to just the first and last columns, leading to savings in area and power consumption.

Finally, we note that the well-known Merkle tree, which does not include modulo-2 addition with the masks, can take advantage of the proposed RESN through swapping nodes, for instance, pairs (ψ_{2^h-2}, ψ_{2^h-1}) and (ψ_{2^h-4},ψ_{2^h-3}) can be swapped in Merkle trees to get the results of the internal nodes, that is, $N_{2^{(h-1)}-1,1} = H(\psi_{2^h-2} \,||\, \psi_{2^h-1})$ and $N_{2^{(h-1)}-2,1} = H(\psi_{2^h-4} \,||\, \psi_{2^h-3})$.

3.2. Unbalanced Binary L-Trees

In case the number of bit strings in the verification key of the chosen OTS is not a power of 2, the resulting Merkle tree is unbalanced. Such unbalanced trees are denoted as L-Trees, where the usage model is confined to hash public keys, for example, those in SPHINCS. The height of an L-Tree is $\lceil log_2 l \rceil$, and it needs $\lceil log_2 l \rceil$ bitmasks.

Error detection for L-Trees through the proposed RESN is possible; yet, one needs to be careful to correctly swap the nodes. We have shown such a structure in Figure 5. The l leaves of an L-Tree are the elements of a WOTS$^+$ public key, and the tree is constructed as full binary trees; however, a left node that has no right sibling (typically in the last row) is lifted to a higher level of the L-Tree until it becomes the right sibling of another node, as seen in Figure 5. The node (leaf in this case) ψ_{2^h-2} without any right string is lifted up and the error detection is performed through RESN for pairs (ψ_{2^h-4}, ψ_{2^h-3}) and (ψ_{2^h-6},ψ_{2^h-5}) to derive the same output for both of the internal nodes, that is, $N_{2^{h-1}-2,1} = H((\psi_{2^h-4} \,||\, \psi_{2^h-3}) \oplus Q_0)$ and $N_{2^{h-1}-3,1} = H((\psi_{2^h-6} \,||\, \psi_{2^h-5}) \oplus Q_0)$.

3.3. Uniquely Structured Binary L-Trees

There are structures for binary L-trees that require close attention with respect to the approaches we have presented for fault diagnosis. Here, we go over one interesting case; nevertheless, this, by no means, confines the architectures for uniquely structured binary L-trees to such variants.

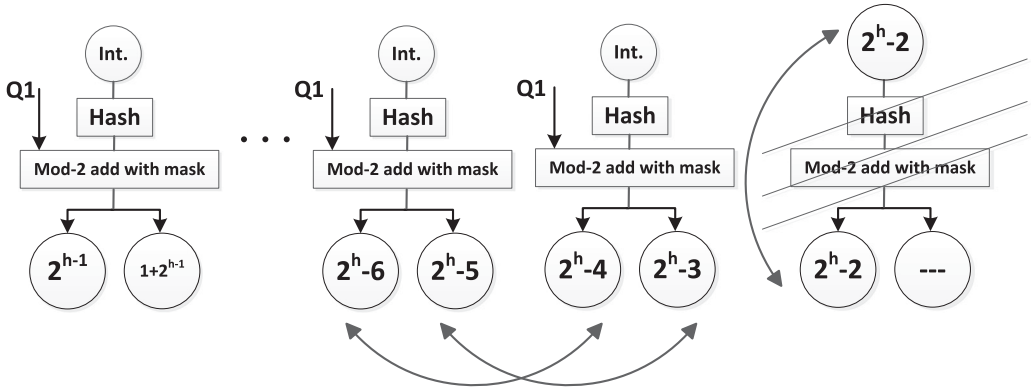

Fig. 5. Error detection in unbalanced L-Trees through RESN.

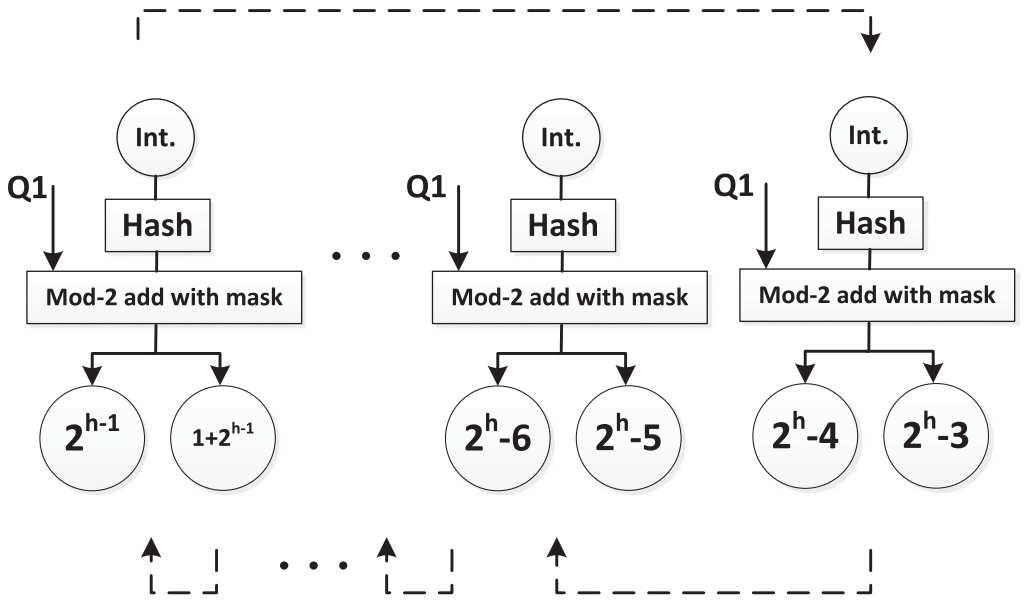

Fig. 6. Modified structure of the proposed RESN through adjacent relocation.

Odd-Pair Node/Leaf Construction

It is possible that due to the potential unbalanced nature of L-trees or specific structure of trees, we have an odd number of pairs in leaves, which, in turn, necessitates having RESN modified. Two cases for which such complications could happen are (a) when a left node that has no right sibling is lifted to a higher level of the L-Tree until it becomes the right sibling of another node, as seen in Figure 3, resulting in odd number of pairs or (b) when the number of nodes necessitates that the tree be structured with odd pairs. One can adopt RESN for these cases, where if information about the location of faults is available, then mutual swapping can be done for the nodes located in such spots and the rest is left intact. One other possibility is to move the adjacent pairs to the left or right and perform RESN. This is shown in Figure 6. As seen in this figure, one can move the pairs to the left, for instance, for all the adjacent ones, so RESN can be done for the odd number of pairs in such cases.

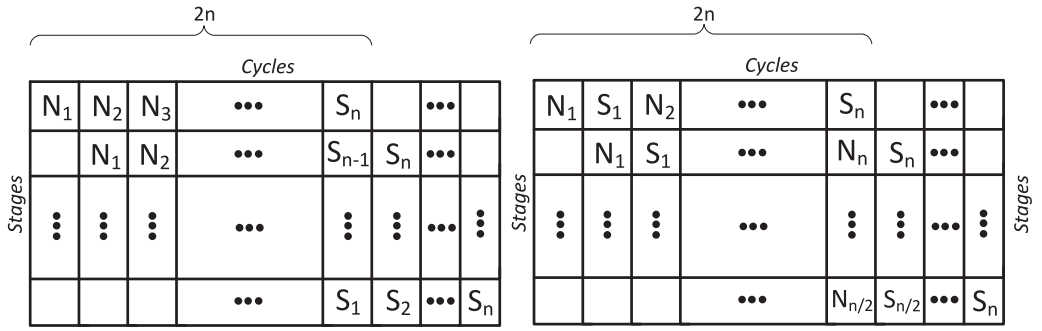

Fig. 7. Throughput alleviation for the proposed scheme for hash tree constructions for normal strings (N) and swapped strings (S) of nodes for $\exists_n$ stages.

3.4. Discussions

The proposed RESN scheme is applicable to hash tree constructions for error detection of both permanent and transient faults. Other schemes, such as recomputing with rotated or shifted operands (n-bit strings), do not provide efficient approaches for error detection of hash tree constructions as discussed in this section. One of the shortcomings of such recomputation is the decreased throughput of the computation (with the fixed applied frequency). To alleviate this problem, suppose one pipeline register has been placed to sub-pipeline the structures to break the timing path to approximately equal halves. Let us denote the two halves of pipelined stages by $\exists_1$ and $\exists_2$. The original input is first applied to the architecture and in the second cycle, while the second half of the circuit executes the first input, the second input (see Figure 7) or the encoded variant of the first input is fed to the first half of the circuit. This trend (which can be scaled to n stages as seen is Figure 7) is consecutively executed for normal strings (N) and swapped strings (S) of nodes for $\exists_n$ stages (see Figure 7). We have shown two possibilities for such a scheme. In the first one, output data availability has precedence over reliability (while both are achieved, the output data are derived first and fault diagnosis is performed after). Nevertheless, in the second approach, error detection is performed for each sub-segment of input data while the entire output is derived after such an order is followed. Depending on the requirements in terms of reliability and availability, one can tailor these approaches to fulfill such constraints.

Such an approach, although it increases the latency, which may not be admissible in some applications, ensures lower degradation in the throughput (and achieving higher frequencies) at the expense of more area overhead.

4. ALGORITHM-DEPENDENT ERROR DETECTION APPROACHES

In this section, we present schemes based on the specific hash functions used, that is, BLAKE and SPONGENT. In the next sections, simulation results and implementation benchmark are presented.

4.1. Error Detection Schemes for ChaCha

We present the error detection approaches for the ChaCha stream cipher, which is used in the BLAKE hash function of the hash tree construction here. Figure 8 shows the structure of ChaCha that is the main building block of BLAKE (the three steps for BLAKE compression function, that is, initialization, round functions, and finalization, as well as the control signal for selecting the initialization or round function steps construct BLAKE). We note that based on our ASIC implementation evaluations, initialization and finalization steps constitute less than 5% of the area of BLAKE.

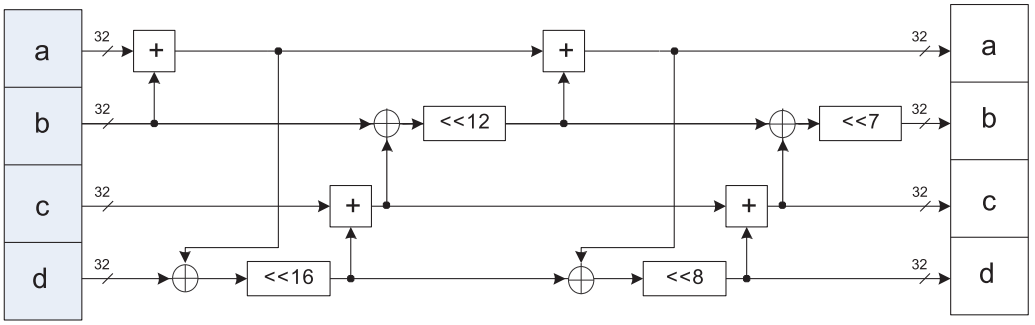

Fig. 8. ChaCha's quarter-round operations.

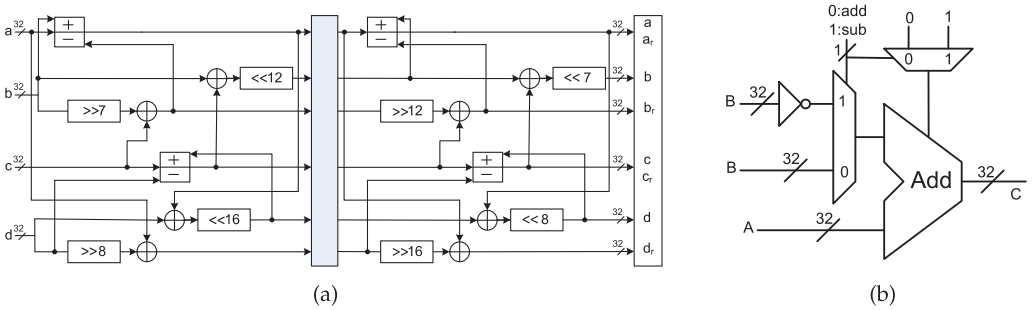

Fig. 9. The proposed complementary scheme for ChaCha with pipelined stages.

In what follows, we present two schemes for detection of errors in ChaCha: the complementary scheme and the REEO approach. In the former, we embed the inverse of the quarter round in the original structure; this is shown in Figure 9. Specifically, for the 32-bit first and third entries, a and c (from the total of four entries to the left-hand side of Figure 8), adder/subtractors are used instead of single adders (compare Figure 8 with Figure 9). In the hardware implementation of the adder/subtractor unit, adding is performed as normal, while subtraction is done by complementing an input and having carry-in as one, that is, twos complement process. The 32-bit second and fourth entries in Figure 9(b and d) are also complemented by reverting the rotation operations (compare Figure 9 with Figure 8). Finally, in the right-hand side, we obtain the outputs of the original, a–d, and the reverse algorithm, a_r–d_r. It is shown in the next section that with high error coverage for both transient and permanent faults, the proposed scheme results in acceptable hardware overhead. It is noted that the pipelined stages shown in this scheme (Figure 9) result in a large decrease in throughput degradation.

Our presented complementary scheme has high error coverage, which is of benefit for reliability-constrained applications. Although taking advantage of our proposed embedding approach reduces the hardware overhead of this scheme, resource-constrained devices might just be able to tolerate low hardware overheads. For ChaCha and to get a low-overhead error detection approach, one can use the alternative REEO approach in which recomputations are performed through encoded entries (for instance, rotations). The proposed REEO-based scheme is shown in Figure 10 (although we have shown the recomputing with rotated operands [nodes or leaves in the hash-based construction], encoding schemes are not certainly confined to this approach). The pipelined original structure in this figure is executed 2 times with original and rotated inputs (for the four 32-bit inputs in Figure 10). We note that for the XOR and rotation operations in

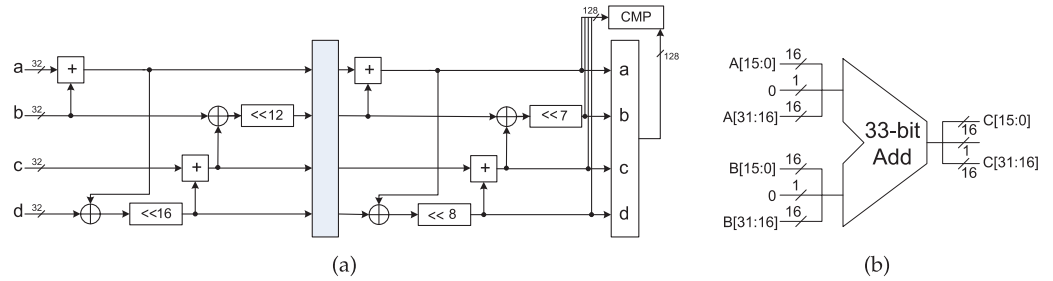

Fig. 10. The proposed recomputation with encoded hash tree construction nodes (leaves) scheme for ChaCha.

Figure 10, one would only need to rotate the original inputs. However, for modular addition, based on a modification we have made, a zero is inserted between the 32-bit rotated operands (A and B as examples) before addition and then, after addition, the middle bit is discarded to obtain C. It is worth mentioning that after each addition, we require to discard the middle-bit (which is the carry-out of the last-bit addition of the non-rotated operands), because it can be non-zero. Finally, a comparison is performed as shown in Figure 10 in which the original result is compared with back-rotated results.

We also present fault detection schemes based on dual-rail checkers for ChaCha. In this scheme, self-checking adders based on dual-rail encoding is utilized for modular adders within ChaCha. A variant of self-checking adder design utilizes two n-bit ripple carry adders to pre-compute the sum bits with complemented values of C_{in}, that is, 0 and 1, and the original value of C_{in} is used to select the actual sum bits [Akbar and Lee 2014]. One could employ this adder for detection of faults in the modular addition unit of ChaCha. Figure 11 shows the design module of a 4-bit self-checking carry-select adder; an n-bit model of the same design module is employed in ChaCha. The inputs here are given to the two-pair two-rail checker; S_{0N} is the sum output of the full adder with inputs (A_n,B_n) with the initial C_{in} equal to zero. The output of the XNOR and S_{1N} is always complementary to each other and, hence, is chosen as the inputs to the two-pair two-rail checker. This construction can be utilized for detecting the faults in the modular adder unit of ChaCha.

4.2. Error Detection Schemes for SPONGENT

Sponge functions, such as the one used in SPONGENT, provide a particular way to generalize hash functions to more general functions whose output length is arbitrary. Sponge constructions, likewise, consist of an iterated construction building a variable-length-input–variable-length-output function based on a fixed length permutation (or transformation) (given a finite number of input bits, it produces an n-bit hash value).

Fault diagnosis for this hash function can be presented for its individual phases, that is, three phases in SPONGENT: (a) an initialization phase wthat includes message padding and then the message is cut into blocks of r bits; (b) an absorbing phase in which the r-bit input message blocks are XORed into the first r bits of the state, interleaved with applications of the permutation π_b (operating on a state of a fixed number b of bits); and (c) a squeezing phase where the first r bits of the state are returned as output, interleaved with applications of the permutation π_b until n bits are returned.

Let us present the scheme for the first phase. This phase includes message padding wthat is free in hardware. In other words, there is no architecture involved in padding, similarly to shift, rotate, and the like, and thus any added fault can be detected using parity (or similar signatures). Specifically, here, and for the case of parity, the predicted

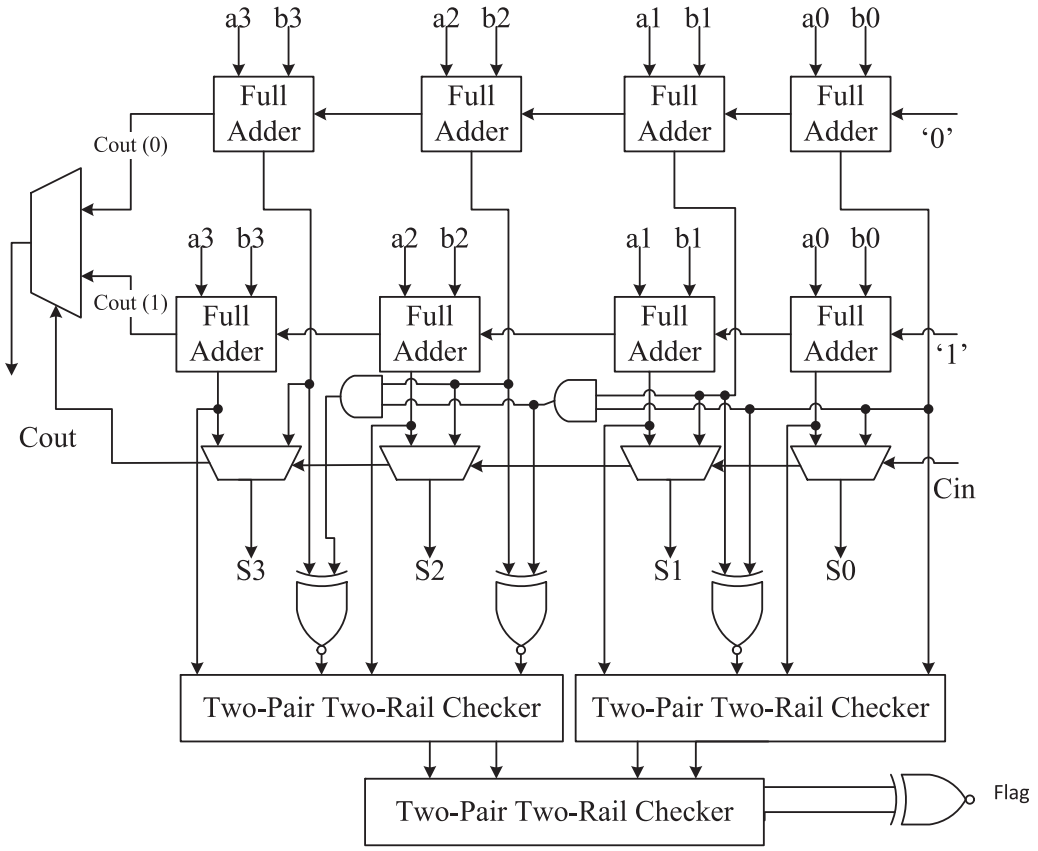

Fig. 11. Scheme based on self-checking carry-select adder for the modular adder of ChaCha.

Table I. The Entries for the S-box (sBoxLayer$_b$) and the Derived Interleaved Parities

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
E (01)	D (01)	B (10)	0 (00)	2 (01)	1 (01)	4 (10)	F (00)	7 (10)	A (11)	8 (10)	5 (11)	9 (11)	C (00)	3(00)	6(11)

and actual parities would be the same and thus no cost is involved. These parities with no cost can be used separately for the r-bit blocks in this phase.

In the second phase, absorbing, these blocks are XORed into the first r bits of the state, the fault diagnosis of which is straightforward through signatures, for example, parities, or recomputing with encoded/rotated/shifted operands. We will go over the permutation π_b separately. Finally, in the squeezing phase, π_b is the major function used whose fault detection is presented as follows:

Fault Diagnosis of π_b: In π_b, SPONGENT has the sBoxLayer$_b$, pLayer$_b$, and lCounter$_b$ building blocks. Let us present the fault detection schemes devised for these building blocks.

For sBoxLayer$_b$, where a 4-bit to 4-bit S-box is applied $b/4$ times, one can utilize signatures, for example, interleaved parities, and store them in the memory table to detect the faults. As seen in Table I, we have derived these parities for the least/most significant two-bit parts of each entry in the S-box, presented in parentheses. To detect input parity errors and some internal memory (data or decode) errors, we propose replacing the original 8-bit bit-holders with 10-bit ones. Depending on the

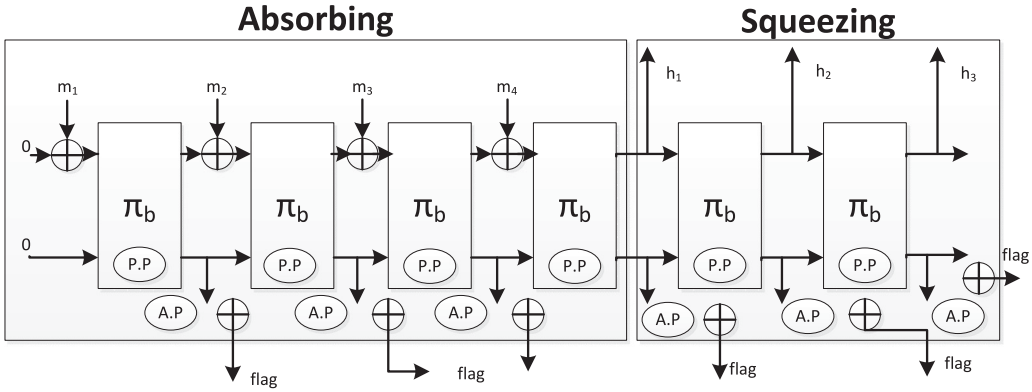

Fig. 12. Fault detection scheme for SPONGENT.

security objectives and overhead to be tolerated, one can tailor the detection scheme, for example, using other signature variants.

The building block pLayer$_b$ that involves bit permutations, where they move bit j of the state to bit position $P_b(j) = j.b/4 \bmod b - 1$ for $j \in \{0, \ldots, b-2\}$ and to bit position $P_b(j) = b - 1$ if $j = b - 1$ is free in hardware and any predicted signature, for example, parities, can be derived for such constructions. Finally, in lCounter$_b$, which is based on LFSRs, parity predictions are done as a viable approach. The entire approach for detecting faults in SPONGENT is shown in Figure 12. As seen in this figure, sponge construction based on a b-bit permutation π_b is shown. m_i are r-bit message blocks and h_i are parts of the hash value. We have also shown the signatures, for example, predicted/actual parity calculations using P.P (predicted parity) and A.P (actual parity) blocks, as well as the comparison units to derive the error indication flags.

5. ERROR SIMULATIONS

Throughout this article, both single and multiple stuck-at faults have been considered. With respect to fault analysis attacks, due to the technological constraints, single stuck-at fault may not be applicable for an attacker to flip exactly only one bit to gain more information. Thus, multiple stuck-at faults are also considered in this article. We consider the BLAKE-64 structure that instantiates a variant of the ChaCha block to construct a hash function used in hash trees. Moreover, architectures for SPONGENT have been simulated. According to the fault model presented in this section, both stuck-at zero and stuck-at one faults are injected in multiple random locations. We consider the transient faults to occur at both runs independently. However, for permanent faults, each fault can occur at the same location for both runs with the same polarity. Therefore, the number of fault locations for this type is half of that for the transient one.

For single-bit fault injection, for BLAKE-64, we generated 1,000,000 random inputs and, for SPONGENT, the random inputs were 350,000 and for both types of stuck-at faults (we note that this number of random inputs was chosen to have a practical simulation process; however, single stuck-at faults were injected exhaustively). For multiple-bit fault injection simulations, the same number of injections were performed (a number of multiple-bit faults were injected per each random input) with random locations. Single stuck-at faults were detected 100% using both complementary and REEO approaches. We note that these detection schemes consider both natural and malicious faults and are not just capable of detecting random multiple faults. For testing this, we have also injected double and triple adjacent faults, byte faults, and four different types of isolated faults that mimic fault injections in specific locations.

Table II. False-Alarm Assessments for Multiple, Random Faults in ChaCha (Signature-Based Approach)

Type of faults	Injected faults	Detected faults	False alarms	Percent false-alarms
Stuck-at zero	1,000	995	5	0.50%
	10,000	9,945	31	0.03%
	20,000	19,711	110	0.55%
	25,000	24,892	198	0.79%
Stuck-at one	1,000	954	7	0.70%
	10,000	9,899	67	0.67%
	20,000	19,869	122	0.61%
	25,000	24,950	195	0.78%

For multiple stuck-at faults, we analyzed two to eight bit faults for transient and permanent types as well as random number of bits. The results show that for both transient and permanent faults, the detection ratio percentage is roughly 99.9%.

5.1. False-Alarms

The cryptographic architectures that include fault detection schemes need to be immune against possible false-alarms that can eventually cause distrust and may result in falsely abandoning the infrastructure. False-alarms could have adverse effects on the utilization of cryptographic solutions. Specifically, if such alarms get repetitive, cryptographic architectures wthat include fault detection mechanisms may be affected by them. Such false-alarms might hinder the normal operations of cryptographic algorithms by inducing distrust to users who may eventually abandon all the security solutions.

For the proposed schemes for ChaCha and SPONGENT, let us consider the cases in which the fault model deals with multiple, random stuck-at faults (although this case has been used as the case study here, it, by no means, confines the discussions here to just these types of faults). For such cases, a number of causes may result in having false-alarms in ChaCha and SPONGENT. For ChaCha, because we are dealing with complementary or recomputing schemes, as two main schemes, no false-alarm has been observed. This is because any injected fault that produces an error is detected after all the computations are done and no intermediate detection scheme is used for detection (which could potentially alert for faults which are masked). For the dual-rail fault detection scheme of the modular addition units in ChaCha and also for the SPONGENT hash function, because we are dealing with multiple signatures, there might be cases in which we detect faults in an inner sub-part that will not be eventually translated into errors in the output due to the masking of such faults.

Through simulations, considering different number of stuck-at zeros and ones, we have identified such cases. The results are shown in Tables II and III for both ChaCha and SPONGENT. As seen in these tables, different numbers of faults are injected for the two types, and the number of masked faults and false-alarms are shown. These false-alarms show that there exists natural defect(s) or fault attack(s) in the architectures; nevertheless, these do not result in erroneous outputs for that particular simulation instance. The false-alarm percentage is typically low as seen in these tables.

6. ASIC IMPLEMENTATIONS

This section presents the results of our ASIC syntheses performed for the original and the error detection structures (constructing a hash function used in hash trees such as L-Tree, full binary tree, Merkle tree, and the like) to benchmark the overheads induced.

Table III. False-Alarm Assessments for Multiple, Random Faults in SPONGENT

Type of faults	Injected faults	Detected faults	False alarms	Percent false-alarms
Stuck-at zero	1,000	987	8	0.80%
	10,000	9,923	86	0.86%
	20,000	19,911	160	0.80%
	25,000	24,943	211	0.84%
Stuck-at one	1,000	967	9	0.90%
	10,000	9,945	84	0.84%
	20,000	19,918	178	0.89%
	25,000	24,900	194	0.77%

Table IV. Area Overhead and Performance Degradations of the Proposed Schemes for ChaCha

Structure	Area (μm^2) (KGE)	Overhead	Frequency (MHz)	Throughput (Gbps)	Degradation
Original	79,772 (56.5)	—	307	9.6	—
Complementary	106,191 (75.3)	33.1%	538	8.2	14.5%
REEO[1]	87,012 (61.7)	**9.1%**	551	**8.6**	**10.4%**
REEO[2]	92,190 (65.3)	15.5%	**789**	8.2	14.6%

1 and 2: One- and two-stage sub-pipelined architectures.

We note that ASIC is chosen based on the resources available to us (library and tools) and as our presented schemes are not dependent on the hardware platform; similar overheads are expected if FPGAs are utilized for the implementations. Through these ASIC syntheses, the overheads in terms of hardware and timing are derived. We have used the TSMC 65-nm standard-cell library for the original and the error detection structures and Synopsys Design Compiler [Synopsys 2015].

To benchmark the performance of the proposed schemes, we have done implementations for the original and fault diagnosis schemes for ChaCha, whose results are presented in Table IV. Moreover, based on the sub-pipelining approach we presented in this article, one can alleviate the inherent performance degradations of the proposed structures. Specifically, with the expense of adding registers for deep sub-pipelining (for instance, one stage sub-pipelining in Table IV), higher frequencies are achieved that make the degradations in throughput less intense. We note that, in Table IV, the areas (in terms of μm^2), maximum working frequencies (in terms of MHz), and throughputs (in terms of Gbps) have been obtained. In order to make the area results meaningful when switching technologies, we have provided the NAND-gate equivalency. This is performed using the area of a NAND gate in the utilized TSMC 65-nm CMOS library, which is 1.41μm^2.

In Table IV, for the original BLAKE and the proposed schemes, the 4G BLAKE structure is utilized. In this structure, four parallel internal operations are performed simultaneously and twice. According to Table IV, for the complementary scheme, the area overhead and throughput degradation are 33.1% and 14.5%, respectively. These are lower for the REEO-based scheme, that is, 9.1% and 10.4%, respectively. We note that, as expected, the overheads for the complementary schemes are higher than those for REEO-based ones. However, the advantage of the complementary schemes is their slightly higher error coverage. We have also shown the results with two-stage sub-pipelining and as seen, higher frequencies are achieved (789MHz); nevertheless, the throughput overhead is increased. Based on the overhead tolerance and fault detection ratio to achieve, one can tailor the proposed approaches for various objectives to reach.

We have also shown the results of our ASIC implementations for SPONGENT and its fault diagnosis architectures in Table V. As seen in this table, we have implemented

Table V. Area Overhead and Performance Degradations of the Proposed Schemes for $SPONGENT_1$ and $SPONGENT_2$

Structure	Area (μm^2) (GE)	Overhead	Throughput (Kbps)	Degradation
$SPONGENT_1$	1,990 (1,412)	—	$\simeq$12	—
Fault Detection	2,467 (1,750)	24%	$\simeq$11	8.3%
$SPONGENT_2$	2,700 (1,915)	—	$\simeq$8	—
Fault Detection	3,307 (2,345)	22.5%	$\simeq$6.5	18.7%

two variants of SPONGENT and reported the areas (GE using the area of a NAND gate in the utilized TSMC 65-nm CMOS library, which is $1.41\mu m^2$) and throughputs and their corresponding overheads. The first variant is SPONGENT with pre-image 80 and second pre-image 40 (hash 88 bits) and datapath 88 bits, and the second variant is SPONGENT with pre-image 120 and second pre-image 64 (hash 128 bits) and datapath 136 bits. We denote the former $SPONGENT_1$ and the latter $SPONGENT_2$, as seen in Table V. From this table, for $SPONGENT_1$, the area of the original design is 1,990 (1,412) (μm^2) (GE) and its throughput is $\simeq$12 (Kbps). For the corresponding fault detection architecture, these are 2,467 (1,750) (μm^2) (GE) (overhead of 24%) and $\simeq$11 (Kbps) (overhead of 8.3%), respectively. Moreover, as seen in Table V, for $SPONGENT_2$, the area of the original design is 2,700 (1,915) (μm^2) (GE) and its throughput is $\simeq$8 (Kbps). For the corresponding fault detection architecture, these are 3,307 (2,345) (μm^2) (GE) [overhead of 22.5%] and $\simeq$6.5 (Kbps) (overhead of 18.7%), respectively.

7. CONCLUSIONS

Post-quantum cryptographic implementation attacks and natural faults need to be detected through efficient countermeasures. In this article, various fault diagnosis approaches for hash-based post-quantum signatures are proposed. The merit of the proposed schemes is that they are a step forward towards reliability and fault attack immunity of resistant hash trees in future potential post-quantum systems. Based on the reliability requirements and available resources, one may select the error detection schemes suitable for these architectures. We have developed a method for swapping nodes and leaves in binary hash trees, L-Trees, and Merkle trees to detect faults, where recomputations with rotated/shifted, and, in general, encoded operands fail. We have also discussed various complications in special hash tree constructions. Moreover, for the inner hash functions BLAKE and SPONGENT, we have presented a number of diagnosis methods that are capable of reaching high error coverage (this includes analysis of false-alarms) with acceptable area overhead and performance degradation.

REFERENCES

M. A. Akbar and J. A. Lee. 2014. Comments on self-checking carry-select adder design based on two-rail encoding. *IEEE Trans. Circ. Syst. I, Reg. Pap.* 61, 7, 2212–2214.

S. Ali, D. Mukhopadhyay, and M. Tunstall. 2013. Differential fault analysis of AES: Towards reaching its limits. *J. Cryptogr. Eng.* 3, 2, 73–97.

J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. 2010. BLAKE hash function. Retrieved from https://131002.net/blake/blake.pdf.

S. Bayat-Sarmadi and M. Anwar Hasan. 2007. On concurrent detection of errors in polynomial basis multiplication. *IEEE Trans. VLSI Syst*. 15, 4, 413–426.

S. Bayat-Sarmadi, M. Mozaffari-Kermani, and A. Reyhani-Masoleh. 2014. Efficient and concurrent reliable realization of the secure cryptographic SHA-3 algorithm. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 33, 7, 1105–1109.

D. J. Bernstein, D. Hopwood, A. Hlsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn. 2014. SPHINCS: Practical stateless hash-based signatures. *IACR Cryptology ePrint Archive*, 1–30.

D. J. Bernstein, D. Hopwood, A. Hlsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn. 2015. SPHINCS: Practical stateless hash-based signatures. In *Proc. EUROCRYPT*. 368–397.

D. J. Bernstein. 2007. The Salsa20 family of stream ciphers Salsa. Retrieved from http://cr.yp.to/snuffle/salsafamily-20071225.pdf.

D. J. Bernstein. 2008. ChaCha, a variant of Salsa20. Retrieved from http://cr.yp.to/chacha/chacha-20080120.pdf.

D. J. Bernstein, J. Buchmann, and E. Dahmen (Eds.). 2009. *Post-Quantum Cryptography*. Springer. http://www.springer.com/us/book/9783540887010.

A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varc, and I. Verbauwhede. 2013. SPONGENT: The design space of lightweight cryptographic hashing. *IEEE Trans. Comput.* 62, 10, 2041–2053.

D. Boneh, R. DeMillo, and R. Lipton. 1997. On the importance of checking cryptographic protocols for faults. In *Proc.Int. Conf. Eurocrypt*. 37–51.

J. Buchmann, E. Dahmen, and A. Hlsing. 2011. XMSS - a practical forward secure signature scheme based on minimal security assumptions. In *Proc. Post-Quantum Cryptogr*. 117–129.

G. Di Natale, M. Doulcier, M. L. Flottes, and B. Rouzeyre. 2009. A reliable architecture for parallel implementations of the advanced encryption standard. *J. Electron. Test.: Theor. Appl.* 25, 4, 269–278.

J. Ding and D. Schmidt. 2005. Rainbow, a new multivariable polynomial signature scheme. In *Proc. 3rd Int. Conf. ACNS*. 164–175.

S. Fenn, M. Gossel, M. Benaissa, and D. Taylor. 1998. On-line error detection for bit-serial multipliers in $GF(2^m)$. *J. Electron. Test.: Theor. Appl.* 13, 29–40.

O. Goldreich. 2004. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, Cambridge, UK.

L. K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proc. ACM Symp. Theor. Comput.* 212–219.

T. Guneysu and P. Lyubashevsky. 2012. Practical lattice-based cryptography: A signature scheme for embedded systems. (unpublished).

X. Guo and R. Karri. 2013. Recomputing with permuted operands: A concurrent error detection approach. *IEEE Trans.Comput.-Aid. Des. Integr. Circ. Syst.* 32, 10, 1595–1608.

X. Guo and R. Karri. 2014. Low-cost concurrent error detection for GCM and CCM. *J. Electron. Test.* 30, 6, 725–737.

X. Guo, D. Mukhopadhyay, C. Jin, and R. Karri. 2014. NREPO: Normal basis recomputing with permuted operands. In *Proc.HOST*. 118–123.

X. Guo, D. Mukhopadhyay, C. Jin, and R. Karri. 2015. Security analysis of concurrent error detection against differential fault analysis. *J. Cryptogr. Eng.* 5, 3, 153–169.

A. Hlsing, L. Rausch, and J. Buchmann. 2013. Optimal parameters for XMSS. In *Proc. Security Engineering and Intelligence Informatics*. 194–208.

M. Mozaffari Kermani and R. Azarderakhsh. 2013. Efficient fault diagnosis schemes for reliable lightweight cryptographic ISO/IEC standard CLEFIA benchmarked on ASIC and FPGA. *IEEE Trans. Ind. Electron.* 60, 12, 5925–5932.

M. Mozaffari Kermani and A. Reyhani-Masoleh. 2006. Parity-based fault detection architecture of s-box for advanced encryption standard. In *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 572–580.

M. Mozaffari Kermani and A. Reyhani-Masoleh. 2007. A structure-independent approach for fault detection hardware implementations of the advanced encryption standard. In *Proc. IEEE Workshop Fault Diagnosis and Tolerance in Cryptography.* 47–53.

M. Mozaffari Kermani and A. Reyhani-Masoleh. 2010. Concurrent structure-independent fault detection schemes for the advanced encryption standard. *IEEE Trans. Comput*. 59, 5, 608–622.

M. Mozaffari Kermani and A. Reyhani-Masoleh. 2011. A high-performance fault diagnosis approach for the AES SubBytes utilizing mixed bases. In *Proc. IEEE Workshop Fault Diagnosis and Tolerance in Cryptography.* 80–87.

M. Mozaffari Kermani and A. Reyhani-Masoleh. 2011. A lightweight high-performance fault detection scheme for the advanced encryption standard using composite fields. *IEEE Trans. Very-large Scale Integr. Syst*. 19, 1, 85–91.

L. Lamport. 1979. *Constructing Digital Signatures from a One Way Function*. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory.

P. Maistri and R. Leveugle. 2008. Double-data-rate computation as a countermeasure against fault analysis. *IEEE Trans. Comput*. 57, 11, 1528–1539.

T. G. Malkin, F. X. Standaert, and M. Yung. 2006. A comparative cost/security analysis of fault attack countermeasures. In *Proc. Int. Workshop Fault Diagnosis and Tolerance in Cryptography*, 159–172.

R. Merkle. 1990. A certified digital signature. In *Proc. CRYPTO*. 218–238.

M. Mozaffari-Kermani, R. Azarderakhsh, and A. Aghaie. 2015. Reliable and error detection architectures of Pomaranch for false-alarm-sensitive cryptographic applications. *IEEE Trans. VLSI Syst.* 23, 12, 2804–2812.

M. Mozaffari-Kermani, K. Tian, R. Azarderakhsh, and S. Bayat-Sarmadi. 2014. Fault-resilient lightweight cryptographic block ciphers for secure embedded systems. *Embed Syst Lett.* 6, 4, 89–92.

R. Overbeck and N. Sendrier. 2009. Code-based cryptography. In *Proc. Post-Quantum Cryptography*, 95–145.

C. Peikert. 2014. Lattice cryptography for the Internet. IACR. (unpublished)

P. W. Shor. 1997. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM J. Comput.* 26, 5.

F. Song. 2014. A note on quantum security for post-quantum cryptography. In *Proc. Post-Quantum Cryptography.* 246–265.

Synopsys. 2016. Homepage. Retrieved from www.synopsys.com.

M. Tunstall, D. Mukhopadhyay, and S. Ali. 2011. Differential fault analysis of the advanced encryption standard using a single fault. In *Proc. Int. Conf. Information Security Theory and Practice*, 224–233.

C. H. Yen and B. F. Wu. 2006. Simple error detection methods for hardware implementation of advanced encryption standard. *IEEE Trans. Comput.* 55, 6, 720–731.

Received November 2015; revised March 2016; accepted April 2016