Reliable Constructions for the Key Generator of Code-based Post-quantum Cryptosystems on FPGA

ALVARO CINTAS CANTO, Marymount University, USA
MEHRAN MOZAFFARI KERMANI, University of South Florida, USA
REZA AZARDERAKHSH, Florida Atlantic University, USA

Advances in quantum computing have urged the need for cryptographic algorithms that are low-power, low-energy, and secure against attacks that can be potentially enabled. For this post-quantum age, different solutions have been studied. Code-based cryptography is one feasible solution whose hardware architectures have become the focus of research in the NIST standardization process and has been advanced to the final round (to be concluded by 2022–2024). Nevertheless, although these constructions, e.g., McEliece and Niederreiter public key cryptography, have strong error correction properties, previous studies have proved the vulnerability of their hardware implementations against faults product of the environment and intentional faults, i.e., differential fault analysis. It is previously shown that depending on the codes used, i.e., classical or reduced (using either quasi-dyadic Goppa codes or quasi-cyclic alternant codes), flaws in error detection could be observed. In this work, efficient fault detection constructions are proposed for the first time to account for such shortcomings. Such schemes are based on regular parity, interleaved parity, and two different cyclic redundancy checks (CRC), i.e., CRC-2 and CRC-8. Without losing the generality, we experiment on the McEliece variant, noting that the presented schemes can be used for other code-based cryptosystems. We perform error detection capability assessments and implementations on field-programmable gate array Kintex-7 device xc7k70tfbv676-1 to verify the practicality of the presented approaches. To demonstrate the appropriateness for constrained embedded systems, the performance degradation and overheads of the presented schemes are assessed.

CCS Concepts: • **Hardware** → *Application specific integrated circuits*; *Hardware reliability screening;*

Additional Key Words and Phrases: Code-based cryptography, low-power fault detection, McEliece cryptosystem, post-quantum cryptography

ACM Reference format:
Alvaro Cintas Canto, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2022. Reliable Constructions for the Key Generator of Code-based Post-quantum Cryptosystems on FPGA. *ACM J. Emerg. Technol. Comput. Syst.* 19, 1, Article 5 (December 2022), 20 pages.
https://doi.org/10.1145/3544921

5

This work has been supported by the U.S. National Science Foundation (NSF) through Award No. SaTC-1801488.
Authors' addresses: A. Cintas Canto, Marymount University, 2807 North Glebe Road, Arlington, VA 22207; email: acintas@marymount.edu; M. Mozaffari Kermani, University of South Florida, 4202 E. Fowler Avenue, Tampa, FL 33620; email: mehran2@usf.edu; R. Azarderakhsh, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431; email: razarderakhsh@fau.edu.

1550-4832/2022/12-ART5 $15.00
https://doi.org/10.1145/3544921

1 INTRODUCTION

The potential of the advent of high-performance and low-power quantum computers has heightened the necessity for the development of public-key cryptosystems that are safe against faults that such quantum-based computing systems may empower. In fact, the Shor's quantum algorithm efficiently factors integers in polynomial time, allowing conventional cryptosystems to be broken. In late 2017, the **National Institute of Standards and Technology (NIST)** launched a project to standardize one or more quantum computer resistant public-key cryptographic algorithms [1], which is currently in its final round since July 2020. It is expected that in 2024, the details for a portfolio of standardized algorithms are revealed. Such standardized algorithms will be alternated to the current classical public-key cryptosystems.

Different algorithms have been studied for this post-quantum age, denoted as **post-quantum cryptography (PQC)**. Among the different types of post-quantum cryptographic algorithms, code-based cryptography is a potential approach for resisting quantum computer-based attacks. The McEliece cryptosystem is a type of code-based cryptography whose security is based on the hardness of decoding a general linear code, possibly chosen in a specific family, e.g., quasi-dyadic Goppa codes and quasi-cyclic alternant codes. The McEliece cryptosystem security and implementation complexity have been scrutinized over years. As efficient examples, implementations of the **McEliece cryptoprocessor (MECS)** have been proposed in References [2–8].

Classic McEliece has progressed to the current and last stage of the NIST PQC standardization process. However, the McEliece post-quantum algorithm is still vulnerable to side-channel attacks [9]. Additionally, fault analysis attacks are studied in Reference [10] to prove that the probability when the McEliece construction does not repair an error is not negligible. Mounting attacks and recovering the secret information through fault attacks in the McEliece cryptosystem are also discussed on other works [11, 12]

The McEliece cryptosystem spends the majority of its runtime executing arithmetic operations on finite fields to perform the key generation process. Among all the finite-field arithmetic, inversion takes the longest time to compute. Many approaches have been studied to improve the performance of inversions in $GF(2^m)$ with polynomial basis. The **Fermat's little theorem (FLT)** and the **Itoh-Tsujii algorithm (ITA)** are two of the most used methods for computing inversions in $GF(2^m)$. ITA was originally designed to be used with elements over $GF(2^m)$ using normal basis [13]; nonetheless, recent works demonstrate that ITA can be utilized with different field element representations [14, 15]. Such methods significantly utilize squarings and multiplications, involving hundreds of gates. Thus, these architectures are vulnerable to faults and implementing them robust to natural and intended faults is a difficult challenge. These structures not only require little overhead, but they also require sufficient error coverage.

1.1 Previous Works

For sensitive systems, degraded performance can lead to disastrous results; consequently, research has explored strategies to reduce errors and provide higher reliability with acceptable overhead [16–24]. In Reference [16], a fingerprint-based technique for detecting malicious programs in hardware is presented. Fault diagnosis approaches based on multiterm signatures against false-alarms, which may be unacceptable in critical intelligent infrastructures, are presented in Reference [17]. In References [18, 19], fault detection mechanisms are presented for the lightweight cryptographic block cipher QARMA and polynomial basis inversions, respectively. Moreover, error detection constructions based on recomputation with encoded (shifted) operands and recomputation with encoded (negated) operands for the Ring-LWE and for the ring polynomial multiplication and modular reduction of Ring-LWE are implemented in References [20, 21], respectively. Last, error detection schemes are proposed for secure cryptographic GCM structures in Reference [22], for

Table 1. Comparison with Other Works

Work	Error Detection Scheme	Limitations
[30]	Parity, Multi-parity	Singular parity offers up to 50% error coverage and both singular and multi-parity can be vulnerable to intelligent fault injections
[20]–[23]	Recomputing	Recomputing can add large overhead, since the operations are being performed twice. This leads to an increased delay overhead of at least 100%, unless pipelining is used, which would increase the number of registers and consequently, the area overhead is increased
[24], [18], [19]	CRC-10, CRC-3, CRC-5	CRC is an efficient choice to protect the systems against intelligent fault injections and adds acceptable overheads. These have been used for different applications in works [18], [19], and [24]; however, those works do not provide flexibility in terms of security and overhead. This article derives four different schemes for the Key Generator of McEliece cryptosystem that can be combined to provide flexibility depending on the user needs

Hash-Counter-Hash tweakable enciphering constructions in Reference [23], and for cryptographic applications using multipliers in Reference [24]. In Table 1, a summary of the limitations of these works on fault detection is shown. We note that the error detection schemes are not confined to one specific cryptographic algorithm, for example see References [25–29] for those related to the AES and lightweight cryptography.

We present the first work on fault detection in the underlying blocks of the McEliece cryptosystem Key Generator, based on regular parity, interleaved parity, CRC-2, and CRC-8. Fault detection is essential in the generation of the keys, especially for remote systems where the creation of fault-free keys is required for the overall system dependability. The hardware implementation of the Key Generator is the most complicated inside McEliece, since it has the largest area complexity. Our suggested techniques are suitable to the generation of the control matrix H, and we have also incorporated fault detection techniques in the other units of the Key Generator. Nonetheless, the underlying blocks that execute finite-field operations can be employed not just in these constructions but also in other cryptographic systems. In Reference [30], fault detection techniques based on parities are proposed for the composite-field operations of the McEliece cryptosystem. This work completes [30] by performing fault detection in finite fields and adding **cyclic redundancy checks (CRC)** as a fault detection technique. Although we have presented our approach for the Key Generator of the McEliece cryptosystem and implemented the different schemes on **field-programmable gate array (FPGA)**, the presented models are suitable to other code-based cryptographic algorithms, e.g., Niederreiter cryptosystem. These models are also platform-oblivious, anticipating comparable outcomes on **application-specific integrated circuit (ASIC)** platforms.

1.2 Contributions

The following is a summary of our contributions in this work:

- We construct sets of formulations for the various finite-field blocks of the McEliece cryptosystem, e.g., addition, subtraction, multiplication, squaring, and inversion, based on regular parity, interleaved parity, CRC-2, and CRC-8. To account for the entire Key Generator, we additionally offer fault detection techniques in the remaining units of the Key Generator.
- The presented fault detection techniques are employed in the distinct units of the Key Generator to maximize the likelihood of error detection, since it is generally formed by multiplications and inversions over $GF(2^{13})$.
- The fault coverage of the presented fault detection methods is examined. To assess the different overheads of the suggested techniques, we implemented our schemes on FPGA by adding them to the original sub-blocks of McEliece cryptosystem Key Generator.

The following is the outline of the article: Preliminaries are discussed in Section 2, where the McEliece cryptosystem is introduced. Section 3 presents the proposed fault detection

constructions based on regular parity, interleaved parity, and CRC for the different finite-field blocks in the McEliece cryptosystem. In Section 4, the presented fault detection techniques are implemented to calculate the different overheads when the derived schemes are used in the original constructions. Moreover, we benchmark our derived work by implementing the presented designs on FPGA. Finally, concluding observations are given in Section 5.

2 PRELIMINARIES

There are three main parameters in the McEliece cryptosystem: m, which is used for the code subspace dimension; t, which is the maximum number of faults that the system can correct; and n, which stands for the code length. The cryptosystem presented in this article uses parameters $m = 13$, $t = 128$, and $n = 8{,}192$, since they were one of the proposed security metrics to NIST in 2020 [31]; although the presented schemes are oblivious of these metric sizes.

The McEliece cryptosystem has three main processes: Key generation, which consists of the generation of two keys, e.g., private and public and keys, required to maintain the data safe; encryption, which uses the public key to create the ciphertext; and decryption, which utilizes the private key to get the initial data. The private key is produced by a generator matrix while the public key is provided by a control matrix. Initially, the Key Generator produces at random a monic irreducible polynomial of degree t such as $f(\alpha) = \alpha^t + f_{t-1}\alpha^{t-1} + \cdots + f_1\alpha + f_0$, known as the Goppa polynomial. To create the Goppa polynomial, the coefficients of a basic finite field $GF(2^m)$ are utilized. In the NIST submission, this basic finite field has 8,192 elements when $m = 13$, i.e., $\alpha_0, \alpha_1, \ldots, \alpha_{8191}$, which are all 13-bit vectors. The private key, which consists of the Goppa polynomial and a permutation matrix P, is kept hidden, since the control matrix H is created by using such key and three other matrices designated as X, Y, and Z. Thereafter, a permutation utilizing the matrix P is done to produce the public key. This process yields a large public key $\tilde{\mathbb{H}}$, which gets shortened by converting it into a binary form H_2 over $GF(2)$ and by utilizing the matrices Π_{mt} and R to convert it into a systematic form $\tilde{G}$. Finally, $\tilde{G}$ is transposed into G, obtaining the public key, represented as R^T. Algorithm 1 shows how the pair of keys are generated.

For the process of encryption in MECS, an l-bit plaintext m and the public key R^T are required. A random n-bit error vector e, a random $(k - l)$-bit vector r_1, and a random l-bit vector r_2 are generated next. Then, public key R^T is expanded to $G = [R^T|\Pi_k]$, a hash function $h = hash(m||r_2)$ is performed, and a safe plaintext $\tilde{m} = r_1||h$ is created to be encoded into $z' = \tilde{m}G$. Finally, the ciphertext z is calculated by performing $z = (z' \oplus e)||(hash(r_1) \oplus m)||(hash(e) \oplus r_2)$.

The decryption process in MECS uses the ciphertext z and the private key $(P, g(\alpha))$ to obtain the plaintext m. First, the ciphertext z is split into (z_1, z_2, z_3), where z_1 is n bits long and z_2 and z_3

ALGORITHM 1: MECS key generation

1: Choose the parameters m, t, and n.
2: Calculate k according to m, t, and n where $k = n - mt$.
3: Randomly create a monic, irreducible polynomial $f(\alpha) = \alpha^t + f_{t-1}\alpha^{t-1} + \cdots + f_1\alpha + f_0$ using the coefficients in $GF(2^m)$ and degree of t.
4: Create the auxiliary matrices $t \times t$ matrix X, $t \times n$ matrix Y, and $n \times n$ matrix Z.
5: Calculate the $t \times n$ control matrix $H = XYZ$.
6: Randomly pick a permutation matrix P and compute the permuted control matrix $\tilde{\mathbb{H}} = HP^T$.
7: Transform $\tilde{\mathbb{H}}$ into H_2 over $GF(2)$ and then, into the form $\tilde{G} = [\Pi_{mt}|R]$.
8: $\tilde{G}$ is transposed into G obtaining R^T.
9: Return R^T as its public key, and $(P, f(\alpha))$ as its private key.

are l bits long. z_1 is permuted into $c = z_1 P$ and an error vector e' is reconstructed by employing the Patterson algorithm [32]. Then, the error vector is permuted $e = e' P^T$ and z' is reconstructed by performing $z' = z_1 \oplus e$. Next, z_1 is split into r_1 ($k - l$ bits long), h (l bits long), and $\tilde{m}$ ($n - k$ bits long). Finally, the plaintext is reconstructed $m = z_2 \oplus hash(r_1)$ as well as $r_2 : r_2 = hash(e) \oplus z_3$, and if $hash(m||r_2) \equiv h$, then the plaintext has no errors and it is returned, meaning that the entire process of decryption is completed.

3 PROPOSED FAULT DETECTION ARCHITECTURES

Differential fault analysis (DFA) compares a correct output with a defective one (produced by a natural cause or a third party) generally to obtain the private key. We can observe several fault models based on the sort of attack. These models depend on the amount of bits compromised, where the faults are located, how the faults are introduced, and the duration of the faults. Due to technological limitations, an adversary may not be able to flip precisely one bit to capture sensitive information. In practice, the attacker attempts to introduce as few faults as possible (ideally single faults of varying intensities) to minimize the effort. Biased fault models with a single-bit (more probable in low fault intensity), two-bit, three-bit, and four-bit (more frequent in higher intensities) may be utilized to mimic fault intensity fluctuation. In this work, techniques that can identify multiple stuck-at faults (both stuck-at one and stuck-at zero cases), adjacent (for interleaved cases), and single or multiple stuck-at faults are addressed, i.e., regular parity, interleaved parity, CRC-2, and CRC-8. These schemes also aim to detect transient and permanent internal faults on the Key Generator. We take into consideration an acceptable tradeoff between the fault detection abilities and the overheads to be accepted while providing the relevant error detection techniques. Because of their low overhead and good error coverage, the schemes presented in this work are suitable for embedded devices.

The Key Generator has the largest area complexity and, as a result, it is the most extensive hardware implementation inside McEliece. The *H-generator* is the most involved and complex block in the design of the Key Generator. It generates a control matrix H required to get the public key of the McEliece cryptosystem. As previously stated, H is generated by multiplying the matrices X, Y, and Z using the *G-memory*, *H-memory*, *Horner*, $GF(2^m)$ *Multiplication*, $GF(2^m)$ *Inverse*, and $GF(2^m)$ *Generator* blocks. First, an auxiliary matrix t x t X is created by using the *G-memory*, which contains the Goppa polynomial and is expressed as

$$X = \begin{Bmatrix} g_t & 0 & \cdots & 0 \\ g_{t-1} & g_t & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & \cdots & g_t \end{Bmatrix}.$$

Matrix Y is then generated using the $GF(2^m)$ *Generator* block, which produces α_i elements where $i \in \{0, 1, \ldots, 8, 191\}$ (this work uses the composite fields of complex Goppa codes, i.e., $GF((2^{13})^{128})$, $GF(2^{13})$ with the field polynomial $p(\alpha) = \alpha^{13} + \alpha^4 + \alpha^3 + \alpha + 1$, and $GF(2)$; however, numbers can vary depending on the security parameters utilized). Matrix Y is a t x n matrix expressed as

$$Y = \begin{Bmatrix} 1 & 0 & \cdots & 0 \\ \alpha_0 & \alpha_1 & \cdots & \alpha_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{t-1} & \alpha_1^{t-1} & \cdots & \alpha_{n-1}^{t-1} \end{Bmatrix}.$$

ALGORITHM 2: Multiplicative Inversion Addition-Chain Itoh-Tsujii Algorithm

1: $\beta_0 = A(\alpha)$
2: for i from 1 to t do
3: $\quad \beta_i = [\beta_{i_1}]^{2^{c_{i_2}}} \cdot \beta_{i_2} \pmod{p(\alpha)}$
4. return $((\beta_t)^2 \pmod{p(\alpha)})$

The inversion of $f(\alpha_i)$ is then used to calculate a matrix Z size n x n, requiring the $GF(2^m)$ *Inverse* block to obtain

$$Z = \begin{Bmatrix} g(\alpha_0) & 0 & \cdots & 0 \\ 0 & g(\alpha_1)^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g(\alpha_{n-1})^{-1} \end{Bmatrix}.$$

The *H-Generator* operates as follows: Matrices X and Y are multiplied first using the *Horner* block, which transforms high degree multiplications into easier and more efficient ones, e.g., the operation $f_1 + f_2\alpha_2 + f_3\alpha_2^2$ is calculated as $f_1 + (f_2 + f_3\alpha_2)\alpha_2$. To execute *(XY)Z*, $GF(2^m)$ *Multiplication* blocks are needed, since α_i is multiplied by the corresponding element accessible from XY to get $f(\alpha_i)$, e.g., $f_1 + f_2\alpha_2 + f_3\alpha_0^2$, which is available in the *H-memory*, is multiplied by α_0 to obtain $f(\alpha_0)$. Then, utilizing the $GF(2^m)$ *Inverse* block, $f(\alpha_i)$ is inverted.

The process of obtaining a particular element $A^{-1} \epsilon\, GF(2^m)$ so $A \cdot A^{-1} = 1$ is denoted as performing the multiplicative inverse of the element $A \neq 0$ over $GF(2^m)$. The FLT and ITA methods are investigated in this article to perform the multiplicative inverse of any finite-field element over $GF(2^m)$. According to FLT, the inverse of a finite-field element A is calculated as $A^{2^m-2} \equiv A^{-1} \bmod p(\alpha)$. However, the FLT algorithm yields to 2^m-2 multiplications over $GF(2^m)$ in hardware implementations, needing additional memory to hold the precomputed data. There have been many studies to reduce the amount of gates needed for finite-field inversions, e.g., Kaliski inversion, square-and-multiply algorithm, and ITA algorithm. The latter approach, which was developed by Itoh and Tsujii, greatly reduces the total amount of finite-field multiplications involved in the exponentiation by effectively using addition chains. The inverse of a finite-field element A is represented as $A^{-1} = [\beta_{m-1}(A)]^2$, where $\beta_k(A) = A^{2^k-1} \epsilon\, GF(2^m)$ and $k \epsilon\, \mathbb{N}$. To compute $\beta_{m-1}(A)$, [15] uses a recursive sequence with an addition chain for $m-1$ to calculate $\beta_{m-1}(A)$. To compute the addition chain $C = \{c_1, c_2, \ldots, c_t\}$ using $p(\alpha)$ or field polynomial of m degree, we need $c_1 = 1$ and $c_t = m-1$. If c_i is odd, then $c_{i-1} = c_i - 1$; if c_i is even, then $c_{i-1} = c_i/2$. Moreover, Algorithm 2 shows the Multiplicative Inversion Addition-Chain ITA. To calculate the inversion of an element in $GF(2^{13})$ using ITA with addition chains, 4 multiplications and 12 squarings in $GF(2^{13})$ are needed.

The hardware design to execute finite-field multiplications over $GF(2^m)$ is split into three modules, i.e., α, *sum*, and *pass-thru* modules. The α module reduces the output modulo $F(\alpha)$ after multiplying a finite-field element by α, the *sum* module utilizes m number of XOR gates to add two elements in $GF(2^m)$ (finite-field additions only use the *sum* module), and the *pass-thru* module multiplies an element in $GF(2^m)$ with an element in $GF(2)$. However, to perform finite-field squarings, only two modules are needed, i.e., α^2 (where a finite-field element is multiplied by α^2) and *sum* modules.

3.1 Regular and Interleaved Parity

Derivations for the α module with regular parity are formulated in the work of Reference [33]. This type of parity is suitable for single faults; however, it does not detect an even amount of faults. Therefore, the goal of our initial derivations is to ensure that contiguous faults are identified as

well. Next, Theorem 3.1 is proposed for error detection in the α module based on interleaved parity, where element A is the input and X is the output. Without compromising generality, m is assumed to be odd in Theorem 3.1; however, it is simple to adapt it for even m.

THEOREM 3.1. *Consider a $GF(2^m)$ element A, where $p_{Ae} = \sum_{i=0}^{\frac{m-1}{2}} a_{2i}$ represents the even parity bits of A, and $p_{Ao} = \sum_{i=1}^{\frac{m-1}{2}} a_{2i-1}$ represents the odd parity bits of A. Let $f_i \in GF(2)$ for $i = 0, 1, \ldots, m-1$ and m, which is used as the code subspace dimension, be odd. Then, the predicted parities of the output X denoted as $\hat{p}_{Xe}$ and $\hat{p}_{Xo}$ for even and odd bits, respectively, are*

$$\hat{p}_{Xe} = a_{m-1} + \sum_{i=1}^{\frac{m-1}{2}} (a_{2i-1} + a_{m-1} \cdot f_{2i}), \tag{1}$$

$$\hat{p}_{Xo} = \sum_{i=1}^{\frac{m-1}{2}} (a_{(2i-1)-1} + a_{m-1} \cdot f_{2i-1}), \tag{2}$$

and when $m = 13$, $\hat{p}_{Xe} = p_{Ao}$ and $\hat{p}_{Xo} = p_{Ae} + a_{12}$ are obtained.

PROOF. The X coordinates are calculated by utilizing the following formula:

$$x_i = \begin{cases} a_{i-1} + a_{m-1} \cdot f_i & 1 \le i \le m-1, \\ a_{m-1} & i = 0, \end{cases} \tag{3}$$

which divides the predicted parity $\hat{p}_X$ as

$$\begin{aligned} \hat{p}_X &= a_{m-1} + \sum_{i=1}^{m-1} (a_{i-1} + a_{m-1} \cdot f_i) \\ &= a_{m-1} + \sum_{i=1}^{\frac{m-1}{2}} (a_{2i-1} + a_{m-1} \cdot f_{2i}) \\ &\quad + \sum_{i=1}^{\frac{m-1}{2}} (a_{(2i-1)-1} + a_{m-1} \cdot f_{2i-1}) \\ &= \hat{p}_{Xe} + \hat{p}_{Xo}. \end{aligned}$$

The field polynomial used in our design is $p(\alpha) = \alpha^{13} + \alpha^4 + \alpha^3 + \alpha + 1$, obtaining $f_{13} = f_4 = f_3 = f_1 = f_0 = 1$. Then, from Equations (1) and (2), we have

$$\begin{aligned} \hat{p}_{Xe} &= a_{12} + a_1 + a_3 + a_{12} + a_5 + a_7 + a_9 + a_{11} \\ &= p_{Ao} \end{aligned}$$

and

$$\begin{aligned} \hat{p}_{Xo} &= a_0 + a_{12} + a_2 + a_{12} + a_4 + a_6 + a_8 + a_{10} \\ &= p_{Ae} + a_{12}. \end{aligned}$$

This brings the proof to a close. □

Elements A and B in $GF(2^m)$ are added in the *sum* module to produce the even and odd predicted parities of output D, denoted as $\hat{p}_{De}$ and $\hat{p}_{Do}$, respectively, obtaining $\hat{p}_{De} = p_{Ae} + p_{Be}$ and $\hat{p}_{Do} = p_{Ao} + p_{Bo}$.

Last, in the *pass-thru* module, a $GF(2)$ element b is multiplied by the parity bits of A, which are split into p_{Ae} and p_{Ao}, producing output G. Next, the even and odd predicted parities of G, represented as $\hat{p}_{Ge}$ and $\hat{p}_{Go}$, are split into $\hat{p}_{Ge} = b \cdot p_{Ae}$ and $\hat{p}_{Go} = b \cdot p_{Ao}$, respectively.

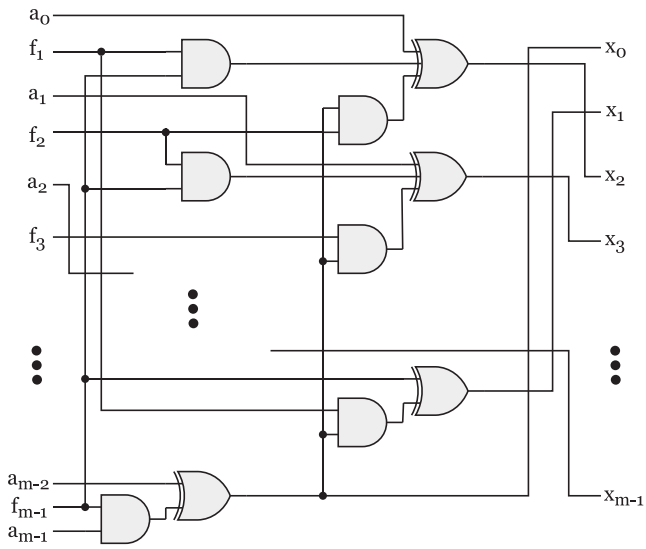

Fig. 1. The proposed design for α^2 module, where a_i and f_i's symbolize the inputs and x_i's symbolize the outputs.

To perform a finite-field squaring, α^2 and *sum* modules are required. In the α^2 module, element A is multiplied by α^2, obtaining

$$A(\alpha) \cdot \alpha^2 = a_{m-1} \cdot \alpha^{m+1} + a_{m-2} \cdot \alpha^m + \cdots + a_0 \cdot \alpha^2, \tag{4}$$

where

$$\alpha^{m+1} = f_{m-1} \cdot \alpha^m + f_{m-2} \cdot \alpha^{m-1} + \cdots + f_0 \cdot \alpha \bmod p(\alpha)$$

and

$$\alpha^m = f_{m-1} \cdot \alpha^{m-1} + f_{m-2} \cdot \alpha^{m-2} + \cdots + f_0 \bmod p(\alpha),$$

using $p(\alpha)$ as the field polynomial. Moreover, using Equation (4), the X coordinates are represented as

$$x_i = \begin{cases} a_{m-1} \cdot f_{i-1} + (a_{m-1} \\ \cdot f_{m-1} + a_{m-2}) \cdot f_i \\ + a_{i-2} & 2 \le i \le m-1, \\ a_{m-1} + (a_{m-1} \cdot f_{m-1} \\ + a_{m-2}) \cdot f_1 & i = 1, \\ a_{m-1} \cdot f_{m-1} + a_{m-2} & i = 0. \end{cases} \tag{5}$$

The hardware design of the α^2 module is presented in Figure 1, where a_0-a_{m-1} are the coefficients of input A, f_0-f_{m-1} are the field polynomial $p(\alpha)$ coefficients, and α_0-α_{m-1} are denoted as the output X coefficients. Additionally, the derivations from Equation (5) are shown in Figure 1, using several XOR and AND gates to obtain such formulations. For instance, the output x_0 is obtained by XORing the input a_{m-2} with the result of adding through an AND gate the inputs a_{m-1} and f_{m-1}, the output x_1 is obtained by XORing a_{m-1} with the result of $(a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1$ (using two AND gates and an extra XOR gate), and so on. To derive the regular and interleaved parities of the α^2 module, Theorems 3.2 and 3.3 are presented next.

Theorem 3.2. *Consider a $GF(2^m)$ element A, where $p_A = \sum_{i=0}^{m-1} a_i$ represents the parity bits of A. Let $f_i \in GF(2)$ for $i = 0, 1, \ldots, m-1$ and m be used as the code subspace dimension. Then, the predicted parity of the output X is*

$$\begin{aligned}\hat{p}_X = a_{m-1} \cdot f_{m-1} + a_{m-2} + a_{m-1} + (a_{m-1} \\ \cdot f_{m-1} + a_{m-2}) \cdot f_1 + \sum_{i=2}^{m-1} (a_{m-1} \cdot f_{i-1} \\ + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2}),\end{aligned} \tag{6}$$

and when $m = 13$, $\hat{p}_X = p_A + a_{11} + a_{12}$ is obtained.

Proof. Our proposed scheme uses the field polynomial $p(\alpha) = \alpha^{13} + \alpha^4 + \alpha^3 + \alpha + 1$. Therefore, $f_{13} = f_4 = f_3 = f_1 = f_0 = 1$. Then from Equation (6), one obtains

$$\begin{aligned}\hat{p}_X = a_{11} + a_{12} + a_{11} + a_{12} + a_0 + a_{11} + a_1 + a_{12} + a_{11} \\ + a_2 + a_{12} + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 + a_9 + a_{10} \\ = p_A + a_{11} + a_{12}.\end{aligned}$$

This brings the proof to a close. □

Theorem 3.3. *Consider a $GF(2^m)$ element A, where $p_{Ae} = \sum_{i=0}^{\frac{m-1}{2}} a_{2i}$ represents the even parity bits of A, and $p_{Ao} = \sum_{i=1}^{\frac{m-1}{2}} a_{2i-1}$ represents the odd parity bits of A. Let $f_i \in GF(2)$ for $i = 0, 1, \ldots, m-1$ and m, which is used as the code subspace dimension, be odd. Then, predicted parities of the output X denoted as $\hat{p}_{Xe}$ and $\hat{p}_{Xo}$ for even and odd bits, respectively, are*

$$\begin{aligned}\hat{p}_{Xe} = a_{m-1} \cdot f_{m-1} + a_{m-2} + \sum_{i=1}^{\frac{m-1}{2}} (a_{m-1} \\ \cdot f_{2i-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_{2i} + a_{2i-2}),\end{aligned} \tag{7}$$

$$\begin{aligned}\hat{p}_{Xo} = a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1 + \sum_{i=1}^{\frac{m-3}{2}} (a_{m-1} \\ \cdot f_{2i} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_{2i+1} + a_{2i-1}),\end{aligned} \tag{8}$$

and when $m = 13$, $\hat{p}_{Xe} = p_{Ae} + a_{12}$ and $\hat{p}_{Xo} = p_{Ao} + a_{11}$ are obtained.

Proof. The predicted parity $\hat{p}_X$ is divided into

$$\begin{aligned}\hat{p}_X = a_{m-1} \cdot f_{m-1} + a_{m-2} + a_{m-1} + (a_{m-1} \\ \cdot f_{m-1} + a_{m-2}) \cdot f_1 + \sum_{i=1}^{m-1} (a_{m-1} \cdot f_{2i-1} \\ + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_{2i} + a_{2i-2}) \\ = \hat{p}_{Xe} + \hat{p}_{Xo}.\end{aligned}$$

The field polynomial utilized in our design is $p(\alpha) = \alpha^{13} + \alpha^4 + \alpha^3 + \alpha + 1$; therefore, one obtains $f_{13} = f_4 = f_3 = f_1 = f_0 = 1$. Then, from Equations (7) and (8), we have

$$\begin{aligned}\hat{p}_{Xe} = a_{11} + a_{12} + a_0 + a_{12} + a_{11} + a_2 + a_4 + a_6 + a_8 + a_{10} \\ = p_{Ae} + a_{12}\end{aligned}$$

and

$$\begin{aligned}\hat{p}_{Xo} = a_{12} + a_{11} + a_{11} + a_1 + a_{12} + a_3 + a_5 + a_7 + a_9 \\ = p_{Ao} + a_{11}.\end{aligned}$$

This brings the proof to a close. □

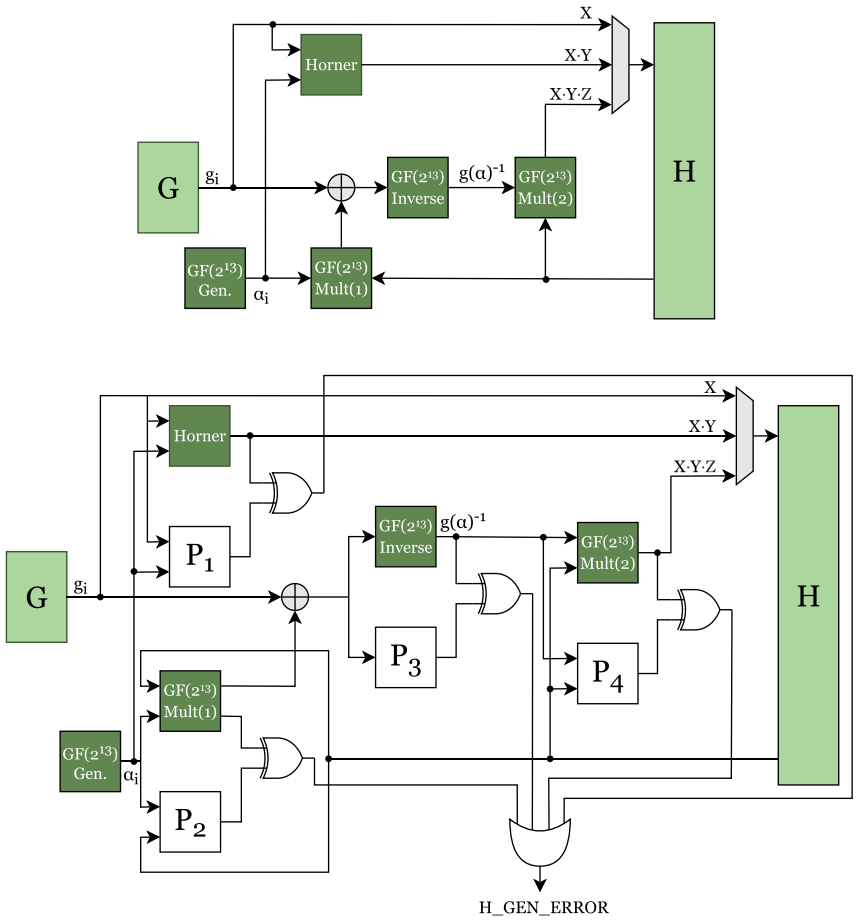

Fig. 2. The presented fault detection scheme of the *H-generator* for MECS.

Figure 2 shows both the original *H-generator* architecture (top) and the *H-generator* with the presented fault detection blocks (bottom). The proposed design produces the matrix H by multiplying matrices X, Y, and Z. In Figure 2, the *H-generator* may employ both regular and interleaved parities represented as $P_1 - P_4$. The different fault detection blocks have been simplified in Figure 2 as four big white blocks; however, each of those P blocks contain many XOR gates, OR gates, and error flags. In Section 4, we calculate the amount of signatures, which relate to the footprint of the output of an error-detecting block, for the entire Key Generator. Nevertheless, let us go over a specific example to show how the P blocks behave. For instance, the $GF(2^{13})$ *Mult(2)* block performs a total of $8,192 \cdot 128$ multiplications to obtain XYZ. Each of those multiplications uses 12 α, 12 *sum*, and 13 *pass-thru* modules, which translates into $(8{,}192 \cdot 128)_{mult} \cdot (12_{\alpha} + 12_{sum} + 13_{pass})$ sub-outputs. Each of these sub-outputs are compared with the predicted sub-outputs of the P_4 block, obtaining $(8{,}192 \cdot 128)_{mult} \cdot (12_{\alpha} + 12_{sum} + 13_{pass})$ or close to 3.9×10^7 signatures, which are then ORed with each other. We note that the term signature here refers to appended bits used for error detection through error-detecting codes and not the typical signatures commonly used for proof of authenticity in cryptography. If the fault detection scheme used is based on interleaved parity, then the number of error flags will double. As deducted from Theorem 3.1 for finite-field

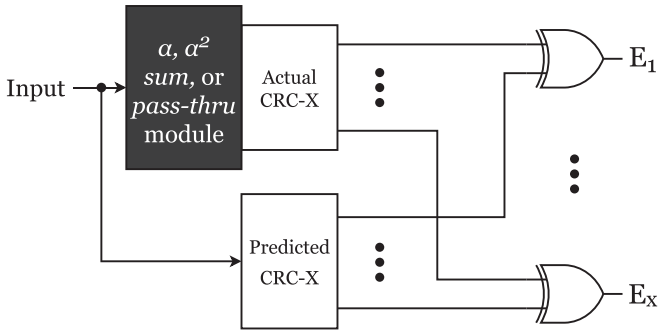

Fig. 3. CRC applied to the original α, α^2, *sum*, and *pass-thru* modules.

multiplications, P_2 and P_4 blocks contain the different gates to obtain Equations (1) and (2). After the Horner derivations are performed, P_1 is used, using Equations (1) and (2) to provide error detection. Regular and interleaved parities for multiplications and squarings over $GF(2^m)$ formulated in Theorem 3.1 and Theorems 3.2 and 3.3, respectively, are performed by the P_3 block. Therefore, the P_3 block uses the Equations (1), (2), and (6)–(8) to provide error detection capabilities to the $GF(2^{13})$ inverse block. All the different error detection blocks are connected together through an OR gate that indicates if a fault has been detected in any block of the *H-generator*.

3.2 CRC

CRC uses cyclic error-correcting codes. First, a generator polynomial $g(\alpha)$ is selected to perform CRC. Next, a long division of polynomials is calculated, where $g(\alpha)$ becomes the divisor, the data becomes the dividend, the remainder generates the result, and the quotient is disregarded. Last, the data is appended with a specified number of check bits, which are examined when the output is retrieved to identify any faults. The CRCs that are used along this work are customizable depending on the security considerations and the amount of overhead that may be accepted. To put it another way, for applications like gaming consoles where performance is crucial (because they are plugged in, their power usage is not), the CRC size may be increased. Nonetheless, smaller CRCs are desirable for constrained devices.

CRCs in the *sum* and *pass-thru* modules need fewer formulations than those in the α and α^2 modules. The predicted CRC-1 for the *sum* module is equivalent to the parity bits of the inputs A and B in $GF(2^m)$ addition, which give us $\hat{p}_X = p_A + p_B$. Furthermore, CRC for the *pass-thru* module, where b is an element in $GF(2)$, corresponds to $\hat{p}_X = b \cdot p_A$. Instead of adding all the parity bits as in CRC-1, the *sum* and *pass-thru* modules for each CRC-X scheme verify X bits at a time. The NIST field $GF(2^{13})$ is utilized next in conjunction with CRC-2 and CRC-8. Figure 3 shows how CRC is added to the original α, α^2, *sum*, and *pass-thru* modules. The number of error flags, denoted as E_x, is directly related to the CRC scheme used.

3.2.1 CRC for α Module. In the α module, multiplying an element in $GF(2^{13})$ by α produces

$$\begin{aligned} A(x) \cdot x = {} & a_{12} \cdot \alpha^{13} + a_{11} \cdot \alpha^{12} + a_{10} \cdot \alpha^{11} + a_9 \cdot \alpha^{10} \\ & + a_8 \cdot \alpha^9 + a_7 \cdot \alpha^8 + a_6 \cdot \alpha^7 + a_5 \cdot \alpha^6 + a_4 \cdot \alpha^5 \\ & + a_3 \cdot \alpha^4 + a_2 \cdot \alpha^3 + a_1 \cdot \alpha^2 + a_0 \cdot \alpha, \end{aligned} \tag{9}$$

where

$$\alpha^{13} = f_{12}\alpha^{12} + f_{11}\alpha^{11} + \cdots + f_1\alpha + f_0 \bmod p(\alpha).$$

The irreducible polynomial $p(\alpha) = \alpha^{13} + \alpha^4 + \alpha^3 + \alpha + 1$ is then employed to get

$$\begin{aligned} A(\alpha) \cdot \alpha \equiv\ & a_{12}\alpha^4 + a_{12}\alpha^3 + a_{12}\alpha + a_{12} + a_{11}\alpha^{12} \\ & + a_{10}\alpha^{11} + a_9\alpha^{10} + a_8\alpha^9 + a_7\alpha^8 + a_6\alpha^7 + a_5\alpha^6 \\ & + a_4\alpha^5 + a_3\alpha^4 + a_2\alpha^3 + a_1\alpha^2 + a_0\alpha \bmod p(\alpha). \end{aligned} \tag{10}$$

- For the case study of $m = 13$ with CRC-2 in the α module, $g_0(\alpha) = \alpha^2 + \alpha + 1$ is utilized as the generator polynomial, which is used to find its derivations as follows:

$$\begin{aligned} \alpha^2 &\equiv \alpha + 1 \bmod g_0(\alpha), \\ \alpha^3 &\equiv 1 \bmod g_0(\alpha), \\ \alpha^4 &\equiv \alpha \bmod g_0(\alpha), \\ \alpha^5 &\equiv \alpha + 1 \bmod g_0(\alpha), \\ \alpha^6 &\equiv 1 \bmod g_0(\alpha), \\ \alpha^7 &\equiv \alpha \bmod g_0(\alpha), \\ \alpha^8 &\equiv \alpha + 1 \bmod g_0(\alpha), \\ \alpha^9 &\equiv 1 \bmod g_0(\alpha), \\ \alpha^{10} &\equiv \alpha \bmod g_0(\alpha), \\ \alpha^{11} &\equiv \alpha + 1 \bmod g_0(\alpha), \\ \alpha^{12} &\equiv 1 \bmod g_0(\alpha). \end{aligned}$$

  Next, to determine the predicted CRC-2 equation for $GF(2^{13})$ in the α module, denoted as $(PCRC2_{13})$, $g_0(\alpha)$ is applied in Equation (10), obtaining

$$\begin{aligned} A(\alpha) \cdot \alpha \equiv\ & a_{11} + a_{10}(\alpha + 1) + a_9\alpha + a_8 + a_7(\alpha \\ & + 1) + a_6\alpha + a_5 + a_4(\alpha + 1) + a_3\alpha + a_2 + a_1(\alpha \\ & + 1) + a_0\alpha \bmod g_0(\alpha), \end{aligned}$$

  or

$$\begin{aligned} PCRC2_{13} = (& a_{10} + a_9 + a_7 + a_6 + a_4 + a_3 \\ & + a_1 + a_0)\alpha + (a_{11} + a_{10} + a_8 + a_7 + a_5 \\ & + a_4 + a_2 + a_1). \end{aligned} \tag{11}$$

  Then, the coefficients from Equation (10) are renamed to determine the actual CRC-2 equation for $GF(2^{13})$ in the α module $(ACRC2_{13})$: a_{11} as $\gamma_{12}, \ldots, a_0$ as γ_1,

$$\begin{aligned} A(x) \cdot x \equiv\ & \gamma_{12}\alpha^{12} + \gamma_{11}\alpha^{11} + \gamma_{10}\alpha^{10} + \gamma_9\alpha^9 \\ & + \gamma_8\alpha^8 + \gamma_7\alpha^7 + \gamma_6\alpha^6 + \gamma_5\alpha^5 + \gamma_4\alpha^4 \\ & + \gamma_3\alpha^3 + \gamma_2\alpha^2 + \gamma_1\alpha^1 + \gamma_0 \bmod g_0(\alpha), \end{aligned} \tag{12}$$

  and we apply $g_0(\alpha)$ as follows:

$$\begin{aligned} A(\alpha) \cdot \alpha \equiv\ & \gamma_{12} + \gamma_{11}(\alpha + 1) + \gamma_{10}\alpha + \gamma_9 \\ & + \gamma_8(\alpha + 1) + \gamma_7\alpha + \gamma_6 + \gamma_5(\alpha + 1) + \gamma_4\alpha \\ & + \gamma_3 + \gamma_2(\alpha + 1) + \gamma_1\alpha + \gamma_0 \bmod g_0(\alpha), \end{aligned}$$

  or

$$\begin{aligned} ACRC2_{13} = (& \gamma_{11} + \gamma_{10} + \gamma_8 + \gamma_7 + \gamma_5 + \gamma_4 \\ & + \gamma_2 + \gamma_1)\alpha + (\gamma_{12} + \gamma_{11} + \gamma_9 + \gamma_8 + \gamma_6 + \gamma_5 \\ & + \gamma_3 + \gamma_2 + \gamma_0). \end{aligned} \tag{13}$$

- For the case study of $m = 13$ with CRC-8 in the α module, $g_1(\alpha) = \alpha^8 + \alpha^2 + \alpha + 1$ is utilized as the generator polynomial, which is used to find its derivations as follows:

$$\begin{aligned}
\alpha^8 &\equiv \alpha^2 + \alpha + 1 \bmod g_1(\alpha),\\
\alpha^9 &\equiv \alpha^3 + \alpha^2 + \alpha \bmod g_1(\alpha),\\
\alpha^{10} &\equiv \alpha^4 + \alpha^3 + \alpha^2 \bmod g_1(\alpha),\\
\alpha^{11} &\equiv \alpha^5 + \alpha^4 + \alpha^3 \bmod g_1(\alpha),\\
\alpha^{12} &\equiv \alpha^6 + \alpha^5 + \alpha^4 \bmod g_1(\alpha).
\end{aligned}$$

Next, to determine the predicted CRC-8 equation for $GF(2^{13})$ in the α module, denoted as $(PCRC8_{13})$, $g_1(\alpha)$ is applied in Equation (10), obtaining

$$\begin{aligned}
A(\alpha)\cdot\alpha \equiv\ & a_{12}(\alpha^4 + \alpha^3 + \alpha + 1) + a_{11}(\alpha^6 + \alpha^5 + \alpha^4)\\
&+ a_{10}(\alpha^5 + \alpha^4 + \alpha^3) + a_9(\alpha^4 + \alpha^3 + \alpha^2) + a_8(\alpha^3\\
&+ \alpha^2 + \alpha) + a_7(\alpha^2 + \alpha + 1) + a_6\alpha^7 + a_5\alpha^6 + a_4\alpha^5\\
&+ a_3\alpha^4 + a_2\alpha^3 + a_1\alpha^2 + a_0\alpha \bmod g_1(\alpha),
\end{aligned}$$

or

$$\begin{aligned}
PCRC8_{13} =\ & a_6\alpha^7 + (a_{11} + a_5)\alpha^6 + (a_{11} + a_{10}\\
&+ a_4)\alpha^5 + (a_{12} + a_{11} + a_{10} + a_9 + a_3)\alpha^4 +\\
&(a_{12} + a_{10} + a_9 + a_8 + a_2)\alpha^3 + (a_9 + a_8 + a_7\\
&+ a_1)\alpha^2 + (a_{12} + a_8 + a_7 + a_0)\alpha + (a_{12} + a_7).
\end{aligned} \tag{14}$$

Then, the coefficients from Equation (10) are renamed to determine the actual CRC-8 equation, denoted as $ACRC8_{13}$, for $GF(2^{13})$ in the α module: a_{11} as $\gamma_{12}, \ldots, a_0$ as γ_1,

$$\begin{aligned}
A(\alpha)\cdot\alpha \equiv\ & \gamma_{12}\alpha^{12} + \gamma_{11}\alpha^{11} + \gamma_{10}\alpha^{10} + \gamma_9\alpha^9\\
&+ \gamma_8\alpha^8 + \gamma_7\alpha^7 + \gamma_6\alpha^6 + \gamma_5\alpha^5 + \gamma_4\alpha^4\\
&+ \gamma_3\alpha^3 + \gamma_2\alpha^2 + \gamma_1\alpha + \gamma_0 \bmod g_1(\alpha),
\end{aligned} \tag{15}$$

and we apply $g_0(\alpha)$ as follows:

$$\begin{aligned}
A(\alpha)\cdot\alpha \equiv\ & \gamma_{12}(\alpha^6 + \alpha^5 + \alpha^4) + \gamma_{11}(\alpha^5 + \alpha^4 + \alpha^3)\\
&+ \gamma_{10}(\alpha^4 + \alpha^3 + \alpha^2) + \gamma_9(\alpha^3 + \alpha^2 + \alpha) + \gamma_8(\alpha_2\\
&+ \alpha + 1) + \gamma_7\alpha^7 + \gamma_6\alpha^6 + \gamma_5\alpha^5 + \gamma_4\alpha^4 + \gamma_3\alpha^3\\
&+ \gamma_2\alpha^2 + \gamma_1\alpha + \gamma_0 \bmod g_1(\alpha),
\end{aligned}$$

or

$$\begin{aligned}
ACRC8_{13} =\ & \gamma_7\alpha^7 + (\gamma_{12} + \gamma_6)\alpha^6 + (\gamma_{12} + \gamma_{11}\\
&+ \gamma_5)\alpha^5 + (\gamma_{12} + \gamma_{11} + \gamma_{10} + \gamma_4)\alpha^4 + (\gamma_{11}\\
&+ \gamma_{10} + \gamma_9 + \gamma_3)\alpha^3 + (\gamma_{10} + \gamma_9 + \gamma_8 + \gamma_2)\alpha^2\\
&+ (\gamma_9 + \gamma_8 + \gamma_1)\alpha + (\gamma_8 + \gamma_0).
\end{aligned} \tag{16}$$

The NIST field $GF(2^{13})$ is utilized next with CRC-2 and CRC-8 for the α^2 module; nonetheless, the presented error detection techniques can be applied with any field sizes or CRCs.

3.2.2 CRC for α^2 Module. In the α^2 module, multiplying any element in $GF(2^{13})$ by α^2 produces

$$\begin{aligned} A(\alpha)\cdot\alpha^2 = {} & a_{12}\cdot\alpha^{14} + a_{11}\cdot\alpha^{13} + a_{10}\cdot\alpha^{12} \\ & + a_9\cdot\alpha^{11} + a_8\cdot\alpha^{10} + a_7\cdot\alpha^9 + a_6\cdot\alpha^8 \\ & + a_5\cdot\alpha^7 + a_4\cdot\alpha^6 + a_3\cdot\alpha^5 + a_2\cdot\alpha^4 \\ & + a_1\cdot\alpha^3 + a_0\cdot\alpha^2, \end{aligned} \tag{17}$$

where $\alpha^{14} = f_{12}\alpha^{13} + f_{11}\alpha^{12} + \cdots + f_1\alpha^2 + f_0\alpha \bmod p(\alpha)$ and $\alpha^{13} = f_{12}\alpha^{12} + f_{11}\alpha^{11} + \cdots + f_1\alpha + f_0 \bmod p(\alpha)$. For $m = 13$, $p(\alpha) = \alpha^{13} + \alpha^4 + \alpha^3 + \alpha + 1$, which is the irreducible polynomial, is employed to get

$$\begin{aligned} A(\alpha)\cdot\alpha^2 \equiv {} & a_{12}\alpha^5 + a_{12}\alpha^4 + a_{12}\alpha^2 + a_{12}\alpha + a_{11}\alpha^4 \\ & a_{11}\alpha^3 + a_{11}\alpha + a_{11} + a_{10}\alpha^{12} + a_9\alpha^{11} + a_8\alpha^{10} \\ & + a_7\alpha^9 + a_6\alpha^8 + a_5\alpha^7 + a_4\alpha^6 + a_3\alpha^5 + a_2\alpha^4 \\ & + a_1\alpha^3 + a_0\alpha^2 \bmod p(\alpha). \end{aligned}$$

- For $m = 13$ with CRC-2 in the α^2 module, $g_0(\alpha)$ is utilized in Equation (17) to determine the predicted CRC-2 equation for $GF(2^{13})$, denoted as $PCRC2_{13}$, obtaining

$$\begin{aligned} A(\alpha)\cdot\alpha^2 \equiv {} & a_{10} + a_9(\alpha+1) + a_8\alpha + a_7 + a_6(\alpha \\ & + 1) + a_5\alpha + a_4 + a_3(\alpha+1) + a_2\alpha + a_1 + a_0(\alpha \\ & + 1) \bmod g_0(\alpha), \end{aligned}$$

  or

$$\begin{aligned} PCRC2_{13} = {} & (a_9 + a_8 + a_6 + a_5 + a_3 + a_2 + a_0) \\ & \cdot\alpha + (a_{10} + a_9 + a_7 + a_6 + a_4 + a_3 + a_1 + a_0). \end{aligned} \tag{18}$$

  Then, the coefficients from Equation (17) are renamed to determine the actual CRC-2 equation for $GF(2^{13})$ in the α^2 module, denoted as $ACRC2_{13}$, getting the same derivations as for the α module.
- For $m = 13$ with CRC-8 in the α^2 module, $g_1(\alpha)$ is utilized in Equation (17) to determine the predicted CRC-8 equation for $GF(2^{13})$, denoted as $PCRC8_{13}$, obtaining

$$\begin{aligned} A(\alpha)\cdot\alpha^2 \equiv {} & a_{12}(\alpha^5 + \alpha^4 + \alpha^2 + \alpha) + a_{11}(\alpha^4 + \alpha^3 \\ & + \alpha + 1) + a_{10}(\alpha^6 + \alpha^5 + \alpha^4) + a_9(\alpha^5 + \alpha^4 + \alpha^3) \\ & + a_8(\alpha^4 + \alpha^3 + \alpha^2) + a_7(\alpha^3 + \alpha^2 + \alpha) + a_6(\alpha^2 \\ & + \alpha + 1) + a_5\alpha^7 + a_4\alpha^6 + a_3\alpha^5 + a_2\alpha^4 + a_1\alpha^3 \\ & + a_0x^2 \bmod g_1(\alpha), \end{aligned}$$

  or

$$\begin{aligned} PCRC8_{13} = {} & a_5\alpha^7 + (a_{10} + a_4)\alpha^6 + (a_{12} + a_{10} + a_9 \\ & + a_3)\alpha^5 + (a_{12} + a_{11} + a_{10} + a_9 + a_8 + a_2)\alpha^4 + (a_{11} \\ & + a_9 + a_8 + a_7 + a_1)\alpha^3 + (a_{12} + a_8 + a_7 + a_6 \\ & + a_0)\alpha^2 + (a_{12} + a_{11} + a_7 + a_6)\alpha + (a_{11} + a_6). \end{aligned} \tag{19}$$

  Then, the coefficients from Equation (17) are renamed to determine the actual CRC-8 equation for $GF(2^{13})$ in the α^2 module, denoted as $ACRC8_{13}$, getting the same derivations as for the α module.

Table 2. Steps Required to Complete the Inverse of an Element A in $GF(2^{13})$ Employing the Addition Chain C

Step	$\beta_{V_i}(\alpha)$	$\beta_{V_j+U_k}(\alpha)$	Exponentiation
1	$\beta_1(\alpha)$	–	A
2	$\beta_2(\alpha)$	$\beta_{1+1}(\alpha)$	$(\beta_1)^{2^1}\beta_1 = A^{2^2-1}$
3	$\beta_3(\alpha)$	$\beta_{2+1}(\alpha)$	$(\beta_2)^{2^1}\beta_1 = A^{2^3-1}$
4	$\beta_6(\alpha)$	$\beta_{3+3}(\alpha)$	$(\beta_3)^{2^3}\beta_3 = A^{2^6-1}$
5	$\beta_{12}(\alpha)$	$\beta_{6+6}(\alpha)$	$(\beta_6)^{2^6}\beta_6 = A^{2^{12}-1}$

4 ERROR COVERAGE AND FPGA IMPLEMENTATIONS

As previously stated, multiplications in $GF(2^m)$ require three distinct modules (α, *sum,* and *pass-thru* modules), squarings in $GF(2^m)$ utilize the α^2 and *sum* modules, and additions in $GF(2^m)$ require just the *sum* module. For $m = 13$ using a bit-parallel design with regular parity, 12 α, 12 *sum*, and 13 *pass-thru* modules are utilized to perform each multiplication in $GF(2^{13})$; 12 α^2 and 12 *sum* modules are required for each squaring in $GF(2^{13})$; and just 12 *sum* modules are used on each addition in $GF(2^{13})$. The number of signatures for regular parity utilized by the P_i blocks, where $1 \leq i \leq 4$, is determined as:

(1) A 128×128 matrix X is generated to calculate the matrix XY. To achieve efficient polynomial multiplications, the Horner algorithm is performed. Each column of the $128 \times 8{,}192$ XY matrix needs a total of 127 multiplications and 127 additions in $GF(2^{13})$, resulting in a total of $8{,}192_{column} \cdot 127_{mult.} \cdot (12_{\alpha} + 12_{sum} + 13_{pass})$ signatures for finite-field multiplications and $8{,}192_{column} \cdot 127_{add.} \cdot 12_{sum}$ signatures for finite-field additions. Therefore, the *Horner* block requires more than $5 \cdot 10^7$ signatures.

(2) In Figure 2, it is shown that to derive the matrix Z, the $GF(2^{13})$ *Mult(1)* block is needed. Such block performs 8,192 multiplications and 8,192 additions in $GF(2^{13})$, resulting in a total of $8{,}192_{mult.} \cdot (12_{\alpha} + 12_{sum} + 13_{pass})$ signatures for finite-field multiplications, and $8{,}192_{add.} \cdot 12_{sum}$ signatures for finite-field additions. Therefore, the $GF(2^{13})$ *Mult(1)* block requires more than $4 \cdot 10^5$ signatures.

(3) Next, a total of 8,192 inversions in $GF(2^m)$ are performed by the $GF(2^{13})$ *Inverse* block shown in Figure 2. For $m = 13$, the addition chain utilized is $C = \{1, 2, 3, 6, 12\}$. Table 2 shows the different steps required to obtain the inverse of $A \epsilon GF(2^{13})$ using addition chains. In Table 2, the integers in the calculated addition chain are denoted as V_i, $V_j = V_{i-1}$, and $U_k = V_i - V_j$, requiring 4 multiplications and 12 squaring in $GF(2^{13})$. Therefore, a total of $8{,}192_{inv.} \cdot (4_{mult.} \cdot (12_{\alpha} + 12_{sum} + 13_{pass}) + 12_{add.} \cdot (12_{\alpha} + 12_{sum}))$ or close to $3.6 \cdot 10^6$ operations and signatures are required. We note that each finite-field multiplication and squaring requires a total of seven and six clock cycles, respectively, specifying a main clock time constraint of 20 ns, which corresponds to a frequency of 50 MHz.

(4) Then, the $GF(2^{13})$ *Mult(2)* block from Figure 2 requires $8{,}192 \cdot 128$ multiplications to obtain XYZ. Moreover, the $GF(2^{13})$ *Mult(2)* block uses $(8{,}192 \cdot 128)_{mult} \cdot (12_{\alpha} + 12_{sum} + 13_{pass})$ signatures to perform all the finite-field multiplications needed, totaling about $3.9 \cdot 10^7$ signatures.

(5) After the control matrix H is generated, other steps are computed. The matrix H is multiplied with a matrix P to be permuted, obtaining $\hat{\mathbb{H}}$. Our fault detection schemes can be integrated in this process, since it requires approximately $8.6 \cdot 10^9$ multiplications ($8{,}192 \cdot 8{,}192 \cdot 128$) and approximately $8.6 \cdot 10^9$ additions ($8{,}192 \cdot 8{,}192 \cdot 128$) in $GF(2^{13})$. The Gauss Systemizer unit, which conducts row permutations and XOR additions of two rows, is also used in the key generation. Permutations can be achieved in this situation by

Table 3. Area and Delay Results Obtained After Adding Fault Detection into the Original Horner and Inversion Architectures on FPGA Kintex-7 (Device xc7k70tfbv676-1)

Architecture	Area (occupied slices)	Delay (ns)
Original Horner block	2,976	28.267
Horner Reg. Parity (predicted/actual/compressor)	3,138 (5.44%)	28.541 (Neg. over.)
Horner Inter. Parity (predicted/actual/compressor)	3,402 (14.31%)	29.410 (Neg. over.)
Horner CRC-2 (predicted/actual/compressor)	3,285 (10.38%)	28.850 (Neg. over.)
Horner CRC-8 (predicted/actual/compressor)	3,572 (20.03%)	29.803 (5.43%)
Original Inversion block	783	28.820
Inversion Reg. Parity (predicted/actual/compressor)	976 (24.65%)	29.013 (Neg. over.)
Inversion Inter. Parity (predicted/actual/compressor)	1,083 (38.31%)	28.995 (Neg. over.)
Inversion CRC-2 (predicted/actual/compressor)	1,121 (43.17%)	28.720 (Neg. over.)
Inversion CRC-8 (predicted/actual/compressor)	1,166 (48.91%)	31.124 (7.99%)

Table 4. Power and Throughput Results Obtained After Adding Fault Detection into the Original Horner and Inversion Architectures on FPGA Kintex-7 (Device xc7k70tfbv676-1)

Architecture	Power (mW) @50 MHz	Throughput (Gbps)	Efficiency (Gbps/slices)
Original Horner block	0.144	0.460	1.54×10^{-4}
Horner Reg. Parity (predicted/actual/compressor)	0.147 (Neg. over.)	0.455 (Neg. over.)	1.45×10^{-4} (5.84%)
Horner Inter. Parity (predicted/actual/compressor)	0.157 (9.03%)	0.442 (Neg. over.)	1.30×10^{-4} (15.58%)
Horner CRC-2 (predicted/actual/compressor)	0.154 (6.94%)	0.451 (Neg. over.)	1.37×10^{-4} (11.04%)
Horner CRC-8 (predicted/actual/compressor)	0.158 (9.72%)	0.446 (Neg. over.)	1.25×10^{-4} (18.83%)
Original Inversion block	0.101	0.451	5.76×10^{-4}
Inversion Reg. Parity (predicted/actual/compressor)	0.108 (6.93%)	0.448 (Neg. over.)	4.59×10^{-4} (20.31%)
Inversion Inter. Parity (predicted/actual/compressor)	0.110 (8.91%)	0.448 (Neg. over.)	4.14×10^{-4} (28.12%)
Inversion CRC-2 (predicted/actual/compressor)	0.112 (10.89%)	0.453 (Neg. over.)	4.04×10^{-4} (29.86%)
Inversion CRC-8 (predicted/actual/compressor)	0.113 (11.88%)	0.418 (–7.31%)	3.58×10^{-4} (37.84%)

rewiring, which does not require to add signatures, and XOR additions (perform between 13-bit vectors), which can integrate the signatures mentioned above.

The formula $100 \cdot (1 - (\frac{1}{2})^{\#sign})\%$, where $\#sign.$ stands as the number of signatures, is used to compute the fault coverage percentage of the presented schemes. Moreover, the presented regular parity has a high fault coverage percentage of close to $100 \cdot (1-(\frac{1}{2})^{10^8})\%$, the presented interleaved parity and CRC-2 have a fault coverage percentage of close to $100 \cdot (1-(\frac{1}{2})^{2 \cdot 10^8})\%$, and the presented CRC-8 has a fault coverage percentage of close to $100 \cdot (1 - (\frac{1}{2})^{8 \cdot 10^8})\%$. Additionally, only the signatures required for one block out of the four in our presented method illustrated in Figure 1 would be considered for local faults. For regular parity, the fault coverage is close to $100 \cdot (1 - (\frac{1}{2})^{3.6 \cdot 10^6})\%$, $100 \cdot (1 - (\frac{1}{2})^{7.2 \cdot 10^6})\%$ for interleaved parity and CRC-2, and $100 \cdot (1 - (\frac{1}{2})^{2.9 \cdot 10^7})\%$ for CRC-8, if the errors are restricted to the $GF(2^{13})$ *Inverse* block.

In Tables 3 and 4, the overheads of our fault detection schemes are shown in terms of area (occupied slices), delay, power (with a 50 MHz frequency), throughput, and performance for the *Horner* and the $GF(2^{13})$ *Inverse* block, where $neg.\ over.$ stands for negligible overhead. The presented constructions are not fully pipelined, and they are implemented on Xilinx FPGA family Kintex-7 device xc7k70tfbv676-1 using the Vivado tool and Verilog as the hardware design language. However, we note that because our schemes are platform-oblivious, the outcome is not necessarily

influenced by the intended platform. To get the area, the Vivado's place utilization report reads CLBs, which are the main resources for creating general purpose combinational and sequential circuits. To compute the delay, we utilize Vivado's Timing Constraints Wizard, specifying a main clock time constraint of 20 ns, which corresponds to a frequency of 50 MHz. The total on-chip power, which is the power utilized internally within the FPGA and is calculated by combining device static power and design power, is also reported. Throughput is obtained by dividing the total number of output bits over the delay, efficiency is obtained by dividing throughput over area, and performance is obtained by dividing throughput over slices.

Let us go over an example to show how our fault detection schemes are incorporated to the Key Generator. First, the top module calls a functional unit, e.g., *Horner* block, $GF(2^{13})$ *Inverse* block, $GF(2^{13})$ *Mult(1)* block, $GF(2^{13})$ *Mult(2)* block, and $GF(2^{13})$ *Gen.* block. For instance, if the *Horner* block is called, this functional unit calls many multiplications and additions in $GF(2^m)$. When each finite-field multiplication is called, our error detection scheme is called as well to compare the predicted output of that specific finite-field operation and the actual output with XOR gates, producing an error flag. Last, when the *Horner* unit has made all the calls to the different finite-field multiplications and additions, it compares all the error flags with OR gates to check that all the finite-field operations have been performed free of faults. Additionally, we have implemented the *Inverse* block with different finite-field sizes, i.e., $GF(2^{12})$, which is the other finite-field option for the McEliece cryptosystem NIST submission and is considered a Category 3 parameter set in terms of expected strength (as opposed to $GF(2^{13})$, which is considered Category 5), and $GF(2^{11})$, to show the feasibility of our schemes for other fields. For the *Inverse* block using the finite field $GF(2^{12})$, the area overheads are 37.92% (451 slices), 42.51% (466 slices), and 46.79% (480 slices) when using normal parity, interleaved parity, and CRC-2, respectively; the delay overheads are 6.14% (14.796 ns), negligible (13.654 ns), and negligible (13.704 ns) when using normal parity, interleaved parity, and CRC-2, respectively; and the power overheads and 7.06% (0.091 mW), 5.88% (0.090 mW), and 5.88% (0.090 mW) when using normal parity, interleaved parity, and CRC-2, respectively. For the *Inverse* block using the finite field $GF(2^{11})$, to show the feasibility, the area overheads for the first two cases are 33.50% (267 slices), 19.50% (239 slices), the delay overheads are 6.30% (4.519 ns), 8.89% (4.629 ns), and the power overheads are negligible, i.e., 0.085 mW.

To obtain the area, the utilization report from Vivado reads the occupied slices, which are essential for the implementation of general purpose combinational and sequential circuits. To compute the delay, we utilize Vivado's Timing Constraints Wizard, specifying a main clock time constraint of 20 ns, or a 50 MHz frequency. Total on-chip power, which is the power utilized internally through the FPGA is reported as well and is calculated by combining the design power and the device static power. Last, the number of bits from the output is divided by the delay to get the throughput. As demonstrated in Tables 3 and 4, when stronger schemes with better error coverage are added to the original designs, they result in increased area and power overheads. The overhead difference in terms of delay is small and changes based on the gates employed in each design. Furthermore, it is certain that the larger the overall design is, the lower the overhead is. Since the $GF(2^{13})$ *Inverse* block does fewer operations than the *Horner* block, the overall overheads are higher. Both interleaved parity and CRC-2 are quite comparable, since they have the same amount of error flags, while CRC-8 is the most costly fault detection architecture. This is to be expected, given CRC executes more operations, resulting in larger error coverage, as seen by our studies. The overall area overheads of the strategies proposed in this article are less than 49%, which corresponds to the overhead obtained by the $GF(2^{13})$ *Inverse* block when it uses CRC-8, and more than 5%, which corresponds to the overhead obtained by the *Horner* block when it uses regular parity. Low delay overheads are observed for the presented fault detection schemes. As shown in Table 3, the worst-case scenarios in terms of delay overhead are less than 6% and less than 8%

Table 5. Worst-case Overhead Comparison of the Presented Schemes with Other Fault Detection Works

Work	Fault Detection Scheme	Worst-Case Overhead %				
		Area	Delay	Power	Throughput	Efficiency
[18]	CRC-10	25.51	Not given	Not given	Not given	Not given
[19]	CRC-3	20.81	18.71	10.29	15.77	28.99
[20]	Recom. shifted operands	40.74	21.23	22.41	Not given	Not given
[23]	Recom. customizable swapped entries	5.1	Not given	Not given	13.5	18.0
[24]	CRC-5	18.33	11.25	$\simeq 0$	10.10	Not given
[30]	Multi-parity	8.54	4.68	3.90	2.91	Not given
This work	CRC-8 (Goppa Horner)	20.03	5.43	9.72	Neg. over.	18.83
This work	CRC-8 (Inversion)	48.91	7.99	11.88	7.31	37.84

for the *Horner* and $GF(2^{13})$ *Inverse* block when using CRC-8, respectively. Furthermore, the power overheads added to the original architectures are less than 12%, which is obtained by the $GF(2^{13})$ *Inverse* block using CRC-8. Last, Table 4 shows how the throughput decreases when fault detection is added to the original constructions, obtaining a worst-case throughput overhead scenario of less than 8%. The fault detection schemes used are customizable based on the level of security required and the amount of overheads to be accepted. The scheme sizes can be increased for situations where performance is crucial, while smaller schemes are preferable for deeply-embedded systems.

To the best of the authors' knowledge, there has been no past studies on this sort of fault detection methods for the McEliece's Key Generator has been done. Let us look at some case studies for a qualitative assessment to verify that the overheads generated are reasonable. In Reference [29], fault detection techniques using parity prediction for multiplication in $GF(2^m)$ with normal basis are presented in Reference [34], getting approximately 58% of combined area and delay overhead (worst-case scenario). One of the drawbacks of regular parity prediction is that intelligent fault injection can get around this predictable countermeasure by injecting an even number of faults. Therefore, we present interleaved parity as well as CRC-2 and CRC-8 to resolve this issue. Concurrent error detection constructions to perform the Extended Euclidean-based division over $GF(2^m)$ are provided in Reference [35]. The schemes utilized are based on parity prediction and they have a combined worst-case area and delay overhead of 25.18%. Moreover, Table 5 shows a comparison in terms of worst-case area, delay, and power overheads of the presented schemes with other works on fault detection. These and related earlier research on traditional cryptography demonstrate that the presented fault detection constructions acquire comparable overheads to existing works on error detection, obtaining a tolerable overhead.

5 CONCLUSION

In this work, fault detection schemes are used in the different blocks of the Key Generator and other units of code-based cryptosystems. Key generation has the largest area complexity and, as a result, it is the most involved hardware implementation inside McEliece, using finite-field addition, multiplication, squaring, and inversion operations. McEliece has been advanced to the current and final round in the NIST standardization process as of July 2020. Our work has a special focus on the *H generator,* which provides the control matrix H required to get the McEliece cryptosystem public key. We have derived closed formulations for regular parity, interleaved parity, CRC-2, and CRC-8 for the finite-field blocks, and we have implemented these signatures on FPGA to assess the overheads and performance deterioration of the presented schemes and demonstrate their applicability for constrained embedded systems. The overall area and delay overheads of the strategies proposed in this article are less than 49%, which corresponds to the overhead obtained by the $GF(2^{13})$ *Inverse* block when it uses CRC-8, and more than 5%, which corresponds to the overhead

obtained by the *Horner* block when it uses regular parity. The provided fault detection constructions produce high error coverage at a reasonable overhead, as the results show.

REFERENCES

[1] D. Moody. 2016. Post-quantum cryptography: NIST's plan for the future. Retrieved from https://pqcrypto2016.jp/data/pqc2016_nist_announcement.pdf.

[2] A. Shoufan, T. Wink, H. G. Molter, S. A. Huss, and E. Kohnert. 2010. A novel cryptoprocessor architecture for the McEliece public-key cryptosystem. *IEEE Trans. Comput.* 59, 11 (2010), 1533–1546.

[3] R. Agrawal, L. Bu, and M. A. Kinsy. 2020. Quantum-proof lightweight McEliece cryptosystem co-processor design. In *Proceedings of the IEEE 38th International Conference on Computer Design (ICCD'20)*. 73–79.

[4] M. Lopez-Garcia and E. Cant-Navarro. 2020. Hardware-software implementation of a McEliece cryptosystem for post-quantum cryptography. In *Proceedings of the Future Information and Communication Conference*. 814–825.

[5] L. Mariot, S. Picek, and R. Yorgova. 2021. On McEliece type cryptosystems using self-dual codes with large minimum weight. Cryptology ePrint Archive. Retrieved from https://eprint.iacr.org/2021/837.

[6] J. Roth, E. Karatsiolis, and J. Kramer. 2020. Classic McEliece implementation with low memory footprint. In *Proceedings of the International Conference on Smart Card Research and Advanced Applications*. 34–49.

[7] Z. Li, C. Xing, and S. L. Yeo. 2019. Reducing the key size of Mceliece cryptosystem from automorphism-induced Goppa codes via permutations. In *Proceedings of the IACR International Workshop on Public Key Cryptography*. 599–617.

[8] M. S. Chen and T. Chou. 2021. Classic McEliece on the ARM Cortex-M4. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems* 2021, 3, 125–148.

[9] F. Strenzke, E. Tews, H. G. Molter, R. Overbeck, and A. Shoufan. 2008. Side channels in the McEliece PKC. In *Proceedings of the International Workshop on Post-Quantum Cryptography*. 216–229.

[10] P. L. Cayrel and P. Dusart. 2010. McEliece/Niederreiter PKC: Sensitivity to fault injection. In *Proceedings of the International Conference on Future Information Technology*. 1–6.

[11] F. Strenzke. 2013. Efficiency and implementation security of code-based cryptosystems. Ph.D. Thesis.

[12] P. L. Cayrel, B. Colombier, V. F. Drăgoi, A. Menu, and L. Bossuet. 2021. Message-recovery laser fault injection attack on the classic McEliece cryptosystem. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 438–467.

[13] T. Itoh and S. Tsujii. 1988. A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Info. Comput.* 78, 3 (1988), 171–177.

[14] J. Guajardo and C. Paar. 2002. Itoh-Tsujii inversion in standard basis and its application in cryptography and codes. *Designs, Codes Cryptogr.* 25 (2002), 207–216.

[15] F. Rodriguez-Henriquez, N. A. Saqib, and N. Cruz-Cortes. 2005. A fast implementation of multiplicative inversion over $GF(2^m)$. In *Proceedings of the International Symposium on Information Technology*. 574–579.

[16] B. Liu and R. Sandhu. 2015. Fingerprint-based detection and diagnosis of malicious programs in hardware. *IEEE Trans. Reliabil.* 64, 3 (2015), 1068–1077.

[17] M. Mozaffari-Kermani, R. Azarderakhsh, and A. Aghaie. 2015. Reliable and error detection architectures of Pomaranch for false-alarm-sensitive cryptographic applications. *IEEE Trans. Very Large Scale Integr. Syst.* 23, 2804–2812.

[18] A. Cintas-Canto, M. Mozaffari-Kermani, and R. Azarderakhsh. 2021. CRC-based error detection constructions for FLT and ITA finite field inversions over GF(2^m). *IEEE Trans. Very Large Scale Integr. Syst.* 29, 5 (2021), 1033–1037.

[19] J. Kaur, M. Mozaffari Kermani, and R. Azarderakhsh. 2021. Hardware constructions for lightweight cryptographic block cipher QARMA with error detection mechanisms. *IEEE Trans. Emerg. Top. Comput.*, accepted.

[20] A. Sarker, M. Mozaffari Kermani, and R. Azarderakhsh. 2021. Fault detection architectures for inverted binary Ring-LWE construction benchmarked on FPGA. *IEEE Trans. Circ. Syst. II* 68, 4 (2021), 1403–1407.

[21] A. Sarker, M. Mozaffari Kermani, and R. Azarderakhsh. 2021. Error detection architectures for ring polynomial multiplication and modular reduction of Ring-LWE in $Z = \frac{pZ[x]}{x^n+1}$ benchmarked on ASIC. *IEEE Trans. Reliabil.* 70, 1 (2021), 362–370.

[22] M. Mozaffari Kermani and R. Azarderakhsh. 2019. Reliable architecture-oblivious error detection schemes for secure cryptographic GCM structures. *IEEE Trans. Reliabil.* 68, 4 (2019), 1347–1355.

[23] M. Mozaffari Kermani, R. Azarderakhsh, A. Sarker, and A. Jalali. 2018. Efficient and reliable error detection architectures of Hash-Counter-Hash tweakable enciphering schemes. *ACM Trans. Embed. Comput. Syst.* 17, 2 (2018), 54:1–54:19.

[24] A. Cintas-Canto, M. Mozaffari-Kermani, and R. Azarderakhsh. 2021. Reliable CRC-based error detection constructions for finite field multipliers with applications in cryptography. *IEEE Trans. Very Large Scale Integr. Syst.* 29, 1 (2021), 232–236.

[25] M. Mozaffari Kermani and A. Reyhani-Masoleh. 2009. Fault detection structures of the S-boxes and the inverse s-boxes for the advanced encryption standard. *J. Electr. Testing: Theory Appl.* 25, 4 (2009), 225–245.

[26] M. Mozaffari Kermani and A. Reyhani-Masoleh. 2009. A low-cost S-box for the advanced encryption standard using normal basis. In *Proceedings of the IEEE International Conference Electro/Information Technology (EIT'09)*. 52–55.
[27] S. Subramanian, M. Mozaffari Kermani, R. Azarderakhsh, and M. Nojoumian. 2017. Reliable hardware architectures for cryptographic block ciphers LED and HIGHT. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 36, 10 (2017), 1750–1758.
[28] 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.-Aided Design Integr. Circ. Syst.* 33, 7 (2014), 1105–1109.
[29] M. Mozaffari Kermani and A. Reyhani-Masoleh. 2011. A high-performance fault diagnosis approach for the AES SubBytes utilizing mixed bases. In *Proceedings of the IEEE Workshop Fault Diagnosis and Tolerance in Cryptography (FDTC'11)*, pp. 80–87.
[30] A. Cintas-Canto, M. Mozaffari-Kermani, and R. Azarderakhsh. 2021. Reliable architectures for composite-field-oriented constructions of McEliece post-quantum cryptography on FPGA. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 40, 5 (2021), 999–1003.
[31] D. J. Bernstein, T. Chuo, T. Lange, I. Von Maurich, R. Misoczki, R. Niederhagen, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, and W. Wang. 2017. Classic McEliece: Conservative code-based cryptography. Retrieved from https://classic.mceliece.org/nist/mceliece-20171129.pdf.
[32] N. Patterson. 1975. Algebraic decoding of Goppa codes. *IEEE Trans. Info. Theory* 21, 2 (1975), 203–207.
[33] A. Reyhani-Masoleh and M. A. Hasan. 2002. Error detection in polynomial basis multipliers over binary extension fields. In *Proceedings of the Workshop Cryptographic Hardware and Embedded Systems* (*CHES'02*). 515–528.
[34] C. Y. Lee, P. K. Meher, and J. C. Patra. 2009. Concurrent error detection in bit-serial normal basis multiplication over $GF(2^m)$ using multiple parity prediction schemes. *IEEE Trans. Very Large Scale Integr. Syst.* 18, 8 (2009), 1234–1238.
[35] M. Mozaffari-Kermani, R. Azarderakhsh, C. Y. Lee, and S. Bayat-Sarmadi. 2013. Reliable concurrent error detection architectures for Extended Euclidean-based division over $GF(2^m)$. *IEEE Trans. Very Large Scale Integr. Syst.* 22, 5 (2013), 995–1003.

Received 29 November 2021; revised 10 May 2022; accepted 1 June 2022