Error Detection Constructions for ITA Finite Field Inversions Over GF(2^m) on FPGA Using CRC and Hamming Codes

Alvaro Cintas-Canto, *Member, IEEE*, Mehran Mozaffari Kermani, *Senior Member, IEEE*, and Reza Azarderakhsh, *Member, IEEE*

***Abstract*—Finite field arithmetic operations over GF(2^m) are widely used in critical applications, such as cryptography, coding theory, error-correcting codes, and digital signal processing. Finite field inversions are the most time-consuming operations among other widely-used ones and require a large footprint as well as power/energy to be performed. To reduce such complexity, the Itoh–Tsujii algorithm (ITA) has received prominent attention in the literature; however, implementations using ITA are still vulnerable to natural very-large-scale integration defects. To overcome the challenge of detecting naturally-induced faults in such constructions, for the first time, we propose error detection schemes based on Hamming codes for architectures performing finite field inversions using the ITA algorithm over GF(2^m) with polynomial basis. Additionally, CRC-oriented error detection schemes for inversions in GF(2^m) with normal basis are also studied and new approaches to protect them are presented. In this article, general formulations are provided along with different case studies to show the feasibility of our schemes with any finite field size. Moreover, field-programmable gate array (FPGA) implementations are performed on two Xilinx FPGA families, i.e., Kintex UltraScale+ and Xilinx Virtex-7 UltraScale+, to verify that the overheads added by the error detection architectures to provide reliability are suitable for deeply-constrained embedded systems.**

***Index Terms*—Fault detection, field-programmable gate array (FPGA), finite field inversion, hamming codes.**

Acronyms and Abbreviations

CRC	Cyclic redundancy check.
FLT	Fermat's Little Theorem.
FPGA	Field-programmable gate array.
GNB	Gaussian normal basis.
ITA	Itoh–Tsujii algorithm.
VLSI	Very-large-scale integration.

Manuscript received 30 March 2022; revised 4 June 2022 and 16 August 2022; accepted 11 October 2022. Date of publication 1 November 2022; date of current version 5 June 2023. This work was supported in part by Marymount University through the START under Grant 2450100 and in part by the U.S. National Science Foundation (NSF) through the Award SaTC-1801488. Associate Editor: Tadashi Dohi.

Alvaro Cintas-Canto is with the School of Technology and Innovation, Marymount University, Virginia, VA 22207 USA (e-mail: acintas@marymount.edu).

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

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

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TR.2022.3216014.

Digital Object Identifier 10.1109/TR.2022.3216014

I. Introduction

ARITHMETIC operations over the Galois field GF(2^m) are of special interest for many applications, such as cryptography, coding theory, error-correcting codes, and digital signal processing. The Galois field contains a finite number of elements that satisfy certain rules when performing addition, subtraction, multiplication, inversion, and division. The order of a Galois field is a prime or a power of a prime. The characteristic 2 of such fields is widely used in very-large-scale integration (VLSI) implementations due to the natural representation of the GF(2^m) symbols, which are m-bit vectors. Among the different finite field arithmetic operations, finite field inversion is the most time-consuming due to the large footprint required, needing thousands of gates for large field sizes. Moreover, battery-powered deeply-embedded systems such as implantable medical devices, smart fabrics, and Internet of nano-Things require low-energy realizations of these building blocks.

To reduce the complexity of finite field inversions, many approaches have been studied, see, for example, [1], [2], [3], [4], [5]. Most of these approaches decrease the complexity of finite field inversions by the use of multiplications, squarings, and additions. The Itoh–Tsujii algorithm (ITA) has been studied and implemented in different systems since ITA drastically reduces the number of multiplications needed to perform inversions over GF(2^m). In the early stages, ITA was meant for finite fields with normal basis; however, its versatility and efficacy on finite fields with polynomial basis have been proven [6], [7]. Recent works have explored ITA even further to reduce its complexity [8], [9]. Nevertheless, although ITA reduces the number of multiplications, it still requires many gates to implement the inverse of an element A where $A \epsilon$ GF(2^m). Implementations using such large designs are sensitive and can be compromised by natural defects. Producing a resilient architecture is a difficult challenge, not only because the error coverage has to be high but also because the overhead induced by the error detection blocks has to be suitable for deeply-constrained embedded systems, where one fault could lead to erroneous results.

Providing security is crucial for many applications and there has been extensive research on creating reliable architectures [10], [11], [12], [13]. A widely used method to provide reliability

is by adding error detection schemes into the original architectures [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26]. Moreover, some studies focus on error detection in architectures using GF(2^m) arithmetic operations, mostly in finite field multipliers. In [27], error detection based on 1-b parity prediction for finite field multipliers using polynomial basis over GF(2^m) is performed. One of the drawbacks of singular parity prediction is that it can only provide an error detection of up to 50%, i.e., if the number of faults is even, the system would not detect the faults. Other works such as [28], [29], [30], [31] overcame such limitations by proposing error detection based on multi-bit parity prediction at the expense of larger overheads than 1-b parity prediction schemes. These highly predictable countermeasures can be circumvented by intelligent fault injection; therefore, works [32], [33], [34] explore cyclic redundancy checks (CRCs) as error detection schemes in architectures with GF(2^m) arithmetic. In this article, we not only complete work [34] by deriving and implementing CRC-oriented error detection schemes for inversions over GF(2^m) with normal basis, but we also propose for the first time error detection constructions based on Hamming codes inversions over GF(2^m) using the ITA algorithm. Hamming codes are a very well-known type of error-correcting codes and achieve the highest possible rate for codes with their block length and minimum distance of three. Our contributions to this work may be summed up as follows:

1) We propose fault detection architectures based on CRC-3 and Hamming codes for finite field inversions over GF(2^m) using ITA with normal and polynomial basis, respectively. Depending on the system security requirements, larger CRCs can be utilized by following similar derivations of the ones presented in this article.
2) Formulations for the different finite fields using ITA with normal and polynomial basis are derived and validated by performing software implementations. In this work, we also provide specific examples with different m values; however, these formulations and error detection schemes can be applied to any field size.
3) The overheads of the presented error detection schemes are analyzed by embedding such schemes to the original ITA implementation. This is done by utilizing Xilinx field-programmable gate array (FPGA) family Virtex-7 for device xc7vx1140tflg1930-i and FPGA family Kintex UltraScale+ for device xcku5p-sfvb784-1LV-i.

The rest of this article is organized as follows: Section II discusses the differences between normal basis, especially Gaussian normal basis (GNB), and polynomial basis. Additionally, it states the equations used to perform finite field addition, squaring, and multiplication with both normal and polynomial basis. Section III is divided into four subsections. First, the fault model in this work is discussed; next, an ITA overview and how it works with finite field inversions are presented; finally, we derive different fault detection based on CRC-3 and Hamming codes for finite field inversions over GF(2^m) using ITA with normal and polynomial basis, respectively. We apply such fault detection schemes into the original finite field inversion construction in Section IV to benchmark the different overheads. Finally, Section V concludes this article.

II. Preliminaries

This section introduces finite fields over GF(2^m) with normal basis and finite fields over GF(2^m) with polynomial basis. The multiplication of elements with polynomial basis is relatively simple compared to normal basis. However, performing squaring with normal basis is much easier than with polynomial basis since it can be done by simply applying cyclic right shifts, which has no cost in hardware. Therefore, both polynomial basis and normal basis are used in different applications depending on the system requirements.

A finite field element A over GF(2^m) with normal basis $\{\alpha, \alpha^2, \alpha^{2^2}, \ldots, \alpha^{2^{m-1}}\}$, where each α is a normal element of GF(2^m), can be represented as

$$A = \sum_{i=0}^{m-1} a_i \alpha^{2^i} \; a_i \, \epsilon \, \{0, \, 1\}. \tag{1}$$

GNB is a type of normal basis in which $m > 1$ and not divisible by 8. According to a more thorough definition in [35], m and t are positive integers such that $p = mt + 1$ is a prime number. A Gauss period of type (m, t) over GF(2) is denoted as follows:

$$\alpha = \sum_{i=0}^{t-1} \delta^{\tau^i}$$

where δ is the primitive $(mt + 1)$th root of unity in GF$(mt + 1)$. To calculate τ, which is the primitive tth root of unity, the following property is applied: $\tau^t = 1 \bmod p$. For example, the GNB with type-4 over GF(2^7) has $\tau = 12$ since $12^4 = 1 \bmod 29$. Moreover, α is calculated as follows: $\alpha = \sum_{i=0}^{t-1} \delta^{12^i} = \delta + \delta^{12} + \delta^{17} + \delta^{28}$.

To add finite field elements A and B with normal basis, we add the coefficients of each element using XOR gates, i.e., $\sum_{i=0}^{m-1}(a_i + b_i)\alpha^{2^i}$. The multiplication of the finite field elements A and B with normal basis that generates an output C can be represented as

$$C = \left(\left(\left(a_{m-1} \alpha B^{2-(m-1)} \right)^2 + a_{m-2} \alpha B^{2^{-(m-2)}} \right)^2 + \cdots \right)^2 + a_0 \alpha B. \tag{2}$$

Readers interested in the mathematics behind this formulation can refer to [28]. Finally, as previously noted, one of the primary advantages of using normal basis is that squaring requires no extra hardware cost and is accomplished by simply performing cyclic right shifts. On normal basis, squaring an element A over GF(2^m) may be represented as

$$A^{2^i} = \sum_{j=0}^{m-1} a_{<j-i>} \alpha^{2^i}. \tag{3}$$

On the other hand, a finite field element A over GF(2^m) with polynomial basis $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ can be represented as

$$A = \sum_{i=0}^{m-1} a_i \alpha^i, \; a_i \, \epsilon \, \{0, \, 1\} \tag{4}$$

where a_i's are the coordinates of the input element A.

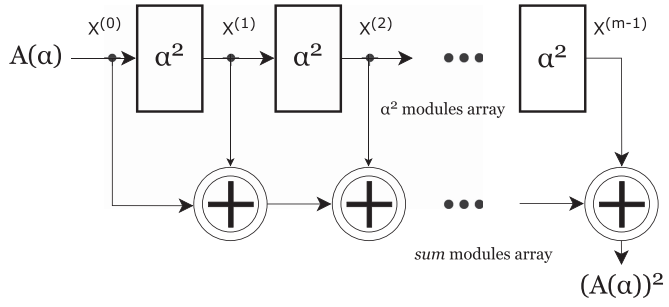

Fig. 1. Finite field squaring over GF(2^m) using polynomial basis.

To add finite field elements A and B with polynomial basis, as with normal basis, we also add the coefficients of each element using XOR gates, i.e., $\sum_{i=0}^{m-1}(a_i + b_i)\alpha^i$. Finite field multiplications with polynomial basis can be represented as

$$\begin{aligned} C &= A \cdot B \bmod f(\alpha) \\ &= \sum_{i=0}^{m-1} b_i \cdot \left(\left(A\alpha^i\right) \bmod f(\alpha)\right) \\ &= \sum_{i=0}^{m-1} b_i \cdot X^{(i)} \end{aligned}$$

where the set of b_i's is the B coefficients, $f(\alpha)$ is the irreducible polynomial, $X^{(i)} = \alpha \cdot X^{(i-1)} \bmod f(\alpha)$, and $X^{(0)} = A$. To perform such multiplication, we use the architecture from the work by Reyhani-Masoleh and Hasan [36]. In [36], three different modules are used to perform multiplications with polynomial basis: *Sum*, *pass-thru*, and α modules. The *sum* module adds two elements in GF(2^m) and it is used to perform finite field addition; the *pass-thru* module multiplies a GF(2^m) element by a GF(2) element; and the α module multiplies an element of GF(2^m) by α such as

$$A(\alpha) \cdot \alpha = a_{m-1} \cdot \alpha^m + a_{m-2} \cdot \alpha^{m-1} + \cdots + a_0 \cdot \alpha \quad (5)$$

where $\alpha^m \equiv f_{m-1} \cdot \alpha^{m-1} + f_{m-2} \cdot \alpha^{m-2} + \cdots + f_0 \bmod f(\alpha)$, reducing the result modulo $f(\alpha)$. Finally, to perform finite field squaring with polynomial basis, we use the *sum* module previously mentioned and the α^2 module, which multiplies an element of GF(2^m) by α^2 such as

$$A(\alpha) \cdot \alpha^2 = a_{m-1} \cdot \alpha^{m+1} + a_{m-2} \cdot \alpha^m + \cdots + a_0 \cdot \alpha^2 \quad (6)$$

where $\alpha^{m+1} \equiv f_{m-1} \cdot \alpha^m + f_{m-2} \cdot \alpha^{m-1} + \cdots + f_0 \cdot \alpha \bmod f(\alpha)$ and $\alpha^m \equiv f_{m-1} \cdot \alpha^{m-1} + f_{m-2} \cdot \alpha^{m-2} + \cdots + f_0 \bmod f(\alpha)$. In Fig. 1, the entire finite field squaring over GF(2^m) using polynomial basis is shown.

III. Proposed Fault Detection Architectures

This section is divided into four different subsections. First, the fault model used through our work for the finite field inversions using ITA is described; we then describe ITA and the benefits of using it; next, both similarities and differences between CRC and Hamming codes are analyzed; fault detection architectures based on CRC for finite field inversions over GF(2^m) using ITA with normal basis are derived after; and finally, fault detection architectures based on Hamming codes for finite field inversions over GF(2^m) using ITA with polynomial basis are proposed. In this article, we work with two different fields, i.e., GF(2^7) with normal basis using CRC and GF(2^{64}) with polynomial basis using Hamming codes. Since the NIST field GF(2^{163}) is used in [34], we wanted to explore different fields to show the flexibility and suitability of our schemes for different types of finite fields. Additionally, small fields are not only good for the sake of brevity of the article, but they also are used in many applications and even in postquantum cryptographic algorithms, e.g., McEliece cryptosystem and Luov cryptosystem.

A. Fault Model

Fault detection is especially important for remote systems where error-free key production is critical for overall system dependability. There are many fault models based on the sort of fault, e.g., the number of bits impacted, where the faults are located, and the fault duration. As a result, special countermeasures are required to safeguard the finite field inversions against such defects.

We present several error detection techniques based on CRC-3 and Hamming codes in this article. These techniques attempt to detect transient and permanent faults on the finite field inversion constructions caused by natural faults in VLSI architectures. These natural defects may be produced in more than one bit. Therefore, to obtain a high error coverage and detect multiple bit faults, we propose error detection schemes based on CRC-3 and Hamming codes, which unlike parity error detection schemes can detect even number of stuck-at faults (stuck-at 0 and stuck-at 1) in addition to single faults. In this article, it is assumed that the comparators are hardened, i.e., the comparators are fault free and not compromised, and that the inputs are not compromised prior to the execution of the finite field inversion unit. Additionally, we assume that both predicted and actual signature blocks are not compromised simultaneously. These assumptions can be made possible using triple-modular redundancy for specific parts of the architecture prone to faults (a compromise between overhead and error coverage). Moreover, physical hardening can be achieved using 1) insulating substrates instead of the usual semiconductor wafers, 2) shielding, and 3) cells with more transistors per cell than usual.

B. Itoh–Tsujii Algorithm

Inversions over GF(2^m) are one of the most complex and time-consuming arithmetic operations with finite fields. The inverse of a finite field element A can be represented as $A^{-1} \in \text{GF}(2^m)$ where $A \cdot A^{-1} = 1$. Due to the complexity of such operations, many different approaches have been studied over time. One of the most famous approaches is Fermat's Little Theorem or FLT. With FLT, finite field inversions can be performed using $m-2$ multiplications over GF(2^m) and $m-1$ squarings over GF(2^m) by following this formula:

$$A^{2^m-2} \equiv A^{-1} \bmod f(\alpha). \quad (7)$$

This was a good improvement in the complexity of finite field inversions; however, it could lead to low performance in deeply-embedded systems, especially when m is large, e.g., GF(2^{163}).

Algorithm 1: Multiplicative Inversion Addition-Chain ITA.

Input: $A \epsilon \text{GF}(2^m)$
Output: $A^{-1} \epsilon \text{GF}(2^m)$
1: $\zeta_0 = A(\alpha)$
2: **for** $i \leftarrow 1$ to t **do**
3: $\zeta_i = [\zeta_{i_1}]^{2^{c_{i_2}}} \cdot \zeta_{i_2} \pmod{f(\alpha)}$
4: **end for**
5: **return** $((\zeta_t)^2 \pmod{f(\alpha)})$

ITA was then introduced to reduce the amount of finite field multiplications to $\log_2(m-1) + H_2(m-1) - 1$, where $H_2(m-1)$ is the Hamming weight. This is done by assigning $2^0 + 2^1 + 2^2 + \cdots + 2^{m-2}$ to $1 + 2^n + 2^{2n} + \cdots + 2^{(k-2)n}$ and decomposing it as follows:

$$1 + 2^n + 2^{2n} + \cdots + 2^{(k-2)n} = \begin{cases} (1+2^n) \times (1+2^{2n} \\ +2^{4n} + \cdots + 2^{(k-3)n}), & \text{if } k-1 \equiv 0 \bmod 2 \\ 1 + 2^n \times (1+2^n) \\ \times (1 \times 2^{2n} + 2^{4n} + \cdots \\ +2^{(k-4)n}), & \text{if } k-1 \equiv 1 \bmod 2. \end{cases} \quad (8)$$

Other techniques employ addition chains effectively to accomplish ITA. In [37], the finite field inverse of an element A is expressed as $A^{-1} = [\zeta_{m-1}(A)]^2$, where $\zeta_k(A) = A^{2^k-1}$ and $k \epsilon \mathbb{N}$. To generate the addition chain $U = \{u_1, u_2, \ldots, u_t\}$, we first assign $u_0 = 0$ and $u_t = m-1$. Next, if u_i is even, $u_{t-1} = u_t/2$, and if u_t is odd, $u_{t-1} = u_t - 1$. In Algorithm 1, the computational steps to perform the Multiplicative Inversion Addition-Chain ITA of a finite field element A in GF(2^m) are provided. Addition chains can be generalized into many directions, and numerous ideas can be found in the literature [38], [39], [40].

In this work, we use both ITA approaches, i.e., ITA without addition chains and ITA utilizing additions chains, in the next two sections, respectively.

C. CRC Versus Hamming Codes

As mentioned previously, error detection techniques based on CRC and Hamming codes are derived in this work. Both CRC and Hamming codes are binary linear codes with error detection capabilities. The latter is also used for error correction; however, CRC can only be used for error detection. CRC is widely used to detect transmission errors in a frame or block of data while Hamming codes are used to detect and correct bit errors that can occur when computer data are moved or stored. Furthermore, another notable difference is that Hamming codes, depending on the Hamming code employed, only work on data of a specific size, but the CRC is a convolutional code that works on data of any size. In Table I, a summary of other works on fault detection and their limitations is shown.

Algorithm 2: Finite field multiplication over GF(2^m) with normal basis.

Input: $A \epsilon \text{GF}(2^m)$, $B \epsilon \text{GF}(2^m)$
Output: $C \epsilon \text{GF}(2^m)$
1: $C_0 \leftarrow 0$
2: **for** $j \leftarrow 1$ to m **do**
3: $C_j = C_{j-1}^2 + a_{m-j} \alpha B^{2^{-(m-j)}}$
4: **end for**
5: $C = C_m$
6: **return** C

The goal of this article is not to choose one scheme over the other since both have their own advantages but to propose for the first time error detection constructions based on Hamming codes inversions over GF(2^m) using the ITA algorithm.

For the sake of comparison in terms of area (occupied slices), delay (ns), and power (mW) with the clock frequency of 50 MHz between CRC-3 and Hamming codes, we have added them to the original architectures of a finite field multiplier using elements in GF(2^4) using polynomial basis and implemented them on Xilinx FPGA family Artix-7 device xc7a12tcpg238-3, Kintex-7 device xc7k70tfbg484-3, and Spartan-7 device xc7s6cpga196-2 using the Vivado tool. As we can see in Table II, the area obtained by Hamming codes is lower than that obtained with CRC-3, but the delay is higher. Therefore, one could choose one or the other based on the system requirements. In Section IV, more in-depth implementations are carried out along with an analysis on the error detection capabilities of each.

D. Fault Detection Based on CRC-3 of Finite Field Inversions Over GF(2^m) With Normal Basis Using ITA

In this article, CRC provides two sets of equations, which we can call predicted signatures and actual signatures (do not mistake this kind of signatures with digital signatures). Predicted signatures are mathematically calculated before the operation takes place. These signatures are then compared via XOR gates with the actual signatures, which is the actual output divided into smaller parts (the smaller they are, the larger the CRC is, e.g., CRC-3 divides the signatures into three parts). To implement CRC, a generator polynomial $g(\alpha)$ is needed. In this subsection, we work with the field GF(2^7) using CRC-3 and the original ITA, since addition chains do not reduce the number of multiplications for the case of GF(2^7). Different CRC schemes and field sizes can be easily adopted by following the derivations of this work.

Following (8), the field GF(2^7) has the following decomposition using ITA:

$$1 + 2 + 2^2 + \cdots + 2^5 = (1+2) \times \left(1 + 2^2 \times \left(1 + 2^2\right)\right).$$

Finite field squarings with normal basis are done by just cyclic right shifting while finite field multiplications with normal basis are more complex as seen in (2). We use Algorithm 2 to compute (2).

TABLE I
SUMMARY OF OTHER WORKS ON FAULT DETECTION AND THEIR CHARACTERISTICS/LIMITATIONS

Work	Fault Detection Scheme	Characteristics/Limitations
[21], [27], [28], [29], [30], [31], [36]	Singular parity	Singular parity only offers up to a 50% error coverage, i.e., if the number of faults is even, the system would not detect the faults.
[19], [22], [28], [29], [30], [31], [41]	Multi-parity	Both singular and multi-parity can be vulnerable to intelligent fault injections.
[10], [11], [16], [17], [20], [23], [24]	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.
[32], [33], [34]	CRC	CRC is an efficient scheme to protect the systems against intelligent fault injections and adds acceptable overheads

TABLE II
ILLUSTRATIVE EXAMPLE WITH FPGA IMPLEMENTATION RESULTS FOR FINITE FIELD MULTIPLIERS USING CRC-3 AND HAMMING CODES FOR ELEMENTS IN GF(2^4) WITH POLYNOMIAL BASIS

Architecture	Area (occupied slices)	Delay (ns)	Power (mW) @50 MHz	Xilinx FPGA Family & Device
CRC-3	8	1.777	0.061	Artix-7
Hamming Codes	7	2.258	0.061	xc7a12tcpg238-3
CRC-3	8	1.153	0.083	Kintex-7
Hamming Codes	7	1.300	0.083	xc7k70tfbg484-3
CRC-3	8	2.032	0.020	Spartan-7
Hamming Codes	7	2.067	0.020	xc7s6cpga196-2

TABLE III
F VALUES OF TYPE-4 GNB IN GF(2^7)

n	1	2	3	4	5	6	7	8	9	10
F(n)	0	1	5	2	1	6	5	3	3	2
n	11	12	13	14	15	16	17	18	19	20
F(n)	4	0	4	6	6	4	0	4	2	3
n	21	22	23	24	25	26	27	28	-	-
F(n)	3	5	6	1	2	5	1	0	-	-

The αB multiplier, which multiplies α with the finite field element B, performs the most complex step in Algorithm 2, and it is where the fault error detection schemes are incorporated. To calculate αB, $\alpha = \delta + \delta^{2^m} + \cdots + \delta^{2^{m(t-1)}}$ produces the normal basis and $B = b_{F(0)} + b_{F(1)}\delta + \cdots + b_{F(p-1)}\delta^{p-1}$, where $F(2^i 2^{mj} \bmod p) = i, 0 \le i \le m-1, 0 \le j \le t-1$. In Table III, the F values of type-4 GNB in GF(2^7) can be seen.

Furthermore, αB in GNB form is calculated as

$$\alpha B = B^{(1)} + B^{(2^m \bmod p)} + \cdots + B^{(2^{m(t-1)} \bmod p)} \quad (9)$$

where

$$B^{(i)} = \sum_{j=0}^{p-1} b_{F(j-i)}\delta^j.$$

The normal basis form of (9) is represented as

$$\alpha B = \sum_{i=0}^{m-1} \bar{b_i}\alpha^{2^i}$$

where $\bar{b_i} = \sum_{i=0}^{t-1} b_{F(2^i - 2^{mj})} + b_{F(0)}$. If t is even, $b_{F(0)}$ is omitted [28].

For the case study of GF(2^7), $B = (b_0, b_1, \ldots, b_6)$ and $\alpha = \delta + \delta^{12} + \delta^{17} + \delta^{28}$. The redundant basis of B is then represented as

$$\begin{aligned} B = {} & b_0\delta + b_1\delta^2 + b_5\delta^3 + b_2\delta^4 + b_1\delta^5 + b_6\delta^6 \\ & + b_5\delta^7 + b_3\delta^8 + b_3\delta^9 + b_2\delta^{10} + b_4\delta^{11} + b_0\delta^{12} \\ & + b_4\delta\gamma^{13} + b_6\delta^{14} + b_6\delta^{15} + b_4\delta^{16} + b_0\delta^{17} + b_4\delta^{18} \\ & + b_2\delta^{19} + b_3\delta^{20} + b_3\delta^{21} + b_5\delta^{22} + b_6\delta^{23} + b_1\delta^{24} \\ & + b_2\delta^{25} + b_5\delta^{26} + b_1\delta^{27} + b_0\delta^{28}. \end{aligned} \quad (10)$$

Next, (9) and (10) are used to obtain $\alpha B = (\delta + \delta^{12} + \delta^{17} + \delta^{28})B$, or $\alpha B = B^{(1)} + B^{(12)} + B^{(17)} + B^{(28)}$, where

$$\begin{aligned} B^{(1)} = {} & b_0 + b_0\delta^2 + b_1\delta^3 + b_5\delta^4 + b_2\delta^5 + b_1\delta^6 \\ & + b_6\delta^7 + b_5\delta^8 + b_3\delta^9 + b_3\delta^{10} + b_2\delta^{11} + b_4\delta^{12} \\ & + b_0\delta^{13} + b_4\delta^{14} + b_6\delta^{15} + b_6\delta^{16} + b_4\delta^{17} + b_0\delta^{18} \\ & + b_4\delta^{19} + b_2\delta^{20} + b_3\delta^{21} + b_3\delta^{22} + b_5\delta^{23} + b_6\delta^{24} \\ & + b_1\delta^{25} + b_2\delta^{26} + b_5\delta^{27} + b_1\delta^{28}. \end{aligned}$$

$$\begin{aligned} B^{(12)} = {} & b_0 + b_4\delta + b_2\delta^2 + b_3\delta^3 + b_3\delta^4 + b_5\delta^5 \\ & + b_6\delta^6 + b_1\delta^7 + b_2\delta^8 + b_5\delta^9 + b_1\delta^{10} + b_0\delta^{11} \\ & + b_0\delta^{13} + b_1\delta^{14} + b_5\delta^{15} + b_2\delta^{16} + b_1\delta^{17} + b_6\delta^{18} \\ & + b_5\delta^{19} + b_3\delta^{20} + b_3\delta^{21} + b_2\delta^{22} + b_4\delta^{23} + b_0\delta^{24} \\ & + b_4\delta^{25} + b_6\delta^{26} + b_6\delta^{27} + b_4\delta^{28}. \end{aligned}$$

$$\begin{aligned} B^{(17)} = {} & b_0 + b_4\delta + b_6\delta^2 + b_6\delta^3 + b_4\delta^4 + b_0\delta^5 \\ & + b_4\delta^6 + b_2\delta^7 + b_3\delta^8 + b_3\delta^9 + b_5\delta^{10} + b_6\delta^{11} \\ & + b_1\delta^{12} + b_2\delta^{13} + b_5\delta^{14} + b_1\delta^{15} + b_0\delta^{16} \\ & + b_0\delta^{18} + b_1\delta^{19} + b_5\delta^{20} + b_2\delta^{21} + b_1\delta^{22} + b_6\delta^{23} \\ & + b_5\delta^{24} + b_3\delta^{25} + b_3\delta^{26} + b_2\delta^{27} + b_4\delta^{28}. \end{aligned}$$

$$B^{(28)} = b_0 + b_1\delta^1 + b_5\delta^2 + b_2\delta^3 + b_1\delta^4 + b_6\delta^5$$

TABLE IV
$\alpha B^{2^{-I}}$ PREDICTED SIGNATURES USING CRC-3

Step	Predicted signatures
$\alpha B^{2^{-6}}$	$(b_3+b_6)+(b_1+b_2+b_3+b_5+b_6)\alpha+(b_1+b_3+b_4)\alpha^2$
$\alpha B^{2^{-5}}$	$(b_1+b_2+b_4+b_5)+(b_1+b_3+b_5+b_6)\alpha+\alpha^2$
$\alpha B^{2^{-4}}$	$(b_0+b_1+b_2+b_5+b_6)+(b_1+b_2+b_3+b_5+b_6)\alpha+(b_1+b_4+b_6)\alpha^2$
$\alpha B^{2^{-3}}$	$(b_1+b_3)+(b_0+b_1+b_3+b_6)\alpha+(b_3+b_5)\alpha^2$
$\alpha B^{2^{-2}}$	$(b_1+b_6)+(b_1+b_2+b_4+b_6)\alpha+(b_0+b_1+b_2+b_3+b_4+b_5)\alpha^2$
$\alpha B^{2^{-1}}$	$(b_1+b_2+b_5)+(b_0+b_6)\alpha+(b_1+b_4+b_6)\alpha^2$
$\alpha B^{2^{0}}$	$(b_3+b_4+b_6)+(b_0+b_3)\alpha+(b_3+b_4+b_6+b_2)\alpha^2$.

$$+ b_5\delta^6 + b_3\delta^7 + b_3\delta^8 + b_2\delta^9 + b_4\delta^{10} + b_0\delta^{11} + b_4\delta^{12} + b_6\delta^{13} + b_6\delta^{14} + b_4\delta^{15} + b_0\delta^{16} + b_4\delta^{17} + b_2\delta^{18} + b_3\delta^{19} + b_3\delta^{20} + b_5\delta^{21} + b_6\delta^{22} + b_1\delta^{23} + b_2\delta^{24} + b_5\delta^{25} + b_1\delta^{26} + b_0\delta^{27}.$$

Utilizing these formulations and (9), we can finally obtain

$$\alpha B = (b_4+b_4+b_1)\alpha + (b_0+b_2+b_6+b_5)\alpha^2 + (b_1+b_3+b_6+b_2)\alpha^{2^2} + (b_5+b_3+b_4+b_1)\alpha^{2^3} + (b_1+b_6+b_4+b_5)\alpha^{2^4} + (b_5+b_2+b_3+b_3)\alpha^{2^5} + (b_6+b_2+b_0+b_0)\alpha^{2^6}. \quad (11)$$

It is noted that (11) is utilized when $i = 7$ in Algorithm 1, which makes $\alpha B^{2^{-(7-7)}}$ or αB. We utilize the following equation to deduce the remainder of the formulations for other i's:

$$B^{2^{-i}} = \sum_{j=0}^{m-1} b_{<j-i>}\alpha^{2^i}. \quad (12)$$

For the purpose of brevity, the formulations and error detection techniques for the other i's in Algorithm 1 are not derived in this study, but they are provided in Table IV.

To provide error detection constructions based on CRC signatures for the αB module, we follow the next steps.

1) Selection of the generator polynomial $g(\alpha) \equiv \alpha^N + \alpha^{N-1} + \cdots + \alpha + 1$ for a specific CRC-N, where N denotes the number of error flags each module has.
2) Derivation of the generator polynomial, which is arranged as $\alpha^N = \alpha^{N-1} + \cdots + \alpha + 1 \bmod g(\alpha)$, into different equations by multiplying α to each side. Repeat until a substitution for α^{m-1} or g_{m-1} is obtained.
3) Substitution of different α's of
$$\alpha B = (b_{m-1} + b_{m-2} + \cdots + b_1 + b_0)(\alpha + \alpha^2 + \cdots + \alpha^{2^{m-2}} + \alpha^{2^{m-1}})$$
with the different equations calculated in Step 2 to obtain
$$\alpha B = (b_{m-1} + b_{m-2} + \cdots + b_1 + b_0)(g_0 + g_1 + \cdots + g_{m-2} + g_{m-1})$$

In this article, a generator polynomial $g(\alpha) = \alpha^3 + \alpha + 1$ for CRC-3 is chosen to generate fault detection techniques in the computation of αB. With that $g(\alpha)$, a set of equations is then developed

$$\alpha^4 \equiv \alpha^2 + \alpha \bmod g(\alpha)$$
$$\alpha^5 \equiv \alpha^3 + \alpha^2 \equiv \alpha^2 + \alpha + 1 \bmod g(\alpha)$$
$$\alpha^6 \equiv \alpha^3 + \alpha^2 + \alpha \equiv \alpha^2 + 1 \bmod g(\alpha). \quad (13)$$

To obtain the predicted signatures, we apply (13) into (11), producing

$$\text{Predicted} = (b_4+b_4+b_1) + (b_0+b_2+b_6+b_5)\alpha + (b_1+b_3+b_6+b_2)\alpha^2 + (b_5+b_3+b_4+b_1)(\alpha+1) + (b_1+b_6+b_4+b_5)(\alpha^2+\alpha) + (b_5+b_2+b_3+b_3)(\alpha^2+\alpha+1) + (b_6+b_2+b_0+b_0)(\alpha^2+1)$$

or

$$\text{Predicted} = (b_3+b_4+b_6) + (b_0+b_3)\alpha + (b_3+b_4+b_6+b_2)\alpha^2. \quad (14)$$

Finally, to obtain the actual signatures, we rename the coefficients of (11) and apply (13) to obtain

$$\text{Actual} = (\gamma_0) + (\gamma_1)\alpha + (\gamma_2)\alpha^2 + (\gamma_3)(\alpha+1) + (\gamma_4)(\alpha^2+\alpha) + (\gamma_5)(\alpha^2+\alpha+1) + (\gamma_6)(\alpha^2+1)$$

or

$$\text{Actual} = (\gamma_0+\gamma_3+\gamma_5+\gamma_6) + (\gamma_1+\gamma_3+\gamma_4+\gamma_5)\alpha + (\gamma_2+\gamma_4+\gamma_5+\gamma_6)\alpha^2. \quad (15)$$

We note that the actual signatures are the same for all i's. The fault detection technique based on CRC-3 for the multiplication of αB is depicted in Fig. 2. For CRC-3, three separated error flags, denoted as E_i, are produced by XORing the predicted signatures with the actual signatures. E_1 is the result of XORing the predicted signature $(b_3+b_4+b_6)$ with the actual signature $(\gamma_0+\gamma_3+\gamma_5+\gamma_6)$, obtained after α is multiplied with the field element B. In the same manner, (b_0+b_3) is XOR with $(\gamma_1+\gamma_3+\gamma_4+\gamma_5)$ obtaining E_2, and $(b_3+b_4+b_6+b_2)$ is XOR with $(\gamma_2+\gamma_4+\gamma_5+\gamma_6)$ obtaining E_3.

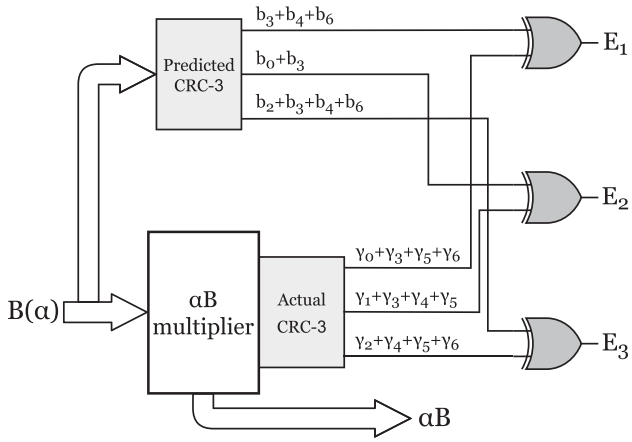

Fig. 2. αB multiplication with the proposed error detection scheme based on CRC.

TABLE V
STEPS NEEDED TO PERFORM THE INVERSE OF $A \epsilon$ GF(2^{64}) USING ITA WITH ADDITION CHAINS

Step	$\zeta_{V_i}(x)$	$\zeta_{V_j+Y_k}(x)$	Exponentiation
1	$\zeta_1(x)$	–	A
2	$\zeta_2(x)$	$\zeta_{1+1}(x)$	$(\zeta_1)^{2^1}\zeta_1 = A^{2^2-1}$
3	$\zeta_3(x)$	$\zeta_{2+1}(x)$	$(\zeta_2)^{2^1}\zeta_1 = A^{2^3-1}$
4	$\zeta_6(x)$	$\zeta_{3+3}(x)$	$(\zeta_3)^{2^3}\zeta_3 = A^{2^6-1}$
5	$\zeta_7(x)$	$\zeta_{6+1}(x)$	$(\zeta_6)^{2^1}\zeta_1 = A^{2^7-1}$
6	$\zeta_{14}(x)$	$\zeta_{7+7}(x)$	$(\zeta_7)^{2^7}\zeta_7 = A^{2^{14}-1}$
7	$\zeta_{15}(x)$	$\zeta_{14+1}(x)$	$(\zeta_{14})^{2^1}\zeta_1 = A^{2^{15}-1}$
8	$\zeta_{30}(x)$	$\zeta_{15+15}(x)$	$(\zeta_{15})^{2^{15}}\zeta_{15} = A^{2^{30}-1}$
9	$\zeta_{31}(x)$	$\zeta_{30+1}(x)$	$(\zeta_{30})^{2^1}\zeta_1 = A^{2^{31}-1}$
10	$\zeta_{62}(x)$	$\zeta_{31+31}(x)$	$(\zeta_{31})^{2^{31}}\zeta_{31} = A^{2^{62}-1}$
11	$\zeta_{63}(x)$	$\zeta_{62+1}(x)$	$(\zeta_{62})^{2^1}\zeta_1 = A^{2^{63}-1}$

E. Fault Detection Based on Hamming Codes of Finite Field Inversions Over GF(2^m) With Polynomial Basis Using ITA

Hamming codes are a type of error-correcting code and are often called perfect codes because they achieve the highest possible rate for codes with their block length and minimum distance of three. Hamming codes are classified into two categories depending on the structure of the encoder output, e.g., systematic encoding and nonsystematic encoding. In this work, we use systematic encoding, which separates the data and the parity bits, with a (7,4) Hamming block encoder. What this means is that it takes 4-b of data and produces a 7-b codeword, which are 4 data bits and 3 parity bits. To produce such codewords, we multiply each 4-b of data with a generator matrix G in the form of

$$\text{codeword} = \text{message} \times G \tag{16}$$

where

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \tag{17}$$

Hamming codes are incorporated to the α and α^2 modules needed to perform finite field multiplications and inversions, respectively. In this section, we use GF(2^{64}) as our case study and ITA using the addition chain $U = \{1, 2, 3, 6, 7, 14, 15, 30, 31, 62, 63\}$. In Table V, the different steps needed to perform ITA using addition chains for GF(2^{64}) are shown. As it can be seen on the left-most column, a total of 10 finite field multiplications and 63 finite field squarings are needed to perform the inverse of $A \epsilon$ GF(2^{64}) (only 62 squarings are shown in Table V; however, an extra squaring is needed since $A^{-1} = [\zeta_{m-1}(A)]^2$ using ITA with addition chains).

To derive the set of formulations needed to provide error detection in the α and α^2 modules, (5) and (6) are used. Since a (7,4) Hamming block encoder is utilized in this work, the polynomials from (5) and (6) are divided into 4-b blocks. For our case study, 64 is divisible by 4; however, if the field m is not divisible by 4, this can be easily fixed by appending the necessary number of "0" bits to the input message until it is divisible by 4.

1) α Module With Hamming Codes: For the α module with $m = 64$, $f(\alpha) = \alpha^{64} + \alpha^4 + \alpha^3 + \alpha + 1$, and using a (7,4) Hamming block, we use (5) and divide the input into 16 submessages m such as

$$\begin{aligned} m_{15} &= a_{62}\alpha^{63} + a_{61}\alpha^{62} + a_{60}\alpha^{61} + a_{59}\alpha^{60} \\ m_{14} &= a_{58}\alpha^{59} + a_{57}\alpha^{58} + a_{56}\alpha^{57} + a_{55}\alpha^{56} \\ m_{13} &= a_{54}\alpha^{55} + a_{53}\alpha^{54} + a_{52}\alpha^{53} + a_{51}\alpha^{52} \\ &\cdots \\ m_2 &= a_{10}\alpha^{11} + a_9\alpha^{10} + a_8\alpha^9 + a_7\alpha^8 \\ m_1 &= a_6\alpha^7 + a_5\alpha^6 + a_4\alpha^5 + (a_{63} + a_3)\alpha^4 \\ m_0 &= (a_{63} + a_2)\alpha^3 + a_1\alpha^2 + (a_{63} + a_0)\alpha + a_{63}. \end{aligned} \tag{18}$$

Next, each codeblock from (18) is multiplied with the last three columns of generator matrix G from (17), to obtain the following 48 (or 16×3) predicted signatures:

$$\begin{aligned} \text{predicted}_{47} &= a_{62}\alpha^{63} + a_{61}\alpha^{62} + a_{59}\alpha^{60} \\ \text{predicted}_{46} &= a_{62}\alpha^{63} + a_{61}\alpha^{62} + a_{60}\alpha^{61} \\ \text{predicted}_{45} &= a_{61}\alpha^{62} + a_{60}\alpha^{61} + a_{59}\alpha^{60} \\ \text{predicted}_{44} &= a_{58}\alpha^{59} + a_{57}\alpha^{58} + a_{55}\alpha^{56} \\ &\cdots \\ \text{predicted}_3 &= a_5\alpha^6 + a_4\alpha^5 + (a_{63} + a_3)\alpha^4 \\ \text{predicted}_2 &= (a_{63} + a_2)\alpha^3 + a_1\alpha^2 + a_{63} \\ \text{predicted}_1 &= (a_{63} + a_2)\alpha^3 + a_1\alpha^2 + (a_{63} + a_0)\alpha \\ \text{predicted}_0 &= a_1\alpha^2 + (a_{63} + a_0)\alpha + a_{63}. \end{aligned} \tag{19}$$

We note that the first bit of the message to be multiplied with G is a_{62}, then $(a_{63} + a_0)\alpha$, and so on, since a_{63} is the least significant bit. To obtain the actual signatures, we rename the coefficients of (18): a_{62} as γ_{63}, a_{61} as γ_{62},..., and a_{63} as γ_0

$$\begin{aligned} \text{actual}_{47} &= \gamma_{63}\alpha^{63} + \gamma_{62}\alpha^{62} + \gamma_{60}\alpha^{60} \\ \text{actual}_{46} &= \gamma_{63}\alpha^{63} + \gamma_{62}\alpha^{62} + \gamma_{61}\alpha^{61} \\ \text{actual}_{45} &= \gamma_{62}\alpha^{62} + \gamma_{61}\alpha^{61} + \gamma_{60}\alpha^{60} \\ \text{actual}_{44} &= \gamma_{59}\alpha^{59} + \gamma_{58}\alpha^{58} + \gamma_{56}\alpha^{56} \\ &\cdots \\ \text{actual}_3 &= \gamma_6\alpha^6 + \gamma_5\alpha^5 + \gamma_4\alpha^4 \\ \text{actual}_2 &= \gamma_3\alpha^3 + \gamma_2\alpha^2 + \gamma_0 \\ \text{actual}_1 &= \gamma_3\alpha^3 + \gamma_2\alpha^2 + \gamma_1\alpha \\ \text{actual}_0 &= \gamma_2\alpha^2 + \gamma_1\alpha + \gamma_0. \end{aligned} \tag{20}$$

Once the predicted and the actual signatures are implemented, they are XORed with each other to check for natural faults. An entire finite field multiplier needs $m - 1$ α modules, m *pass-through* modules, and $m - 1$ *sum* modules, while a finite field squaring uses $m - 1$ α^2 modules and $m - 1$ *sum* modules as shown in Fig. 1. In Fig. 3, we zoom in to show how the proposed error detection schemes based on Hamming codes are realized when they are applied to just one α module or one α^2

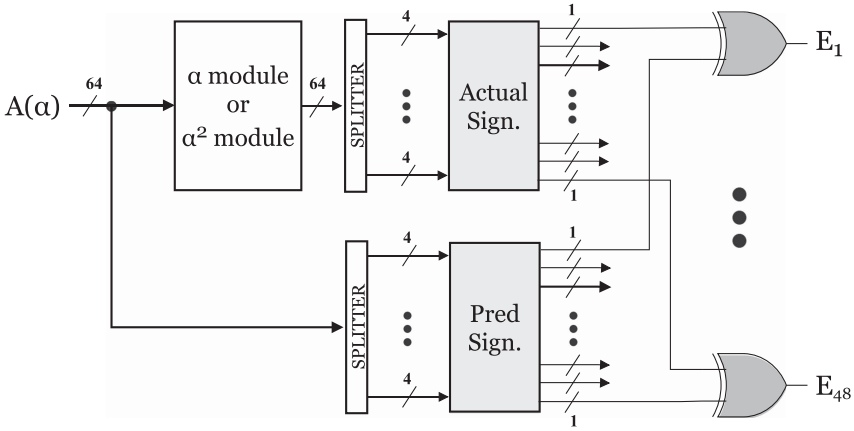

Fig. 3. Error detection schemes based on Hamming codes for the α and α^2 modules.

module (presented in the next section). If there would not be error detection, Fig. 3 would just have an input, the α or α^2 module, and the output. However, with Hamming codes, the finite field element A (or input) goes into two different blocks, as it can be seen in Fig. 3. In the top part, A goes through the α *module* or α^2 *module* block to perform (5) or (6), respectively, as it would do without any error detection unit. Then, the resulting 64-b output is split into 16 parts, each of them containing 4-b. The *Actual Sign.* block performs the calculations from (20) and each output bit of the *Actual Sign.* block contains each signature of (20). Meanwhile, in the bottom part, the input A is directly split into 16 parts. Following (19) for the case of multiplication [or (22) for the case of squaring], 48 predicted signatures (1-b each) are obtained in the *Predicted Sign*. block. Each of these predicted signatures is then XORed with the actual signatures to check if an error has been found, e.g., $predicted_0$ and $actual_0$ are the inputs of the first XOR gate, which will output $EF_1 = 1$ if $predicted_0$ and $actual_0$ are different, and therefore, this would mean that an error has been found.

2) α^2 Module With Hamming Codes: For the α^2 module with $m = 64$, $f(\alpha) = \alpha^{64} + \alpha^4 + \alpha^3 + \alpha + 1$, and using a (7,4) Hamming block, we use (6) and divide the input into 16 submessages m such as

$$\begin{aligned}
m_{15} &= a_{61}\alpha^{63} + a_{60}\alpha^{62} + a_{59}\alpha^{61} + a_{58}\alpha^{60}\\
m_{14} &= a_{57}\alpha^{59} + a_{56}\alpha^{58} + a_{55}\alpha^{57} + a_{54}\alpha^{56}\\
m_{13} &= a_{53}\alpha^{55} + a_{52}\alpha^{54} + a_{51}\alpha^{53} + a_{50}\alpha^{52}\\
&\cdots\\
m_2 &= a_9\alpha^{11} + a_8\alpha^{10} + a_7\alpha^9 + a_6\alpha^8\\
m_1 &= a_5\alpha^7 + a_4\alpha^6 + (a_{63} + a_3)\alpha^5 + (a_{63} + a_{62} + a_2)\alpha^4\\
m_0 &= (a_{62} + a_1)\alpha^3 + (a_{63} + a_0)\alpha^2 + (a_{63} + a_{62})\alpha + a_{62}.
\end{aligned} \tag{21}$$

Next, each codeblock from (18) is multiplied with the last three columns of generator matrix G from (17), to obtain the following 48 (or 16×3) predicted signatures

$$\begin{aligned}
\text{predicted}_{47} &= a_{61}\alpha^{63} + a_{60}\alpha^{62} + a_{58}\alpha^{60}\\
\text{predicted}_{46} &= a_{61}\alpha^{63} + a_{60}\alpha^{62} + a_{59}\alpha^{61}\\
\text{predicted}_{45} &= a_{60}\alpha^{62} + a_{59}\alpha^{61} + a_{58}\alpha^{60}\\
\text{predicted}_{44} &= a_{57}\alpha^{59} + a_{56}\alpha^{58} + a_{54}\alpha^{56}\\
&\cdots\\
\text{predicted}_3 &= a_4\alpha^6 + (a_{63} + a_3)\alpha^5 + (a_{63} + a_{62} + a_2)\alpha^4\\
\text{predicted}_2 &= (a_{62} + a_1)\alpha^3 + (a_{63} + a_0)\alpha^2 + a_{62}\\
\text{predicted}_1 &= (a_{62} + a_1)\alpha^3 + (a_{63} + a_0)\alpha^2 + (a_{63} + a_{62})\alpha\\
\text{predicted}_0 &= (a_{63} + a_0)\alpha^2 + (a_{63} + a_{62})\alpha + a_{62}.
\end{aligned} \tag{22}$$

We note that the first bit of the message to be multiplied with G is a_{62}, then $(a_{63} + a_{62})\alpha$, and so on, since a_{62} is the least significant bit. To obtain the actual signatures, we rename the coefficients of (18): a_{61} as γ_{63}, a_{60} as γ_{62},..., and a_{62} as γ_0, obtaining the same formulations as (20).

In this article, we work with two finite fields to show how the proposed error detection schemes are applied; however, by following the previous derivations, any field size can use the presented schemes.

IV. Error Coverage and FPGA Implementations

To calculate the error coverage given by the various error detection techniques discussed in this article, the total number of signatures must be determined. When the total signatures s are calculated, the error coverage is given by performing $100 \cdot (1 - (\frac{1}{2})^s)\%$. We note that one actual signature with one predicted signature is equivalent to just one signature in this formula.

To perform the finite field inversion of an element in GF(2^7) with normal basis using ITA and CRC-3 as the error detection technique, a total of $3_{\text{mult}} \cdot (6_{\alpha B} \cdot (3_{\text{EF}}))$ or 54 signatures are needed, where mult denotes the number of finite field multiplications, αB stands for the number of αB operations in each multiplication, and EF is the number of error flags in each αB operation. These 54 signatures translate into an error coverage of $100 \cdot (1 - (\frac{1}{2})^{54})\%$ or close to 100%.

On the other hand, to perform the finite field inversion of an element in GF(2^{64}) with polynomial basis using ITA and

TABLE VI
FPGA IMPLEMENTATION RESULTS OF THE PROPOSED FAULT DETECTION SCHEMES BASED ON CRC-3 FOR FINITE FIELD INVERSION USING ITA AND ELEMENTS IN GF(2^7) WITH NORMAL BASIS. PERFORMED ON XILINX FPGA FAMILY VIRTEX-7 DEVICE XC7VX1140TFLG1930-I

Architecture	Area (occupied slices)	Delay (ns)	Power (mW) @50 MHz	Throughput (Gbps)	Efficiency (Gbps/Slices)
Original ITA	294	1.452	0.599	4.821	0.016
ITA with CRC-3	408 (38.78%)	1.609 (10.81%)	0.610 (1.84%)	4.351 (-9.75%)	0.011 (-31.25%)

TABLE VII
FPGA IMPLEMENTATION RESULTS OF THE PROPOSED FAULT DETECTION SCHEMES BASED ON HAMMING CODES FOR FINITE FIELD INVERSION USING ITA AND ELEMENTS IN GF(2^{64}) WITH POLYNOMIAL BASIS. PERFORMED ON XILINX FPGA FAMILY KINTEX ULTRASCALE+ DEVICE XCKU5P-SFVB784-1LV-I

Architecture	Area (occupied slices)	Delay (ns)	Power (mW) @50 MHz	Throughput (Gbps)	Efficiency (Gbps/Slices)
Original ITA	3347	10.377	0.571	6.167	1.84×10^{-3}
ITA with Hamming codes	4147 (23.90%)	13.675 (31.78%)	Negli. Over.	4.680 (-24.11%)	1.12×10^{-3} (-39.13%)

Hamming codes as the error detection technique, 10 finite field multiplications, and 63 finite field squarings are needed. Each finite field multiplication uses 63 α modules, and each α module utilizes 48 signatures. Moreover, each finite field squaring uses 63 α^2 modules, and each α module utilizes 48 signatures. Therefore, the total number of signatures is $10_{\text{mult}} \cdot (63_{\alpha} \cdot (48_{\text{EF}})) + 63_{\text{squaring}} \cdot (63_{\alpha^2} \cdot (48_{\text{EF}}))$ or 220 752. Moreover, the error coverage is $100 \cdot (1 - (\frac{1}{2})^{220,752})\%$ or very close to 100%. We note that our error detection schemes would not detect faults for very specific cases using CRC-3. The three schemes for hardening mentioned in Section III-A can be combined with the proposed methods to alleviate such cases. Looking at Table IV, we can corroborate that the proposed architecture would detect all faults besides: If there is a single bit flip located at the coefficients b_1 or b_5, or a double bit flip at coefficients b_4 and b_6 when the step αB^{2^0} is being performed; when a double bit flip happens at the coefficients b_2 and b_5 when the step $\alpha B^{2^{-1}}$ is being performed; when a single bit flip occurs at the coefficient b_3 or a double bit flip happens at the coefficients b_2 and b_4, b_0 and b_5, b_0 and b_3, and b_0 and b_5 when the step $\alpha B^{2^{-2}}$ is being performed; when a double bit flip at coefficients b_0 and b_6 when the step $\alpha B^{2^{-3}}$ is being performed; when a single bit flip located at the coefficients b_2 or b_4, or a double bit flip at coefficients b_1 and b_6 or b_2 and b_5 when the step $\alpha B^{2^{-4}}$ is being performed; when a single bit flip located at the coefficient b_0 or a double bit flip at coefficients b_1 and b_5 when the step $\alpha B^{2^{-5}}$ is being performed; and when a single bit flip located at the coefficient b_0 or b_5 when the step $\alpha B^{2^{-6}}$ is being performed. If an attacker has the expensive resources to mount single-bit flip attacks, the error coverage of CRC-3 for single bit flips in the step αB^{2^0} is 71.43%; 100% in the step $\alpha B^{2^{-1}}$; 85.71% in the step $\alpha B^{2^{-2}}$; 100% in the step $\alpha B^{2^{-3}}$; 71.43% in the step $\alpha B^{2^{-4}}$; 85.71% in the step $\alpha B^{2^{-5}}$; and 71.43% in the step $\alpha B^{2^{-6}}$. These calculations have been done taking into account that there is a total of 7 coefficients where the error can be injected. Additionally, if the attacker performs double bit flips, the error coverage of CRC-3 for double bit flips in the step αB^{2^0} is 95.24%; 95.24% in the step $\alpha B^{2^{-1}}$; 80.95% in the step $\alpha B^{2^{-2}}$; 95.24% in the step $\alpha B^{2^{-3}}$; 90.48% in the step $\alpha B^{2^{-4}}$; 95.24% in the step $\alpha B^{2^{-5}}$; and 100% in the step $\alpha B^{2^{-6}}$. These calculations have been done taking into account that there is a total of 21 possible double bit flips where the errors can be injected. On the other hand, the proposed architecture using Hamming codes would be able to detect all 1-b, 2-b, 3-b, and 4-b errors. This type of error detection is mutually exclusive, meaning that all errors are detected having not too many redundant error detection flags.

We have implemented the proposed fault detection techniques to demonstrate that they provide high error coverage with acceptable overhead. These implementations are performed on Xilinx FPGA family Virtex-7 device xc7vx1140tflg1930-i and family Kintex UltraScale+ device xcku5p-sfvb784-1LV-i using the Vivado tool and Verilog as the hardware design entry. Table VI shows the overheads of the fault detection architectures based on CRC-3 when added to the original finite field inversions constructions with normal basis using ITA. The overheads are presented in terms of area (occupied slices), delay (ns), power (mW) with the clock frequency of 50 MHz, throughput (Gb/s), and efficiency (Gb/s/Slices). The first three are given directly by the Vivado tool while the latter two are calculated by dividing the total number of output bits over the delay and by dividing first the total number of output bits over the delay and then over the area, respectively. As shown in Table VI, the area overhead is less than 39% and the delay overhead is close to 11%. Table VII presents the overheads of the fault detection architectures based on Hamming codes when added to the original finite field inversions constructions with polynomial basis using ITA. Table VII also presents the overheads in terms of area, delay, power, throughput, and efficiency (we note that Negli. Over. stands for negligible overhead). As it can be seen in Table VII, the area overhead is less than 24% and the delay overhead is less than 32%.

There has not been any prior work done on error detection based on CRC-3 and Hamming codes for finite field inversions using ITA with normal basis and polynomial basis elements, respectively, to the best of our knowledge. For qualitative comparison to verify that the overheads incurred are acceptable, let us go over some case studies on error detection in GF(2^m) arithmetic hardware. One of the goals of this article was to complete [34] by performing error detection on elements with normal basis. In [34], fault detection schemes for the inverse of an element in GF(2^{163}) using polynomial basis are proposed, obtaining an

area overhead of close to 26%. In [26], error detection based on parity prediction for normal basis multiplication is performed, obtaining a worst case delay overhead of 58.2% and worst case area overhead of 27%. Additionally, parity prediction signatures provide an error detection of up to 50%, i.e., if the number of faults is even, the approach would not be able to detect the faults. As shown in Table I, duplication/recomputing is also used in a variety of works. Duplication provides a high error detection percentage; however, it can add a 100% overhead unless different techniques (such as pipelining for recomputation) are carried. Reliable concurrent error detection architectures for extended Euclidean-based division over GF(2^m) are provided in [41]. The schemes utilized are based on parity prediction and they have a worst case area overhead of 25.18%. These and other similar works on error detection show that the overheads obtained in this work are acceptable.

V. Conclusion

Providing reliable architectures that allow error detection has been the focus of extensive research. Finite field inversions are used by many different architectures and it is essential that such constructions are capable of detecting natural defects in VLSI. In this work, we have presented error detection schemes based on CRC and Hamming codes for finite field inversions using ITA with normal basis and polynomial basis elements. These schemes employ different formulations and derivations that have been meticulously calculated with software implementations. Additionally, we have implemented the proposed fault detection architectures by adding them to the original finite field inversion constructions on Xilinx FPGA Virtex-7 and Kintex UltraScale+. When fault detection schemes based on CRC-3 are applied to the original ITA finite field inversion constructions for GF(2^m) elements normal basis, the area overhead is less than 39% and the delay overhead is close to 11%. When fault detection schemes based on Hamming codes are applied to the original ITA finite field inversion constructions for GF(2^m) elements with polynomial basis, the area overhead is less than 24% and the delay overhead is less than 32%. The results obtained in this work were compared to other works on error detection to prove that the presented fault detection techniques achieve a high error coverage with acceptable overheads.

References

[1] A. Ibrahim, T. Alsomani, and F. Gebali, "Unified systolic array architecture for finite field multiplication and inversion," *Comput. Elect. Eng.*, vol. 61, pp. 104–115, 2017.

[2] H. Yi, S. Tang, and R. Vemuri, "Fast inversions in small finite fields by using binary trees," *Comput. J.*, vol. 59, no. 7, pp. 1102–1112, 2016.

[3] J. Li, Z. Li, C. Xue, J. Zhang, W. Gao, and S. Cao, "A fast modular inversion FPGA implementation over $GF(2^m)$ using modified x^{2^n} unit," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2018, pp. 1–5.

[4] H. El-Razouk and A. Reyhani-Masoleh, "New architectures for digit-level single, hybrid-double, hybrid-triple field multiplications and exponentiation using Gaussian normal bases," *IEEE Trans. Comput.*, vol. 65, no. 8, Aug. 2016, Art. no. 2495.

[5] A. Reyhani-Masoleh, H. El-Razouk, and A. Monfared, "New multiplicative inverse architectures using Gaussian normal basis," *IEEE Trans. Comput.*, vol. 68, no. 7, pp. 991–1006, Jul. 2019.

[6] J. Guajardo and C. Paar, "Itoh-Tsujii inversion in standard basis and its application in cryptography and codes," *Des., Codes, Cryptogr.*, vol. 25, 2002, Art. no. 207.

[7] F. Rodriguez-Henriquez, N. A. Saqib, and N. Cruz-Cortes, "A fast implementation of multiplicative inversion over $GF(2^m)$," in *Proc. Int. Symp. Inf. Technol.*, 2005, pp. 574–279.

[8] J. Hu, W. Guo, J. Wei, and R. C. C. Cheung, "Fast and generic inversion architectures over $GF(2^m)$ using modified Itoh–Tsujii algorithms," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 62, no. 4, pp. 367–371, Apr. 2015.

[9] M. Kalaiarasi, V. R. Venkatasubramani, and S. Rajaram, "A parallel quad Itoh-Tsujii multiplicative inversion algorithm for FPGA platforms," in *Proc. IEEE 3rd ISEA Conf. Secur. Privacy*, 2020, pp. 31–35.

[10] A. Sarker, M. Mozaffari-Kermani, and R. Azarderakhsh, "Error detection architectures for ring polynomial multiplication and modular reduction of Ring-LWE in $Z = \frac{\mathbb{Z}/p\mathbb{Z}[x]}{x^n+1}$ benchmarked on ASIC," *IEEE Trans. Rel.*, vol. 70, no. 1, pp. 362–370, Mar. 2021.

[11] M. M. Kermani and R. Azarderakhsh, "Reliable architecture-oblivious error detection schemes for secure cryptographic GCM structures," *IEEE Trans. Rel.*, vol. 68, no. 4, pp. 1347–1355, Dec. 2019.

[12] M. H. Saračević et al., "Data encryption for Internet of Things applications based on Catalan objects and two combinatorial structures," *IEEE Trans. Rel.*, vol. 70, no. 2, pp. 819–830, Jun. 2021.

[13] Y. Zhu, R. Yu, D. Ma, and W. C. C. Chu, "Cryptographic attribute-based access control (ABAC) for secure decision making of dynamic policy with multiauthority attribute tokens," *IEEE Trans. Rel.*, vol. 68, no. 4, pp. 1330–1346, Dec. 2019.

[14] D. Abbasinezhad-Mood and M. Nikooghadam, "Efficient design of a novel ECC-based public key scheme for medical data protection by utilization of NanoPi fire," *IEEE Trans. Rel.*, vol. 67, no. 3, pp. 1328–1339, Sep. 2018.

[15] M. Elhoseny and K. Shankar, "Reliable data transmission model for mobile ad hoc network using signcryption technique," *IEEE Trans. Rel.*, vol. 69, no. 3, pp. 1077–1086, Sep. 2020.

[16] P. Ahir, M. M. Kermani, and R. Azarderakhsh, "Lightweight architectures for reliable and fault detection Simon and Speck cryptographic algorithms on FPGA," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 4, pp. 109:1–109:17, Sep. 2017.

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

[18] J. S. Coron, A. Roy, and S. Vivek, "Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2014, pp. 170–187.

[19] M. M. Kermani, A. Jalali, R. Azarderakhsh, J. Xie, and K.-K. R. Choo, "Reliable inversion in $GF(2^8)$ with redundant arithmetic for secure error detection of cryptographic architectures," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 3, pp. 696–704, Mar. 2018.

[20] S. Subramanian, M. M. Kermani, R. Azarderakhsh, and M. Nojoumian, "Reliable hardware architectures for cryptographic block ciphers LED and HIGHT," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 36, no. 10, pp. 1750–1758, Oct. 2017.

[21] M. M. Kermani and A. Reyhani-Masoleh, "Reliable hardware architectures for the third-round SHA-3 finalist Grostl benchmarked on FPGA platform," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, 2011, pp. 325–331.

[22] M. M. Kermani and A. Reyhani-Masoleh, "A high-performance fault diagnosis approach for the AES SubBytes utilizing mixed bases," in *Proc. IEEE Workshop Fault Diagnosis Tolerance Cryptogr.*, 2011, pp. 80–87.

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

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

[25] M. M. Kermani and A. Reyhani-Masoleh, "Fault detection structures of the S-boxes and the inverse S-boxes for the advanced encryption standard," *J. Electron. Testing: Theory Appl.*, vol. 25, no. 4, pp. 225–245, Aug. 2009.

[26] M. M. Kermani and A. Reyhani-Masoleh, "A low-cost S-box for the advanced encryption standard using normal basis," in *Proc. IEEE Int. Conf. Electro/Inf. Technol.*, 2009, pp. 52–55.

[27] A. Reyhani-Masoleh and M. Anwar Hasan, "Fault detection architectures for field multiplication using polynomial bases," *IEEE Trans. Comput.*, vol. 55, no. 9, pp. 1089–1103, Sep. 2006.
[28] C. Y. Lee, P. K. Meher, and J. C. Patra, "Concurrent error detection in bit-serial normal basis multiplication over $GF(2^m)$ using multiple parity prediction schemes," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 18, no. 8, pp. 1234–1238, Aug. 2010.
[29] A. Cintas-Canto, M. Mozaffari-Kermani, and R. Azarderakhsh, "Reliable architectures for composite-field-oriented constructions of McEliece post-quantum cryptography on FPGA," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 40, no. 5, pp. 999–1003, May 2021.
[30] C. Fei, F. Zhou, N. Wu, F. Ge, J. Wen, and P. Qin, "A scalable bit-parallel word-serial multiplier with fault detection on $GF(2^m)$," in *Proc. IEEE 20th Int. Conf. Commun. Technol.*, 2020, pp. 1660–1664.
[31] C. Y. Lee and J. Xie, "High capability and low-complexity: Novel fault detection scheme for finite field multipliers over $GF(2^m)$ based on MSPB," in *Proc. IEEE Int. Symp. Hardware Oriented Secur. Trust*, 2019, pp. 21–30.
[32] A. Cintas-Canto, M. Mozaffari-Kermani, and R. Azarderakhsh, "Reliable CRC-based error detection constructions for finite field multipliers with applications in cryptography," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 29, no. 1, pp. 232–236, Jan. 2021.
[33] A. Cintas-Canto, M. Mozaffari Kermani, and R. Azarderakhsh, "Reliable constructions for the key generator of code-based post-quantum cryptosystems on FPGA," *ACM J. Emerg. Technol. Comput. Syst.*, 2022.
[34] A. Cintas-Canto, M. Mozaffari-Kermani, and R. Azarderakhsh, "CRC-based error detection constructions for FLT and ITA finite field inversions over $GF(2^m)$," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 29, no. 5, pp. 1033–1037, May 2021.
[35] A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, *Applications of Finite Fields*. Norwell, MA, USA: Kluwer, 1993.
[36] A. Reyhani-Masoleh and M. A. Hasan, "Error detection in polynomial basis multipliers over binary extension fields," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2002, pp. 515–528.
[37] L. Li and S. Li, "Fast inversion in $GF(2^m)$ with polynomial basis using optimal addition chains," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2017, pp. 1–4.
[38] K. Jarvinen, V. Dimitrov, and R. Azarderakhsh, "A generalization of addition chains and fast inversions in binary fields," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2421–2432, Sep. 2015.
[39] B. Rashidi, "High-speed hardware implementation of Gaussian normal basis inversion algorithm over F_{2^m}," *Microelectron. J.*, 63, pp. 138–147, 2017.
[40] T. Shahroodi, S. Bayat-Sarmadi, and H. Mosanaei-Boorani, "Low-latency double point multiplication architecture using differential addition chain over $GF(2^m)$," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 66, no. 4, pp. 1465–1473, Apr. 2019.
[41] M. Mozaffari-Kermani, R. Azarderakhsh, C. Y. Lee, and S. Bayat-Sarmadi, "Reliable concurrent error detection architectures for Extended Euclidean-based division over $GF(2^m)$," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 22, no. 5, pp. 995–1003, May 2014.

Alvaro Cintas-Canto (Member, IEEE) received the B.Sc. degree in computer engineering and the M.Sc. degree in computer and electrical engineering from Tennessee Tech University, Cookeville, TN, USA, in 2016 and 2018, respectively, and the Ph.D. degree in computer science and engineering from the Department of Computer Science and Engineering, University of South Florida, Tampa, FL, USA, in 2021.

In 2018, he joined the Cryptographic Engineering and Hardware Security Research lab, performing research on error detection of postquantum cryptographic schemes. Since 2021, he has been an Assistant Professor with the School of Technology and Innovation, Marymount University, Arlington, VA, USA. His current research interests include hardware security, postquantum cryptography, cryptographic engineering, finite field and its applications, and high-performance embedded systems design.

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

He was with the Advanced Micro Devices as a Senior ASIC/layout Designer, integrating sophisticated security/cryptographic capabilities into accelerated processing. In 2012, he joined the Electrical Engineering Department, Princeton University, NJ, as an NSERC Postdoctoral Research Fellow. From 2013 to 2017, he was an Assistant Professor with the Rochester Institute of Technology, and since 2017, he has been with the Computer Science and Engineering Department, University of South Florida, where he is currently an Associate Professor.

Dr. Kermani is an Associate Editor for the IEEE TRANSACTIONS ON VLSI SYSTEMS, the *ACM Transactions on Embedded Computing Systems*, and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS.He has been the TPC member for a number of conferences including HOST (Publications Chair), CCS (Publications Chair), DAC, DATE, RFIDSec, LightSec, WAIFI, FDTC, and DFT. He was a recipient of the prestigious Natural Sciences and Engineering Research Council of Canada Post-Doctoral Research Fellowship in 2011 and the Texas Instruments Faculty Award (Douglas Harvey) in 2014. He is also the awardee for USF 2021 Faculty Outstanding Research Achievement Award, USF Nexus Initiative Global Award, and USF College of Engineering's 2018 Outstanding Junior Research Achievement Award.

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

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

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