A Secure Architecture for Modular Division over a Prime Field against Fault Injection Attacks

Xiaoting Hu 1,2 and Zhongping Qin 1,*

1 School of Software Engineering, Huazhong University of Science and Technology, Wuhan 430074, China; hxt@jsnu.edu.cn
2 School of Computer Science and Technology, Jiangsu Normal University, Xuzhou 221116, China
* Correspondence: zpqin@hust.edu.cn

Received: 26 January 2020; Accepted: 26 February 2020; Published: 2 March 2020

Abstract: Fault injection attacks pose a serious threat to many cryptographic devices. The security of most cryptographic devices hinges on a key block called modular division (MD) over a prime field. Although a lot of research has been done to implement the MD over a prime field in hardware efficiently, studies on secure architecture against fault injection attack are very few. A few of the studies that focused on secure architecture against fault injection attack can only detect faults but not locate faults. In this regard, this paper designs a novel secure architecture for the MD over a prime field, which can not only detect faults, but also can locate the error processing element. In order to seek the best optimal performance, four word-oriented systolic structures of a main function module (MFM) were designed, and three error detection schemes were developed based on different linear arithmetic codes (LACs). The MFM structures were combined flexibly with the error detection schemes. The time and area overheads of our architecture were analyzed through the implementation in an application-specific integrated circuit (ASIC), while the error detection and location capabilities of our architecture were demonstrated by C++ simulation, in comparison to two existing methods. The results show that our architecture can detect single-bit error (SBE) with 100% accuracy and locate the erroneous processing element (PE), and correctly identify most of the single PE errors and almost all of the multi-PE errors (when there are more than three erroneous PEs). The only weakness of our architecture is the relatively high time and area overhead ratios.

Keywords: modular division (MD); secure architecture; fault injection attacks; concurrent error detection; systolic implementation

1. Introduction

Currently, there are various important integrated circuit (IC) devices, ranging from pivotal calculators to security-sensitive devices. Many of these IC devices face the risk of fault injection attacks. The consequences of these attacks include sudden failure of the IC and the leak of key secret information [1–4]. Over the years, many fault injection methods have emerged, namely, heavy ion radiation, electromagnetic interference, and laser exposure, posing an increasingly high threat to IC devices.

In the field of IC security, widespread attention has been paid to the protection against fault injection attacks [5–10], resulting in multiple measures to prevent the ICs from being attacked by fault injection. The typical measures include physical protection [11], the hardware/time redundancy (module duplication/re-computation) method [12–15], and the error detection codes (EDC)-based technique [16,17,18]. Among them, the EDC-based technique achieves the best tradeoff between fault coverage and hardware/time overheads [19]. Recently, Mustafa et al. [20] presented a novel differential fault attack (DFA)-aware floor-planning technique, which mitigates the threat from different fault
attacks. The authors confirmed that this technique, involving no algorithm, circuit nature, or protocol, falls into the category of physical protection. Despite being an attractive approach to design secure ICs, the DFA-aware technique is not directly linked to our research, which focuses on the EDC-based technique for fault detection.

Finite field arithmetic has important applications in cryptography and coding theory. One of the most essential and complex operation over a finite field is the modular division/inversion (MD/MI) over a prime field. The secure implementation of this operation preludes the security of related cryptographic devices.

Because of modular division in cryptography, researchers have paid much attention to it. Various algorithms have been presented to compute modular division. They can be classified into three categories: repeated exponentiation, the extended Euclidean algorithm, and the extended binary GCD algorithm. The extended GCD algorithm is most suitable to be implemented in hardware. Many implementation architectures of this algorithm have been presented [26–33]. These works made great contributions to improve hardware implementation performance of modular division, however, they did not guarantee the architectures were reliable when the chip was injected into the fault.

Considering the reliability of finite field arithmetic, researchers have also developed many fault detection schemes [6,10,34–40]. Some scholars [34–36] explored deep into concurrent error detection schemes for division/division over the Galois field GF(2^m). Based on time redundancy, Bayat Sarmadi and Hasan [34] put forward a security scheme for MD over polynomial basis and normal basis finite fields but did not implement the proposed scheme. Mozaffari Kermani and Reyhani Masoleh [35] proposed a fault detection scheme for multiplicative division on GF(2^m) based on parity prediction. Mozaffari Kermani et al. [36] applied a similar technique to ensure the structural reliability of an extended Euclidean algorithm over GF(2^m). The parity prediction scheme is efficient for MD over GF(2^m), but not the best choice for arithmetic operations like MD over a prime field. This is because the parity prediction of arithmetic operations is usually more complicated than logic operations. The complexity leads to large area and time overheads. Compared with parity code, arithmetic codes are efficient in protecting arithmetic operations [37]. Some fault-tolerant schemes have been investigated based on different arithmetic codes [6,10,38,39], yet fail to tackle MD over a prime field.

Focusing on modular division over a prime field, Hu et al. [40] came up with a concurrent error detection scheme based on linear arithmetic code (LAC), and applied it to systolic implementation of MD. As shown in Figure 1, the scheme consists of a main function module (MFM) and an error detection module (EDM). The former is responsible for the MD computation, and the latter for error detection of the MFM, using the LAC. The EDM involves three sub-modules, namely, actual check part generator (ACPG), check part predictor (CPP), and comparator (CMP). Among them, the ACPG accepts the results of each iteration from the MFM, and generates their actual check parts; the CPP predicts the check parts of iterative results; the CMP compares the actual check parts with the predicted ones—if the two parts are consistent in each iteration, then no fault has been injected in the MFM, and the system will continue to run; otherwise, the architecture will issue an alarm about the probability of fault attacks in the MFM. The MD output is regarded as valid, if and only if the actual and predicted check parts of any iteration are equal to each other. On the upside, the scheme in [40] can detect errors accurately with limited area and time overheads. On the downside, this scheme cannot report the detected errors in real-time because it takes the entire n-bit iterative result as the detection cell so that the system cannot output the iterative detection result until the last word of the iteration result is valid. Neither could this scheme locate the detected errors.
This paper extends our work in [40] to present a new secure MD architecture that can not only detect, but also locate, the error. Inspired by the concurrent error detection scheme for the MD in [40], this paper proposes a novel MD architecture capable of effectively detecting and locating erroneous processing elements, and applicable to related cryptographic implementations, laying the basis for prevention of natural faults and fault attacks. The contributions of this paper are as follows:

1. This paper extends our work in [40] to present a new secure MD architecture that can not only detect, but also locate, the error.
2. Twelve combinations of four word-oriented systolic implementations of MFM and three error-detecting schemes with different LAC values were explored to seek the best tradeoff between area, time overheads, and error detection capability. These combinations were modeled using Verilog language and synthesized using Synopsys Design Complier with the TMSC (Taiwan Semiconductor Manufacturing) 90nm CMOS (Complementary metal-oxide-semiconductor) standard cell library. Their functions were also verified using Modelsim.
3. Random fault injections were simulated using the C++ program and the simulation result shows that the proposed architecture can detect single-bit error (SBE) with 100% accuracy and locate the erroneous processing element (PE). The detection capability of single-PE error (multiple-bit error is injected into one PE) and multi-PE error vary with the value of LAC. However, it reaches 99.898% when the number of erroneous processing elements is three or more.
4. In addition, the proposed architecture can greatly shorten the delay in error reporting.

The remainder of this paper is organized as follows: Section 2 briefly reviews the MD algorithm and the LAC; Section 3 sets up our architecture and describes its algorithm; Section 4 analyzes the error detection and location capabilities of our architecture; Section 5 presents the application-specific integrated circuit (ASIC) implementation results of our architecture, and compares them with those of existing schemes; Section 6 puts forward the conclusions of this research.

2. Preliminaries

2.1. The MD Algorithm

Let $X$, $Y$, and $M$ be three integers, where the greatest common divisor $\text{GCD}(Y, M) = 1$. The MD problem aims to find an integer $R$ satisfying $RY \equiv X \pmod{M}$. Chen and Qin [31] optimized the extended binary GCD algorithm for the MD over a prime field, which requires less iteration than the other existing algorithms. Its equivalent description is shown as Algorithm 1.

---

**Figure 1.** The concurrent error detection scheme for the modular division (MD) in [40].
The relevant parameters are defined as in Table 1 below.

| Algorithm 1 Equivalent Description of Modular division algorithm over prime field in [31] |
|---------------------------------------------------------------|
| **Input:** X, Y, M( M is a prime), 0 ≤ Y ≤ M, 0 ≤ X ≤ M |
| **Output:** R(Y ≡ X mod M) |
| 1: Initialization |
| 2: i = 0, φ^0 = 1, δ^0 = 1 |
| 3: [A^0, B^0, R^0, S^0] ← [Y, M, X, 0] |
| 4: Modular Division Computation |
| 5: While A^0 ≠ ±1 |
| 6: i = i + 1 |
| 7: Control Element |
| 8: α^i = |
| 0 if A^i mod 2 = 0 |
| 1 else if (A^i+1 + B^i-1) mod 4 = 0 |
| -1 else if (A^i-1 - B^i-1) mod 4 = 0 |
| 9: β^i = |
| φ^i-1 if (R^i-1 + α^i S^i-1) mod 2 = 0 |
| -φ^i-1 otherwise |
| 10: φ^i = |
| φ^i-1 if (R^i-1 + α^i S^i-1) mod 2 = 0 |
| -φ^i-1 otherwise |
| 11: λ^i = |
| 0 if (α = 0) or (δ^i ≤ 0) |
| 1 otherwise |
| 12: δ^i = |
| 1 + δ^i-1 if (λ^i = 0) |
| 1 - δ^i-1 if (λ^i = 1) |
| 13: Compute Iteration Output |
| A^i = (A^i-1 + α^i B^i-1)/2 |
| B^i = (B^i-1 if (λ^i = 0) |
| S^i = (S^i-1 if (λ^i = 1) |
| R^i = (R^i-1 + α^i S^i-1 + β^i M)/2 |
| 14: |
| S^i = (S^i-1 if (λ^i = 0) |
| R^i = (R^i-1 if (λ^i = 1) |
| 15: end for |
| 16: end while |
| 17: if (A^i = -1) then R^i = -R^i |
| 18: R = R^i |
| 19: Return R |

2.2. The Linear Arithmetic Code (LAC)

**Definition 1 (LAC).** Let A be an integer, and p be a small odd number (p <= A). Then, (A, A mod p) is defined as a LAC word of A, where A mod p, denoted as |A|_p, is the check part of the LAC word, and p is the check factor.

The LAC can be added and multiplied as normal integers:

|A + B|_p = ||A|_p + |B|_p|_p and |A × B|_p = ||A|_p × |B|_p|_p |

In theory, p can be any relatively small prime number. In practice, however, the p value greatly affects the performance of different applications. This paper adjusts the p value for specific MFM structure, provided that it satisfies p = 2^l - 1, where l = 2,3,5 (namely, p = 3,7,31).

3. Proposed Secure Architecture and Its Algorithm Description

This section firstly proposes a secure architecture for the MD, and then shows its algorithm description. The relevant parameters are defined as in Table 1 below.
Table 1. Parameter definitions.

| Symbol | Description |
|--------|-------------|
| \( p \) | The check factor of the LAC |
| \( n \) | The bit number of modulus \( M \) |
| \( w \) | The word size of PE (\( w\in\{8, 16, 32, 64\} \)) |
| \( e \) | The number of words in operands \( e = \lceil n/w \rceil \) |
| \( X^i \) | The value of \( X \) at the \( i \)-th iteration of Algorithm 2 |
| \( X_k \) | The \( k \)-th word of \( X \) (\( X \) can be A, B, R, or S) |
| \( L(X_k) \) | The least significant bit (LSB) of \( X_k \) |
| \( L^*(X_0) \) | Bits 1 to 0 from \( X_0 \) |
| \( H(X_k) \) | Bits \( w-1 \) to 1 from \( X_k \) |
| \( Z \) | The set \{A, B, R, S\} |
| \( Z_k \) | The set \{A\_k, B\_k, R\_k, S\_k\} |
| \( L(Z_k) \) | The set \{L(A\_0), L(B\_0), L(R\_0), L(S\_0)\} |
| \( L^*(Z_0) \) | The set \{L\_k(A\_0), L\_k(B\_0), L\_k(R\_0), L\_k(S\_0)\} |
| \( H(Z_k) \) | The set \{H(A\_k), H(B\_k), H(R\_k), H(S\_k)\} |
| \( X||Y \) | The concatenation of \( X \) and \( Y \) |
| \( CA_k \) | The 1-bit carry-out signal from the computation of the \( k \)-th word of A |
| \( CR_k \) | The 2-bit carry-out signal from the computation of the \( k \)-th word of R |

As shown in Figure 2, the proposed secure architecture contains two modules, namely, the MFM (marked by green line) and the EDM (marked by red line). The former is employed to compute the MD and the latter to detect the error in MFM. The computing process of the secure architecture is summarized as Algorithm 2. In this algorithm, Lines 4, 18–20 describe the error detection function, which is mapped into EDM of Figure 2, and the left lines describe the word-oriented computing process of MD, which is mapped to MFM.

MFM architecture of Figure 2 is similar with that in [40]. In this architecture, the \( n \)-bit operands A, B, R, S are split into \( e \) w-bit words, each of which is processed by a PE. A total of \( e \) PEs are combined into a systolic array to process them word by word where \( e = \lceil n/w \rceil \). Specifically, the control element (CE) executes the operations in Lines 9–15, producing control variables (\( \alpha, \beta, \phi, \lambda, \delta \)), initial carry-in signals (1-bit \( CA_{-1} \) and 2-bit \( CR_{-1} \)), and the \( f \) signal that controls the initialization in Lines 2–4; the PE\( k \), the physical mapping of the \( k \)-th step in Line 17 in for-loop, targets the \( k \)-th word of A, B, R, and S. From right to left, a total of \( e \) PEs are connected to complete the computation of one iteration (\( e = 4 \) in Figure 2). In addition, in MFM, the testing element (TE) module is used to execute the condition judgment operation in Line 6. Finally, the output control element (OCE) executes the operations in Lines 23–24, ensuring that the MD problem outputs parallel results. This paper implements the secure architecture for four types of MFM structures, namely, Type-8, Type-16, Type-32, and Type-64. The number in the name of each type equals \( w \), the word size of PE. Note that, in this paper, we do not show the implementation of MFM; the implementation details of MFM can be found in [40].
Algorithm 2 Proposed word-oriented MD Algorithm with concurrent error detection

**Input:** $X = \sum_{k=0}^{c-1} X_k 2^{kw}, \ Y = \sum_{k=0}^{c-1} Y_k 2^{kw}, \ M = \sum_{k=0}^{c-1} M_k 2^{kw}$

**Output:** $R(\{Y, X \mod M\})$

1: **Initialization**
2: $i = 0$, $\phi^0 = 1$, $\delta^0 = 1$
3: $\{A^0, B^0, R^0, S^0\} \leftarrow \{Y, M, X, 0\}$
4: for $(k = 0 \text{ to } k < c)$ do
5:     $A^k, B^k, R^k, S^k = [Y_kL_p, M_kL_p, X_kL_p, 0]$
6:     if $(\{A_k^0, B_k^0, R_k^0, S_k^0\})$ then exit
7: end for
8: // Modular division computation
9: $\alpha^i = \begin{cases} 0 & \text{if } L(A_0^{i-1}) = 0 \\ 1 & \text{else if } (\phi(L(A_0^{i-1}) + L(R_0^{i-1}))) \mod 4 = 0 \\ -1 & \text{else if } (\phi(L(A_0^{i-1}) - L(R_0^{i-1}))) \mod 4 = 0 \end{cases}$
10: $\beta^i = \begin{cases} \phi^{i-1} & \text{if } (R_0^{i-1} + \alpha S_0^{i-1}) \mod 2 = 0 \\ \phi^{i-1} & \text{otherwise} \end{cases}$
11: $\delta^i = \begin{cases} \phi^{i-1} & \text{if } (R_0^{i-1} + \alpha S_0^{i-1}) \mod 2 = 0 \\ -\phi^{i-1} & \text{otherwise} \end{cases}$
12: $\lambda^i = \begin{cases} 1 & \text{otherwise} \end{cases}$

---

**Figure 2.** The proposed secure systolic architecture for the MD.
provides an ED to each PE. Hence, the validity of any Ef will lead to an error alarm and the termination of the MD process; otherwise, the MD process will continue. The logic diagram of \( \text{ED}_k \) encompasses \( \text{ACP}_g \), CPP, and CMP. Among them, \( \text{ACP}_g \) executes the operation of Line 18 to generate the actual check part of \( \{A_k, B_k, R_k, S_k\} \). CPP executes the operation of Line 19 to predict the check part of \( \{A_k, B_k, R_k, S_k\} \). CMP compares the actual and predicted check parts. If the two parts are unequal, CMP will issue an alarm \( \text{Ef}_k \) about the high possibility of a fault injection attack on PE, and the system will terminate the MD process; otherwise, the MD process will continue. The logic diagrams of ACP and CPP are shown as Figures 4 and 5, respectively. Note that our architecture provides an ED to each PE. Hence, the validity of any Ef will lead to an error alarm and the termination of the MD, which overcomes the defect of the EDM in [40]—taking the entire n-bit iterative result as the detection so that the system cannot output the iterative detection result until the last word of the iteration result is valid.
and the system will terminate the MD process; otherwise, the MD process will continue. The logic diagrams of ACPG_k and CPP_k are shown as Figures 4 and 5, respectively. Note that our architecture provides an ED to each PE. Hence, the validity of any E_f will lead to an error alarm and the termination of the MD, which overcomes the defect of the EDM in [40]—taking the entire n-bit iterative result as the detection so that the system cannot output the iterative detection result until the last word of the iteration result is valid.

4. Testing and Comparison of Error Detection Capability

4.1. Attacker Model

Since the emergence of fault injection, various attack methods have been created to inject fault into semiconductors [41], which makes it hard to define the fault model. Referring to the relevant literature, this paper puts forward the following common hypotheses on the capability of attackers: (1) Attackers are incapable of directly invading, modifying, or rebuilding the circuit structure, using powerful fault injection techniques like a focused ion beam [42]. It is a most powerful assumption of the capability of attacker. If attackers are capable of rebuilding the circuit, then our secure architecture is not able to resist this attack. However, the attack is high-cost and is hard to carry out in practice. It needs
very expensive consumables and a strong technical background [24]. (2) Attackers are incapable of tampering with the clock signal [10]. Although tampering with the clock signal is a viable option for an attacker, it is a common assumption for a secure architecture designer because the designer is usually focused on protecting the data path of the chip but not the clock. (3) Attackers are capable of injecting a fault into either MFM or EDM, but not both at the same time. However, the probability that a fault is injected into MFM and EDM at the same time and escape from a comparison result is very small in practice. (4) Attackers are incapable of controlling the error pattern in the MFM output. It is also a strong assumption of the capability of an attacker, that if an attacker is capable of injecting a fault in MFM and causes an error pattern that our architecture cannot detect, our secure scheme will not work.

4.2. Five Types of Error Models

In this paper, the five types of error models were adopted to verify the error detection capability of our secure architecture.

1. Single-bit error (SBE): The injected fault causes a one-bit error in the PE output.
2. Single-PE single-cycle error (SPSCE): The injected fault causes an error in the output of one PE, and the error lasts only one cycle.
3. Single-PE multi-cycle error (SPMCE): The injected fault causes an error in the output of one PE, and the error lasts multiple cycles in a row.
4. Multi-PE single-cycle error (MPSCE): The injected fault causes an error in the output of multiple PEs, and the errors last only one cycle.
5. Same-iteration multi-PE error (SIMPE): The injected fault causes an error in the output of multiple continuous PEs in the same iteration.

The five error models are visualized in Figure 6, where each black rectangle represents a PE stricken by the injected fault, and each black dot represents an erroneous bit in PE output.

![Figure 6. Five types of error models.](image)

4.3. Simulation and Comparison of Error Detection Capabilities

In order to analyze the error detection capability of the proposed systolic MD architecture, we first verified the proposed secure architecture using the C++ program, then simulated fault injection and got the error detection capability of the architecture based on 100,000 testing cases. The testing results were compared with the result of the error detection scheme in [40] in Table 2. In addition, we also applied Mozaffari Kermani’s multi-column parity prediction scheme to our MFM, and using the same way, investigated its error detection capability. Here, we need to clarify a fact—in this paper we borrowed Mozaffari Kermani’s multi-column parity prediction idea and applied it to MD over a prime field, but in [36], Mozaffari Kermani applied it to MD over GF(2^m). The MD operations over a prime field and GF(2^m) are different, thus the cost of error detection is different.
Table 2. The comparison of error detection capability.

| MFM Arch. | Scheme      | $p$  | Prob. | Error Model | Error PE Location |
|-----------|-------------|------|-------|-------------|-------------------|
|           |             |      |       | SBE         | SPSCE         | SPMCE         | MPSCE         | SIMPE         |               |
|           |             |      |       | One-bit Error | One-PE Error | Two-PE Error | Three-PE Error | Two-PE Error | Three-PE Error | Two-PE Error | Three-PE Error |   |
| Type-8    | Parity [36] | 8bit | P(%)  | 100        | 100          | 100          | 100          | 100          | 100          | 100          | 100          | × |
| Style-I [40] | 2^8 – 1  | P(%) | 100   | 99.895     | 100          | 100          | 100          | 100          | 100          | 100          | 100          | × |
| Style-II (Ours) | 3        | P(%) | 100   | 85.432     | 97.878       | 99.691       | 97.878       | 99.691       | 97.878       | 99.691       | ✓ |
| Style-II (Ours) | 7        | P(%) | 100   | 95.227     | 99.772       | 99.898       | 99.772       | 99.898       | 99.772       | 99.898       | ✓ |
| Style-II (Ours) | 31       | P(%) | 100   | 99.012     | 99.990       | 100          | 99.990       | 100          | 99.990       | 100          | ✓ |
| Type-16   | Parity [36] | 8bit | P(%)  | 100        | 100          | 100          | 100          | 100          | 100          | 100          | × |
| Style-I [40] | 2^{16} – 1 | P(%) | 100   | 100        | 100          | 100          | 100          | 100          | 100          | 100          | × |
| Style-II (Ours) | 3        | P(%) | 100   | 85.595     | 97.925       | 99.701       | 97.925       | 99.701       | 97.925       | 99.701       | ✓ |
| Style-II (Ours) | 7        | P(%) | 100   | 95.293     | 99.778       | 99.990       | 99.778       | 99.990       | 99.778       | 99.990       | ✓ |
| Style-II (Ours) | 31       | P(%) | 100   | 99.097     | 99.992       | 100          | 99.992       | 100          | 99.992       | 100          | ✓ |
| Type-32   | Parity [36] | 8bit | P(%)  | 100        | 100          | 100          | 100          | 100          | 100          | 100          | × |
| Style-I [40] | 2^{32} – 1 | P(%) | 100   | 100        | 100          | 100          | 100          | 100          | 100          | 100          | × |
| Style-II (Ours) | 3        | P(%) | 100   | 85.597     | 97.925       | 99.701       | 97.925       | 99.701       | 97.925       | 99.701       | ✓ |
| Style-II (Ours) | 7        | P(%) | 100   | 95.294     | 99.778       | 99.990       | 99.778       | 99.990       | 99.778       | 99.990       | ✓ |
| Style-II (Ours) | 31       | P(%) | 100   | 99.097     | 99.992       | 100          | 99.992       | 100          | 99.992       | 100          | ✓ |
| Type-64   | Parity [36] | 8bit | P(%)  | 100        | 100          | 100          | 100          | 100          | 100          | 100          | × |
| Style-I [40] | 2^{64} – 1 | P(%) | 100   | 100        | 100          | 100          | 100          | 100          | 100          | 100          | × |
| Style-II (Ours) | 3        | P(%) | 100   | 85.598     | 97.925       | 99.701       | 97.925       | 99.701       | 97.925       | 99.701       | ✓ |
| Style-II (Ours) | 7        | P(%) | 100   | 95.295     | 99.779       | 99.990       | 99.779       | 99.990       | 99.779       | 99.990       | ✓ |
| Style-II (Ours) | 31       | P(%) | 100   | 99.098     | 99.992       | 100          | 99.992       | 100          | 99.992       | 100          | ✓ |
For a simplified description, we used Parity, Style-I, and Style-II to represent the error detection scheme in [36,40], and in this paper. Table 2 shows simulation results of all three schemes. As shown in Table 2, for the SBE model, all three methods detected 100% of the errors. For the other error models, Style-II exhibited a poorer error detection capability than Style-I and Parity, due to its extremely small p value. However, Style-II architectures with $p = 7$ and $p = 31$ were similar to Style-I and Parity in error detection performance. When there were three or more erroneous PEs, Style-II could detect 99.898% or more errors, slightly behind that of Style-I and Parity. Despite the lag in detection ability, Style-II has an advantage over the two contrastive methods—once an error was detected in a PE, the erroneous PE could be located 100%.

Table 2 also shows that the error detection capability of Style-I varied with the MFM structures—the longer the word size of the PE in the MFM, the better the error detection capability. By contrast, Style-II’s error detection capability changed only slightly with the MFM structures. For Style-II, the error detection capability increases with the check factor $p$, under the same MFM structure.

5. Analysis on Time and Area Overheads

This section mainly presents the time and area overheads of our architecture (Style-II) with three different check factors ($p = 3, 7, \text{and } 31$) under each MFM structure. In order to get time and area overheads, we first modeled the proposed architecture using Verilog, then verified using Modelsim, and finally synthesized the circuit by Synopsys Design Vision with TSMC 90nm CMOS standard cell library. For comparison, Style-I and Parity methods were also modeled and synthesized under the same conditions. The synthesized results are given in Table 3, where time (area) overhead ratio refers to the quotient between extra time (area) overhead and MFM time (area).

Firstly, as shown in Table 3, the time and area overhead ratios of Style-II always surpassed those of Style-I, whichever the check factor, but they were lower than those of Parity at $p = 3$ or 7. For example, when the MFM structure belonged to Type-8, the mean time and area overhead ratios of Style-I were 0.51% and 41.31%, respectively; those of Style-II with $p = 3$ were 11.86% and 45.67%, respectively; those of Style-II with $p = 7$ were 20.70% and 77.84%, respectively; those of Parity were 30.07% and 72.23%, respectively; those of Style-II with $p = 31$ were 49.64% and 123.22%.

Secondly, we noticed that, when the $p$ value was fixed, the time overhead ratio increased with the operand size $n$, the area overhead ratio decreased with the growth in $n$, and the product of time and area overhead ratios decreased with $n$. For example, for Style-II with $p = 7$, the mean time and area overhead ratios were 18.50% and 104.80%, respectively, when $n = 128$, and 23.64% and 46.80%, respectively, when $n = 1024$. Judging by only time and area overheads, Style-II’s performance is negatively correlated with $p$-value. However, considering overall performance including time overheads, area overheads, and error detection capability, Style-II with $p = 7$ should be a better choice compared with Style-II with $p = 3$ or 31.

Thirdly, under the fixed $p$-value, Style-II’s performance varied with the MFM structures. The larger the word size of the PE in the MFM, the greater the time and area overhead ratios. For example, when Style-II with $p = 3$ was applied to Type-8 MFM, the mean time and area overhead ratios were 11.86% and 45.67%, respectively; when it was applied to Type-16 MFM, the two ratios were 33.98% and 51.02%, respectively. Hence, Style-II works better in error detection of short-word systolic implementation of the MFM. On the contrary, Style-I is more suitable for error detection of long-word systolic implementation of the MFM. In both methods, the time and area overheads are negatively correlated with the operand size $n$. In other words, the two methods are more efficient for large integer MD problems.

Finally, Style-II showed a much shorter delay in error reporting than Style-I and Parity. The delay of Style-II increased with $w$ and $p$, but remained basically constant when the MFM structure was stable and the operand size varied. Overall, Style-II with $p = 7$ and Type-8 MFM strikes a good balance between time overheads, area overheads, and error detection capability.
### Table 3. Comparison of time and area overheads.

| MFM Types     | Scheme | $P$  | Operand Size (bit) | Total Area (K gate) | Extra Overhead | Extra Overhead Ratio | Error-Reporting Delay (ns) |
|---------------|--------|------|-------------------|---------------------|----------------|----------------------|---------------------------|
|               |        |      |                   |                     | Extra Cycles   | Area (K gate) | Time (%) | Area (%) |
| **Type-8**    |        | 8bit |                   |                     |                |                     |                          |
| Parity [36]   |        | 128  | 61.06             | 3                   | 29.96          | 50.11               | 30.82                     |
|               |        | 256  | 131.16            |                     | 59.68          | 85.33               | 76.33                     |
|               |        | 512  | 308.49            |                     | 119.12         | 63.49               | 29.76                     |
|               |        | 1024 | 799.28            |                     | 237.99         | 42.40               | 29.59                     |
| Style-I [40]  | $2^8 - 1$ | 128  | 48.84             | 2                   | 17.74          | 1.09                | 57.06                     |
|               |        | 256  | 105.70            |                     | 24.22          | 0.54                | 47.88                     |
|               |        | 512  | 257.57            |                     | 68.20          | 0.27                | 36.01                     |
|               |        | 1024 | 697.53            |                     | 136.25         | 0.13                | 24.27                     |
| **Style-II (Ours)** |        | 3    | 1111              | 2                   | 32.59          | 6.65                | 57.06                     |
|               |        | 256  | 1052              |                     | 37.33          | 13.14               | 53.08                     |
|               |        | 512  | 264.33            |                     | 74.96          | 15.40               | 37.40                     |
|               |        | 1024 | 705.71            |                     | 130.94         | 24.27               | 27.21                     |
| **Type-16**   |        | 7    | 1136              | 2                   | 63.69          | 18.50               | 104.80                    |
| Parity [36]   |        | 128  | 83.73            |                     | 65.19          | 20.43               | 91.20                     |
|               |        | 256  | 98.00             |                     | 136.67         | 20.10               | 27.40                     |
|               |        | 512  | 319.20            |                     | 257.57         | 20.22               | 68.56                     |
|               |        | 1024 | 814.37            |                     | 705.71         | 23.64               | 46.80                     |
| Style-I [40]  | $2^{16} - 1$ | 7    | 800               | 2                   | 83.73          | 45.06               | 169.24                    |
|               |        | 256  | 800               |                     | 97.73          | 45.06               | 148.68                    |
|               |        | 512  | 396.64            |                     | 205.27         | 45.06               | 108.40                    |
|               |        | 1024 | 970.66            |                     | 415.89         | 45.06               | 74.97                     |
| **Style-II (Ours)** |        | 31   | 800               | 2                   | 493.96         | 16.33               | 120.53                    |
|               |        | 256  | 800               |                     | 83.73          | 16.33               | 148.68                    |
|               |        | 512  | 396.64            |                     | 205.27         | 16.33               | 108.40                    |
|               |        | 1024 | 970.66            |                     | 415.89         | 16.33               | 74.97                     |
| **Type-16**   |        | 7    | 800               | 3                   | 800            | 42.22               | 67.45                     |
| Parity [36]   |        | 128  | 59.35             |                     | 128.40         | 49.39               | 169.24                    |
|               |        | 256  | 769               |                     | 122.37         | 49.39               | 148.68                    |
|               |        | 512  | 619.91            |                     | 266.26         | 49.39               | 108.40                    |
| Style-I [40]  | $2^{16} - 1$ | 16   | 800               | 3                   | 59.35          | 49.39               | 74.97                     |
|               |        | 256  | 800               |                     | 97.73          | 49.39               | 148.68                    |
|               |        | 512  | 396.64            |                     | 205.27         | 49.39               | 108.40                    |
|               |        | 1024 | 970.66            |                     | 415.89         | 49.39               | 74.97                     |
| **Style-II (Ours)** |        | 31   | 800               | 3                   | 800            | 42.22               | 67.45                     |
|               |        | 256  | 800               |                     | 83.73          | 12.52               | 122.81                    |
|               |        | 512  | 396.64            |                     | 205.27         | 12.52               | 102.30                    |
|               |        | 1024 | 970.66            |                     | 415.89         | 12.52               | 77.37                     |
Table 3. Cont.

| MFM Types | Scheme | $P$ | Operand Size (bit) | $f$ (MHZ) | Total Area (K gate) | Extra Overhead | Extra Overhead Ratio | Error-Reporting Delay (ns) |
|-----------|--------|----|--------------------|----------|---------------------|---------------|----------------------|--------------------------|
|           |        |     |                    |          |                     | Extra Cycles | Area (K gate)        | Time (%)                 | Area (%)                 |
|           |        |     |                    |          |                     |               |                     |                          |                          |
| Type-32   |        | 32bit |                    |          |                     |               |                     |                          |                          |
|           |   Parity [36] | 128 | 32bit              | 128      | 59.79               | 35.13         | 46.65               | 36.54                    | 142.44                   | 5.80                     |
|           |           | 256 |                    | 114.76   | 64.27               | 45.82         | 122.90              | 8.70                     |
|           |           | 512 |                    | 237.73   | 123.88              | 45.41         | 108.81              | 14.50                    |
|           |           | 1024|                    | 517.68   | 246.09              | 45.20         | 89.91               | 26.10                    |
|           |   Style-I [40] | 128 | 256                 | 34.11    | 9.45                | 38.31         | 4.00                 | 25.91                    | 18.00                    |
|           |           | 256 |                    | 71.15    | 19.67               | 38.21         | 6.00                 | 40.09                    |                          |
|           |           | 512 |                    | 151.91   | 38.05               | 33.43         | 10.00               |                          |
|           |           | 1024|                    | 346.06   | 74.47               | 44.94         | 2.80                 |                          |
|           | Style-II (Ours) | 128 | 3                    | 800      | 41.95               | 17.51         | 25.95               | 71.62                    | 2.50                     |
|           |           | 256 |                    | 82.24    | 30.82               | 35.51         | 59.94               | 2.70                     |
|           |           | 512 |                    | 175.16   | 61.30               | 40.26         | 53.85               | 2.80                     |
|           |           | 1024|                    | 395.10   | 122.51              | 40.13         | 44.94               | 2.80                     |
| Type-64   |        | 64bit |                    |          |                     |               |                     |                          |                          |
|           |   Parity [36] | 128 | 64bit               | 667      | 45.40               | 21.01         | 51.14               | 16.13                    | 3.00                     |
|           |           | 256 |                    | 93.43    | 42.02               | 50.57         | 81.75               | 3.00                     |
|           |           | 512 |                    | 197.89   | 83.05               | 55.29         | 72.94               | 3.10                     |
|           |           | 1024|                    | 440.67   | 167.95              | 55.14         | 61.59               | 3.10                     |
|           |   Style-I [40] | 128 | 7                    | 667      | 43.90               | 15.64         | 64.51               | 81.79                    | 3.20                     |
|           |           | 256 |                    | 93.43    | 42.02               | 50.57         | 81.75               | 3.00                     |
|           |           | 512 |                    | 197.89   | 83.05               | 55.29         | 72.94               | 3.10                     |
|           |           | 1024|                    | 440.67   | 167.95              | 55.14         | 61.59               | 3.10                     |
|           | Style-II (Ours) | 128 | 31                   | 625      | 51.40               | 27.01         | 61.22               | 110.71                   | 3.20                     |
|           |           | 256 |                    | 667      | 93.43               | 42.02         | 50.57               | 81.75                    | 3.00                     |
|           |           | 512 |                    | 197.89   | 83.05               | 55.29         | 72.94               | 3.10                     |
|           |           | 1024|                    | 440.67   | 167.95              | 55.14         | 61.59               | 3.10                     |
| Type-64   |        | 64bit |                    |          |                     |               |                     |                          |                          |
|           |   Parity [36] | 128 | 64bit               | 625      | 55.64               | 31.81         | 47.13               | 133.46                   | 3.20                     |
|           |           | 256 |                    | 111.73   | 63.07               | 46.28         | 129.62              | 6.40                     |
|           |           | 512 |                    | 229.41   | 125.61              | 45.87         | 122.18              | 9.60                     |
|           |           | 1024|                    | 478.77   | 250.68              | 45.66         | 109.90              | 16.00                    |
|           |   Style-I [40] | 128 | 7                    | 625      | 34.03               | 10.19         | 0.77                | 42.78                    | 2.20                     |
|           |           | 256 |                    | 667      | 61.25               | 12.59         | 0.38                | 25.89                    | 4.40                     |
|           |           | 512 |                    | 120.81   | 18.01               | 0.19          | 17.52               | 6.60                     |
|           |           | 1024|                    | 360.40   | 256.96              | 0.09          | 12.66               | 11.00                    |
| Type-64   |        | 64bit |                    |          |                     |               |                     |                          |                          |
|           |   Parity [36] | 128 | 7                    | 645      | 43.36               | 19.51         | 41.99               | 81.79                    | 3.10                     |
|           |           | 256 |                    | 87.30    | 39.01               | 41.44         | 80.73               | 3.10                     |
|           |           | 512 |                    | 181.10   | 75.99               | 45.73         | 73.70               | 3.20                     |
|           |           | 1024|                    | 385.02   | 153.23              | 45.59         | 66.90               | 3.20                     |
|           |   Style-II (Ours) | 128 | 31                   | 555      | 45.90               | 18.51         | 64.89               | 92.45                    | 3.60                     |
|           |           | 256 |                    | 93.68    | 45.37               | 64.26         | 93.92               | 3.60                     |
|           |           | 512 |                    | 191.73   | 88.62               | 68.50         | 85.95               | 3.70                     |
|           |           | 1024|                    | 540      | 402.41              | 173.36        | 68.34               | 75.69                    | 3.70                     |
6. Conclusions

This paper extends the work in [40] to put forward a new LAC-based secure architecture for the MD over a prime field against fault injection attacks. Instead of taking the long n-bit iteration result as a detection cell, this paper takes the short w-bit word as the detection cell to implement the function of locating the erroneous processing element. In this paper, four word-based MFM systolic structures with different word sizes and three error detection schemes with different values of linear arithmetic code were explored to seek an optimal tradeoff between different performance indexes. These combination architectures were modeled using Verilog and synthesized by Synopsys Design Vision with the TSMC 90nm CMOS standard cell library to get time and area overheads. The error detection and location capability of proposed architectures were also investigated using the C++ simulation method. The same methods were also used to test the performance of the architectures based on the Style-I scheme and the Parity scheme. The simulation results show that the proposed architecture with $p = 7$ and Type-8 MFM strikes a good balance between time overheads, area overheads, and error detection capability. Despite having greater area overheads than Style-I, the architecture enjoys unique advantages in the location of the error processing element and timely error reporting. The research results help to find and locate fault attacks quickly. However, the large time and area overheads of this architecture maybe limited its application; we need to optimize its implementation in future research to expand its application range.

Author Contributions: Conceptualization, X.H. and Z.Q.; software, X.H.; validation, X.H. and Z.Q.; formal analysis, X.H. and Z.Q.; investigation, X.H.; writing—original draft preparation, X.H. and Z.Q.; writing—review and editing, X.H. and Z.Q.; visualization, X.H. and Z.Q.; supervision, Z.Q.; project administration, X.H. and Z.Q.; funding acquisition, Z.Q. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China under Grant No.61702237, the Natural Science Foundation of Jiangsu Province, China under Grant No. BK20150241.

Acknowledgments: The authors would like to thank the reviewers for their constructive comments. The authors thank Changxuan Liu for her help in writing. We also would like to thank Mustafa Khairallah for his comment about DFA-aware technology.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Voyiatzis, A.G.; Serpanos, D.N. A fault-injection attack on Fiat-Shamir cryptosystems. In Proceedings of the 24th International Conference on Distributed Computing Systems Workshops, Tokyo, Japan, 23–24 March 2004; Volume 5, pp. 618–621.
2. Schmidt, J.M.; Hutter, M.; Plos, T. Optical fault attacks on AES: A threat in violet. In Proceedings of the Sixth International Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2009, Lausanne, Switzerland, 6 September 2009; pp. 13–22.
3. Trichina, E.; Korkikyan, R. Multi fault laser attacks on protected CRT-RSA. In Proceedings of the 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography, Santa Barbara, CA, USA, 21 August 2010; pp. 75–86.
4. Karaklajic, D.; Schmidt, J.M.; Verbauwhede, I. Hardware designer’s guide to fault attacks. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 2013, 21, 2295–2306. [CrossRef]
5. Karpovsky, M.; Kulikowski, K.J.; Taubin, A. Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard. In Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN 2004), Florence, Italy, 28 June–1 July 2004; pp. 93–101.
6. Gaubatz, G.; Sunar, B. Robust finite field arithmetic for fault-tolerant public-key cryptography. In Fault Diagnosis and Tolerance in Cryptography, Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography, Yokohama, Japan, 10 October 2006; Springer: Berlin/Heidelberg, Germany, 2006; pp. 196–210.
7. Eddine Cherif, B.D.; Bendiabdellah, A.; Tabbakh, M. Diagnosis of an inverter IGBT open-circuit fault by hilbert-huang transform application. Traitement du Signal 2019, 36, 127–132. [CrossRef]
8. Berzati, A.; Canovas, C.; Goubin, L. In (security) against fault injection attacks for CRT-RSA implementations. In Proceedings of the 2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography, Washington, DC, USA, 10 August 2008; pp. 101–107.

9. Dominguez-Oviedo, A.; Hasan, M.A. Error detection and fault tolerance in ECSM using input randomization. *IEEE Trans. Dependable Secur. Comput.* 2008, 6, 175–187. [CrossRef]

10. Wang, Z.; Karpovsky, M.; Joshi, A. Secure multipliers resilient to strong fault-injection attacks using multilinear arithmetic codes. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 2011, 20, 1036–1048. [CrossRef]

11. IBM 4764 PCI-X Cryptographic Coprocessor. 2011. Available online: http://www.ibm.com/support/knowledgecenter/9119-FHB/p7hcd/icc474.htm (accessed on 28 February 2020).

12. Patel, J.H.; Fung, L.Y. Concurrent error detection in ALU’s by recomputing with shifted operands. *IEEE Trans. Comput.* 1982, 31, 589–595. [CrossRef]

13. Johnson, B.W.; Aylor, J.H.; Hana, H.H. Efficient use of time and hardware redundancy for concurrent error detection in a 32-bit VLSI adder. *IEEE J. Solid-State Circuits* 1988, 23, 208–215. [CrossRef]

14. Li, J.; Swartzlander, E.E. Concurrent error detection in ALUs by recomputing with rotated operands. In Proceedings of the 1992 IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, Dallas, TX, USA, 4–6 November 1992; pp. 109–116.

15. Chiou, C.W.; Lee, C.Y.; Lin, J.M.; Hou, T.W.; Chang, C.C. Concurrent error detection and correction in dual basis multiplier over GF (2m). *IET Circuits Devices Syst.* 2009, 3, 22–40. [CrossRef]

16. Lo, J.C.; Thanawastien, S.; Rao, T.R.N. Concurrent error detection in arithmetic and logical operations using Berger codes. In Proceedings of the 9th Symposium on Computer Arithmetic, Santa Monica, CA, USA, 6–8 September 1989; pp. 233–240.

17. Nicolaidis, M.; Duarte, R.O.; Manich, S.; Figueras, J. Fault-secure parity prediction arithmetic operators. *IEEE Des. Test Comput.* 1997, 14, 60–71. [CrossRef]

18. Reyhani-Masoleh, A.; Hasan, M.A. Fault detection architectures for field multiplication using polynomial bases. *IEEE Trans. Comput.* 2006, 55, 1089–1103. [CrossRef]

19. Goubatz, G.; Sunar, B.; Karpovsky, M.G. Non-linear residue codes for robust public-key arithmetic. In *Fault Diagnosis and Tolerance in Cryptography, Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography, Yokohama, Japan, 10 October 2006*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 173–184.

20. Hariri, A.; Reyhani-Masoleh, A. Fault detection structures for the Montgomery multiplication over binary extension fields. In Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007), Vienna, Austria, 10 September 2007; pp. 37–46.

21. Bayat-Sarmadi, S.; Hasan, M.A. On concurrent detection of errors in polynomial basis multiplication. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 2007, 15, 413–426. [CrossRef]

22. Hariri, A.; Reyhani-Masoleh, A. Concurrent error detection in montgomery multiplication over binary extension fields. *IEEE Trans. Comput.* 2011, 60, 1341–1353. [CrossRef]

23. Wang, Z.; Karpovsky, M.; Sunar, B. Multilinear codes for robust error detection. In Proceedings of the 2009 15th IEEE International On-Line Testing Symposium, Lisbon, Portugal, 24–26 June 2009; pp. 164–169.

24. Barenghi, A.; Breveglieri, L.; Koren, I.; Naccache, D. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proc. IEEE 2012*, 100, 3056–3076. [CrossRef]

25. Khairallah, M.; Sadhukhan, R.; Samanta, R.; Breier, J.; Bhasin, S.; Chakraborty, R.S.; Mukhopadhyay, D. DFARPA: Differential fault attack resistant physical design automation. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 1171–1174.

26. Stein, J. Computational problems associated with Racah algebra. *J. Comput. Phys.* 1967, 1, 397–405. [CrossRef]

27. Brent, R.P.; Kung, H.T. Systolic VLSI arrays for linear time GCD computation. In *VLSI ’83:VLSI Design of Digital System*; Elsevier Science Pub. Co.: Amsterdam, The Netherlands, 1983; pp. 145–154.

28. Takagi, N. A VLSI algorithm for modular division based on the binary GCD algorithm. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 1998, 81, 724–728.

29. Kaihara, M.; Takagi, N. A Hardware Algorithm for Modular Multiplication/Division. *IEEE Trans. Comput.* 2005, 54, 12–21. [CrossRef]

30. Chen, G.; Bai, G.; Chen, H. A New Systolic Architecture for Modular Division. *IEEE Trans. Comput.* 2007, 56, 282–286. [CrossRef]
31. Chen, C.; Qin, Z. Efficient algorithm and systolic architecture for modular division. *Int. J. Electron.* 2011, 98, 813–823. [CrossRef]

32. Choi, P.; Lee, M.; Kong, J.; Kim, D.K. Efficient Design and Performance Analysis of a Hardware Right-shift Binary Modular Division Algorithm in GF(p). *J. Semicond. Technol. Sci.* 2017, 17, 425–437.

33. Chervyakov, N.; Lyakhov, P.; Babenko, M.; Nazarov, A.; Deryabin, M.; Lavrinenko, I.; Chervyakov, N. A High-Speed Division Algorithm for Modular Numbers Based on the Chinese Remainder Theorem with Fractions and Its Hardware Implementation. *Electronics* 2019, 8, 261. [CrossRef]

34. Bayat-Sarmadi, S.; Hasan, M. Concurrent Error Detection in Finite-Field Arithmetic Operations Using Pipelined and Systolic Architectures. *IEEE Trans. Comput.* 2009, 58, 1553–1567. [CrossRef]

35. Kermani, M.M.; Reyhani-Masoleh, A. Concurrent Structure-Independent Fault Detection Schemes for the Advanced Encryption Standard. *IEEE Trans. Comput.* 2010, 59, 608–622. [CrossRef]

36. Mozaffari-Kermani, M.; Azarderakhsh, R.; Lee, C.Y.; Bayat-Sarmadi, S. Reliable Concurrent Error Detection Architectures for Extended Euclidean-Based Division Over GF(2^m). *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 2013, 22, 995–1003. [CrossRef]

37. Nicolaidis, M.; Duarte, R. Fault-secure parity prediction Booth multipliers. *IEEE Des. Test Comput.* 1999, 16, 90–101. [CrossRef]

38. Yumbul, K.; Erdem, S.S.; Savas, E. On Selection of Modulus of Quadratic Codes for the Protection of Cryptographic Operations against Fault Attacks. *IEEE Trans. Comput.* 2012, 63, 1182–1196.

39. Yang, Q.; Hu, X.; Qin, Z. Secure Systolic Montgomery Modular Multiplier Over Prime Fields Resilient to Fault-Injection Attacks. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 2014, 23, 1889–1902. [CrossRef]

40. Hu, X.; Qin, Z.; Yang, Q. A Secure Modular Division Algorithm Embedding with Error Detection and Low-Area ASIC Implementation. *J. Signal Process. Syst.* 2019, 1–13. Available online: https://link.springer.com/article/10.1007/s11265-019-01481-6 (accessed on 20 February 2020).

41. Benso, A.; Prinetto, P. (Eds.) *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*; Springer Science & Business Media: Berlin, Germany, 2003; Volume 23, pp. 7–39.

42. Torrance, R.; James, D. The state-of-the-art in IC reverse engineering. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 363–381.