Secure Multipliers Resilient to Strong Fault-Injection Attacks Using Multilinear Arithmetic Codes

Zhen Wang, *Member, IEEE,* Mark Karpovsky, *Fellow, IEEE,* and Ajay Joshi, *Member, IEEE*

***Abstract*—Public-key cryptographic devices are vulnerable to fault-injection attacks. As countermeasures, a number of secure architectures based on linear and nonlinear error detecting codes were proposed. Linear codes provide protection only against primitive adversaries which no longer represents practice. On the other hand nonlinear codes provide protection against strong adversaries, but at the price of high area overhead (200–400%). In this paper we propose a novel error detection technique, based on the random selection of linear arithmetic codes for each encryption and the corresponding decryption operation. Under mild assumptions the proposed construction achieves near nonlinear code error detection performance at a lower cost (at most 50% area overhead) due to the fact that no nonlinear operations are needed for the encoder and decoder.**

***Index Terms*—Arithmetic Codes, Side-Channel Attacks, Cryptography, Multipliers.**

I. Introduction

Cryptographic devices are widely used in applications like ATM cards and commercial electronics. These devices are vulnerable to side-channel attacks such as timing analysis attacks [1], power analysis attacks [2] and fault-injection attacks [3], [4]. Due to their active and adaptive nature, fault based attacks are one of the most powerful types of side-channel attacks. Since a fault attack was demonstrated by Boneh et al. in [5] in 1996, numerous papers have been published proposing a variety of fault attacks on both public-key and private-key cryptographic devices. One of the most efficient fault-injection attacks on AES-128, for example, requires only two faulty ciphertexts to retrieve all 128 bits of the secret key [6]. Without proper protection against fault-injection attacks, the security of cryptographic devices can never be guaranteed.

Error detecting codes are often used in cryptographic devices to detect errors caused by injected faults and prevent the leakage of useful information to attackers. Most of the proposed error detecting codes are linear codes like parity codes, Hamming codes and AN codes [7]. Protection architectures based on linear codes concentrate their error detecting abilities on errors with small multiplicities or errors of particular types, e.g. errors with odd multiplicities or byte errors. However, in the presence of unanticipated errors linear codes can provide little protection. Linear 1-d parity codes, for example, can detect no errors with even multiplicities. By carefully selecting faults and injection methods an attacker can with high probability bypass the protection based on linear codes and still be able to break the security of cryptographic devices in a reasonably short time.

In [8], robust algebraic codes were proposed as an alternative to classical linear codes to protect cryptographic devices implementing AES against fault-injection attacks. In [9], robust arithmetic residue codes were proposed for the design of fault tolerant cryptographic devices performing arithmetic operations. Instead of concentrating the error detecting abilities on particular types of errors, robust codes provide nearly equal protection against all error patterns. Hence robust codes eliminate the weakness of linear codes which can be exploited by attackers to mount successful fault attacks. Moreover, the detection of errors for robust codes are message-dependent. If the same error stays for more than one clock cycle, even if the injected fault manifests as an error that cannot be detected at the current clock cycle, it is still possible that the error will be detected at the next clock cycle when a new message arrives. Thereby, the advantage of robust codes will be more significant for **lazy channels** where errors have high probabilities to repeat themselves for several clock cycles. Variants of both algebraic and arithmetic robust codes – partially robust and minimum distance robust codes – were proposed in [10]. The corresponding architectures allow various tradeoffs in terms of robustness and hardware overhead.

The main disadvantage of robust codes is the large hardware overhead when implementing nonlinear operations for the encoding and decoding circuits. In this paper, we propose a novel error detection technique based on the idea of randomly selecting a code from multiple linear codes for each encoding and the corresponding decoding operation. The resulting codes are called **multilinear codes**. These codes have similar error detection capabilities to robust codes while requiring much less hardware overhead due to the fact that no nonlinear operations are needed for the encoder and decoder.

In this paper, the constructions of multilinear arithmetic codes are presented. As an illustrative example, the design of secure multipliers (widely used as sub-blocks in public-key cryptosystems) based on multilinear arithmetic codes will be shown. We assume that countermeasures are implemented in the cryptographic device preventing the attackers from tampering with the clock signals. We further assume that a low-rate true random number generator (e.g. [11]) is available. In fact, most cryptographic devices incorporate a true random number generator by default for key initialization, random pad computation, challenge generation etc.

The error detection capability of the proposed secure multiplier architecture was simulated in C++ and compared to architectures based on linear and partially robust arithmetic

Zhen Wang, Mark Karpovsky and Ajay Joshi are with the Reliable Computing Laboratory, Department of Electrical and Computer Engineering, Boston University, 8 Saint Marys Street, Boston, MA, USA. The work of the second author has been partially supported by the NSF grant CNR 1012910.

codes to demonstrate the advantages of the proposed error detection techniques. The paper was extended from our previous work on applications of multilinear arithmetic codes in [12]. The constructions of multilinear algebraic codes and the analysis of fault detection capabilities of architectures based on multilinear algebraic codes was discussed in [13]. The application of multilinear algebraic codes in the design of secure FSMs was shown in [14].

The rest part of the paper is organized as follows. In Section II, the error and attacker models used throughout the paper are described. In Section III, we analyze the repeatability of errors when injecting faults into Wallace Tree multipliers to further motivate the usage of robust and multilinear arithmetic codes. In Section IV we formalize the design and propose several constructions of multilinear arithmetic codes. The hardware overhead and the error and fault detection capabilities of 16-bit unsigned secure Wallace tree multipliers based on linear, multilinear and partially robust arithmetic codes are compared in Section V.

II. Error and Attacker Model

In this paper we concentrate on the analysis of the error detection capabilities for systematic arithmetic codes and the security of multipliers based on these codes. Different from the widely used non-systematic AN codes [7], the codewords of systematic arithmetic codes contain two parts: the information part and the redundant part. Any codeword c can be written in the format of $(x, y), x \in Z_{2^k}, y \in Z_{2^r}$, where k is the number of information bits, r is the number of redundant bits and Z_{2^k} is the additive group of integers $\{0, 1, \cdots, 2^k - 1\}$. We denote by $e = (e_x, e_y)$ the error vector and $\tilde{c} = (|x+e_x|_{2^k}, |y+e_y|_{2^r})$ the distorted codeword in which $e_x \in Z_{2^k}, e_y \in Z_{2^r}$, $+$ is the arithmetic addition and $|\cdot|_p$ is the modulo p operation.

Let C be an arithmetic code. An error $e = (e_x, e_y)$ is masked by a codeword $c = (x, y) \in C$ if $\tilde{c} = (|x+e_x|_{2^k}, |y+e_y|_{2^r})$ also belongs to C. Given an error e, the error masking probability $Q(e)$ is calculated as follows:

$$Q(e) = \frac{\{c | c \in C, \tilde{c} \in C\}}{|C|}. \quad (1)$$

If an error is masked by all codewords of the code, $Q(e) = 1$ and the error is called **undetectable**. If $0 < Q(e) < 1$, the error is called **conditionally detectable**. Different from algebraic codes, arithmetic codes rarely have undetectable errors. To illustrate the advantage of multilinear arithmetic codes, we compare the number and the probability of **bad errors** – errors e with $Q(e) \geq 0.5$ – for linear arithmetic codes and the proposed multilinear arithmetic codes. Since bad errors are the most difficult to detect, we will show that the transition from linear to multilinear arithmetic codes results in a drastic reduction of the number of bad errors and an improvement of the error detection ability of the code.

Remark 1: For bad errors, randomly attaching a 1-bit tag to the original message and verifying the tag after the computation can achieve an equal or even better error detecting capabilities than using more complex error control codes. Similar analysis can be conducted if we want to compare the number of errors with $Q(e) \geq \beta, \beta < 0.5$.

Fault attacks can be performed in many different ways. The most investigated mechanisms of fault injections in the cryptography communities include introducing variations in power supplies [15], [16], [17], [18], perturbing the silicon of the chip using white light or laser guns (light attacks) [3], [19], [20], [15], [21], [22] and generating eddy current on the surface of the chip using magnetic field (electromagnetic attacks) [23], [20].

Fault attacks can be classified according to the capabilities of the attackers to control the parameters of the injected faults such as timing, locations, the type of the faults and the error patterns [4], [21]. With the vast arsenal of fault injection methods and techniques available to the attacker, the type of faults and the error patterns appearing as manifestations of the injected faults at the outputs of the device-under-attack is hard to model and predict. In [18], for example, the author showed that the number of faults can be controlled by reducing the supply voltage to a certain level. However, as the technology moves into deep-micro realm, it becomes harder and harder for the attacker to control the specific error patterns at the output of the device [22]. Moreover, to our best knowledge, all the known fault injection mechanisms can only provide a limited spatial and timing resolution. For instance, the affected area of the laser gun, which is one of the most powerful fault injection methods, is determined by the technologies and the width of the laser beam [22]. The time between two consecutive shot of the laser gun is affected by the speed of recharging and the delay between the trigger signal and the shot [21].

In this paper we concentrate on protecting the data path of cryptographic devices against fault injection attacks. We assume that countermeasures are implemented in the cryptographic device preventing the attackers from tampering with the clock signal and the control circuits (e.g. finite state machines and state registers). This is a common assumption in many papers discussing countermeasures against fault injection attacks for the data path of cryptographic devices [24], [10].

Throughout the paper we assume a strong attacker model in which an attacker knows everything about the hardware architecture of the device including the codes used to detect errors. Specifically, the attackers may be able to inject faults which only affect the original multiplier (but not the redundant portion used for error detection). We assume that the attacker cannot fully control the manifested nonzero error patterns at the output of the device. The manifested error patterns are determined by factors such as the number of affected gates and the input patterns to the device, etc. We further assume that the attacker cannot change the faults at each clock cycle (**slow fault-injection mechanisms**). Once faults are injected and an error is generated, the faults stay for several clock cycles before new faults can be injected and tend to manifest themselves as the same error patterns at the output of the device. This is the case for several well known fault-injection methodologies mentioned in the last paragraph. We call this kind of channels where errors have high probabilities to repeat themselves for several consecutive clock cycles **lazy channels** or **channels with memory**. Even if the attacker injects new faults, it is still possible that the new faults will have similar

locations thus manifest themselves in a similar way to previous faults. For example, the author in [21] showed that two shots of a laser gun fired in rapid succession on a 32-bit ARM processor produces the same type of errors. The reason is that the fault locations cannot be adjusted in a short time due to the inflexible laser bench.

As it will be shown in the following sections, the advantages of multilinear arithmetic codes in terms of error detection capabilities are two-fold. First, they are better than linear arithmetic codes in a sense that they have a much smaller number of bad errors. Second, multilinear arithmetic codes have much higher error detection probabilities than linear codes for lazy channels hence they will effectively prevent the attacker from implementing a successful fault induction attack under the aforementioned strong attacker model.

III. Repeatability of Errors When Injecting Faults into Wallace Tree Multipliers

To support the statement that slow fault-injection methodologies may result in repeating errors, we conduct fault-injection simulations in C++ for unsigned and signed (2's complement) Wallace tree multipliers.

Multipliers based on Wallace trees[25] are commonly used in various applications due to their faster speed compared to other alternatives. In general, the propagation delay of a n-bit Wallace tree multiplier is on the order of $O(log(n))$ in terms of logic gates. When combined with the Booth encoding technique, Wallace tree multipliers can be used for 2's complement multiplications. The gate level netlists for both signed and unsigned Wallace tree multipliers are modeled in C++. In order to inject faults into the device, we insert a multiplexer at the output of every logic gate as shown in Figure 1. To simplify the analysis, we assume that the injected faults are either stuck-at-0 or stuck-at-1 faults. When *fault_enable* is asserted, *faulty_output* is selected and the observed output of the gate is determined by the internal fault model. We further assume that the attackers are able to inject multiple faults into the devices and the injected fault (or faults) may affect more than one logic gate. Ten thousands of simulations have been performed for every fixed number of affected gates. For each simulation, locations of the affected gates are randomly picked up and one million input operand pairs to the multipliers are randomly generated.

The injected faults may or may not manifest as non-zero error patterns at the output of the multiplier. The probability of manifestation increases as more gates are affected. We also note that when only 1 gate is affected and the fault manifests,

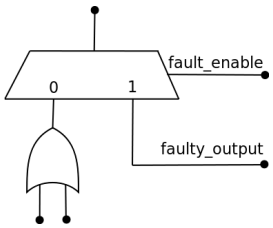

Fig. 1: Fault-injection into a single gate

TABLE I: The estimated repeatability of errors for faults injected into signed(2's complement) and unsigned Wallace tree multipliers

Type of the Multipliers	The number of faulty gates				
	1	2	3	4	5
16-bit Unsigned	0.5430	0.3166	0.2520	0.1857	0.1323
32-bit Unsigned	0.5174	0.2967	0.2301	0.1624	0.1110
16-bit Signed	0.3127	0.1904	0.1313	0.0839	0.0516
32-bit Signed	0.2730	0.1811	0.1197	0.0721	0.0418

it will always manifest as the same non-zero error pattern at the output of the multiplier. Moreover, it is highly probable that the manifested non-zero error pattern is in the format of $\pm 2^i$, where i is an integer (single errors). As the number of the affected gates increases, both the number of possible error patterns and the average multiplicity of errors will increase.

For fixed faults, the error pattern $e = (e_x, e_y)$ observed at the output of the multiplier may vary for different input pairs. Assume that every multiplication takes one clock cycle to finish. Let e_t be the observed error pattern at the tth clock cycle. The repeatability of errors can be defined by the following equation.

$$P_R = P(e_{t+1} = e_t, e_t \neq 0). \tag{2}$$

Table I shows the average repeatability of errors when up to 5 logic gates are affected by the injected faults for 16-bit and 32-bit signed and unsigned Wallace tree multipliers. For the 16-bit unsigned Wallace tree multiplier, the average repeatability of errors is higher than 0.5 when only one gate is affected. The repeatability of errors decreases as the number of affected gates increases. The signed Wallace tree multipliers based on the Booth encoding technique has smaller repeatability of errors compared to the unsigned Wallace tree multipliers. We also note that longer operand size will result in smaller error repeatability for both signed and unsigned Wallace tree multipliers.

In practice, more than 5 logic gates may be affected by the injected faults. However, in general the increase of the affected logic gates does not always result in a decrease of the repeatability of errors as we observed for Wallace tree multipliers. Generally speaking, the more linear the device is, the higher the repeatability of the error is. It is easy to verify that for linear network composed of only XOR gates (e.g. adders in Galois field $GF(2^m)$, where m is an integer), the repeatability of the error is 1 no matter how many gates are affected. Similarly, for faults injected into memories (e.g. SRAM array, FLASH), the faults tend to manifest as the same error pattern regardless of the number of memory cells that are affected. Moreover, we note that even a small repeatability of errors (e.g. 0.05 for the 32-bit signed Wallace tree multiplier) can be sufficient for robust and multilinear arithmetic codes to increase their error detection capabilities.

Remark 2: For secure applications, robust and multilinear arithmetic codes can benefit from **design for repeatabilities**. The circuit can be designed and synthesized in such a way that the repeatability of errors is high. In this case, the error detection capabilities of robust and multilinear arithmetic

codes can be drastically increased. Thereby, the security level of the system protected by these codes will be much higher. For example, reducing the average fanout of gates can result in a smaller number of possible error patterns once the fault is fixed. As a result, the repeatability of errors will increase, assuming a similar probability of fault manifestation. We also note that for linear networks consisting of only XOR gates, $P_R = 1$ assuming a simple stuck-at fault model.

IV. Constructions of Codes and Analysis of Numbers of Bad Errors

We first analyze the error detection properties of linear arithmetic codes.

Theorem 1: (**Linear Arithmetic Codes**) Let C be a linear arithmetic code defined by

$$C = \{(x, y)|x \in Z_{2^k}, y = f(x) \in Z_{2^r}\}, \tag{3}$$

in which $f(x) = |x|_p$, where p is an integer larger than 2 and $|\cdot|_p$ represents the modulo p reduction operation. Denote by $e = (e_x, e_y)$ an additive error, $e_x \in Z_{2^k}, e_y \in Z_{2^r}$, $r = \lceil \log_2 p \rceil$. The distorted codeword is $\tilde{c} = c + e = (|x + e_x|_{2^k}, |y + e_y|_{2^r})$, $c \in C$. As $\frac{p}{2^k} \to 0$, the number of bad errors is upper bounded by

$$2 \cdot (2^k + p - 2^k(H_{p-1} - H_{\lfloor \frac{p}{2} \rfloor}) - \frac{2^{k-1}}{p}), \tag{4}$$

and is lower bounded by

$$2 \cdot (2^k - 2^k(H_{p-1} - H_{\lfloor \frac{p}{2} \rfloor}) - \frac{2^{k-1}}{p}), \tag{5}$$

where H_n represents the n-th harmonic number. For large p the difference $H_{p-1} - H_{\lfloor \frac{p}{2} \rfloor}$ converges to $\ln 2$. In this case the probability of bad errors converges to $0.3 \cdot 2^{-r+1}$ as $\frac{p}{2^k} \to 0$.

If no errors occur to the redundant part of the code ($e_y = 0$), the number of bad errors $e = (e_x, 0)$ is upper bounded by $2 \cdot \lceil \frac{2^{k-1}}{p} \rceil$ and is lower bounded by $2 \cdot \lfloor \frac{2^{k-1}}{p} \rfloor$. As $\frac{p}{2^k} \to 0$, the probability of bad errors in the format of $e = (e_x, 0)$ converges to 2^{-2r}.

Proof: To simplify the analysis, we divide the errors into two classes according to the value of $x + e_x$.

1) $x + e_x < 2^k$, we have $|x + e_x|_{2^k} = x + e_x$, $f(x + e_x) = |x + e_x|_p$.
   a) $|x|_p + e_y < p$, then $||x|_p + e_y|_{2^r} = |x|_p + e_y$. An error (e_x, e_y) is masked if and only if $|x + e_x|_p = |x|_p + e_y$. Or equivalently $|e_x|_p = e_y$. For a codeword x to mask a given error (e_x, e_y), the following conditions must be satisfied:

$$x + e_x < 2^k, \tag{6}$$
$$|x|_p + e_y < p, \tag{7}$$
$$|e_x|_p = e_y. \tag{8}$$

   From (7) and (8) we have $|x|_p < p - |e_x|_p$. For any given $|x|_p < p - |e_x|_p$, the number of x satisfying (6) is bounded by $\lceil \frac{2^k - e_x}{p} \rceil$. Thereby for a given error (e_x, e_y), the total number of codewords that mask the error is $\lceil \frac{2^k - e_x}{p} \rceil \cdot (p - |e_x|_p)$. For bad errors the error masking probability is larger or equal to 0.5. Thus

$$2^{-k} \cdot \lceil \frac{2^k - e_x}{p} \rceil \cdot (p - |e_x|_p) \geq 0.5. \tag{9}$$

   As $\frac{p}{2^k} \to 0$, the error masking probability can be estimated by removing the ceiling function. Thereby (9) can be re-written as follows:

$$2^{-k} \cdot \frac{2^k - e_x}{p} \cdot (p - |e_x|_p) \geq 0.5. \tag{10}$$

   Thereby,

$$e_x \leq \frac{2^{k-1}(p - 2 \cdot |e_x|_p)}{p - |e_x|_p}. \tag{11}$$

   We know that $e_x \geq 0$, so

$$0 \leq |e_x|_p \leq \lfloor \frac{p}{2} \rfloor. \tag{12}$$

   The total number of e_x satisfying (11) and (12) is upper bounded by (For simplicity, let $i = |e_x|_p$.)

$$\sum_{i=0}^{\lfloor \frac{p}{2} \rfloor} (\frac{1}{p} \cdot \frac{2^{k-1}(p - 2i)}{p - i} + 1), \tag{13}$$

   and is lower bounded by

$$\sum_{i=0}^{\lfloor \frac{p}{2} \rfloor} (\frac{1}{p} \cdot \frac{2^{k-1}(p - 2i)}{p - i}), \tag{14}$$

   So the number of bad errors in this class is bounded by (13) and is lower bounded by (14).
   b) $p \leq |x|_p + e_y < 2^r$, errors in this class will never be masked because the redundant part is an invalid value.
   c) $|x|_p + e_y \geq 2^r$, then $||x|_p + e_y|_{2^r} = |x|_p + e_y - 2^r$. An error (e_x, e_y) is masked if and only if $|x + e_x|_p = |x|_p + e_y - 2^r$. Or equivalently $|e_x|_p = |e_y - 2^r|_p$. It is easy to show that $e_y - 2^r \in [-p+1, 0]$, so $|e_y - 2^r|_p = p + e_y - 2^r$. For a codeword x to mask a given error (e_x, e_y), the following conditions must be satisfied:

$$x + e_x < 2^k, \tag{15}$$
$$|x|_p + e_y \geq 2^r, \tag{16}$$
$$|e_x|_p = e_y + p - 2^r. \tag{17}$$

   From (16) and (17) we have $|x|_p \geq p - |e_x|_p$. For a certain value of $|x|_p \geq p - |e_x|_p$, the number of x satisfying (15) is bounded by $\lceil \frac{2^k - e_x}{p} \rceil$. Thereby for a given error (e_x, e_y), the total number of codewords that mask the error is $\lceil \frac{2^k - e_x}{p} \rceil \cdot |e_x|_p$. For bad errors the error masking probability is larger or equal to 0.5. Thus

$$2^{-k} \cdot \lceil \frac{2^k - e_x}{p} \rceil \cdot |e_x|_p \geq 0.5. \tag{18}$$

As $\frac{p}{2^k} \to 0$, the error masking probability can be estimated by removing the ceiling function. Thereby (18) can be re-written as follows:

$$2^{-k} \cdot \frac{2^k - e_x}{p} \cdot |e_x|_p \geq 0.5. \tag{19}$$

So

$$e_x \leq \frac{1}{|e_x|_p} \cdot 2^{k-1} \cdot (2|e_x|_p - p). \tag{20}$$

Because $e_x \geq 0$,

$$|e_x|_p \geq \lceil \frac{p}{2} \rceil. \tag{21}$$

The total number of e_x satisfying (20) and (21) is upper bounded by (For simplicity, let $i = |e_x|_p$.)

$$\sum_{\lceil \frac{p}{2} \rceil}^{p-1} (\frac{1}{p} \cdot \frac{2^{k-1}(2i-p)}{i} + 1), \tag{22}$$

and is lower bounded by

$$\sum_{\lceil \frac{p}{2} \rceil}^{p-1} (\frac{1}{p} \cdot \frac{2^{k-1}(2i-p)}{i}). \tag{23}$$

So the number of bad errors in this class is upper bounded by (22) and is lower bounded by (23).

From the above analysis, the total number of bad errors for the case when $x + e_x < 2^k$ is upper bounded by $2^k + p - 2^k \sum_{i=\lceil \frac{p}{2} \rceil}^{p-1} \frac{1}{i} - \frac{2^{k-1}}{p}$ and is lower bounded by $2^k - 2^k \sum_{i=\lceil \frac{p}{2} \rceil}^{p-1} \frac{1}{i} - \frac{2^{k-1}}{p}$

2) $x + e_x \geq 2^k$, we have $|x + e_x|_{2^k} = x + e_x - 2^k$, $f(x + e_x) = |x + e_x - 2^k|_p$. Following the same analysis, we can show that the number of bad errors in this class is also upper bounded by $2^k + p - 2^k \sum_{i=\lceil \frac{p}{2} \rceil}^{p-1} \frac{1}{i} - \frac{2^{k-1}}{p}$ and lower bounded by $2^k - 2^k \sum_{i=\lceil \frac{p}{2} \rceil}^{p-1} \frac{1}{i} - \frac{2^{k-1}}{p}$.

Thereby for linear arithmetic codes, an upperbound of the number of bad errors is

$$\begin{aligned} & 2 \cdot (2^k + p - 2^k \sum_{i=\lceil \frac{p}{2} \rceil}^{p-1} \frac{1}{i} - \frac{2^{k-1}}{p}) \\ = \quad & 2 \cdot (2^k + p - 2^k (H_{p-1} - H_{\lfloor \frac{p}{2} \rfloor}) - \frac{2^{k-1}}{p}). \end{aligned}$$

Similarly, a lowerbound of the number of bad errors is

$$2 \cdot (2^k - 2^k (H_{p-1} - H_{\lfloor \frac{p}{2} \rfloor}) - \frac{2^{k-1}}{p}).$$

If no errors occur to the redundant part of the code, $e_y = 0$. For the case when $x + e_x < 2^k$, a codeword x mask an error $e = (e_x, e_y = 0)$ if and only if $|e_x|_p = e_y = 0$. It is easy to prove that the number of errors in this class is upper bounded by $\lceil \frac{2^{k-1}}{p} \rceil$ and is lower bounded by $\lfloor \frac{2^{k-1}}{p} \rfloor$. Similarly, when $x + e_x \geq 2^k$, the number of bad errors in the format of $(e_x, 0)$ is also upper bounded by $\lceil \frac{2^{k-1}}{p} \rceil$ and is lower bounded by $\lfloor \frac{2^{k-1}}{p} \rfloor$. So the total number of bad errors occurring to the information part of the code is between $2 \cdot \lfloor \frac{2^{k-1}}{p} \rfloor$ and $2 \cdot \lceil \frac{2^{k-1}}{p} \rceil$. ■

Remark 3: For a more general analysis, the probability of bad errors (no matter whether $e_y = 0$) is computed as the number of bad errors divided by 2^{k+r}. The attacker capability is not taken into account during the computation. When an attacker is able to inject faults that generates errors with $e_y = 0$, the chance that the injected faults manifest as a bad error will be 2^r times larger than the probability of bad errors in the format of $e = (e_x, 0)$ given in Theorem 1. Obviously, the chance for the attacker to generate a bad error will vary depending on the fault types he can inject and the corresponding error patterns.

For linear arithmetic codes, the number of bad errors in the format of $e = (e_x, 0)$ decreases as p increases. When $p > 2^{k-1}$, there are nearly no bad errors in the format of $e = (e_x, 0)$. However, the total number of bad errors is still very large for linear arithmetic codes.

In general, the hardware overhead for the encoder of the code is mostly affected by the number of redundant bits $r = \lceil \log_2(p) \rceil$. The smallest fraction of bad errors for linear arithmetic code is of the order of 2^{-r}. The only way to reduce the fraction is to increase the number of redundant bits, which is costly in terms of the hardware overhead. To reduce the number of bad errors while maintaining the number of redundant bits, partially robust codes based on nonlinear functions were proposed in [10].

Construction 1: **[10]** Let $x \in GF(2^k)$, p be a prime number larger than 2 and $r = \lceil log_2 p \rceil$. Denote by $|\cdot|_p$ the modulo p reduction operation. The arithmetic code C composed of all vectors $(x, |x^2|_p)$, in which $|x^2|_p \in GF(2^r)$, is a partially robust arithmetic code.

Partially robust $(x, |x^2|_p)$ codes have nearly no bad errors and can provide better protection of cryptographic devices than linear arithmetic codes assuming a slow fault-injection mechanism [10]. However, $(x, |x^2|_p)$ codes rely on nonlinear squaring operations and have larger overhead than linear arithmetic codes. Moreover, $(x, |x^2|_p)$ codes have worse detection capabilities of errors in the format of $e = (e_x, 0)$ (Section V-B3).

We next propose two constructions of multilinear arithmetic codes based on the idea of randomly selecting among multiple linear arithmetic codes for each encoding and the corresponding decoding operation. For each multiplication, one randomly selected code is used to encode the message before the multiplication and decode the possibly distorted codeword after the multiplication. For different multiplications, different codes may be used. Intuitively, when we randomly select among multiple codes, even if an error is missed by one of the codes, it may still be detected by other codes. Suppose we randomly select among t codes with equal probabilities. Let $p_i(e), 1 \leq i \leq t$ be the probability that an error e is masked by the ith code. It is easy to show that the average probability that the error e is masked when we randomly selecting among these t codes can be computed as

$$p(e) = \sum_{i=1}^{t} p_i(e)/t \tag{24}$$

With different error detecting properties, the t codes will have different distribution of error masking probabilities

$p_i(e), 1 <= i <= t$. When randomly selecting among them, even if some $p_i(e)$ are larger than 0.5 (For single arithmetic codes, the error is bad.), it is highly probable that the average error masking probability $p(e)$ will still be smaller than 0.5 due to the fact that other $p_i(e)$ are very small. Specifically, when we randomly selecting from two codes, the only possible bad errors are errors masked by both of the codes or errors masked by one of the codes with probability one. Obviously, this constrain will drastically reduce the number of bad errors.

The proposed multilinear codes have similar number of bad errors to $(x, |x^2|_p)$ partially robust codes. One construction will result in a hardware overhead close to architectures based on linear arithmetic codes. The other construction will have much better error detection capabilities of errors in the format of $e = (e_x, 0)$ than linear and partially robust arithmetic codes.

Theorem 2: ($[|\mathbf{x}|_\mathbf{p}, |\mathbf{2x}|_\mathbf{p}]$ **Multilinear Code**) Let C_1, C_2 be two arithmetic systematic codes defined by

$$C_i = \{(x, y)|x \in Z_{2^k}, y = f_i(x) \in Z_{2^r}\}, i \in \{1, 2\},$$

where $f_1(x) = |x|_p$, $f_2(x) = |2x|_p$, p is an integer larger than 2 and $|\cdot|_p$ is the modulo p operation. Denote by $e = (e_x, e_y)$ the arithmetic errors and $\tilde{c} = c + e = (|x + e_x|_{2^k}, |y + e_y|_{2^r})$ the distorted codeword, where $e_x \in Z_{2^k}, e_y \in Z_{2^r}$ and $r = \lceil \log_2 p \rceil$ is the number of redundant bits. If we randomly select C_1 and C_2 to encode the original messages with equal probability, the total number of bad errors is upper bounded by $2 \cdot \lceil \frac{2^{k-1}}{p} \rceil$ and is lower bounded by $2 \cdot \lfloor \frac{2^{k-1}}{p} \rfloor$. As $\frac{p}{2^k} \to 0$, the probability of bad errors for $[|x|_p, |2x|_p]$ multilinear codes converges to 2^{-2r}.

Proof: A non-zero error e is masked by a linear arithmetic code when one of the four cases shown in Table II is satisfied. (Please refer to the proof of Theorem 1 for more details.) When we randomly select C_1 and C_2 with equal probability, an error $e = (e_x, e_y)$ is bad if only it is masked by both of the codes or it is masked by one code with probability 1. More specifically, e is bad if and only if the total number of codewords in C_1 and C_2 that mask e is larger or equal to 2^k.

1) For a given error e, when C_1 is in Case1 (i.e. $x + e_x < 2^k, f_1(x) + e_y < p, f_1(e_x) = e_y$) or Case2 and C_2 is in Case3 or Case4, the total number of codewords masking the error e is less than 2^k. So there are no bad errors in this class. Similarly, when C_1 is in Case3 or Case4 and C_2 is in Case1 or Case2, there are no bad errors.
2) C_1 is in Case1 and C_2 is in Case2. For C_1, $|x|_p + e_y < p$, the possible number of $|x|_p$ is $p - e_y$. For C_2, $|2x|_p + e_y \geq 2^r$, the possible number of $|x|_p$ is $p - 2^r + e_y$. For each possible value of $|x|_p$, the number of x is $\lceil \frac{2^k - e_x}{p} \rceil$. It is easy to prove that the total number of x masking the error is less than 2^k. So there are no bad errors in this class. Similarly we can prove that for the following three cases there are also no bad errors.
   a) C_1 is in Case2, C_2 is in Case1;
   b) C_1 is in Case3, C_2 is in Case4;
   c) C_1 is in Case4, C_2 is in Case3.
3) When C_1 and C_2 both belong to Case2, $x + e_x < 2^k$, we have $|x + e_x|_{2^k} = x + e_x$, $|x|_p + e_y \geq 2^r$ and $|2x|_p + e_y \geq 2^r$. In this case $||x|_p + e_y|_{2^r} = |x|_p + e_y - 2^r$, $||2x|_p + e_y|_{2^r} = |2x|_p + e_y - 2^r$. For C_1, an error (e_x, e_y) is missed if and only if

$$|x + e_x|_p = |x|_p + e_y - 2^r \Rightarrow |e_x|_p = |e_y - 2^r|_p. \quad (25)$$

For C_2, an error (e_x, e_y) is missed if and only if

$$|2 \cdot (x + e_x)|_p = |2x|_p + e_y - 2^r \Rightarrow |2e_x|_p = |e_y - 2^r|_p. \quad (26)$$

From (25) and (26) we have $|e_x|_p = |e_y - 2^r|_p = e_y + p - 2^r = 0$. For an error to be masked by both of the codes, the following conditions must be satisfied:

$$x + e_x < 2^k, \quad (27)$$
$$|x|_p + e_y \geq 2^r, \quad (28)$$
$$|2x|_p + e_y \geq 2^r, \quad (29)$$
$$|e_x|_p = e_y + p - 2^r = 0. \quad (30)$$

From (30), $e_y = 2^r - p \Rightarrow |x|_p + e_y < 2^r, |2x|_p + e_y < 2^r$. So no errors in this case will be masked by both of the codes. Errors in this class are all non-bad errors. Similarly, when C_1 and C_2 both belong to Case4, there are no bad errors.
4) When C_1 and C_2 both belong to Case1, $x + e_x < 2^k$, we have $|x + e_x|_{2^k} = x + e_x$, $f_1(x + e_x) = |x + e_x|_p$, $f_2(x + e_x) = |2 \cdot (x + e_x)|_p$. $|x|_p + e_y < p$ and $|2x|_p + e_y < p$. In this case $||x|_p + e_y|_{2^r} = |x|_p + e_y$, $||2x|_p + e_y|_{2^r} = |2x|_p + e_y$. For C_1, an error (e_x, e_y) is missed if and only if

$$|x + e_x|_p = |x|_p + e_y \Rightarrow |e_x|_p = e_y. \quad (31)$$

For C_2, an error (e_x, e_y) is missed if and only if

$$|2 \cdot (x + e_x)|_p = |2x|_p + e_y \Rightarrow |2e_x|_p = e_y. \quad (32)$$

From (31) and (32) we have $|e_x|_p = e_y = 0$. For a codeword x to mask a given error (e_x, e_y), the following conditions must be satisfied:

$$x + e_x < 2^k, \quad (33)$$
$$|x|_p + e_y < p, \quad (34)$$
$$|2x|_p + e_y < p, \quad (35)$$
$$|e_x|_p = e_y = 0. \quad (36)$$

When (36) is satisfied, (34) and (35) are also satisfied. For each (e_x, e_y) such that $|e_x|_p = e_y = 0$, the total number of codewords in C_1 and C_2 that mask the error is $2 \cdot (2^k - e_x)$. For bad errors this number should be larger or equal to 2^k. Thus

$$2 \cdot (2^k - e_x) \geq 2^k \quad (37)$$
$$\Rightarrow \quad e_x \leq 2^{k-1} \quad (38)$$

From (36) and (38), the number of non-zero bad errors is upper bounded by $\lceil \frac{2^{k-1}}{p} \rceil$ and is lower bounded by $\lfloor \frac{2^{k-1}}{p} \rfloor$. Similarly, when C_1 and C_2 both belong to Case3, the number of bad errors is upper bounded by $\lceil \frac{2^{k-1}}{p} \rceil$ and is lower bounded by $\lfloor \frac{2^{k-1}}{p} \rfloor$.

So an upperbound of the total number of bad errors is $2 \cdot \lceil \frac{2^{k-1}}{p} \rceil$. A lowerbound of the total number of bad errors is $2 \cdot \lfloor \frac{2^{k-1}}{p} \rfloor$. ■

TABLE II: Classification of masked errors for linear arithmetic codes

Case1	Case2	Case3	Case4
$x+e_x<2^k$	$x+e_x<2^k$	$x+e_x\geq 2^k$	$x+e_x\geq 2^k$
$f_i(x)+e_y<p$	$f_i(x)+e_y\geq 2^r$	$f_i(x)+e_y<p$	$f_i(x)+e_y\geq 2^r$
$f_i(e_x)=e_y$	$f_i(e_x)=e_y+p-2^r$	$f_i(e_x-2^k)=e_y$	$f_i(e_x-2^k)=e_y+p-2^r$

The number of bad errors for $[|x|_p, |2x|_p]$ multilinear codes is much smaller than that for the linear arithmetic codes (see (4)). All bad errors are in the format of $e=(e_x,0)$. The security level of systems protected by the $[|x|_p, |2x|_p]$ codes can be further increased by implementing a merged design of the original device and the encoder generating redundant bits of the output of the protected device. In that case, the injected faults will have high probability to affect not only the original device but also the encoder that generates the redundant bits of the code. The probability of errors in the format of $e=(e_x,0)$ will be efficiently reduced. As a result, the error detection capabilities and the security level of the system will be increased.

If the original device and the encoder are separated and the attacker is able to inject faults only to he original device, $[|x|_p, |2x|_p]$ multilinear codes do not have any advantages over linear arithmetic codes in terms of the error detecting capability. In this case, the system should be protected using multi-modulus multilinear codes shown in Theorem 3 presented in the left part of the section.

A more general case of Theorem 2 is to randomly select from $L\leq p-1$ codes defined by $C_i=\{x, f_i(x)\}$ where $f_i(x)=|ix|_p, 1\leq i<p$. However, from the proof of the above theorem it is easy to show that increasing the number of codes from which we randomly select a code for encoding and decoding will not reduce the number of bad errors in this situation. We next present a construction based on using multiple modulii. The resulting codes will be different from $[|x|_p, |2x|_p]$ codes in the following two aspects.

- ♦ They have much less bad errors in the format of $e=(e_x,0)$;
- ♦ Increasing the number of codes from which we randomly select a code for encoding and decoding will further reduce the number of bad errors.

Theorem 3: (**Multi-modulus Multilinear Code**) Let C_1, C_2 be two systematic arithmetic codes defined by

$$C_i=\{(x,y)|x\in Z_{2^k}, y=f_i(x)\in Z_{2^r}\}, i\in\{1,2\},$$

in which $f_1(x)=|x|_p$, $f_2(x)=|x|_q$ where p,q are co-prime numbers larger than 2, $r=\max(\lceil\log_2 p\rceil, \lceil\log_2 q\rceil)$ and $|\cdot|_p$ is the modulo p operation. Denote by $e=(e_x\in Z_{2^k}, e_y\in Z_{2^r})$ the arithmetic additive errors and $\tilde{c}=c+e=(|x+e_x|_{2^k}, |y+e_y|_{2^r})$ the distorted codeword. If we randomly select C_i with equal probability to encode the original messages, the number of bad errors in the format of $e=(e_x,0)$ is upper bounded by

$$2(\lceil\frac{2^{k-1}}{pq}\rceil+\lceil\frac{2^k}{pq}\rceil), \tag{39}$$

and is lower bounded by

$$2(\lfloor\frac{2^{k-1}}{pq}\rfloor+\lfloor\frac{2^k}{pq}\rfloor). \tag{40}$$

When $pq << 2^k$ and q is close to p, the probability of bad errors occurring to the information part of multi-modulus multilinear codes converges to $3\cdot 2^{-3r}$.

Proof: Since $e_y=0$, we have $|x|_p+e_y<p$ and $|x|_q+e_y<q$. For each linear code, errors are masked if and only if one of the following two conditions are satisfied (see the proof of Theorem 1).

Case1 : $x+e_x<2^k, f_i(e_x)=0$.

Case2 : $x+e_x\geq 2^k, f_i(e_x-2^k)=0$.

If we randomly select C_i with equal probability, an error $e=(e_x,0)$ is masked by a probability at least 0.5 if and only if the total number of codewords belonging to C_i which mask the error is larger or equal to 2^k. For a given non-zero error $e=(e_x,0)$, there are three possible situations as stated below.

1) Both C_i belong to Case1, $|e_x|_p=|e_x|_q=0$. The total number of codewords belonging to C_i which mask the error e is $2\cdot(2^k-e_x)$. Hence the error is bad if and only if $e_x\leq 2^{k-1}$. Since $|e_x|_p=|e_x|_q=0$, the number of bad errors in this class is upper bounded by $\lceil\frac{2^{k-1}}{pq}\rceil$ and is lower bounded by $\lfloor\frac{2^{k-1}}{pq}\rfloor$ ($e_x=2^{k-1}$ does not satisfy $|e_x|_p=0$ and $|e_x|_q=0$).
2) Both C_i belong to Case2, following similar analysis we can prove that the number of bad errors in this class is upper bounded by $\lceil\frac{2^{k-1}}{pq}\rceil$ and is lower bounded by $\lfloor\frac{2^{k-1}}{pq}\rfloor$.
3) When C_i belong to different cases, it is easy to prove that as long as $e=(e_x,0)$ satisfies $|e_x|_p=0, |e_x|_q=|2^k|_q$ or $|e_x|_p=|2^k|_p, |e_x|_q=0$, the total number of codewords masking the error is always 2^k. Hence the error is always bad. The number of bad errors in this class is upper bounded by $2\lceil\frac{2^k}{pq}\rceil$ and is lower bounded by $2\lfloor\frac{2^k}{pq}\rfloor$.

■

Remark 4: When randomly selecting from two codes, experimental results show that the total number of bad errors $e=(e_x,e_y)$ for the multi-modulus codes is comparable to that of $[|x|_p, |2x|_p]$ multilinear codes and is much smaller than that of linear arithmetic codes. The idea of using multiple residues as the redundant part of the code has already been presented in [7]. With two residues, the codeword was in the format of $(x, |x|_p, |x|_q)$. We want to emphasize that our construction is different from multi-residue codes proposed in [7] since at each clock cycle our code has only one residue for the redundant part. Instead of using multiple residues simultaneously, we use only one for each encoding and decoding operation and randomly select the modulus for different operations.

Multi-modulus codes have much less bad errors in the format of $e=(e_x,0)$ than linear and $[|x|_p, |2x|_p]$ multilinear arithmetic codes (see (39)). Table III shows the estimated number of bad errors in this class for all three constructions.

TABLE III: Number of bad errors in the format of $e = (e_x, 0)$ for linear and multilinear codes ($k = 32$)

	$p = 5$	$p = 241$	$p = 563$	$p = 883$	$p = 1237$	$p = 2767$
LinearArithmetic	8.6×10^8	1.8×10^7	7.6×10^6	4.9×10^6	3.5×10^6	1.6×10^6
$[\|x\|_p, \|2x\|_p]$ multilinear codes	8.6×10^8	1.8×10^7	7.6×10^6	4.9×10^6	3.5×10^6	1.6×10^6
Multi-modulus codes ($L = 2$)	8.6×10^8	2.2×10^5	4.1×10^4	1.7×10^4	8.5×10^3	1.7×10^3

The number of information bits of the codes in the Table is $k = 32$. For multi-modulus codes, p corresponds to the larger modulus. The other modulus is selected to be the largest possible prime number less than p, e.g. when $p = 241$, the other modulus is 239. As p increases, the number of bad errors for multi-modulus codes decreases much faster than for the other two alternatives. When $p = 2767 (r = 12)$, the multi-modulus code has only 1.7×10^3 bad errors in the format of $(e_x, 0)$ while the other two codes have about 1.6×10^6.

These characteristics of multi-modulus codes are beneficial in many different situations. For example, when the attacker can identify and inject faults only to the original device, or the encoder of the code is only a small part of the cryptographic system and is separated from the original device so that most of the injected single (or even double) faults affect only the original device, the generate errors will be in the format of $e = (e_x, 0)$. In this case, systems protected by multi-modulus codes will have a higher security level than architectures based on other alternatives. Systems with different error rates for the original device and the predictor can also benefit from this characteristic of multi-modulus codes. Recently design based on multi-voltage regions is proposed to reduce the total power consumption of the system [26]. In the region with the smaller voltage level, circuits are more vulnerable to soft errors and are more probable to have errors caused by problems such as timing violations [27]. As a result, the error rate for circuits in this region will be higher. If the original device operates at a lower voltage level than the predictor, multi-modulus codes can provide better protections due to the fact that they have higher detection capabilities of errors in the format of $e = (e_x, 0)$.

Different from $[|x|_p, |2x|_p]$ codes, for multi-modulus codes increasing the number of codes from which we randomly select a code for encoding and decoding can further reduce the total number of bad errors. Table IV shows the simulation results for a 8-bit multipliers protected by multi-modulus codes with different number of modulii. The second line corresponds to the case when a single linear arithmetic code is used. When we randomly select from multiple linear arithmetic codes with four different modulusi, the number of bad errors in the format of $e = (e_x, 0)$ is only 13, which is more than 100 times better than architectures based on linear arithmetic codes.

Remark 5: From Table IV, when we randomly select from two linear arithmetic codes with different modulii, the number of bad errors in the format of $e = (e_x, 0)$ is a little bit larger than the result given by (39). This is because when using arithmetic codes to protect multipliers, the output of the multiplier, hence the information bits of the arithmetic codes, is not uniformly distributed. Moreover, some combinations of information bits in $Z_{2^{2k}}$ may never occur at the output of a k-bit multiplier. However, simulation results show that in this situation multilinear codes still largely over-perform linear arithmetic codes and all the advantages of multilinear codes are preserved.

Table V summarizes the probability of bad errors of linear and multilinear arithmetic codes. In the next section we will present secure multiplier architectures based on $[|x|_p, |2x|_p]$ and multi-modulus codes

V. Secure Multipliers Based on Linear, Multilinear and Partially Robust Arithmetic Codes

The multiplier is a basic block in many public key cryptographic devices. Due to its arithmetic nature of the operations, arithmetic error model is most often used for such devices. We assume that faults manifest as additive arithmetic errors at the output of the multiplier and the predictor [1]. The error is in the format of $e = (e_x, e_y), e_x \in Z_{2^k}, e_y \in Z_{2^r}$, where k is the number of information bits and r is the number of redundant bits. In this section, we analyze and compare the hardware overhead, the number of bad errors and the fault detection capabilities for architectures protected by linear, multilinear (Section IV) and robust arithmetic codes [10].

A. Hardware Overhead

The general architecture of multipliers protected by block codes contains three parts: the original multiplier, the predictor that generates the redundant bits of the code and the error detection network (EDN). The detailed architectures for secure multipliers protected by linear and multilinear arithmetic codes are shown in Figure 2. For the architecture based on $(x, |x^2|_p)$ partially robust arithmetic codes, please refer to [10].

The predictor for the linear arithmetic codes contains one multiplier in Z_p. Except for the r-bit comparator, the only operation implemented in the error detection network is a modulo p operation. The hardware overhead mainly comes from the $r = \lceil \log_2(p) \rceil$ bit modulo p multiplier, whose complexity is of the order of $O(r^2)$, and the modulo p operation in EDN, whose complexity is $O(k)$. (k is the number of information bits).

Compared with architectures based on linear arithmetic codes, the architecture utilizing $[|x|_p, |2x|_p]$ multilinear codes only needs one extra r-bit multiplexer and one extra multiply-by-2 operation in Z_p for both the predictor and the EDN. Multiply-by-2 operation is equal to shifting the operands by 1 bit, which is trivial in terms of the hardware overhead. The complexity of a r-bit multiplexer is in general of the order

[1]The term *predictor* is used in this context to refer to the circuit that computes the redundant bits of the output of the operation directly from the inputs. In our case the predictor computes the redundant bits of the multiplication result.

TABLE IV: Number of bad errors when selecting from linear arithmetic codes with different modulii

Modulii	Bad Errors	Bad Errors in the Format of $e = (e_x, 0)$
$p_1 = 31$	47857	2113
$p_1 = 31, p_2 = 29$	1781	249
$p_1 = 31, p_2 = 29, p_3 = 23$	1651	180
$p_1 = 31, p_2 = 29, p_3 = 23, p_4 = 19$	133	13

TABLE V: Probability of bad errors for linear and multilinear codes

Probability of	Linear Arithmetic Codes	$[\lvert x\rvert_p, \lvert 2x\rvert_p]$ ML codes	Multi-modulus ML codes
Bad errors	$\approx 0.3 \cdot 2^{-r+1}$	$\approx 2^{-2r}$	$\approx 2^{-2r*}$
Bad errors $e = (e_x, 0)$	$\approx 2^{-2r}$	$\approx 2^{-2r}$	$\approx 3 \cdot 2^{-3r}$

* : Based on experimental results.

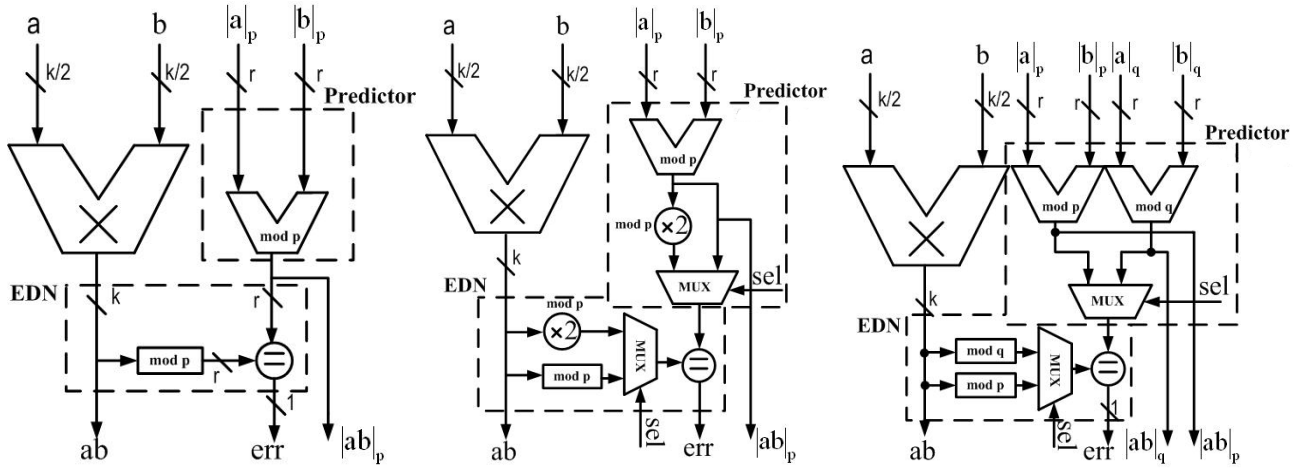

Fig. 2: Hardware architectures for multipliers protected by (a) linear arithmetic codes, (b) $[|x|_p, |2x|_p]$ multilinear codes and (c) Multi-modulus multilinear codes

of $O(r)$. Thereby this architecture has comparable hardware overhead to linear arithmetic codes.

The protection architecture based on multi-modulus multilinear codes needs one more multiplier in Z_q for the predictor. When $p << 2^k$, which is often the case in real life, q should be selected as the largest prime number that is smaller than p if we want to minimize the number of bad errors. A multiplier in Z_q will have about the same hardware complexity as the multiplier in Z_p and this will double the overhead for the predictor. However, we claim that a merged design of the two multipliers for the predictor should be implemented. First, from the security point of view, separate redundant data path may be used by attackers to derive the secret information of the devices, e.g. the attacker can inject faults into one redundant path of the device which will never influence the other. A merged design can effectively solve the problem because most of the faults injected into the redundant part of the device will affect the generation of redundant bits for both codes. Second, the hardware overhead of the predictor will be reduced if we merge the design of the two multipliers. A more aggressive approach is to design the original multiplier and the predictor of the code together as discussed in Section IV.

Remark 6: There is a tradeoff between the error detection capabilities and the hardware overhead when we select p and q. Specialized p and q can significantly reduce the hardware complexity of the modulo operation, e.g. using Mersenne primes.

To compare the hardware area overhead, we modeled 16-bit Wallace tree multipliers protected by different alternatives in Verilog and synthesized them in RTL design compiler using Nangate 45nm technology. The area comparison is based on synthesized results. We expect similar results after placing and routing to the first order. The results are shown in Table VI. We select p to be 31. For multi-modulus multilinear codes, q is selected to be 29. The percentage overhead is computed by dividing the estimated gate area of the predictor and EDN by the estimated area of the Wallace tree multiplier. As expected, secure multipliers based on $[|x|_p, |2x|_p]$ codes have similar overhead to architectures based on linear arithmetic codes. Architectures based on multi-modulus codes require the largest overhead, which is around 50%. The benefit of these codes is that they have the best error detection capabilities against errors in the format of $e = (e_x, 0)$. In Section V-B3, we will show that this characteristics of multi-modulus codes will make them the best alternative against fault-injection attacks when the design of the predictor is separated from the original multiplier. Moreover, the hardware overhead of architectures based on multi-modulus codes will be drastically reduced if

TABLE VI: Hardware area overhead for architectures based on linear, multilinear and partially robust arithmetic codes ($k = 32, r = 5, p = 31, q = 29$)

Code	Predictor	EDN	Total
Linear Arithmetic Codes	9.76%	10.68%	20.44%
$[\|x\|_p, \|2x\|_p]$ Multilinear Codes	10.37%	11.75%	22.12%
Multi-modulus Multilinear Codes	16.14%	37.57%	53.71%
$(x, \|x^2\|_p)$ Partially Robust Arithmetic Codes	14.60%	14.71%	29.31%

we select q to be a Mersenne prime. In fact, 50% overhead is still much smaller than overheads for architectures based on robust arithmetic codes, which is around $200\% - 400\%$ [9], [10].

B. Experimental Results on Comparison of Error and Fault Detection Capabilities for Linear, Partially Robust and Multilinear Arithmetic Codes

To demonstrate the advantages of multilinear codes and partially robust codes over linear codes for building secure multipliers against fault-injection attacks, we conduct simulations to analyze and compare the number of bad errors and the fault detection capabilities of the four alternatives presented in the last section. For all the simulations, we assume the operands of the multipliers are 16 bits. Each code has 32 information bits ($k = 32$). p and q are selected to be 31 and 29 respectively.

1) Number of Bad Errors: In this simulation, we randomly generate 5000 non-zero errors $e = (e_x, e_y)$. For each e, we randomly select one million messages in $Z_{2^{32}}$ and encode them using linear, $[|x|_p, |2x|_p]$, multi-modulus and $(x, |x^2|_p)$ partially robust arithmetic codes. The distorted codewords $\tilde{c} = (\tilde{x}, \tilde{y}) = (|x + e_x|_{2k}, |y + e_y|_{2^r})$ are decoded by the error detection network. The number of codewords masking each error is recorded. The distribution of error masking probabilities of the 5000 non-zero errors is shown in Table VII. Most of the errors are masked by a probability of less than 10% for all the alternatives. Linear arithmetic codes have 149 bad errors which are masked by a probability of at least 0.5. The numbers of bad errors for $[|x|_p, |2x|_p]$, multi-modulus and $(x, |x^2|_p)$ codes are similar and are much smaller than that of the linear arithmetic codes, which can result in better fault detection capabilities assuming a slow fault-injection mechanism (fault stays for several consecutive clock cycles). Compared to multilinear codes, $(x, |x^2|_p)$ codes have much less errors that are masked by a probability of more than 10%. However, we will show later in this section that $(x, |x^2|_p)$ codes actually have the worst error detection capabilities of errors in the format of $e = (e_x, 0)$ and is only suitable for designs where the multiplier and the predictor are synthesized together. Moreover, $(x, |x^2|_p)$ codes have larger overhead than $[|x|_p, |2x|_p]$ multilinear codes. The disadvantage of overhead for $(x, |x^2|_p)$ will become more significant as k increases.

2) Fault Detection Capabilities When Both the Multiplier and the Predictor are Affected: Suppose both the original multiplier and the predictor are affected by the injected faults, which manifest as a non-zero error $e = (e_x, e_y)$ at the output of the device. Assume that each multiplication is completed in one clock cycle and e stays for T consecutive clock cycles (slow fault-injection mechanism). If e is detected at least once among the T clock cycles, we say that e is detected. Otherwise e is masked. In this simulation, we randomly select 10 millions possible error patterns e and assume that e may stay up to 3 clock cycles. The average error masking probabilities of e for the four presented alternatives are shown in Table VIII. All codes have similar error detection capabilities when e stays for only one clock cycle. However, when $T = 2$, the error masking probabilities of $[|x|_p, |2x|_p]$ and multi-modulus codes are already nearly half of that of linear arithmetic codes. As T increases, the advantage of $[|x|_p, |2x|_p]$ and multi-modulus codes become more significant. As expected, when both the original multiplier and the predictor are affected by the injected faults, $[x, |x^2|_p]$ codes have the best error and fault detection capabilities among the four alternatives.

TABLE VIII: Fault masking probabilties when both the original multiplier and the predictor are affected ($k = 32, r = 5, p = 31, q = 29$)

T	Linear	$[\|x\|_p, \|2x\|_p]$	Multi-modulus	$(x, \|x^2\|_p)$
T=1	3.12%	3.12%	3.12%	3.12%
T=2	1.81%	0.95%	0.95%	0.10%
T=3	1.25%	0.38%	0.35%	0.003%

* T is the number of clock cycles that a fault stays.

3) Fault-Injection Simulations For the Case When Only the Multiplier is Affected: Suppose the design of the multiplier and the predictor is separated and the attacker manages to inject faults only to the original multiplier. In order to analyze the fault detection capabilities, we conduct gate-level fault-injection simulations in C++ on 16-bit secure Wallace tree multipliers protected by different alternatives. The gate level netlist is derived from Verilog models. Each gate may have stuck-at-0 or stuck-at-1 faults. We assume that 2 to 4 gates ($2 \leq N \leq 4$) may be affected by the injected faults and the faults stay for up to 3 consecutive clock cycles ($T \leq 3$). At each clock cycle, a new pair of operands are randomly generated and multiplied. If the manifested error is detected for at least one clock cycle, we say that the fault is detected.

Table IX summarizes the fault masking probabilities for all combinations of N and T. When a certain number of gates are affected (N is fixed), larger T will result in smaller fault masking probabilities. When $T = 1$, the average fault masking probabilities increase as N increases. However, when $T > 2$, the average fault masking probabilities will drop as N increases. This is because for larger N, errors are more probable to manifest as different non-zero errors at the output of the device. For smaller N, it is more likely that even if

TABLE VII: Error masking probabiliy distributions for secure multipliers based on linear, multilinear and partially robust arithmetic codes ($k = 32, r = 5, p = 31, q = 29$)

Code	$< 10\%$	$10\% - 20\%$	$20\% - 30\%$	$30\% - 40\%$	$40\% - 50\%$	$\geq 50\%$ (Bad Errors)
linear	4656	59	51	46	39	149(29.8%)
$[\|x\|_p, \|2x\|_p]$	4492	182	142	107	74	3(0.06%)
Multi-modulus	4444	196	140	118	97	5 (0.1%)
$(x, \|x^2\|_p)$	4996	0	0	0	0	4(0.08%)

the fault stays for several consecutive clock cycles, it only manifests in one clock cycle. In this case, the fault detection capabilities will not increase.

Linear arithmetic codes and $[|x|_p, |2x]_p$ codes have the same error detection capabilities for errors in the format of $e = (e_x, 0)$. The reason is when $e_y = 0$, the error masking equations for $(x, |x|_p)$ and $(x, |2x|_p)$ codes are $|e_x|_p = 0, |2e_x|_p = 0$ and $|e_x - 2^k|_p = 0, |2(e_x - 2^k)| = 0$ depending on the ranges of $x+e_x$. Obviously, $|e_x|_p = 0$ is equivalently to $|2e_x|_p = 0$ and $|e_x - 2^k|_p = 0$ is equivalent to $|2(e_x - 2^k)| = 0$. Thereby, $e = (e_x, 0)$ is masked by $[|x|_p, |2x]_p$ codes if and only if it is masked by linear arithmetic codes.

$(x, |x^2|_p)$ codes have the worst error detection capabilities of errors in the format of $e = (e_x, 0)$ among the four alternatives. When $e_y = 0$, the error masking equation for $(x, |x^2|_p)$ is $|2e_x x + e_x^2|_p = 0$ and $|2(e_x - 2^k) + (e_x - 2^k)^2|_p = 0$ for different ranges of $x+e_x$. When $|e_x| = 0$ or $|e_x - 2^k|_p = 0$, $|2e_x x + e_x^2|_p = 0$ or $|2(e_x - 2^k) + (e_x - 2^k)^2|_p = 0$ is always true. But the inverse statement is incorrect. Thereby, $(x, |x^2|_p)$ will mask more errors in the format of $e = (e_x, 0)$ than linear and $[|x|_p, |2x]_p$ arithmetic codes.

When only the original multiplier is affected, multi-modulus codes have the best error detection capabilities. When $T = 2$, the fault masking probabilities of multi-modulus codes are nearly half of the fault masking probabilities of linear and $[|x|_p, |2x]_p$ arithmetic codes with the same N. The advantage of multi-modulus codes becomes larger as T increases.

4) Selection of Arithmetic Codes for Secure Multipliers: From the above analysis, linear arithmetic codes have a lot of bad errors – errors masked with a probability of at least 0.5 – which may compromise the security level of the system. $[|x|_p, |2x|_p]$, multi-modulus and $(x, |x^2|_p)$ codes have much less bad errors than linear arithmetic codes (Table V). $[|x|_p, |2x|_p]$ and $(x, |x^2|_p)$ are more suitable for designs where the original multipliers and the predictors are synthesized together. $(x, |x^2|_p)$ have better fault and error detection capabilities while $[|x|_p, |2x|_p]$ require less hardware overhead. The selection of these two codes depends on specific applications. When the designs of the multiplier and the predictor are separated and only the multiplier is affected by the injected faults, $[|x|_p, |2x|_p]$ and $(x, |x^2|_p)$ are no better than linear arithmetic codes. In this case, we should select multi-modulus codes which have the best detection capabilities against errors in the format of $e = (e_x, 0)$.

VI. Conclusions

In this paper we propose to use multilinear codes to protect cryptographic devices against strong fault-injection attacks. Several constructions of multilinear arithmetic codes are presented. The hardware overhead and the error and fault detection capabilities of secure multipliers based on multilinear codes are analyzed and compared to those based on linear and partially robust arithmetic codes. Simulation results show that multilinear and partially robust arithmetic codes have much smaller number of bad errors (errors masked by a probability of at least 0.5) and can provide better protection than linear arithmetic codes assuming a slow fault-injection mechanism. $[|x|_p, |2x|_p]$ codes have similar overhead to linear arithmetic codes with the same number of redundant bits. $(x, |x^2|_p)$ and multi-modulus codes have slightly higher overhead than linear arithmetic codes. But the overhead is at most around 50% and is much smaller than the overhead of architectures based on robust arithmetic codes, which is around $200\% - 400\%$.

If the designs of the predictor and the original multiplier are separated and the injected faults affect only the multiplier, multi-modulus code is the best alternative. In this case, the fault masking probability of architectures based on multi-modulus codes is almost twice smaller than architectures based on the other codes when the fault stays for only one clock cycle. The advantage of multi-modulus codes will become even more significant as the fault stays longer.

If the faults affect both the multiplier and the predictor, $(x, |x^2|_p)$ codes have the best fault detection capabilities. $[|x|_p, |2x|_p]$ code has similar performance to multi-modulus codes and require the least hardware overhead among multilinear and partially robust arithmetic codes. The selection of codes depends on specific applications. Multi-modulus codes can be generalized to the case of randomly selecting from L codes $C_i = \{(x, |x|_{p_i}), 1 \leq i \leq L\}$, where p_i are different prime numbers larger than 2. The number of bad errors will be further reduced as L increases.

Acknowledgment

We would like to thank Prasanna Rao from Boston University and Professor Berk Sunar from Worcester Polytechnic Institute for their contributions and valuable advices for the paper.

References

[1] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Lecture Notes in Computer Science*, 1996.

[2] P. Kocherand, J. Jaffe, and B. Jun, "Differential power analysis," in *Lecture Notes in Computer Science*, 1999.

[3] S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *Revised Papers from the 4th International Workshop on Cryptographic Hardware and*

TABLE IX: Fault masking probabilties when only the original device is affected ($k = 32, r = 5, p = 31, q = 29$)

	Linear			$[\|x\|_p, \|2x\|_p]$			Multi-modulus			$(x, \|x^2\|_p)$		
T	N=2	N=3	N=4	N=2	N=3	N=4	N=2	N=3	N=4	N=2	N=3	N=4
1	2.7%	3.5%	3.5%	2.7%	3.5%	3.5%	1.6%	2.4%	2.8%	5.7%	6.5%	6.5%
2	1.0%	0.78%	0.47%	1.0%	0.78%	0.47%	0.5%	0.43%	0.29%	2.1%	1.5%	1.0%
3	0.34%	0.16%	0.06%	0.34%	0.16	% 0.06%	0.14%	0.07%	0.03%	0.71%	0.31%	0.14%

* T is the number of clock cycles that a fault stays. N is the number of gates that a fault affects.

Embedded Systems, (London, UK), pp. 2–12, Springer-Verlag, 2003.

[4] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerers apprentice guide to fault attacks," 2002.

[5] D. Boneh, R. A. Demillo, and R. J. Lipton, "On the importance of eliminating errors in cryptographic computations," in *Journal of Cryptology*, 2001.

[6] G. Piret and J.-J. Quisquater, "A differential fault attack technique against spn structures, with application to the AES and KHAZAD," in *Cryptographic Hardware and Embedded Systems Workshop (CHES)*, 2003.

[7] T. Rao and O. Garcia, "Cyclic and multiresidue codes for arithmetic operations," *IEEE Transactions on Information Theory*, vol. IT-17, no. 1, 1971.

[8] M. Karpovsky., K. Kulikowski, and A.Taubin, "Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard," Proc. Int. Conference on Dependable Systems and Networks (DSN), July 2004.

[9] G. Gaubatz, B. Sunar, and M. G. Karpovsky, "Non-linear residue codes for robust public-key arithmetic," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2006.

[10] K.Kulikowski, Z.Wang, and M.G.Karpovsky, "Comparative analysis of fault attack resistant architectures for private and public key cryptosystems," in *Proc of Int. Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2008.

[11] I. Vasyltsov, E. Hambardzumyan, Y.-S. Kim, and B. Karpinskyy, "Fast digital trng based on metastable ring oscillator," in *In Proceedings of Cryptographic Hardware and Embedded Systems Workshop (CHES)*, 2008.

[12] Z. Wang, M. Karpovsky, B. Sunar, and A. Joshi, "Design of reliable and secure multipliers by multilinear arithmetic codes," in *Information and Communications Security*, vol. 5927 of *Lecture Notes in Computer Science*, pp. 47–62, 2009.

[13] Z. Wang, M. Karpovsky, and B. Sunar, "Multilinear codes for robust error detection," in *IEEE International On-Line Testing Symposium (IOLTS)*, 2009.

[14] Z. Wang and M. Karpovsky, "Robust FSMs for cryptographic devices resilient to strong fault injection attacks," in *16th IEEE International On-Line Testing Symposium (IOLTS'2010)*.

[15] G. Canivet, P. Maistri, R. Leveugle, J. Cldire, F. Valette, and M. Renaudin, "Glitch and laser fault attacks onto a secure aes implementation on a sram-based fpga," *Journal of Cryptology*, pp. 1–22, 2010. 10.1007/s00145-010-9083-9.

[16] J.-M. Schmidt and C. Herbst, "A practical fault attack on square and multiply," in *Proceedings of the 2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, (Washington, DC, USA), pp. 53–58, IEEE Computer Society, 2008.

[17] C. Kim and J.-J. Quisquater, "Fault attacks for crt based rsa: New attacks, new results, and new countermeasures," in *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems* (D. Sauveron, K. Markantonakis, A. Bilas, and J.-J. Quisquater, eds.), vol. 4462 of *Lecture Notes in Computer Science*, pp. 215–228, Springer Berlin / Heidelberg, 2007.

[18] A. Barenghi, G. Bertoni, E. Parrinello, and G. Pelosi, "Low voltage fault attacks on the rsa cryptosystem," *Fault Diagnosis and Tolerance in Cryptography, Workshop on*, vol. 0, pp. 23–31, 2009.

[19] Y. Monnet, M. Renaudin, R. Leveugle, C. Clavier, and P. Moitrel, "Case study of a fault attack on asynchronous des crypto-processors," in *Fault Diagnosis and Tolerance in Cryptography* (L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, eds.), vol. 4236 of *Lecture Notes in Computer Science*, pp. 88–97, Springer Berlin / Heidelberg, 2006.

[20] J. M. Schmidt and M. Hutter, "Optical and em fault-attackson crt-based rsa: Concrete results," in *15th Austrian Workshop on Microelectronics*, 2007.

[21] E. Trichina and R. Korkikyan, "Multi fault laser attacks on protected crt-rsa," *Fault Diagnosis and Tolerance in Cryptography, Workshop on*, vol. 0, pp. 75–86, 2010.

[22] S. Skorobogatov, "Optical fault masking attacks," 2010.

[23] D. Samyde, S. Skorobogatov, R. Anderson, and J. jacques Quisquater, "On a new way to read data from memory," in *SISW2002 First International IEEE Security in Storage Workshop*, 2002.

[24] T. Malkin, F.-X. Standaert, and M. Yung, "A comparative cost/security analysis of fault attack countermeasures," in *Fault Diagnosis and Tolerance in Cryptography* (L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, eds.), vol. 4236 of *Lecture Notes in Computer Science*, pp. 159–172, Springer Berlin / Heidelberg, 2006.

[25] C. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, 1964.

[26] S. Khursheed, B. Al-Hashimi, S. Reddy, and P. Harrod, "Diagnosis of multiple-voltage design with bridge defect," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, pp. 406 –416, mar. 2009.

[27] D. Roberts, T. Austin, D. Blauww, T. Mudge, and

K. Flautner, "Error analysis for the support of robust voltage scaling," pp. 65 – 70, mar. 2005.