Design of Reliable and Secure Multipliers by Multilinear Arithmetic Codes

Zhen Wang[1], Mark Karpovsky[1], Berk Sunar[2], and Ajay Joshi[1]

[1] Boston University, Reliable Computing Laboratory, 8 Saint Marys Street, Boston, MA, USA
{lark,markkar,joshi}@bu.edu,
[2] Worcester Polytechnic Institute, CRIS Laboratory, 100 Institute Road, Worcester, MA 01609
sunar@wpi.edu

Abstract. We propose an efficient technique for the detection of errors in cryptographic circuits introduced by strong adversaries. Previously a number of linear and nonlinear error detection schemes 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 Linear codes provide protection only against primitive adversaries which no longer represents practice.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 (200–300%). Here we propose a novel error detection technique, based on the random selection of linear arithmetic codes. Under mild assumptions the proposed construction achieves near nonlinear code error detection performance at a lower cost (about 50% area overhead) due to the fact that no nonlinear operations are needed for the encoder and decoder.

1 Introduction

Cryptographic 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 architecture 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 types of errors linear codes can provide little protection. Linear 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 architectures 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 which can be used to design 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. Variants of both algebraic and arithmetic robust codes – partially robust and minimum distance robust codes – were proposed in [10]. These architectures allow various tradeoffs in terms of robustness and hardware overhead.

Robust codes are based on nonlinear functions [11] and the robustness of the code is highly related to the nonlinearity of the function. Systematic robust codes, for example, can be constructed by appending a signature generated by a nonlinear function f to the information part of the code. The worst case error masking probability of any nonzero error is bounded by P_f, which gives a measure of the degree of the nonlinearity of f.

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 different method to achieve similar levels of protection. Instead of using nonlinear functions to generate the signature of the code, we randomly select a code from multiple linear codes at each clock cycle. The resulting codes are called **multilinear codes**. The proposed method can have as small number of undetectable errors as classical robust codes while requiring much less hardware overhead due to the fact that no nonlinear operations are needed for the encoder and decoder.

Multipliers are widely used as sub-blocks in public key cryptosystems. In this paper we present constructions of multilinear arithmetic codes and their use to design reliable multipliers. We compare the proposed protection architecture for multipliers with that based on single linear arithmetic code. 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. [12]) 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 capabilities of different architectures are simulated in MATLAB and the advantage of the proposed technique is demonstrated.

The constructions of multilinear algebraic codes and the analysis of fault detection capabilities of architectures based on multilinear algebraic codes were discussed in [13].

The paper is organized as follows. Section 2 describes the error and attacker models we use throughout the paper. In section 3 we formalize the design and propose several constructions based on randomly selecting multiple linear codes. In Section 4 we compare different protection architectures for a fixed precision multiplier. We finish the paper by drawing the conclusions in Section 5.

2 Error and Attacker Model

In this paper we concentrate on the analysis of the error detection abilities for systematic arithmetic codes and the reliability of multipliers based on these codes. Different from the widely used nonsystematic AN codes [7], the codewords of systematic arithmetic

codes contain two parts: the information part and the redundancy 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 $||_p$ 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:

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

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 usually do not have undetectable errors. To illustrate the advantage of the proposed codes, we compare the number of **bad errors**, which are 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.

Throughout the paper we assume a strong attacker model in which an attacker knows everything about the hardware architecture of the device including the code used to detect errors. The attacker can utilize any fault injection methodologies and is able to inject faults with high spatial resolution to generate a specific error vector (e_x, e_y) at the output of the devices. The only limitation on the attacker is that he cannot change the error at each clock cycle. Once faults are injected and an error is generated, the faults stay for several clock cycles and the error tends to repeat. This is the case for several well known fault injection methodologies such as introducing power glitches into the power supply, using laser guns, etc [4]. 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**.

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 it has much smaller number of bad errors. Second, multilinear arithmetic codes have much higher error detection abilities than linear codes in lazy channels hence they will effectively prevent the attacker from implementing a successful fault induction attack under the aforementioned attacker model. To facilitate the analysis and comparison of the error protection architectures based on different codes in lazy channels, we assume that errors last for at least t consecutive clock cycles, $t \geq 1$.

The experimental results for the error detection properties of multipliers protected by different codes are presented in Section 4, which shows that as t increases the error detection probabilities are much higher for architectures based on multilinear arithmetic codes than that based on single linear arithmetic code.

3 Constructions

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*

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

where p is a prime number larger than 2 and $f(x) = |x|_p$ where $||_p$ represents the modulo p reduction operation. Denote by $e = (e_x, e_y)$ an additive error, $e_x \in Z_{2^k}, e_y \in Z_{2^r}$, $r = \lceil \log_2 p \rceil$, $c + e = (|x + e_x|_{2^k}, |y + e_y|_{2^r})$, $c \in C$. When $p << 2^k$, the number of bad errors is upper bounded by

$$2 \cdot (2^k + p - 2^k(H_{p-1} - H_{\lfloor \frac{p}{2} \rfloor}) - \frac{2^{k-1}}{p}). \tag{3}$$

In the equation H_n represents the n-th harmonic number. For large p the difference $H_{p-1} - H_{\lfloor \frac{p}{2} \rfloor}$ converges to $\ln 2$. Thus, for large p, the number of bad errors is upper bounded by $2p - 2^{k+1}(\ln 2 - 1) - 2^k/p$.

If no errors occur to the redundant part of the code, the number of bad errors occurring only to the information part of the code is upper bounded by

$$2 \cdot \lceil \frac{2^{k-1}}{p} \rceil. \tag{4}$$

When p is large and $p << 2^k$, the estimated probability of bad errors is $0.3 \cdot 2^{-r+1}$. The estimated probability of bad errors occurring to the information part is 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:

$$x + e_x < 2^k, \tag{5}$$
$$|x|_p + e_y < p, \tag{6}$$
$$|e_x|_p = e_y. \tag{7}$$

   From (6) and (7) we have $|x|_p < p - |e_x|_p$. For a certain value of $|x|_p < p - |e_x|_p$, the number of x satisfying (5) 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

$$2^{-k} \cdot \lceil \frac{2^k - e_x}{p} \rceil \cdot (p - |e_x|_p) \geq 0.5. \tag{8}$$

When $p << 2^k$, it is reasonable to rewrite (8) as follows:

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

Thereby,

$$e_x \leq \frac{2^{k-1}(p - 2 \cdot |e_x|_p)}{p - |e_x|_p}. \tag{10}$$

We know that $e_x \geq 0$, so

$$0 \leq |e_x|_p \leq \lfloor \frac{p}{2} \rfloor. \tag{11}$$

The total number of e_x satisfying (10) and (11) is upper bounded by (For simplicity, let $i = |e_x|_p$.)

$$\sum_{i=0}^{\lfloor \frac{p}{2} \rfloor} (\frac{1}{p} \cdot \frac{2^{k-1}(p-2i)}{p-i} + 1). \tag{12}$$

So the number of bad errors in this class is bounded by (12), which can be simplified to be

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

(b) $p \leq |x|_p + e_y < 2^r$, errors in this class will never be masked because the redundant part is a value that cannot occur.

(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:

$$x + e_x < 2^k, \tag{14}$$

$$|x|_p + e_y \geq 2^r, \tag{15}$$

$$|e_x|_p = e_y + p - 2^r. \tag{16}$$

From (15) and (16) 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 (14) 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

$$2^{-k} \cdot \lceil \frac{2^k - e_x}{p} \rceil \cdot |e_x|_p \geq 0.5. \tag{17}$$

When $p << 2^k$, we rewrite (17) as follows:

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

So

$$e_x \leq \frac{1}{|e_x|_p} \cdot 2^{k-1} \cdot (2|e_x|_p - p). \tag{19}$$

Because $e_x \geq 0$,

$$|e_x|_p \geq \lceil \frac{p}{2} \rceil. \tag{20}$$

The total number of e_x satisfying (19) and (20) is upper bounded by(For simplicity, let $i = |e_x|_p$.)

$$num < \sum_{\lceil \frac{p}{2} \rceil}^{p-1} (\frac{1}{p} \cdot \frac{2^{k-1}(2i-p)}{i} + 1). \tag{21}$$

So the number of bad errors in this class is bounded by (21), which can be simplified to be

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

From (13) and (22), the total number of bad errors for the case when $x + e_x < 2^k$ is bounded by $2^k + p - 2^k \sum_{i=\lceil \frac{p}{2} \rceil}^{p-1} \frac{1}{i} - \frac{2^{k-1}}{p}$.

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 bounded by $2^k + p - 2^k \sum_{i=\lceil \frac{p}{2} \rceil}^{p-1} \frac{1}{i} - \frac{2^{k-1}}{p}$.

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

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

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 $\lceil \frac{2^{k-1}}{p} \rceil$. 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 $\lceil \frac{2^{k-1}}{p} \rceil$. So the total number of bad errors occurring to the information part of the code is no more than $2 \cdot \lceil \frac{2^{k-1}}{p} \rceil$.

When p is large and $p << 2^k$, the estimated probability of bad errors and bad errors occurring to the information part of the code can be derived directly from (3) and (4). ■

The number of bad errors occurring to the information part of the code decreases as p increases according to (4). When $p > 2^{k-1}$, there are nearly no bad errors in the format of $(e_x, e_y = 0)$. 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 $r = \lceil \log_2(p) \rceil$. Many different p can be selected when r is fixed and some of them are better in terms of the total number of bad errors. Table 1 shows the best p and the corresponding fraction of bad errors for different k and r. The numbers outside the parentheses are the prime numbers which will result in the smallest number of bad errors for the given k and r. The numbers inside the parentheses are the corresponding fractions of bad errors.

When $2^r << 2^k$, we should select p to be the largest possible prime number for the purpose of minimizing the number of bad errors, e.g. when $r = 4$, the best p for the three k values are all 13, which is the largest prime number less than 2^4. However, if r is comparable to k, smaller p can achieve the same error detection capability as larger p.

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.

We next propose a construction of multilinear arithmetic codes that have smaller total number of bad errors than linear codes with the same number of redundant bits r.

	$k = 16$	$k = 32$	$k = 64$
$r = 4$	$13\,(2^{-4.6})$	$13\,(2^{-4.6})$	$13\,(2^{-4.6})$
$r = 8$	$131\,(2^{-8.6})$	$251\,(2^{-8.6})$	$251\,(2^{-8.6})$
$r = 12$	$2053\,(2^{-12.5})$	$2053\,(2^{-12.5})$	$4093\,(2^{-12.5})$
$r = 16$	–	$32771\,(2^{-16.7})$	$65521\,(2^{-16.7})$
$r = 20$	–	$524309\,(2^{-20.7})$	$1048549\,(2^{-20.7})$

Table 1: Best p and the corresponding fractions of bad errors (in parentheses) for different k and r for linear 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 a prime number larger than 2 and $||_p$ is the modulo 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 $e_x \in Z_{2^k}, e_y \in Z_{2^r}$, $r = \lceil \log_2 p \rceil$ 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

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

When $p << 2^k$, the estimated probability of bad errors for $[|x|_p, |2x|_p]$ multilinear codes is 2^{-2r}.

Proof. For each linear code, there are four cases resulting in four sets of bad errors (refer to the proof for Theorem 1), which are shown in Table 2. When we randomly select C_1 and C_2 with equal probability, an error $e = (e_x, e_y)$ 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.

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

Table 2: Classification of bad errors for linear arithmetic codes

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

$$|x + e_x|_p = |x|_p + e_y - 2^r. \tag{24}$$

   Equivalently,

$$|e_x|_p = |e_y - 2^r|_p. \tag{25}$$

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

$$|2 \cdot (x + e_x)|_p = |2x|_p + e_y - 2^r. \tag{26}$$

   Thus

$$|2e_x|_p = |e_y - 2^r|_p. \tag{27}$$

   From (25) and (27) 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:

$$x + e_x < 2^k, \tag{28}$$
$$|x|_p + e_y \geq 2^r, \tag{29}$$
$$|2x|_p + e_y \geq 2^r, \tag{30}$$
$$|e_x|_p = e_y + p - 2^r = 0. \tag{31}$$

   From (31), $e_y = 2^r - p$. So $|x|_p + e_y < 2^r, |2x|_p + e_y < 2^r$. Thereby 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

$$|x + e_x|_p = |x|_p + e_y. \tag{32}$$

Equivalently,

$$|e_x|_p = e_y. \tag{33}$$

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

$$|2 \cdot (x + e_x)|_p = |2x|_p + e_y. \tag{34}$$

Equivalently,

$$|2e_x|_p = e_y. \tag{35}$$

From (33) and (35) 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:

$$x + e_x < 2^k, \tag{36}$$
$$|x|_p + e_y < p, \tag{37}$$
$$|2x|_p + e_y < p, \tag{38}$$
$$|e_x|_p = e_y = 0. \tag{39}$$

When (39) is satisfied, (37) and (38) 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

$$2 \cdot (2^k - e_x) \geq 2^k. \tag{40}$$

Equivalently,

$$e_x \leq 2^{k-1}. \tag{41}$$

From (39) and (41), the number of bad errors is bounded by $\lceil \frac{2^{k-1}+1}{p} \rceil$. Similarly, when C_1 and C_2 both belong to Case3, the number of bad errors is bounded by $\lceil \frac{2^{k-1}}{p} \rceil$.

So an upperbound of the total number of bad errors is

$$\lceil \frac{2^{k-1}+1}{p} \rceil + \lceil \frac{2^{k-1}}{p} \rceil \approx 2 \cdot \lceil \frac{2^{k-1}}{p} \rceil. \tag{42}$$

The estimated probability of bad errors can be derived directly from (23). ■

All bad errors for $[|x|_p, |2x|_p]$ multilinear codes are in the format of $(e_x, e_y = 0)$. They have the same value as bad errors occurring to the information part of the linear arithmetic code. In another word, the proposed multilinear arithmetic code will not help if we assume that e_y is always 0. This situation, however, can be prevented by implementing a merged design of the original device and the encoder of the code since in this

case faults will have high probability to affect not only the original device but also the encoder that generates the redundant part of the code. The advantage of this construction is that it has no other bad errors except for errors occurring to the information part of the code. The total number of bad errors is much smaller than that of linear arithmetic codes. Moreover, it is possible to reduce the number of bad errors to be nearly zero using $[|x|_p, |2x|_p]$ multilinear codes, which is impossible for linear arithmetic codes.

The next construction based on utilizing multiple modulus can drastically reduce the number of bad errors in the format of $(e_x, e_y = 0)$. The theorem can be proved in a similar way to the proof for Theorem 2. Here we omit the proof due to the limit of space.

Theorem 3. *(*$[\mathbf{p}, \mathbf{q}]$ ***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 $r = \max(\lceil \log_2 p \rceil, \lceil \log_2 q \rceil)$, p and q are different prime numbers larger than 2, $f_1(x) = |x|_p$ and $f_2(x) = |x|_q$ ($|\,|_p$ is the modulo operation). Without loss of generality, we assume that $p > q$. 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_1 and C_2 with equal probability to encode the original messages and assume that the error only occurs to the information part of the code, the number of bad errors is upper bounded by

$$2 \cdot (\lceil \frac{2^{k-1}}{pq} \rceil + \lceil \frac{2^k}{pq} \rceil). \tag{43}$$

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

Remark 1. The precise number of bad errors for $[p, q]$ multilinear codes is hard to analyze. However, experimental results indicate that it is comparable to that of $[|x|_p, |2x|_p]$ multilinear codes and is much smaller than that of linear arithmetic codes. The idea of utilizing 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 multiresidue 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.

To demonstrate that $[p, q]$ multilinear codes have less bad errors in the format of $(e_x, e_y = 0)$ than linear and $[|x|_p, |2x|_p]$ multilinear arithmetic codes, we compare the number of bad errors in this class for the three constructions in Table 3. The number of information bits of the codes in the table is 32. For $[p, q]$ multilinear codes, q is selected to be the largest possible prime number less than p, e.g. when $p = 241$, $q = 239$. Linear arithmetic codes and $[|x|_p, |2x|_p]$ multilinear codes have the same number of bad errors occurring to the information part of the code. As p increases, this number for $[p, q]$ multilinear codes decreases much faster than the other two. When $p = 2767 (r = 12)$,

	$p=5$	$p=241$	$p=563$	$p=883$	$p=1237$	$p=2767$
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
$[\|x\|_p, \|2x\|_p]$ 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
$[p,q]$ codes	8.6×10^8	2.2×10^5	4.1×10^4	1.7×10^4	8.5×10^3	1.7×10^3

Table 3: Number of bad errors occurring to the information part of linear and multilinear codes (k=32)

$[p,q]$ multilinear codes has only 1.7×10^3 bad errors in the format of $(e_x,0)$ while the other two have 1.6×10^6.

To end this section, we summarize results on the probability of bad errors and the probability of bad errors occurring to the information part of linear and multilinear codes in Table 4. When bad errors occurring to the information part is more critical, $[p,q]$ multilinear codes should be used. Otherwise, $[|x|_p, |2x|_p]$ multilinear codes should be used because they are better than linear arithmetic codes in terms of the error detection abilities and require less hardware overhead to implement than $[p,q]$ multilinear arithmetic codes (Section 4).

Probability of	Linear Arithmetic Codes	$[\|x\|_p, \|2x\|_p]$ codes	$[p,q]$ codes
bad errors	$\approx 0.3\cdot 2^{-r+1}$	$\approx 2^{-2r}$	$\approx 2^{-2r*}$
bad errors in info. part	$\approx 2^{-2r}$	$\approx 2^{-2r}$	$\approx 3\cdot 2^{-3r}$

* : Based on experimental results.

Table 4: Probability of bad errors and probability of bad errors occurring to the information bits of linear and multilinear codes

4 Protection of Multipliers Using Multilinear Arithmetic Codes

In this section, we propose protection architectures for multipliers based on multilinear arithmetic codes. The multiplier is a basic block in many public key and even in some secret key cryptographic devices. Due to its arithmetic nature of the operations, arithmetic error model is most often used for such devices. In this section, we assume that additive arithmetic errors manifest themselves at the output of the multiplier and the predictor[3]. 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. We analyze and compare the number of bad errors for architectures protected by the three codes presented in Section 3. The advantage of architectures based on multilinear codes in terms of the number of bad errors is demonstrated.

The general hardware 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 which detects errors at the output of the device. The detailed architectures for the alternatives are shown in Figure 1.

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 (EDN) is a modulo p operation. The hardware overhead mainly comes from the $r=$

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

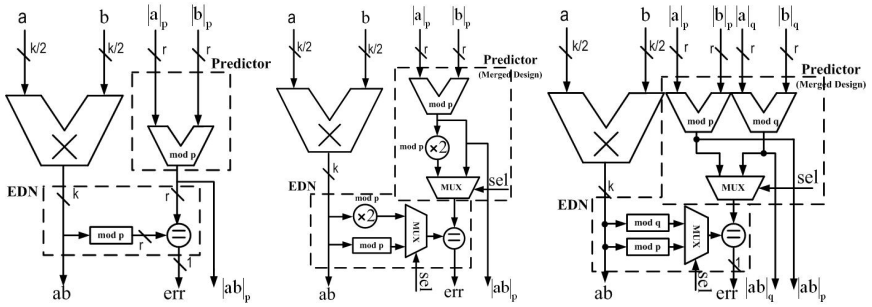

Fig. 1: Hardware arthictectures for multipliers protected by (a) linear arithmetic codes, (b) $[|x|_p, |2x|_p]$ multilinear codes and (c) $[p, q]$ multilinear codes

$\lceil \log_2(p) \rceil$ bit 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 $\times 2$ operation in Z_p for both the predictor and the EDN. $\times 2$ operation is equal to shifting the operands by 1 bit, which is trivial in terms of the hardware overhead. The complexity of an r-bit multiplexer is in general of the order of $O(r)$. Thereby this architecture has comparable hardware overhead to linear arithmetic codes.

The protection architecture based on $[p, q]$ 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 3.

Remark 2. There is a tradeoff between the error detection abilities and the hardware overhead when we select p and q as specialized p and q can significantly reduce the hardware complexity of the modulo operation e.g. using Mersenne primes.

In Table 5 we compare the hardware overhead, the estimated probability of bad errors and the probability of bad errors occurring to the information part of the three architectures for the case when $k = 32, r = 7, p = 2^7 - 1$. For $[p, q]$ multilinear codes $q = 2^5 - 1$. The three designs were modeled in Verilog and synthesized in Synopsis Design Compiler. The hardware overhead for the predictor and EDN for linear arithmetic codes is in total 32%. With 7 redundant bits, the probability of bad errors

for linear arithmetic codes is of the order of 2^{-7}. With similar hardware overhead, $[|x|_p, |2x|_p]$ multilinear code can reduce this probability to 2^{-14}. Experimental results shown that $[p, q]$ multilinear codes have nearly the same probability of bad errors as $[|x|_p, |2x|_p]$ multilinear codes, which is 2^{-14}. Meanwhile the probability of bad errors occurring to the information part is only about $3{\cdot}2^{-21}$, which is the smallest of the three. The disadvantage of this architecture is that it requires more overhead to implement. But we note that 53% is still less than the overhead to implement $(x, |x^2|_p)$ partially robust codes proposed in [10].

	Linear Arith. Codes	$[\|x\|_p, \|2x\|_p]$ ML codes	$[p, q]$ ML codes
Overhead for Predictors and EDN	32%	40%	53%
Probability of bad errors	$\approx 2^{-7}$	$\approx 2^{-14}$	$\approx 2^{-14*}$
Probability of bad errors in the information part	$\approx 2^{-14}$	$\approx 2^{-14}$	$\approx 3 \cdot 2^{-21}$

* : Based on experimental results

Table 5: Hardware overhead and the estimated probability of bad errors for architectures based on linear and multilinear codes ($k = 32, r = 7$)

We next present experimental results of the error detection capabilities of 8-bit multipliers protected by linear, $[|x|_p, |2x|_p]$, $[p, q]$ and $(x, |x^2|_p)$ arithmetic codes to demonstrate the advantages of architectures based on the proposed multilinear codes. All the four alternatives were simulated in MATLAB. The number of bad errors were analyzed and compared. In this experiment, $k = 16, p = 31$. For $[p, q]$ multilinear code $q = 29$. Each operand of the multiplier is a 8-bit binary vector and is randomly selected from 0 to $2^8 - 1$. Additive errors happen at the output of the multiplier and the predictors of the devices. The X axis in Figure 2 is the error masking probability. The Y axis shows the number of errors masked by a certain probability. Errors on the right half of each subfigure are bad errors that are masked by a probability larger or equal to 0.5. As expected, the number of bad errors for (b),(c) and (d) are drastically reduced compared with architectures based on linear arithmetic codes. Compared with (d), the advantage of (b) and (c) is that they require less hardware overhead than the former. (For the estimation of the hardware overhead for $(x, |x^2|_p)$ code, please refer to [10]).

To further demonstrate the advantages of multilinear arithmetic codes, we compare the error/fault detection capabilities of the above four alternatives when errors at the output of the devices repeat. As discussed in Section 2, we assume that an attacker can use any fault injection methodologies and can inject faults with high spatial resolution to generate a specific error at the output of the device. Once injected, the fault/error stays for several clock cycles because the temporal resolution of the fault injection method is limited.

The X-Axis in Figure 3 is the number of consecutive clock cycles that the error lasts, which is denoted by t. Suppose the same error stays at the output for t consecutive clock cycles, the error is said to be detected if it is detected at least once among these t clock cycles. Otherwise, we say that the error is masked. Figure 3(a) compares the error masking probability for linear, $[|x|_p, |2x|_p]$, $[p, q]$ and $(x, |x^2|_p)$ arithmetic codes. The Y-Axis is the average error masking probability.

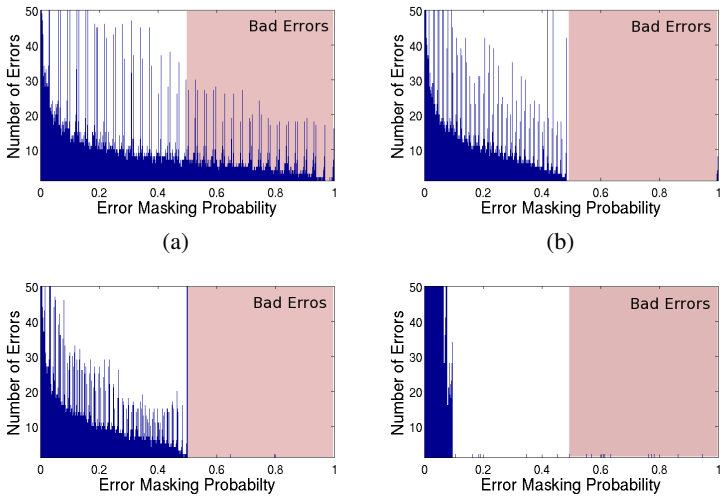

Fig. 2: Number of bad errors for 8-bit multipliers protected by (a) linear arithmetic codes, (b) $[|x|_p, |2x|_p]$ codes, (c) $[p, q]$ codes and (d) $(x, |x^2|_p)$ codes

The average error masking probabilities of all four alternatives decrease as t increases. However, $[|x|_p, |2x|_p]$, $[p, q]$ and $(x, |x^2|_p)$ arithmetic codes have much better error detection capabilities than linear codes for large t. Figure 3(b) plots the ratio of the average error masking probability of $[|x|_p, |2x|_p]$, $[p, q]$ and $(x, |x^2|_p)$ codes to that of the linear arithmetic codes. When the error stays for two consecutive clock cycles, the error detection abilities for multilinear and robust arithmetic codes are already twice better than that of linear codes. $(x, |x^2|_p)$ has the lowest error masking probability when $t = \{2, 3, 4\}$. For $t > 6$, $[|x|_p, |2x|_p]$ codes and $(x, |x^2|_p)$ codes have similar performance. $[p, q]$ codes are the best among the three nonlinear arithmetic codes when $t > 4$ and $p = 31, q = 29$. For smaller q, the error detection ability of $[p, q]$ codes will be worse but still better than linear arithmetic codes. Thereby, the proposed multilinear arithmetic codes can provide comparable or even better error protection abilities than partially robust arithmetic codes $(x, |x^2|_p)$ while requiring much less hardware overhead. Architectures based on nonlinear arithmetic codes are better than that based on linear arithmetic codes in terms of the number of bad errors and have lower error masking probability in lazy channels where errors tend to repeat.

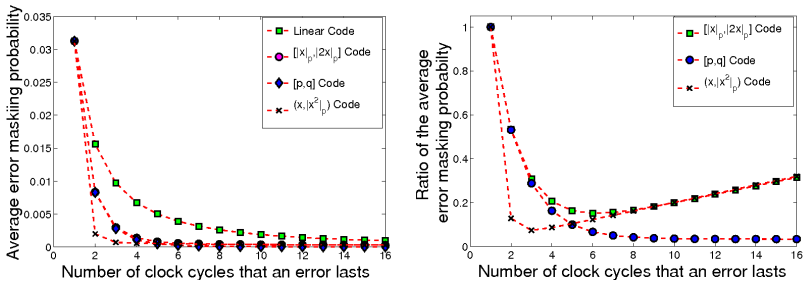

Fig. 3: Error detection properties of 8-bit multipliers protected by linear, multilinear and partially robust arithmetic codes in lazy channels

5 Conclusions

In this paper we presented several constructions of multilinear arithmetic codes and compared their error detection properties to that of linear and partially robust arithmetic codes. Architectures of reliable multiplier based on multilinear arithmetic codes were introduced. Experimental results show that the error detection capability can be significantly improved over that of linear codes at the expense of a very mild increase in the hardware overhead (Table 5). The proposed multilinear codes can achieve as good error detection abilities as that of the partially robust arithmetic codes with smaller hardware overhead and are better choices than linear arithmetic codes in lazy channels where errors tend to repeat. These codes can efficiently prevent the attackers from injecting undetectable faults/errors under the assumption that the temporal resolution of the fault injection methodologies is limited. Finally, we would like to point out that in this paper we considered only the case when multilinear codes are constructed from 2 codes. But the results can be easily generalized to any number of codes.

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. Skorobogatov, "Optical fault induction attacks," in *Cryptographic Hardware and Embedded Systems Workshop (CHES)*, 2002.

[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] C. Carlet and C. Ding, "Highly nonlinear mappings," *Journal of Complexity*, vol. 20, no. 2-3, 2004.

[12] 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.

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