Secure NAND Flash Architecture Resilient to Strong Fault-Injection Attacks Using Algebraic Manipulation Detection Code

Pei Luo
Reliable Computing Lab
Electrical and Computer Engineering
Boston University
Email: luopei@bu.edu

Zhen Wang
Mediatek Wireless, Inc
Email: wang.zhen.mtk@gmail.com

Mark Karpovsky*
Reliable Computing Lab
Electrical and Computer Engineering
Boston University
Email: markkar@bu.edu

***Abstract*—Multi-level cell (MLC) NAND flash memories are widely used because of their high data transfer rate, large storage density and long mechanical durability. Linear error correcting codes (ECC) such as Reed-Solomon (RS) codes and Bose-Chaudhuri-Hocquenghem (BCH) codes are often used for error correction. Although linear codes can efficiently detect and correct random errors, they are not sufficient for protecting NAND flash memories used in cryptographic devices against malicious fault injection attacks. In this paper, we will present an architecture based on the combination of RS codes and Algebraic Manipulation Detection (AMD) codes which can correct any four byte errors and detect any malicious injected errors with a high probability under the strong attack model. This proposed architecture can significantly improve the security level of the MLC NAND flash memories used in cryptographic devices at the cost of only slightly larger latency and area overhead.**

***Keywords*-MLC NAND Flash, Reed-Solomon code, Algebraic Manipulation Detection Code, Error Correction, Fault Injection Attack, Hardware Security**

I. INTRODUCTION

NAND flash memories are widely used in cell phones, digital cameras, USB devices due to the high data transfer rate, low power consumption, large storage density and long mechanical durability [1]. Different from NOR flash and single-level cell (SLC) NAND flash memories, MLC technology is implemented in MLC NAND flash memories to increase the capacity. While MLC technology can increase the capacity of the devices, it will reduce the voltage margin separating the states and thus result in the possibility of more errors. In order to increase the reliability of the MLC NAND flash devices, ECC are often used in the devices to detect and correct the errors.

Most of the works on protecting NAND flash devices against random errors are based on linear codes. In [2], the authors analyzed the error characteristics of MLC flash memory and proposed an error control coding using non-binary low-density parity-check (LDPC) codes. In [3], the authors proposed a high speed architecture based on $(4148, 4096)$ BCH codes correcting quadruple $(t = 4)$ errors. In [4], the authors proposed product codes which use RS codes along rows and Hamming codes along columns which have reduced hardware overhead.

The security of modern devices can be broken by malicious attackers via side channel attacks such as fault injection attacks [5], [6], [7]. While linear error correcting codes are very useful for the detection and correction of random errors, they are not enough against advanced attacks.

In order to improve the security of the system against advanced fault injection attacks, non-linear codes have been used. In [1], the authors proposed two general constructions of nonlinear multi-error correcting codes and compared the architectures to those based on BCH codes and RS codes. In [8], [9], [10], the authors proposed Robust Codes to detect the side channel fault injection attacks. Robust codes can provide nearly equal protection against all error patterns. The error masking probabilities for robust codes are upper-bounded by very small numbers for all non-zero errors [11]. But the limitation of robust codes is that it is assumed that the output of the device is almost uniformly distributed and is not controlled by the attacker [11]. So robust codes can provide protection against weak attacks but it's not good enough against strong attacks.

In order to protect the devices against strong attacks, the authors of [11] implemented AMD codes based on nonlinear encoding functions to detect the fault injection attacks. AMD codes can provide a guaranteed high error detecting probability even if the fault-free outputs as well as the error patterns of a device-under-attack are controlled by the attacker [11], [12].

In this paper, we will present an architecture for MLC NAND flash based on the combination of RS and AMD codes. This architecture is not only able to correct any four byte errors, it can also detect the errors injected by the strong attackers with a very high probability. The results show that the proposed architecture can significantly improve the security of MLC NAND flash devices with minor timing and resource overhead. The simulation results show that while the attacker

*The work of the third author is sponsored by the NSF grant CNR 1012910.
Contact author: Pei Luo, luopei@bu.edu
Track: Hardware Security – Security Architecture

can inject random errors into the common RS architecture MLC NAND flash memories with probability of miscorrection about 4.092%, the proposed architecture can detect most of these miscorrections with a probability $1 - 1.77 \times 10^{-8}$.

The rest part of this paper is organized as follows. We describe the MLC NAND flash memory model and the advanced attack model in Section II. In Section III, we present the definition and construction of AMD codes. In Section IV, the construction of the encoder and decoder based on the combination of RS and AMD codes are presented. In Section V, we present the synthesis and the fault-injection simulation results for the proposed architecture, and compared them with the architecture based only on RS code.

II. MLC NAND Flash Memory and Attack Model

Multi-level cell is able to store multiple bits by separating the threshold voltage into four parts or more, this will increase the bit error rate in the memories because electrons stored in adjacent levels tend to shift more easily from one level to another. Besides such random errors, memories may also be threatened by fault injection attacks. In this section, we will describe MLC NAND flash memories and the attack model we use in this paper.

A. MLC NAND Flash Memory

The threshold voltage of the whole memory array satisfies a Gaussian distribution due to random manufacturing variations [1]. Figure 1 illustrates the threshold voltage distribution of a 2 bits multi-level cell [13], [14], [15], [16].

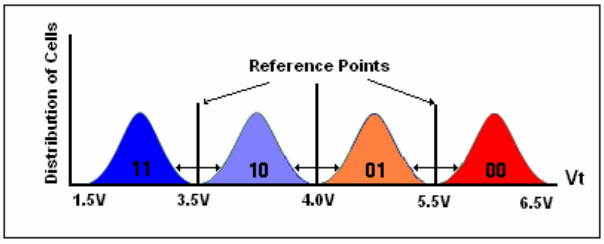

Fig. 1: Voltage threshold of MLC NAND flash

The NAND Flash array is grouped into a series of blocks, which are the smallest erasable entities in a NAND flash device. Each block is composed of some pages, which are the smallest read and program unit [17]. Each page is composed of storage area plus spare area. For example, the page size is 8,936 bytes in MT29F512G08CUCBB from Micron, in which 8,192 bytes are used for storage with extra 744 bytes as the spare area [18].

Because the page size is very large nowadays, the storage area is always divided into several sections to increase the processing speed and decrease the encoding and decoding complexity. For the memory architecture, we assume that the page size of the memory is 2,112 bytes (16,896 bits) in the model we use in this paper, and the storage area is divided into four main sections, spare area are also divided into four parts, each for one main section, as in the Figure 2 [19], [20], [21]. Thus each section will be 512 bytes. We will build our error correcting architecture based on this size.

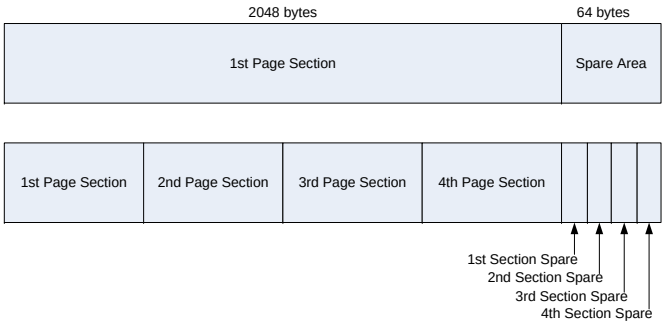

Fig. 2: Section structure of one page

B. Attack Model

We assume an advanced attack model in this paper. Under this model, the attacker knows every detail of the memory, including the error control codes used. The attacker is able to choose the specific inputs to the device during fault injection. At the same time, we assume the attacker has the ability to inject any errors at the output of the device. Thus the attacker will have full control of the fault-free output y, and the faulty output $\tilde{y} = y \oplus e_y$ where e_y is an error pattern and $\oplus$ stands for component wise addition in finite field. Under such a strong attack model, all the previous error correcting codes architectures will be insufficient [1], [11], [12].

Architectures which can provide a guaranteed fault detection probability under such a strong attack model are called strong secure architectures. In this paper, we will describe a strong secure architecture for MLC NAND flash memories based on the combination of RS and AMD codes. We will show that under the strong attack model described in this section, the proposed architecture will still provide a high error (fault) detection probability.

III. AMD codes

Throughout the paper we denote by $\oplus$ the component-wise modulo addition in $GF(q), q = 2^r$. All the results presented in the paper can be easily generalized to the case where $q = p^r$ (p is a prime). An code V with codewords $(y, x, f(y, x))$, where $y \in GF(2^k)$ are the information bits, $x \in GF(2^m)$ are the random bits, and $f(y, x) \in GF(2^r)$ are the redundant bits, will be referred to as a (k, m, r) code.

Definition 3.1: (**Security Kernel**) [11] For any (k, m, r) error detecting code V with the encoding function $f(y, x)$, where $y \in GF(2^k), x \in GF(2^m)$ and $f(y, x) \in GF(2^r)$, the **security kernel** K_S is the set of errors $e = (e_y, e_x, e_f), e_y \in GF(2^k), e_x \in GF(2^m), e_f \in GF(2^r)$, for which there exists y such that $f(y \oplus e_y, x \oplus e_x) \oplus f(y, x) = e_f$ is satisfied for all x.

$$K_S = \{e | \exists y, f(y \oplus e_y, x \oplus e_x) \oplus f(y, x) \oplus e_f = \mathbf{0}, \forall x\}. \quad (1)$$

Under strong attack model, the nonzero errors $e \in K_S$ can be used by the attacker to bypass the protection of the error detection code with the kernel K_S. In order to prevent such attacks, an AMD code should have a kernel K_S composed of only zero vector in $GF(2^n)$, $n = k + m + r$.

Definition 3.2: [22] A (k, m, r) error detecting code is called **Algebraic Manipulation Detection (AMD)** code iff $K_S = \{\mathbf{0}\}$, where $\mathbf{0}$ is the all zero vector in $GF(2^n)$, $n = k + m + r$.

For AMD codes, there are no undetectable errors (errors that are undetected with a probability of 1). For any y and e, the error masking probability can be computed as

$$Q_V(y,e) = 2^{-m}|\{x \,|\, (y, x, f(y,x)) \in V, (y \oplus e_y, x \oplus e_x, f(y,x) \oplus e_f) \in V\}| \quad (2)$$

The security level of a code can be characterized by the worst case of error masking probability $\max_{y,e\neq 0} Q_V(y,e)$. A lower bound on Q_V for AMD codes can be found in [11].

Let $x = (x_1, x_2, \cdots, x_t), x_i \in GF(q), q = 2^r$. Let

$$A(x) = \begin{cases} \bigoplus_{i=1}^{t} x_i^{b+2} & \text{if } b \text{ is odd;} \\ \bigoplus_{i=2}^{t} x_1 x_i^{b+1} & \text{if } b \text{ is even and } t > 1; \end{cases} \quad (3)$$

and

$$B(x,y) = \bigoplus_{1 \le j_0 + j_1 + \cdots + j_{t-1} \le b+1} y_{j_0, j_1, \cdots, j_{t-1}} \prod_{i=0}^{t-1} x_i^{j_i}, \quad (4)$$

where $\prod_{i=1}^{t} x_i^{j_i}$is a monomial of x of a degree at most $b+1$ and $\prod_{i=1}^{t} x_i^{j_i} \notin \Delta B(x)$ where $\Delta B(x)$ is defined as

$$\Delta B(x) = \begin{cases} \{x_1^{b+1}, x_2^{b+1}, \cdots, x_t^{b+1}\} & \text{if } b \text{ is odd;} \\ \{x_2^{b+1}, x_1 x_2^b, \cdots, x_1 x_t^b\} & \text{if } b \text{ is even and } t > 1; \end{cases} \quad (5)$$

Suppose $f(x,y) = A(x) \oplus B(x,y)$, it is easy to verify that the left hand side of the error masking equation $f(x \oplus e_x, y \oplus e_y) \oplus f(x,y) \oplus e_f = 0$ is always a non-zero polynomial of x of a degree up to $b+1$.

Theorem 3.1: [11], [12] Let $f(x,y) = A(x) \oplus B(x,y)$ be a q-ary polynomial with $y \in GF(q^s)$ as coefficients and $x \in GF(q^t)$ as variables, where $1 \le b \le q-3$ and $q = 2^r$. Then the code V composed of all vectors $(y, x, f(x,y))$ is a (k, m, r) AMD code with $m = tr$, $k = (\binom{t+b+1}{t} - 1 - t)r$ and $Q_V = (b+1)2^{-r}$.

Remark 3.1: If b is even when $t = 1$, $A(x)$ can be chosen as x^{b+3} instead of x^{b+2}. In this case, $Q_V = (b+2)2^{-r}$.

Remark 3.2: When k is not a multiple of r, 0's can be appended to y before $f(x,y)$ is computed. The resulting AMD code will have the same Q_v as the AMD code with the AMD code with the same $f(x,y)$, for which k is a multiple of r.

Example 3.1: Let $t = b = 1$, then $m = r$ and $k = (\binom{t+b+1}{t} - 1 - t)r = r$. The encoding function $f(y,x)$ for the AMD code based on Theorem 3.1 is $f(y,x) = y \cdot x \oplus x^3$, where $x, y, f(y,x) \in GF(2^r)$. The resulting AMD code has $Q_V = 2^{-r+1}$.

If $t = 1$ and $b = 3$, then $m = r$, $k = (\binom{t+b+1}{t} - 1 - t)r = 3r$ and $f(y,x) = y_1 \cdot x \oplus y_2 \cdot x^2 \oplus y_3 \cdot x^3 \oplus x^5$, where $y_1, y_2, y_3, x, f(y,x) \in GF(2^r)$. For this code $Q_V = 2^{-r+2}$.

AMD codes for $t = 1$ have been discussed in [22], and details of AMD codes with $t > 1$ can be found in [11], [12], [23]. AMD codes based on different parameters used in this paper will be discussed in Section IV.

IV. Construction of Secure Flash Memory

Although the AMD codes defined in Section III have very high security level and can detect most of the injected errors with very low probability of missing an error, it has no ability to locate and correct the random or injected errors. Because the bit error rate is relatively high in MLC NAND flash, error control codes must be implemented besides AMD codes to detect and correct the random errors. Some linear codes such as BCH code and RS code have very good ability to detect and correct random and burst errors and they are often used in MLC NAND flash memories to improve the reliability [3], [2], [16], [21]. In this paper, we combine RS code and AMD code to construct a secure and reliable structure which can detect most of the malicious attacks and correct random errors.

RS code with symbols from $GF(q_{RS})$ can be defined by a set of three parameters (n_{RS}, k_{RS}, t_{RS}), in which k_{RS} and n_{RS} are the number of symbols before and after encoding respectively, and $t_{RS} = (n_{RS} - k_{RS})/2 = r_{RS}/2$ is the number of symbols which can be corrected among n_{RS} symbols. Symbols take their values in a Galois Field $GF(2^{m_{RS}})$, and are thus represented with m_{RS} bits. The parameter n_{RS} is bounded by $2^{m_{RS}}$ [24]. Through this paper, we use n_{RS}, k_{RS} to denote the length and the number of the information symbols in the RS code while t_{RS} is the number of errors which can be corrected by the RS code.

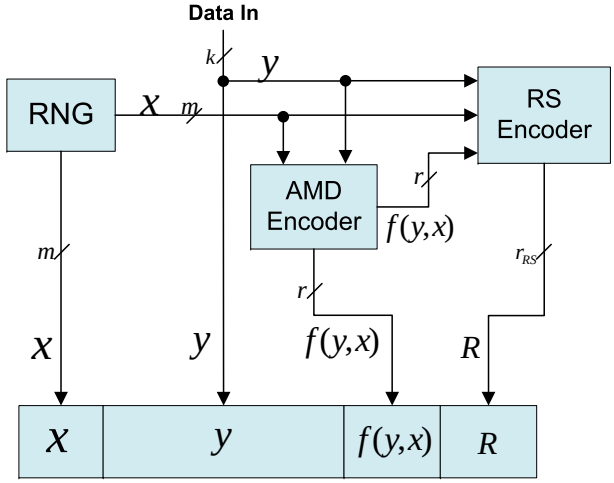

Fig. 3: Encoder structure of the proposed architecture

The encoding architecture based on the combination of RS and AMD code is shown in Figure 3. In this architecture, the **random number generator (RNG)** generates the random numbers $x_i \in GF(2^r)$ for the encoding of AMD code. (We note that the RNG module is already integrated in most of the

modern cryptographic devices thus the proposed architecture needs no extra resource to build the RNG module in such cryptographic applications.) The AMD encoder then computes the AMD redundant bits $f(y,x) \in GF(2^r)$ with the random bits $x \in GF(2^m)$ and information bits $y \in GF(2^k)$. In this architecture, the AMD random number x and the redundancy $f(y,x)$ will also be part of the information bits for RS code such that the errors in x and $f(y,x)$ can also be detected and corrected. So the information part of RS code is $(x, y, f(y,x))$.

The random number x is generated and applied to the RS and AMD encoder before the information bits are read into the encoder module. When the information bits y are read into the encoder module, they are sent to the RS encoder and AMD encoder at the same time. When the information bits are all read into the encoder module, the AMD encoder will finish the encoding and generate the redundancy $f(y,x)$. Then $f(y,x)$ is sent to the RS encoder as a part of the RS information bits. So the encoding of AMD code and RS code can be processed in parallel, thus the architecture will need only several additional clock cycles for the AMD random number and redundant parts as compared with the architecture based on only RS code.

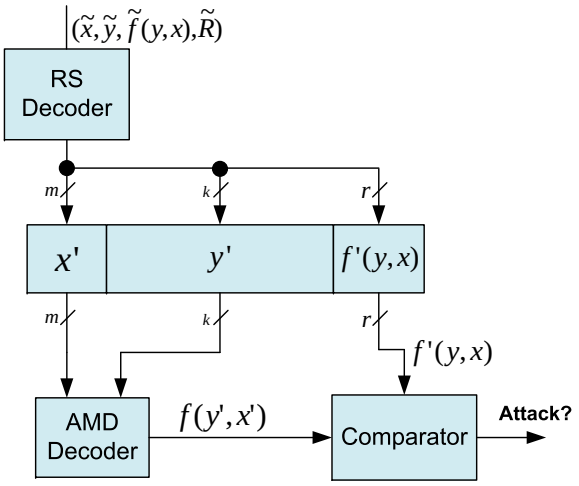

Fig. 4: Decoder structure of the proposed architecture

The decoder architecture is shown in Figure 4. The random number x' is decoded before the information bits in decoding process, thus the decoding of RS and AMD code can be implemented in parallel.

In the decoding process, the RS decoder will output the corrected (but may be still distorted due to the miscorrection) information bits including the random number x', the information part y' and the AMD redundancy $f'(y,x)$ according to the distorted inputs $(\tilde{x}, \tilde{y}, \tilde{f}(y,x), \tilde{R})$. The distortion is due to the possible miscorrection by the RS code. We note that if the information is not distorted, then $x = \tilde{x} = x'$, $y = \tilde{y} = y'$, $f(y,x) = \tilde{f}(y,x) = f'(y,x), R = \tilde{R}$. The AMD encoder then generates $f(y',x')$ according to the recovered x' and y'. The comparator then compares the decoded $f'(y,x)$ and the newly generated $f(y',x')$, if they are not equal, then it means RS code cannot correct all the errors and miscorrection happened.

Apparently, if no more than $t_{RS} = 4$ byte errors happened then the RS decoder can recover all the symbols correctly. Then $x = x'$, $y = y'$ and $f(y,x) = f'(y,x)$, thus $f'(y,x) = f(y',x')$. If there are more than four byte errors in the code words, then the RS decoder can detect most of such situations and refused to correct the errors with the extra being miscorrected. Under such situation, $f(y',x') = f'(y,x)$ holds with a very small probability, which means the AMD code can detect most of the miscorrections.

After we implement RS and AMD codes in the architecture, the encoded message will be in the format $((x_0, \cdots, x_t), (y_1, \cdots, y_b), f(y,x), R)$. In which x_i are the random numbers generated by RNG, y_i are the information bits stored in the flash memories and $f(y,x)$ is the AMD redundancy. The AMD random number x_i, information bits y_i and the redundant part $f(y,x)$ are all in $GF(2^r)$. R is the RS redundancy part with $(x_0, \cdots, x_t)$, $(y_1, \cdots, y_b)$, $f(y,x)$ together as the RS input, where $x_i, y_i, f(y,x) \in GF(2^r)$.

A. Parameters of AMD Code

In the flash memory model we use in this paper, each section size is 512 bytes, or 4,096 bits, so the number of information bits k should be at least 4,096. We choose the length of the random number x_i to be 32 bits, thus $x_i \in GF(2^{32})$, $r = 32$. In this paper, we consider three alternatives:

1) If $t = 1$, according to Theorem 3.1, because the flash section size is 4096 bits, we can get $b = 128$. By (3) and (4), we have $A(x) = x^{131}$, $B(y,x) = \oplus_{1 \le j \le 129} y_j x^j$, $\Delta B(x) = \{x^{129}\}$.
2) If $t = 2$, $b = 14$, $x = (x_0, x_1)$, we have $A(x) = x_0 x_1^{15}$, $B(y,x) = \oplus_{1 \le j_0 + j_1 \le 15} y_{j_0 j_1} x_0^{j_0} x_1^{j_1}$, $\Delta B(x) = \{x_1^{15}, x_0 x_1^{14}\}$.
3) If $t = 3$, $b = 7$, $x = (x_0, x_1, x_2)$, then $A(x) = x_0^9 \oplus x_1^9 \oplus x_2^9$, $B(y,x) = \oplus_{1 \le j_0 + j_1 + j_2 \le 8} y_{j_0 j_1 j_2} x_0^{j_0} x_1^{j_1} x_2^{j_2}$, $\Delta B(x) = \{x_0^8, x_1^8, x_2^8\}$.

We have the AMD parameters of $t = 1, 2, 3$ discussed above and the maximum error masking probabilities Q_V for the corresponding codes estimated by Theorem 3.1 listed in Table I.

TABLE I: Parameters of AMD code

t	r	m	b	k	Q_V
$t = 1$	32	32	128	4,096	3.0×10^{-8}
$t = 2$	32	64	14	4,256	3.49×10^{-9}
$t = 3$	32	96	7	5,152	1.86×10^{-9}

[I] k is the number of maximum length of the AMD code, can be shortened to the size needed
[II] all the computation are in $GF(2^r)$
[III] m is the total length of the random number

From Table I and the definition of AMD code in Section III, we can see that trade-offs between security level and resource can be achieved by choosing different parameters of the code. For example, if we choose $t = 1$, then the extra area we need for the AMD redundancy and resources to compute $f(y,x)$

will be smaller, but the security level Q_V will also be lower. If we want a higher security level against advanced attacks, we will need larger t thus larger redundant area for AMD code and more complicated encoder and decoder circuit. At the same time, we can choose also larger r for AMD code for higher security level, but then the AMD code will be constructed over higher order Galois field thus more resources are needed.

In this paper, we will choose AMD code with $t = 1$ as an example and compare the result with the architecture based on only RS code. AMD encoder and decoder are actually the same. For $t = 1$, the AMD module needs two multipliers M_0 and M_1 to calculate x^j and $x^j y_j$ respectively. The AMD encoder architecture circuit for $t = 1$ is as in Figure 5. In Figure 5, the multiplier M_0 is used to calculate $x^2, x^3, \cdots, x^{131}$ and M_1 is used to calculate $y_j x^j$.

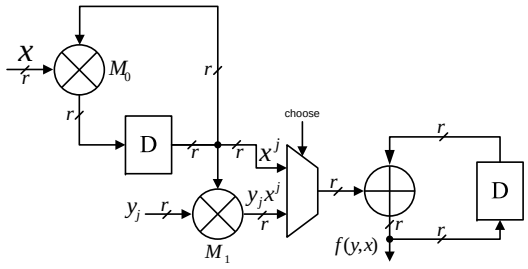

Fig. 5: AMD encoder structure for $t = 1$

For $t = 2$ and $t = 3$, we can use Horner's scheme to implement the AMD encoder and decoder. Or we can also compute x^j before the information bits are read into the system thus to save multipliers. The AMD encoder architecture circuit for $t = 2$ is as in Figure 6. In this architecture, the two multipliers M_0 and M_1 are used to calculate $x_0^2, x_0^3, \cdots, x_0^{15}$ and $x_1^2, x_1^3, \cdots, x_1^{15}$ before read the information bits. Then the multipliers M_0 and M_1 are used to calculate the $y_{j_0 j_1} x_0^{j_0} x_1^{j_1}$.

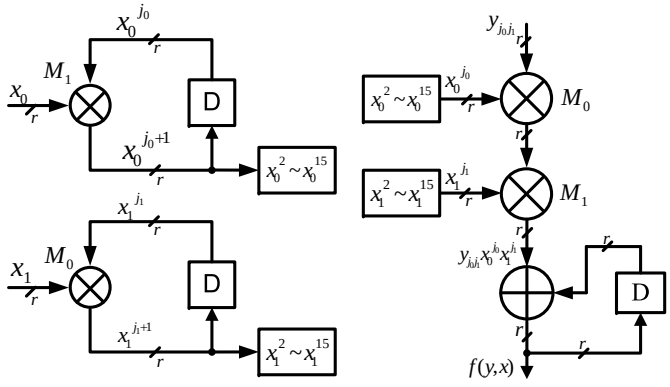

Fig. 6: AMD encoder structure for $t = 2$

B. Parameters of RS Code

In this architecture, the section size is 512 bytes in the MLC NAND flash memory model of Section II. We choose $t = 1$ for AMD code in this paper, so the size of the AMD random number and AMD redundancy are both 32 bits. RS code in the proposed architecture should cover not only the flash memory section, but also the AMD random number and redundancy at the same time, thus $m_{RS} \geq 9$. Higher m_{RS} means higher order Galois field and thus more resource, so we choose $m_{RS} = 9$ in this architecture, which means $n_{RS} = 511$. According to [21], we choose $t_{RS} = 4$ to make a balance between the reliability and resource. Thus the RS code in this architecture is a shorten $(511, 503, 4)$ code.

As described in Section IV-A, we choose $t = 1$ and $r = 32$ for AMD code in this paper, so the AMD random numbers and the redundancy are both 32 bits. The random number x and redundant part $f(y, x)$ are both appended four bits of 0 and then divided into four 9-bits parts. So the whole code can be written as $((x_{00}, x_{01}, x_{02}, x_{03}), (y_0, \cdots, y_{455}), (f_0, f_1, f_2, f_3), R)$. In which $x_{0i} \in GF(2^9)$, $y_i \in GF(2^9)$, $f_i \in GF(2^9)$. Thus the shorten RS code in this architecture is a $(472, 464, 4)$ code. In decoding, the decoder first recovers $(x'_{00}, x'_{01}, x'_{02}, x'_{03})$, then combine these data to get $x' \in GF(2^{32})$. Similar process to recover the $f'(y, x)$.

In this architecture, we use RiBM [25] algorithm which has extremely regular structure and thus is highly advantageous in VLSI layout to decode the RS code [16].

C. Clock cycles overhead

Because the AMD code is constructed over $GF(2^{32})$ and the RS code is constructed over $GF(2^9)$ in this architecture, these two codes need different number of clock cycles for encoding and decoding. In the encoding process, AMD encoder will need 128 clock cycles to compute the $x^j y_j (1 \leq j \leq 128)$ and extra three clock cycles for computing x^{131}. On the other hand, the RS decoder will need 456 clock cycles for the information part and 8 extra cycles to encode x and $f(y, x)$.

In decoding, the RS decoder first detects if the codewords are distorted and whether there are more than four errors using the syndrome. If there are no errors, the decoder outputs the messages directly. If there are less than four errors, the decoder will output the corrected codewords. If there are more than four errors, the decoder will be able to detect it with high probability or miscorrect the codewords with a small probability.

The decoding of AMD code and the decoding of RS code can be processed in parallel, thus no extra time overhead will be needed except for the decoding of x' and $f'(y, x)$, and the comparing of $f'(y, x)$ and $f(y', x')$.

V. Security Level and Hardware Complexity Comparisons

We run random error injection simulation to verify the security level of the proposed MLC NAND flash architecture based on the combination of RS and AMD code. In the simulation, we inject random errors into the $x, y, f(y, x)$ and R randomly. The t_{RS} is 4 in this design, so the RS code can correct any up to 4 byte errors. If we inject more than 4 byte errors, the RS decoder will detect most of them but

miscorrect some. The miscorrected errors will be detected by the AMD code with a high probability. We injected 11 billion error patterns into the system and the simulation results show that if we randomly inject from 5 to 12 byte errors into the proposed architecture, about 4.092% will be miscorrected by RS code and most of the miscorrection will be detected by the AMD code with a probability of missing a miscorrected error as low as only 1.77×10^{-8}. This result is very close to the security level described in Table I.

The simulation results show that the proposed architecture based on RS and AMD codes has much higher security level than the architecture based on only the RS code. For the architecture based only on the RS code, even if the attacker just injects random errors distorting from 5 to 12 bytes randomly into the devices, the attacker can bypass the error detection and correction successfully with a probability about 4.092%. But for our proposed architecture, this probability is as low as 1.77×10^{-8}.

Different parameters of AMD code will cause different resource and time overhead and also different security level. For $t = 1$, $t = 2$ and $t = 3$, the time overhead and the number of multipliers are listed in Table II. The redundant bits in Table II include the random number x and the redundancy $f(y, x)$, so it's the sum of r and m shown in Table I.

TABLE II: Resource requirement of AMD code

t	Number of Redundant bits	Clock cycles overhead	Number of multipliers
$t = 1$ [I]	64	131	1
$t = 1$ [II]	64	0	2
$t = 2$	96	15	2
$t = 3$	128	9	3

[I] This architecture will need only one multiplier but more clock cycles

[II] This architecture will need two multipliers but no extra clock clocks

We note that for $t = 1$, we can compute all the x^j before reading y_j. In this case, we need only one multiplier but more clock cycles. Or we can also compute x^j while we are reading the y_j in parallel, then we will need one more multiplier but no extra clock cycles. We note that in this paper, we use the two-multiplier architecture which is shown in Figure 5 for $t = 1$ to build the proposed NAND flash encoder and decoder.

Actually, for $t = 2$ and $t = 3$, we can also decrease the clock overhead to zero by adding more multipliers in the architecture. From Table II we can see that higher security level will require more resources but need less clock cycles to finish the process. But usually, more clock cycles is not a big problem for AMD module because AMD code is always constructed over higher order Galois field than the RS code and both the encoding and decoding process for AMD code will need much less clock cycles than the RS codes.

Encoder and decoder architectures based on RS and AMD codes will cause different latency and area overhead. The encoder and decoder for the proposed RS&AMD architecture and the architecture based only on RS codes have been modelled in Verilog and synthesized in Cadence Encounter RTL Compiler with the Nangate 45nm Opencell library version v2009_07. The designs were placed and routed using Cadence Encounter. The latency, the area overhead of the encoders and the decoders were estimated using Concurrent Current Source (CCS) model under typical operation condition assuming a supply voltage of 1.1V and a temperature of 25 Celsius degree. The synthesis results for the encoder and decoder are shown in Table III. These results were all obtained based on same simulation conditions.

TABLE III: Synthesis Result of the Encoders and Decoders

Architecture	Latency (ns)	Gates	Area (um^2)
RS encoder	0.991	1,177	939.5
RS AMD encoder	1.934	10,231	8,164.6
RS decoder	1.023	60,659	48,406.1
RS AMD decoder	1.258	62,764	50,086.2

From Table III we can see that the proposed architecture based on RS and AMD code will require more resources than the architecture based on only the RS code. While the encoder for the AMD code constructed over higher order Galois field is more complicate than the RS code, the decoder of AMD is much simpler than the one of RS code.

VI. CONCLUSION

In this paper, we presented an architecture based on RS and AMD codes which can detect and correct up to four byte errors and detect strong attacks. Compared with the architecture based only on RS code, our architecture is more efficient for protecting NAND flash devices against malicious fault injection attacks.

The proposed architecture needs only 64 more redundant bits for every 4096 bits of the original flash and eight more clock cycles for the encoding and decoding compared to the original architecture based on the RS code.

We showed that the proposed architecture can provide both high security and reliability. Under the strong attack model, the chance for an attacker to conduct a successful fault injection attack is guaranteed to be as low as 3×10^{-8}. The simulation results show that under random fault injection attack, the miscorrection probability of RS architecture is 4.092%, and the proposed architecture will detect most of these miscorrections with the probability of missing a miscorrected error about only 1.77×10^{-8}. According to the simulation and synthesis results, we can see that our architecture can significantly improve the security level of the MLC NAND flash memories with only minor time and area overhead.

REFERENCES

[1] Z. Wang, M. Karpovsky, and A. Joshi, "Nonlinear multi-error correction codes for reliable mlc nand flash memo-

ries," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 7, pp. 1221 –1234, july 2012.

[2] Y. Maeda and H. Kaneko, "Error control coding for multilevel cell flash memories using nonbinary low-density parity-check codes," in *Defect and Fault Tolerance in VLSI Systems, 2009. DFT '09. 24th IEEE International Symposium on*, oct. 2009, pp. 367 –375.

[3] W. Liu, J. Rho, and W. Sung, "Low-power high-throughput bch error correction vlsi design for multi-level cell nand flash memories," in *Signal Processing Systems Design and Implementation, 2006. SIPS '06. IEEE Workshop on*, oct. 2006, pp. 303 –308.

[4] C. Yang, Y. Emre, and C. Chakrabarti, "Product code schemes for error correction in mlc nand flash memories," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 12, pp. 2302 –2314, dec. 2012.

[5] S. Skorobogatov, "Local heating attacks on flash memory devices," in *Hardware-Oriented Security and Trust, 2009. HOST '09. IEEE International Workshop on*, july 2009, pp. 1 –6.

[6] ——, "Optical fault masking attacks," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2010 Workshop on*, aug. 2010, pp. 23 –29.

[7] ——, "Flash memory 'bumping' attacks," in *Proceedings of the 12th international conference on Cryptographic hardware and embedded systems*, ser. CHES'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 158–172. [Online]. Available: http://dl.acm.org/citation.cfm?id=1881511.1881526

[8] M. Karpovsky, K. Kulikowski, and A. Taubin, "Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard," in *Dependable Systems and Networks, 2004 International Conference on*, june-1 july 2004, pp. 93 – 101.

[9] M. Karpovsky and A. Taubin, "New class of nonlinear systematic error detecting codes," *Information Theory, IEEE Transactions on*, vol. 50, no. 8, pp. 1818 – 1819, aug. 2004.

[10] K. Kulikowski, Z. Wang, and M. Karpovsky, "Comparative analysis of robust fault attack resistant architectures for public and private cryptosystems," in *Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC '08. 5th Workshop on*, aug. 2008, pp. 41 –50.

[11] Z. Wang and M. Karpovsky, "Algebraic manipulation detection codes and their applications for design of secure cryptographic devices," in *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, july 2011, pp. 234 –239.

[12] ——, "Reliable and secure memories based on algebraic manipulation correction codes," in *On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International*, june 2012, pp. 146 –149.

[13] A. Greg, F. Al, M. Duane, and B. Reaves, "Intel strataflash memory technology overview," Intel, Tech. Rep., .

[14] , "Slc vs. mlc: An analysis of flash memory," Super Talent Technology Corporation, Tech. Rep., .

[15] ——, "Choosing the right nand," Micron Technology, Inc., Tech. Rep., .

[16] B. Chen, X. Zhang, and Z. Wang, "Error correction for multi-level nand flash memory using reed-solomon codes," in *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, oct. 2008, pp. 94 –99.

[17] "Nand flash 101: An introduction to nand flash and how to design it in to your next product," Micron Technology, Inc., Tech. Rep., 2006.

[18] *64Gb, 128Gb, 256Gb, 512Gb Asynchronous/Synchronous NAND Features*.

[19] "Tn-29-05: Ecc module for nand flash via xilinx spartan-3 fpga," Micron Technology, Inc., Tech. Rep., 2005.

[20] "Tn-29-06: Micron nand flash controller via xilinx spartan-3 fpga," Micron Technology, Inc., Tech. Rep., 2005.

[21] M. Mariano, "Ecc options for improving nand flash memory reliability," Micron Technology, Inc., Tech. Rep., 2012.

[22] R. Cramer, Y. Dodis, S. Fehr, C. Padr, and D. Wichs, "Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors," in *Advances in Cryptology C EUROCRYPT 2008*, ser. Lecture Notes in Computer Science, N. Smart, Ed. Springer Berlin / Heidelberg, 2008, vol. 4965, pp. 471–488.

[23] Z. Wang and M. Karpovsky, "New error detecting codes for the design of hardware resistant to strong fault injection attacks."

[24] S. Lin and D. J. Costello, *Error Control Coding, Second Edition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004.

[25] D. Sarwate and N. Shanbhag, "High-speed architectures for reed-solomon decoders," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, no. 5, pp. 641 –655, oct. 2001.