Kalpa Publications in Computing

Volume 7, 2018, Pages 68–82

PROOFS 2018. 7th International Workshop
on Security Proofs for Embedded Systems

Detection and Correction of Malicious and Natural Faults in Cryptographic Modules*

Batya Karp[1], Mael Gay[2], Osnat Keren[1], and Ilia Polian[2]

[1] Faculty of Engineering, Bar-Ilan University, Ramat Gan, Israel
[2] Institute of Computer Engineering and Computer Architecture, University of Stuttgart, Germany

Abstract

Today's electronic systems must simultaneously fulfill strict requirements on security and reliability. In particular, their cryptographic modules are exposed to faults, which can be due to natural failures (e.g., radiation or electromagnetic noise) or malicious fault-injection attacks. We present an architecture based on a new class of error-detecting codes that combine robustness properties with a minimal distance. The new architecture guarantees (with some probability) the detection of faults injected by an intelligent and strategic adversary who can precisely control the disturbance. At the same time it supports automatic correction of low-multiplicity faults. To this end, we discuss an efficient technique to correct single errors while avoiding full syndrome analysis. We report experimental results obtained by physical fault injection on the SAKURA-G FPGA board.

1 Introduction

With the transition to the cyberphysical system (CPS) paradigm, digital circuits are increasingly used for functions that are safety- and security-critical at the same time. For example, emerging car electronics will have to support conventional safety features (like anti-lock braking system or airbag control) and advanced electronic drive-assist functions, which are safety-relevant and must be realized in a failure-proof manner. However, the same electronics will provide the customer with access to social networks and payment functions over the Internet, which makes it a target of deliberate security attacks. Moreover, emerging electronic systems are designed to operate in harsh environments (including temperature extremes, vibration, humidity), increasing the chance of failures due to natural causes: noise and aging. At the same time, their components often lack a "protective perimeter" known from conventional servers located in an access-controlled building and operated by authorized personnel. Cyberphysical infrastructures, vehicles and production systems have parts designed to be placed in public spaces and accessible by anybody, including potential attackers. Therefore, malicious attacks on hardware components can be expected and must be counteracted.

A variety of defences on different abstraction levels have been suggested against natural failures and malicious attacks alike [14]. Here, we relate only to failures and attacks that create

*This research was supported by the ISRAEL SCIENCE FOUNDATION (grant No. 923/16) and by the DFG (German Research Foundation) project Po 1220/7-2.

a tangible and observable change in the input-output behavior of a circuit. In the context of natural failures, we do not consider mechanisms with purely parametric effects (e.g., ones which increase the circuit's power consumption but have no pronounced implications on the logic level). In the malicious case, we restrict ourselves to attacks that actively manipulate the operation of a circuit; purely passive analysis [16] is not in scope of this paper. Such *fault-injection attacks* [21] can aim at disrupting the application's control flow (e.g., jumping over password checks [27] or, in case of cryptographic circuitry, extraction of secret keys via differential fault analysis [3] or fault-sensitivity analysis [15]. We refer by the term "fault" to any logical effect due to either a natural failure or a malicious (active) attack.

Out of various countermeasures against natural and malicious faults, approaches based on error-detecting codes (EDCs) stand out. They can be applied to protect memories, communication channels and combinational circuitry. In the case of natural failures or poorly-controlled malicious fault injections, EDC competes with space- and time-redundancy techniques, including duplication, modular redundancy, and commit-rollback [14]. However, an intelligent attacker with high-precision fault-injection equipment can circumvent this protection by injecting multiple faults into redundant copies such that they cancel each other out. Recent developments such as dual-beam laser fault injectors make this threat practical [24]. On the contrary, special security-oriented EDCs have been proposed [11, 25]. They are designed to counteract a strategic attacker who knows the defences and aims at circumventing them. It can be shown that all *linear codes* offer limited protection against attacks under this assumption, as there are faults that are never detected. Therefore, the usual EDCs like parity or Hamming codes are inadequate in this case, and dedicated non-linear security-oriented codes are required.

In this paper, we consider and optimize architectures that are designed to handle natural and malicious faults. The architectures are based on a recent code construction, the Rabii-Keren (RK) codes [22] in their generalized form [23]. RK codes are defined on the code alphabet of size q, which is a power of 2. For example, $q = 2^4$ ($q = 2^8$) is a natural choice for a circuit with a state organized in 4-bit nibbles (8-bit bytes): a fault that affects k nibbles (bytes) directly corresponds to an error of multiplicity k which can be detected and/or corrected. RK codes combine three properties: a user-defined distance (and therefore the possibility of error correction), a low masking probability (a metric for resilience against malicious attacks), and a high rate (ratio between data and check bits). The code-based architecture is summarized in Fig. 1; notice that the syndrome of the code is intended for processing at the system level, which can also take into account further sources of information to decide whether the detected fault was malicious or not and whether it should be corrected or an alarm should be raised.

The feature to reliably correct errors up to a certain multiplicity is useful for both natural and malicious faults. If a fault could be corrected, the system can proceed with its regular operation, while error-detection without correction requires some handling, e.g., re-execution of the affected computation. Therefore, error-correction is attractive in particular for safety-critical systems with real-time requirements, like aircraft or chemical plants which cannot simply stop operation and wait for fault handling. Note that even if an error can be corrected, the system may still have to record the fault event. Moreover, a system may monitor the faults to decide which of them are due to natural causes or due to malicious tampering. The observed fault effect (fault rate and multiplicity) can be an input to such a monitoring procedure, but in general, further inputs are needed for reliably distinguishing between natural and malicious faults. For instance, a system which operates in a high-radiation environment may be equipped with a radiation sensor; if it reports high radiation, then the fault is likely natural.

Both natural and malicious faults can result in correctable or uncorrectable errors. If a natural fault stems from a minor disturbance (e.g., a low-energy particle discharge), it will

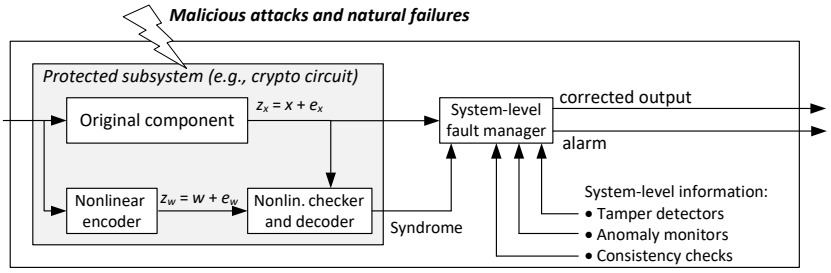

Figure 1: Detection, correction and validation architecture

likely affect only a few bits and the resulting error will be corrected and no further action will be needed. Malicious attacks often have quite strong effects; e.g., if the circuit's clock is glitched or the voltage is lowered [2], many outputs will typically be affected. Errors stemming from such attacks should be detected but it is unrealistic to reliably correct them. Even very pinpointed attacks, like laser fault injections which aim at flipping one particular logic gate output or memory cell, typically start with a tuning phase when (detectable but uncorrectable) multi-bit errors are produced. If an attack is run with a restricted fault model, like single-byte faults in Tunstall's attack on AES [26], the error can be corrected by the RK code with single-error correction capability (distance 3 or larger). Since the correction would happen within the circuit, the attacker would observe no fault-affected ciphertext and therefore would not be able to mount the attack.

In this paper, we present an efficient correction procedure for single errors using RK codes. The procedure, based on a compact *Error Coefficient and Location Table* (ECLT) is a substantial improvement compared with the regular syndrome analysis. We report experiments using a clock glitch based fault injector on cryptographic circuits (full- and small-scale AES [4], LED, PRESENT) on the SAKURA FPGA board. The experiments show that the architectures are capable of detecting errors of arbitrary multiplicity and correcting single errors, reliably recognizing erroneous corrections. The architectures are especially effective when they combine codes with a distance larger than 3 and an additional system-level validation by an outer code.

The remainder of the paper is organized as follows. Section 2 gives background on error-detecting codes. Sections 3 and 4 outline the detection and correction architecture for codes of distance 3 and distance larger than 3, respectively. Experimental results are reported in Section 5. Section 6 concludes the paper.

2 Security oriented codes

Given a vector space $\mathbb{F}_q^n$ of dimension n over $\mathbb{F}_q = GF(q)$, a code $\mathcal{C}$ is a subset of size $|\mathcal{C}|$. A code $\mathcal{C}$ is said to be systematic if every codeword is of the form $c = (x, w(x))$ where $x \in \mathbb{F}_q^k$ is the information portion, and $w \in \mathbb{F}_q^r$ is the redundancy portion.

Let $c \in \mathcal{C}$ be the correct codeword and denote by $\hat{c}$ the distorted word. It is convenient to model a fault that distorts a symbols as an additive error $e = \hat{c} - c$ of Hamming weight a; a is called the error multiplicity. In this paper an error is represented as a q-ary vector $e = (e_x, e_w) \in \mathbb{F}_q^n$ where e_x is the error in the information portion and e_w is the error in the redundancy portion. In addition, the multiplicity of a malicious error is considered here as arbitrary, i.e., $1 \leq a \leq n$.

The effectiveness of a reliability oriented code is usually measured in terms of its decoding error, that is the probability the decoder will fail to correctly decode a tampered word. Since the most probable error has the lowest Hamming weight, these codes are evaluated using the

minimum distance d which is the minimal Hamming distance between all codewords. A security oriented code is evaluated using the maximal error masking probability, $\overline{Q}$, which is the maximum probability that any non-zero error e will map a codeword to another codeword in $\mathcal{C}$. In this sense, when codes are analysed for reliability, the average case is considered, whereas the analysis for security is based on the worst case scenario.

The upper bound on the minimum distance of the code d is *linearly* dependent on the number of redundancy symbols, $d \leq r + 1$; whereas the lower bound on $\overline{Q}$ is *exponentially* dependent on the number of redundancy symbols.

A security oriented code can have a deterministic encoder [9, 1, 17, 10] or incorporate randomness[5, 29, 19] (the latter includes the *non-malleable codes* [6]). The error detection capabilities of codes with random-encoding depend on the entropy of the random number generator (RNG). However, the hardware implementation of a true RNG is expensive and difficult, and the RNG must be shielded from fault injection attacks which could neutralize it. For this reason, codes with deterministic encoding are an attractive alternative. In fact, when properly designed, such codes can be more effective than random codes of the same rate [12]. This work deals with robust codes, which are codes with deterministic encoding.

Notice that an additive error e is masked by a codeword $c \in \mathcal{C}$ if $c \oplus e \in \mathcal{C}$. Similarly, an error e is detected by a codeword $c \in \mathcal{C}$ if $c \oplus e \notin \mathcal{C}$. This leads to the following definition of the error masking probability:

Definition 1. *The error masking probability of an error e, $Q(e)$, is the probability that an error e will be masked by the codewords of $\mathcal{C}$. That is,*

$$Q(e) = \sum_{c \in \mathcal{C}} Pr(c)\delta_C(c \oplus e)$$

where $Pr(c)$ is the probability of the codeword c and δ_C is the characteristic function of $\mathcal{C}$,

$$\delta_C(z) = \begin{cases} 0 & \text{if } z \notin \mathcal{C} \\ 1 & \text{if } z \in \mathcal{C} \end{cases}$$

In the case of uniformly distributed codewords, it is convenient to represent the error masking probability in terms of the autocorrelation function of the code. That is, $Q(e) = R(e)/q^k$ where

$$R(e) = |\{c \,|\, c, c + e \in \mathcal{C}\}| = \sum_{z \in \mathbb{F}_q^n} \delta_C(z)\delta_C(z \oplus e).$$

For some codes the set of codewords that mask an error form a linear subspace. Thus a good code will be a union of small disjoint subspaces. On the relationship between the autocorrelation function and the representation of a code as a union of disjoint subspaces see [13].

The detection kernel of a code, denoted K_d, contains all the error vectors that are never detected by the codewords of $\mathcal{C}$, i.e. all the errors that are masked with probability $Q(e) = 1$.

Definition 2. *(Robust codes) A code $\mathcal{C}$ is called robust if any non-zero error can be detected with some probability greater than zero. Meaning, $Q(e) < 1$ for any non-zero error e, or alternatively, $K_d = \{\mathbf{0}\}$.*

Definition 3. *(Partially Robust codes) A code $\mathcal{C}$ is partially robust if it has a detection kernel of size $1 < |K_d| < |\mathcal{C}|$.*

Linear codes have a detection kernel $K_d = \mathcal{C}$, and therefore linear codes are not robust and cannot be used for security.

There are two known basic high rate binary systematic robust codes: the Quadratic Systematic (QS) code [9], and the Punctured Cubic (PC) code [1], [17]. All other systematic robust codes use these codes as base codes. While the QS code is an optimum robust code when $k = 2sr$ and q is any power of a prime number, and the PC code is a close to optimum robust code for any $1 < r \leq k$ and q is a power of two, neither of these codes have correction capabilities. Some minimum distance partially robust codes exist, for example the Vasil'ev code [28], the Phelps code [20], the one switching code, and the generalized cubic code [7], [18]. While these codes provide the wanted correction capabilities, they are not robust.

In a recent paper Rabii and Keren introduced a construction for a new class of non-linear robust q-ary codes with $q = 2^m$ and error correction capability [22]. The RK codes have higher code rate than concatenated codes, and at the same time, they are more effective [23]. Rabii and Keren did not present encoding, decoding, and error correction algorithms, and their code was not implemented and tested in a realistic environment. This paper aims to close this gap.

The RK code is a non-linear robust q-ary code with $q = 2^m$ and error correction capability. The code is built upon systematic linear codes $[n, k, d]_q$ where the $n - k$ redundant symbols that were originally allocated to increase the minimum distance of the code, are modified to provide both correction capability and robustness. The following (generalized) definition of the RK code is taken from [23].

Construction 1. *(Rabii-Keren code) Let $f : \mathbb{F}_{2^m} \mapsto \mathbb{F}_{2^m}$ be an Almost Perfect Nonlinear (APN) bijective function, and let $G = (I|A)$ be a generator matrix of a systematic linear q-ary code $\mathcal{C}$ with minimum distance d_L where $A = \{a_{ij}\}_{i,j=1}^{k,r}, a_{ij} \in \mathbb{F}_{2^m}$. Let $x = (x_1, x_2, \ldots, x_k)$ where $x_i \in \mathbb{F}_{2^m}$ for $1 \leq i \leq k$. Code $\tilde{\mathcal{C}}$ is defined as follows,*

$$\tilde{\mathcal{C}} = \{(x, w) : x \in \mathbb{F}_{2^m}^k, w = (w_1, w_2, \ldots, w_r) \in \mathbb{F}_{2^m}^r, w_j = \sum_{i=1}^{k} a_{ij} f(x_i)\}$$

The robustness and the effectiveness of the RK code is due to the high non-linearity of f. For odd values of m, the best function to use is the cubic function, $f(x) = x^3$, which is an invertible APN function of (relatively) small implementation cost [23]. However, as shown in [23], it is possible to use other functions, for example $f(x) = x^{-1}$, with even values of m and to pay with a higher error masking probability $Q(e)$. Namely, the error masking probability of the codes is $Q(e) \leq 2/q$ for odd values of m and $4/q$ for even m.

In what follows we present a novel low-cost implementation of error detection and correction architectures based on RK codes. We start with codes of distance $d = 3$ and then generalize the decoder for codes with $d > 3$. We demonstrate the effectiveness of these codes in correcting single erroneous SBox's output and detecting multi-erroneous SBox's outputs in Section 5.

3 Detection and Correction Architecture for Rabii-Keren Codes of Distance 3

In order to use the RK construction, one has to construct a *systematic* generator matrix for the linear code. Algorithm 1 constructs a generator matrix based on the check matrix of a shortened BCH code over an alphabet of size q. Note that $q = 16$ for SBoxes that work on 4-bit nibbles, and $q = 256$ for bytes.

Consider, for example, the $(n = 19, |\mathcal{C}| = 2^{4 \cdot 16}, d = 3)_{16}$ Rabii-Keren code for protecting 16 4-bit SBoxes ($q = 16$) by using 12 redundant bits; The code is based on the $[19, 16, 3]_{16}$ shortened BCH code with distance $d = 3$. The check matrix of the shortened BCH code, $H_{orig,3}$, is the following:

Algorithm 1 Construct A

1: Choose the size of the alphabet, q. (In this paper, $q = 16 = 2^4$.)
2: Choose the dimension of the code, k.
3: Choose the distance of the code, d. (d is assumed to be relatively small.)
4: Determine the root field using the rule: $q^m - 1 > k + (d-1)m$. We want to choose the minimal m that will uphold the condition. (In this paper $m = 2$ was sufficient.)
5: Choose b the power of first root in the sequence of $d-1$ consecutive roots of the generator polynomial. (Usually $b = 1$ is chosen for a simple code, however we found that by choosing $b = 0$ we could use a smaller number of redundancy symbols.)
6: Once d, b, q, m are determined, find r the degree of the generator polynomial.
7: Shorten the code by defining $\tilde{n} = k + r$ given the r that we found.
8: Represent the shortened check matrix

$$H_{orig,d} = \left(\begin{array}{cccc|ccc} 1 & \alpha^b & \cdots & \alpha^{b(k-1)} & \alpha^{bk} & \cdots & \alpha^{b(\tilde{n}-1)} \\ 1 & \alpha^{(b+1)} & \cdots & \alpha^{(b+1)(k-1)} & \alpha^{(b+1)k} & \cdots & \alpha^{(b+1)(\tilde{n}-1)} \\ \vdots & & & \vdots & \vdots & & \vdots \\ 1 & \alpha^{(b+d-2)} & \cdots & \alpha^{(b+d-2)(k-1)} & \alpha^{(b+d-2)k} & \cdots & \alpha^{(b+d-2)(\tilde{n}-1)} \end{array}\right) = (H_l | H_r) \quad (1)$$

9: Compute the check matrix $H_d = (A_d \,|I)$, where $A_d = H_r^{-1} H_l$. Note that H_r is an invertible matrix since the code is cyclic.

$$H_{orig,3} = \left(\begin{array}{cccccccccccccccc|ccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 11 & 5 & 9 & 3 & 1 & 5 & 0 & 13 & 7 & 3 & 3 & 8 & 11 & 6 & 11 & 4 & 6 & 15 \\ 0 & 7 & 11 & 2 & 2 & 1 & 3 & 4 & 3 & 9 & 4 & 10 & 7 & 2 & 12 & 4 & 7 & 0 & 1 \end{array}\right) \begin{array}{l} \leftarrow \alpha^0 \\ \leftarrow \alpha^1 \end{array}$$

Note that the matrix $H_{orig,d}$ defined in Step 8 of Algorithm 1 has elements from $GF(q^m)$. Since the fields $GF(q^m)$ and $GF(q)^m$ are isomorphic, an element in α^i in $H_{orig,d}$ can be represented as an m tuple $(a_{i,0}, \ldots a_{i,m-1})$. In our case, α^i corresponds to the pair $(a_{i,0}, a_{i,1}) \in GF(16)^2$. This equivalent representation was used to define the check matrix $H_{orig,3}$ over $GF(16)$; namely, the numbers in $H_{orig,3}$ represent elements in $\mathbb{F}_{2^4}$ defined with the polynomial $\pi(x) = x^4 + x + 1$.

To simplify the implementation complexity we work with a *systematic* check matrix $H_3 = (A_3, I)$. The matrix A_3 used for RK construction was obtained by multiplying $H_{orig,3}$ by H_r^{-1},

$$A_3 = \begin{pmatrix} 6 & 2 & 12 & 2 & 5 & 12 & 3 & 10 & 0 & 15 & 12 & 10 & 4 & 6 & 9 & 15 \\ 9 & 1 & 3 & 11 & 6 & 4 & 1 & 11 & 10 & 12 & 14 & 4 & 4 & 13 & 3 & 5 \\ 14 & 2 & 14 & 8 & 2 & 9 & 3 & 0 & 11 & 2 & 3 & 15 & 1 & 10 & 11 & 11 \end{pmatrix}.$$

The general process of error correction is the same for a linear code and a non-linear code (the linear version can be obtained by skipping non-linear operations, in this case the inversion). Let $c = (x, w)$ be the correct codeword, and let $z = (z_x, z_w) = (x + e_x, w + e_w)$ be the word received by the checker, and define $y = (y_x, y_w) = (z_x^{-1}, z_w)$. Fig. 3 shows the single error correction architecture for $f(x) = x^{-1}$, and the general correction process described in Algorithm 2 does the following:

The linear decoder computes the syndrome (Algorithm 2, line 1)

$$s = H_d \cdot y^T = (A_d, I)((x + e_x)^{-1}, x^{-1} A_d^T + e_w)^T = (((x + e_x)^{-1} + x^{-1}) A_d^T + e_w)^T$$

Algorithm 2 Correct single error

1: Calculate syndrome $s = H_d y^T$.
2: Normalize the first $(m+1)$ symbols of the syndrome $\hat{s} = (s_1/s_j, s_2/s_j, ..., s_{m+1}/s_j)^T$.
3: Find normalized syndrome in ECLT and determine f_i and i (see, e.g., Tables 1 and 2).
4: If found, use $\hat{y} = y - s_j f_i x^i$ to correct error.
5: Update the syndrome $\tilde{s} = s - s_j f_i h_i$.
6: If the syndrome is equal to zero, there was a single error.
7: Else, more than one error occurred.

In general, each syndrome is associated with an error vector, $\hat{e} = (\hat{e}_x, \hat{e}_w)$, and the decoded $\hat{z}$ is

$$\hat{z} = ((z_x^{-1} + \hat{e}_x)^{-1}, z_w + \hat{e}_w).$$

If the distance $d(z, c)$ is less than or equal to the correction capability of the code, the obtained $\hat{z}$ is in fact the desired codeword c. In other words, if the number of distorted symbols is less than or equal to the correction capability of the code, the decoder will work as desired. Recall that an error is detected if the syndrome s is not zero. To this end, the decoding process is simply to re-encode the information portion $\hat{x} = x + e_x$ into $\hat{w}$, and compare it with the redundant portion of the output vector z; i.e., compare the q-ary vectors $\hat{w}$ and $w + e_w$. This can be seen in Fig. 2 (which can be considered a more detailed version of "protected subsystem" in Fig. 1).

Next, the decoder checks whether the error is correctable (Algorithm 2, lines 2-4); Denote by s_j the first value in s that is not equal to zero and by f_i the first value in h_i that is not equal to zero. Calculate $\hat{s} = s/s_j$ and $\hat{h}_i = h_i/f_i$. $\hat{s} = \hat{h}_i$ and therefore $s = e_i \hat{h}_i / f_i$. Overall we can find the value of $e_i = s_j f_i$ and then correct the single error as $\hat{y} = y - s_j f_i x^i$ where y is the received word. We can easily find s_j when calculating the original syndrome and f_i and i can be stored in an error-coefficient along with the possible corresponding $\hat{h}_i$ values. The *error-coefficient and location table* (ECLT) is a table of size $n \times (r+2)$ where each row contains a different $\hat{h}_i$ of length r, the column indicator i, and the factor f_i. Finally, the decoder verifies that the error has been corrected (lines 5-6).

For example, the table for the $(19, 2^{64}, 3)_{16}$ Rabii-Keren code (described by the above matrix A_3) is shown in Table 1.

Note that the original decoder (based on the original BCH check matrix $H_{orig,3}$) requires a table of $(q-1) \cdot n = 15 \cdot 19$ entries, each with a syndrome vector (3 q-ary symbols or 12 bits), position (5 bits), and the value of the error (4 bits). This amounts to $15 \cdot 19 \cdot 21$ bits. In contrast, the proposed architecture in Fig. 3 employs a table with 19 entries of 18 bit each.

Clearly, when protecting a full scale cipher with 8-bit SBox, the size of the table can be reduced from $255n$ to n entries. The following example demonstrates how the decoding works for the $(19, 2^6 4, 3)_{16}$ RK code described above:

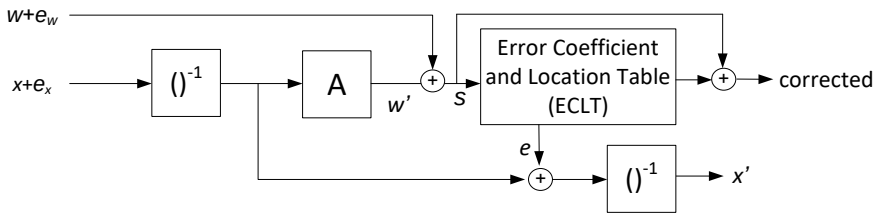

Figure 2: Decoding process for a Rabii-Keren non-linear code where $f(x) = x^{-1}$

Table 1: ECLT for RK code with parameters $n = 19, k = 16, d = 3$

$\hat{h}_i$	i	f_i
1 10 12	0	7
1 9 1	1	9
1 13 6	2	10
1 12 4	3	9
1 15 5	4	11
1 14 5	5	10
1 14 1	6	14
1 13 0	7	12
0 1 13	8	12
1 10 3	9	8
1 6 13	10	10

$\hat{h}_i$	i	f_i
1 5 8	11	12
1 1 13	12	13
1 5 3	13	7
1 6 5	14	2
1 14 7	15	8
1 0 0	16	1
0 1 0	17	1
0 0 1	18	1

Example 1. *Take for example the received word* $\hat{x} = [9, 11, 9, 3, 11, \underline{\mathbf{3}}, 2, 2, 12, 7, 1, 13, 1, 9, 3, 5]$ *and the predicted word* $x = [9, 11, 9, 3, 11, \underline{\mathbf{14}}, 2, 2, 12, 7, 1, 13, 1, 9, 3, 5]$.

Using Algorithm 2, we can detect and correct one error:

1. *After encoding* $\hat{x}$ *using the RK code, the redundancy is* $w = [0, 8, 10]$ *and the syndrome is calculated as* $s = [3; 1; 15]$.

2. *Normalize the syndrome where* $s_j = 3$. *Using calculations over* $GF(16)$, *obtain* $\hat{s} = \hat{h} = [1; 14; 5]$.

3. *Using the ECLT, we find that* $i = 5$, *which corresponds to an error in the sixth symbol of the information portion, and* $f_i = 10$.

4. *We can now calculate* $e_i = s_j f_i = 13$ *and the corresponding error vector* $e = [0, 0, 0, 0, 0, 13, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$. *As can be seen in Fig. 3, in order to find the corrected codeword,* e *must be subtracted from* $\hat{c}^{-1}$ *and the result inverted once again.*

   $y = \hat{c}^{-1} = [2, 5, 2, 14, 5, \underline{\mathbf{14}}, 9, 9, 10, 6, 1, 4, 1, 2, 14, 11, 0, 15, 12]$.

   $y - e = \hat{c}^{-1} - e = [2, 5, 2, 14, 5, \underline{\mathbf{3}}, 9, 9, 10, 6, 1, 4, 1, 2, 14, 11, 0, 15, 12]$. *Once inverted, we do in fact receive the predicted codeword* $c = [9, 11, 9, 3, 11, \underline{\mathbf{14}}, 2, 2, 12, 7, 1, 13, 1, 9, 3, 5, 0, 8, 10]$.

The updated syndrome is then $\tilde{s} = s - s_j f_i h_i = (3, 1, 15)^T - 13 \cdot (12, 4, 9)^T = \mathbf{0}$. *We have successfully corrected the single error; the flag "corrected" will be set to 1.*

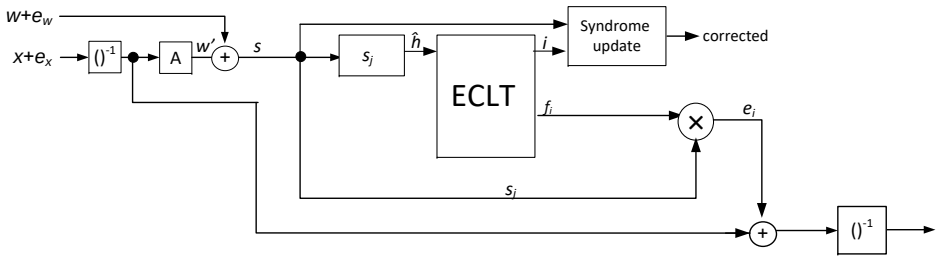

Figure 3: Low-complexity single error correction architecture based on ECLT

4 Detection and Correction Architecture for Rabii-Keren Codes of Distance Larger Than 3

Small-scale natural faults or some precise faults injected by a sophisticated attacker manifest themselves as a single erroneous symbol. However, there is an advantage in protecting the system with a code of distance $d > 3$. A code of distance $d > 3$ allows the correction of a single erroneous symbol (i.e., SBox output) and avoids a miscorrection of up to $d-2$ erroneous symbols.

In this section we show how the idea presented in the previous section can be generalized to codes with $d > 3$. We show that instead of using a table with $(q-1)n$ entries each of size

$$\underbrace{(1+(d-2)m)\log_2 q}_{\text{syndrome}} + \underbrace{\log_2(n)}_{\text{error-location}} + \underbrace{\log_2 q}_{\text{error-value}}$$

bits, one can use a table with n entries each of size $(m+2)\log_2 q + \log_2(n)$.

Let us start with an example before the correctness of this statement is proven.

Example 2. *Consider a* $(23, 2^{16}, 5)_{16}$ *Rabii-Keren code for SBoxes with* $q = 16$ *and distance* $d = 5$. *The check matrix of the code* $H_{orig,5}$ *is the following:*

$$H_{orig,5} = \left(\begin{array}{cccccccccccccccc|cccc|l}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \leftarrow \alpha^0 \\
1 & 11 & 5 & 9 & 3 & 1 & 5 & 0 & 13 & 7 & 3 & 3 & 8 & 11 & 6 & 11 & 4 & 6 & 15 & 13 & 3 & 10 & 6 & \leftarrow \alpha^1 \\
0 & 7 & 11 & 2 & 2 & 1 & 3 & 4 & 3 & 9 & 4 & 10 & 7 & 2 & 12 & 4 & 7 & 0 & 1 & 15 & 12 & 12 & 6 & \\
\hline
1 & 5 & 3 & 5 & 13 & 3 & 8 & 6 & 4 & 15 & 3 & 6 & 0 & 1 & 10 & 15 & 15 & 7 & 8 & 9 & 8 & 5 & 9 & \leftarrow \alpha^2 \\
0 & 11 & 2 & 3 & 3 & 4 & 7 & 12 & 7 & 1 & 12 & 6 & 11 & 3 & 9 & 12 & 11 & 0 & 4 & 14 & 9 & 9 & 15 & \\
\hline
1 & 9 & 5 & 7 & 8 & 11 & 15 & 10 & 0 & 10 & 15 & 11 & 8 & 7 & 5 & 9 & 1 & 1 & 9 & 5 & 7 & 8 & 11 & \leftarrow \alpha^3 \\
0 & 2 & 3 & 9 & 7 & 4 & 1 & 12 & 11 & 11 & 12 & 1 & 4 & 7 & 9 & 3 & 2 & 0 & 2 & 3 & 9 & 7 & 4 &
\end{array}\right)$$

The corresponding matrix A_5 *is*

$$A_5^T = \begin{pmatrix}
1 & 12 & 0 & 15 & 0 & 14 & 13 & 6 & 15 & 12 & 3 & 9 & 15 & 10 & 0 & 15 \\
11 & 12 & 12 & 3 & 15 & 8 & 8 & 2 & 5 & 2 & 2 & 15 & 10 & 13 & 10 & 3 \\
5 & 2 & 12 & 10 & 3 & 12 & 4 & 5 & 4 & 12 & 13 & 9 & 9 & 14 & 13 & 12 \\
10 & 4 & 2 & 0 & 10 & 5 & 7 & 13 & 9 & 5 & 1 & 8 & 5 & 1 & 14 & 1 \\
7 & 8 & 4 & 9 & 0 & 6 & 0 & 6 & 6 & 11 & 12 & 11 & 3 & 6 & 1 & 5 \\
15 & 15 & 8 & 14 & 9 & 5 & 1 & 4 & 12 & 14 & 9 & 2 & 1 & 15 & 6 & 11 \\
12 & 0 & 15 & 0 & 14 & 13 & 6 & 15 & 12 & 3 & 9 & 15 & 10 & 0 & 15 & 14
\end{pmatrix}$$

It follows from the definition of the BCH code that the $3 \times (k+3)$ top left submatrix of $H_{orig,5}$ is identical to $H_{orig,3}$, whereas A_3 and A_5 are different. Nevertheless, the first three $(1+m)$ rows of A_5 have the property that any two or less columns are linearly independent:

Theorem 1. *The first* $(1+m)$ *symbols of the syndrome* s, $s_{[0:m]} \in GF(q)^{m+1}$, *uniquely define the location of the erroneous symbol and its value.*

Proof. To prove the theorem it is sufficient to show that, for the linear code, for any two errors e_1 and e_2 of Hamming weight one, the corresponding partial syndromes $s_{1,[0:m]}$ and $s_{2,[0:m]}$ are distinct.

Recall that the code is of distance $d > 3$, thus the syndromes $s_1 = (A_d|I)e_1^T$ and $s_2 = (A_d|I)e_2^T \in GF(q)^{1+m(d-2)}$ are distinct. Assume (by contradiction) that $s_{1,[0:m]} = s_{2,[0:m]}$. Then,

$$(A_d|I)(e_1 - e_2)^T = (\mathbf{0}_{m+1}, \Delta s)^T$$

Table 2: ECLT for RK code with parameters $n = 23, k = 16, d = 5$

$\hat{h}_i$	i	f_i
1 11 5	0	1
1 1 7	1	10
0 1 1	2	10
1 11 15	3	8
0 1 11	4	8
1 11 7	5	3
1 6 3	6	4
1 14 8	7	7
1 14 6	8	8
1 7 1	9	10
1 15 10	10	14
1 13 1	11	2
1 15 4	12	8
1 3 4	13	12
0 1 3	14	12
1 11 10	15	8
1 0 0	16	1
0 1 0	17	1
0 0 1	18	1

where $\Delta s \in GF(q)^{m(d-3)}$. Since e_1, e_2 consists of elements in $GF(q)$, and $GF(q) \subset GF(q)^m$, the last equality can be written over $GF(q^m)$ as follows:

$$H_{orig,d}(e_1 - e_2)^T - H_r(\mathbf{0}_2, \tilde{\Delta}s)^T = 0$$

where $\tilde{\Delta}s$ is a vector of length $d-2$ over $GF(q^m)$. In other words, we get that the sum of at most $2 + (d-3)$ columns of $H_{orig,d}$ are linearly dependent. This contradicts the fact that $H_{orig,d}$ defines a code of distance d. □

For example, the columns in first $1+m = 3$ rows of A_5 are all distinct, moreover, one is not a multiple of the other. Hence the location of a single erroneous nibble is uniquely defined by $\hat{s}_{[0:2]}$. Table 2 contains the location and error coefficient value for the correctable syndromes. Note that use of Rabii-Keren code as a non-linear error correcting code would require a table with $15^2 \cdot \binom{23}{2} + 15 \cdot \binom{23}{1} = 57,270$ entries, each of $7 \cdot 4 + 2(5+4)) = 46$ bits (2,634,420 bits in total). In contrast, the architecture presented in Fig. 3 requires a table with 23 entries each of $3 \cdot 4 + (5+4) = 21$ bits (483 bits in total).

The following example demonstrates the decoding technique for a $(23, 2^6 4, 5)_{16}$ Rabii-Keren code using the matrix A_5 and the shortened ECLT as described above:

Example 3. *Take the predicted word* $x = [5, 5, 0, \underline{\mathbf{2}}, 13, 12, 1, 2, 10, 12, 11, 13, 14, 13, 12, 4]$ *and the received word* $\hat{x} = [5, 5, 0, \underline{\mathbf{6}}, 13, 12, 1, 2, 10, 12, 11, 13, 14, 13, 12, 4]$.

Using Algorithm 2, we can detect and correct one error as follows:

1. *After encoding* $\hat{x}$ *using the RK code, the redundancy is* $w = [15, 14, 4, 11, 3, 13, 8]$ *and the syndrome is calculated as* $s = [5, 1, 6, 0, 7, 11, 0]$.

2. *Normalize the first* $m + 1 = 3$ *symbols of the syndrome where* $s_j = 5$. *Using calculations over* $GF(16)$, *obtain* $\hat{s} = \hat{h} = [1; 11; 15]$.

3. *Using the ECLT, we find that* $i = 3$, *which corresponds to an error in the fourth symbol of the information portion, and* $f_i = 8$.

4. *We can now calculate* $e_i = s_j f_i = 14$ *and the corresponding error vector* $e = [0, 0, 0, 14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$.

Again, e must be subtracted from $\hat{c}^{-1}$ and the result must be inverted once again.

$y = \hat{c}^{-1} = [11, 11, 0, \underline{\mathbf{7}}, 4, 10, 1, 9, 12, 10, 5, 4, 3, 4, 10, 13, 8, 3, 13, 5, 14, 4, 15]$.

$y - e = \hat{c}^{-1} - e = [11, 11, 0, \underline{\mathbf{9}}, 4, 10, 1, 9, 12, 10, 5, 4, 3, 4, 10, 13, 8, 3, 13, 5, 14, 4, 15]$.

Once inverted, we do in fact receive the predicted codeword

$c = [5, 5, 0, \underline{\mathbf{2}}, 13, 12, 1, 2, 10, 12, 11, 13, 14, 13, 12, 4, 15, 14, 4, 11, 3]$.

We have successfully corrected the single error.

5 Experimental Results

We consider error-detection and correction architectures for four block ciphers: small-scale AES (with state consisting of 4×4 four-bit nibbles instead of bytes); regular AES; LED-64; and PRESENT. The state of AES is organized in bytes and it incorporates 8-bit S-Boxes, whereas all other considered ciphers have nibble-based states and 4-bit S-Boxes. We implemented distance-3 and distance-5 Rabii-Keren (RK) codes over GF(16), that is, one symbol corresponding to four bits. For all ciphers except AES, one symbol corresponds to one element. For AES, we consider an architecture where each byte corresponds to two 4-bit symbols (32 symbols for the complete 128-bit state); note that an error in a single byte may affect two (neighbouring) symbols. A further architecture which overcomes this problem uses two decoders, one over 16 "upper" symbols, where each symbol stands for four most significant bits of a state byte and one over 16 remaining "lower" symbols. On top of the RK code, we implemented a further validation step using quadratic-sum (QS) code as the outer code. It is meant to identify erroneous corrections, similar to the SECDED Hamming code [14]. Note that the QS code alone does not have correction capability.

Table 3 summarizes the considered architectures. Its first 3 columns show the base circuit, distance d of the RK code and whether one or two decoders are used (the latter only happens for full-scale AES as the only byte-oriented cipher). The subsequent 4 columns show the number of information and redundant data of the RK code, first expressed in the numbers of (4-bit) symbols and then in bits. Note that the numbers for the two-decoder architecture are twice the numbers for a single decoder. The check-bit for the QS code is not included in the table.

We ran fault-injection experiments on the mentioned architectures synthesized on Spartan-6 LX75 FPGA on Sakura-G board. We created a faster-than-nominal clock using the FPGA-level digital clock manager and switched to that clock during a specific cycle of encryption. This resulted in a wide distribution of errors of different multiplicity and is a good model of a malicious attack using a rather imprecise equipment. For each architecture, we collected and characterized 1,000,000 *fault events*, where a fault event is one encryption with a specific input (plaintext) whose output (ciphertext) deviated from the fault-free value. We deliberately avoid using the term "fault" or "error" to avoid confusion with scenarios when multiple inputs are used and a fault detected by one of the inputs is counted as detected. This view is appropriate for permanent faults, but fault events considered here are transient.

Each of the fault events is attributed to the following categories. The distribution of the 1,000,000 fault events to four classes C1 – C4 is shown in the next four columns of Table 3.

Class C1: Undetected by the RK code. Faults which were undetected, i.e., resulted in the all-zero syndrome. For a code with distance d, this can only happen for errors of multiplicity d (or multiples of it).

Class C2: Single errors. Faults which affected only one symbol and could be corrected to the original codeword. Note that our experiment set-up keeps the fault-unaffected ciphertext

Table 3: Experimental results on error detection & correction

Architecture			Symbols		Bits		Fault events					
							Non-linear checker				System-level fault manager	
Circuit	d	#Dec	k_q	r_k	k	r	**Class C1** Undetected by the RK code	**Class C2** Single errors (corrected by RK code)	**Class C3** Recognized as suspicious (by RK code)	**Class C4** Erroneous corrections by RK code	**Class S1** Recognized as erroneous by QS outer code	**Class S2** Unrecognised as erroneous by QS outer code
Small-sc.	3	1	16	3	64	12	179	59337	874101	66383	62499 (93.9%)	4063 (6.1%)
AES	5	1	16	7	64	28	0	59337	940662	1	1 (100%)	0 (0%)
AES	3	1	32	3	128	12	239	33	867272	132456	124286 (93.7%)	8409 (6.3%)
	3	2	32	6	128	24	67	33	881006	118894	117475 (98.8%)	1486 (1.2%)
	5	2	32	14	128	56	0	33	999956	11	11 (100%)	0 (0%)
LED-64	3	1	16	3	64	12	234	15	926682	73069	68734 (93.8%)	4569 (6.2%)
	5	1	16	7	64	28	0	15	999985	0	0	0
PRESENT	3	1	16	3	64	12	229	0	927018	72753	68376 (93.7%)	4606 (6.3%)
	5	1	16	7	64	28	0	0	1000000	0	0	0

for reference and therefore can attribute the fault precisely; an actual device under attack would not know the multiplicity of the injected attack.

Class C3: Recognized as suspicious. Faults which resulted in multi-symbol errors and where the correction procedure stopped since it did not find a fitting entry in the ECLT.

Class C4: Erroneous correction. Faults which were corrected but into a different codeword than the original one. This can happen if, e.g., for distance-3 code, an error of multiplicity 2 transforms a codeword into a non-codeword with distance 1 to a different codeword.

Fault events from classes C1 and C4 are potentially critical, as they are associated with errors not properly handled by the RK code. In our architecture, the system-level fault manager based on the QS outer code performs a further validation. Note that the fault events from class C2 are valid corrections which need no further handling, and fault events from class C3 are already recognized as erroneous before the system-level fault manager has been invoked. The fault events from the classes C1 and C4 are subdivided into classes S1 and S2 based on the outcome of this validation:

Class S1: Recognized by QS outer code. Seemingly successful but erroneous corrections which created an inconsistency when recalculating the outer code.

Class S2: Unrecognized by QS outer code. Seemingly successful but erroneous corrections not noticed by the system.

The final two columns of Table 3 present the fault events from these new classes. Note that they sum up to the sum of classes C1 and C4, and that the percentages relate to this sum. For example, the number of fault events for the distance-3 architecture for small-scale AES that need system-level handling is 179 (C1) + 66,383 (C4) = 66,562; out of these, 62,499 or 93.9% are detected by the QS outer code (S1) and the remaining 4,063 or 6.1% are not (S2). Fig. 4 visualizes the four classes C1–C4 and their relationship to system-level classes S1 and S2.

From Table 3, it can be seen that the vast majority of fault events in potentially critical classes C1 and C4 are handled successfully on the system level and are included in class S1. If distance-3 codes are used, less than 1% of fault events go undetected (class S2), the maximum being 8,409 out of 1,000,000 total fault events for the single-decoder AES architecture. However, these cases *never* occur for distance-5 codes.

Even for distance-3 codes, one can assume that, prior to an unnoticed fault, the adversary will have to inject a large number of detected faults, such that the circuit can go into state of alert and, e.g., replace the secret key. The rather low number of successful corrections (single

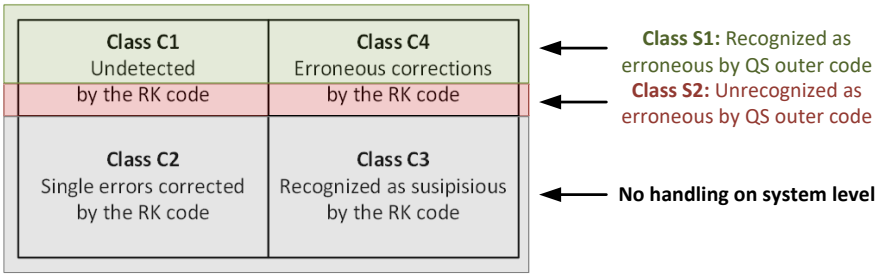

Figure 4: Classification of fault event effects at code and at system level

errors in Table 3) is just the number of single-symbol errors in the fault injection experiment. The code *guarantees* that every single error that shows up will be successfully corrected; this also eliminates the threat of precise single-nibble or single-byte fault injections [26, 8]. The majority of uncorrectable faults are reliably recognized by either the RK code directly or by the outer code. For distance-5 code, the number of erroneous corrections is extremely small (between 0 and 11 out of 1,000,000), and all of them are identified by the outer code.

From the application point of view, the results indicate the suitability of RK-based architectures for mixed detection-correction architectures. In particular, using a code with some "reserves" in terms of detection capability (here: distance-5 code) results in no undetected fault events and no unrecognized erroneous corrections. This means that the architecture can correct low-magnitude disturbances, i.e., single-symbol errors, without much risk of missing attempted attacks. Note that faults are injected into the circuitry, whereas errors are defined on the outputs of the circuit (or of its protected part). Therefore, timing-based fault injections used here can result in errors of different multiplicity, determined by two factors. First, the manipulated (faster-than-nominal) clock runs in parallel through the entire fault-injection campaign, resulting in different and unpredictable deviations between the nominal and the manipulated clock which accumulate over time. When the clock is switched from nominal to manipulated, the next clock edge can occur very quickly, resulting in a large number of failing paths within the circuit and therefore high-multiplicity errors on outputs, or it can happen only slightly before the regular clock edge, such that only one critical path to a circuit output fails. Second, the fault effect propagates through the (unmanipulated) rounds of the cipher after the fault injection, and the diffusion property of the cipher can lead to a higher multiplicity of the error on the circuit's outputs.

Table 4 shows a comparison of the size of our architecture, a purely linear BCH architecture and a triple modular redundant (TMR) architecture code in numbers of needed FPGA configurable logic blocks (CLB). It can be noticed that the robustness, and thus the increase in security, of the RK architecture comes at a low cost compared to the linear (and therefore non-robust) BCH implementation (which also used the ECLT-based approach). The highest increase due to inversions introduced by the RK code is 24% increase; in one case there is even a small decrease due to optimizations during FPGA synthesis. The cost of our architectures exceeds TMR for small basic ciphers, as some required circuitry is cipher-independent. Note, however, that TMR can be interpreted as repetition code and is not robust (the attacker can simply apply the same error to all copies), and therefore its security is inherently worse compared with a robust RK code. In the case of the AES, the number of CLBs are similar which further encourage the use of our architecture.

While the detection and correction performance of the architecture is extremely attractive, the hardware cost of the solution based on advanced non-linear codes is a major limiting factor.

Table 4: Size comparison in numbers of configurable logic blocks (CLBs)

Cipher	Unprot. round	RK ($d = 3$)	BCH ($d = 3$)	RK ($d = 5$)	BCH ($d = 5$)	TMR
Small-scale AES	37	202	169	328	267	108
AES (1 decoder)	173	388	392	–	–	421
AES (2 decoders)	173	465	419	572	462	421
LED-64	39	221	213	257	248	133
PRESENT	23	165	148	240	243	57

For this reason, the ECLT-based approach presented here is an important step towards making these architectures practical. Finally, it is important to note that the used q-ary codes demand more complex operations (multiplications and the inversions) than binary codes. However, it turns out that binary codes with comparable detection and correction properties need considerably more redundancy bits. For example, our distance-5 code over GF(16) requires $r = 56$ redundancy bits for $k = 128$ data bits, whereas a binary BCH code with the same correction capability ($d \geq 2 \cdot 8 + 1$) necessitates $r = 112$ redundancy bits for the same k. Moreover, the decoding is more complex since the ECLT technique from this paper is not applicable and the Berlekamp-Massey algorithm must be used instead. Note that this algorithm cannot be performed in a single cycle, so our higher expenditures in hardware complexity are offset by execution time savings.

6 Conclusions

The precision of physical attacks is steadily improving, giving a strategic attacker the potential to overcome detection strategies based on duplication, modular redundancy, or conventional (linear) error-detecting codes. We presented an architecture based on security-oriented nonlinear codes which can detect and correct errors due to natural and malicious causes. The architecture is based on previously introduced Rabii-Keren codes and discusses the associated encoding, decoding, and error correction algorithms. In particular, we proposed an improved technique for detecting single errors using the Error Coefficient and Location Table (ECLT), which reduces the correction effort by several orders of magnitude and makes this approach feasible for practical use. Experimental results using a physical fault injector on an FPGA show, for several cryptographic circuits, that the architecture can reliably detect and correct faults of arbitrary multiplicity, recognizing erroneous corrections.

References

[1] N. Admaty, S. Litsyn, and O. Keren. Puncturing, expurgating and expanding the q-ary BCH based robust codes. In *IEEE Conv. of Elec. & Electronics Engineers in Israel*, pages 1–5, 2012.

[2] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[3] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of eliminating errors in cryptographic computations. *Jour. Cryptology*, 14(2):101–119, 2001.

[4] C. Cid, S. Murphy, and M. J. B. Robshaw. *Small Scale Variants of the AES*, pages 145–162. Springer Berlin Heidelberg, 2005.

[5] R. Cramer et al. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In *EUROCRYPT*, pages 471–488. Springer, 2008.

[6] S. Dziembowski, K. Pietrzak, and D. Wichs. Non-malleable codes. Cryptology ePrint Archive, Report 2009/608, 2009. http://eprint.iacr.org/2009/608.

[7] S. Engelberg and O. Keren. A comment on the Karpovsky-Taubin code. *IEEE Transactions on Information Theory*, 57(12):8007–8010, 2011.

[8] P. Jovanovic, M. Kreuzer, and I. Polian. A fault attack on the LED block cipher. In *COSADE*, volume 7275 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 2012.

[9] M. Karpovsky, K. Kulikowski, and Z. Wang. Robust error detection in communication and computational channels. In *Int'l Workshop Spectral Methods & Multirate Signal Proc.*, 2007.

[10] M. Karpovsky and A. Taubin. New class of nonlinear systematic error detecting codes. *IEEE Trans. on Information Theory*, 50(8):1818–1819, 2004.

[11] Mark G. Karpovsky and Zhen Wang. Design of strongly secure communication and computation channels by nonlinear error detecting codes. *IEEE Trans. Computers*, 63(11):2716–2728, 2014.

[12] O. Keren and M. Karpovsky. Relations between the entropy of a source and the error masking probability for security-oriented codes. *IEEE Trans. Communications*, 63(1):206–214, 2015.

[13] O. Keren, I. Levin, and R. S. Stankovic. A technique for linearization of logic functions defined by disjoint cubes. I. — Theoretical aspects. *Automation and Remote Control*, 72(3):615–625, 2011.

[14] I. Koren and C.M. Krishna. *Fault-tolerant systems*. Morgan Kaufmann, 2010.

[15] Y. Li, K. Ohta, and K. Sakiyama. New fault-based side-channel attack using fault sensitivity. *IEEE Trans. Information Forensics and Security*, 7(1):88–97, 2012.

[16] S. Mangard, E. Oswald, and T. Popp. *Power analysis attacks – revealing the secrets of smart cards*. Springer, 2007.

[17] Y. Neumeier and O. Keren. Robust generalized punctured cubic codes. *IEEE Transactions on Information Theory*, 60(5):2813–2822, 2014.

[18] Yaara Neumeier and Osnat Keren. A new efficiency criterion for security oriented error correcting codes. In *2014 19th IEEE European Test Symposium (ETS)*, pages 1–6. IEEE, 2014.

[19] X. T. Ngo, S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm. Linear complementary dual code improvement to strengthen encoded circuit against hardware Trojan horses. In *IEEE Int'l Symp. on Hardware Oriented Security and Trust*, pages 82–87, 2015.

[20] KT Phelps. A combinatorial construction of perfect codes. *SIAM Journal on Algebraic Discrete Methods*, 4(3):398–403, 1983.

[21] I. Polian and F. Regazzoni. Counteracting malicious faults in cryptographic circuits. In *IEEE European Test Symp.*, 2017.

[22] H. Rabii and O. Keren. A new construction of minimum distance robust codes. In *International Castle Meeting on Coding Theory and Applications*, pages 272–282. Springer, 2017.

[23] H. Rabii and O. Keren. A new class of security oriented error correcting robust codes. In *Cryptography and Communications (accepted)*, 2018.

[24] B. Selmke, J. Heyszl, and G. Sigl. Attack on a DFA protected AES by simultaneous laser fault injections. In *FDTC*, pages 36–46. IEEE Computer Society, 2016.

[25] V. Tomashevich, Y. Neumeier, R. Kumar, O. Keren, and I. Polian. Protecting cryptographic hardware against malicious attacks by nonlinear robust codes. In *DFT*, pages 40–45, 2014.

[26] M. Tunstall, D. Mukhopadhyay, and S. Ali. Differential fault analysis of the Advanced Encryption Standard using a single fault. In *Works. Inform. Security Theory & Practice*, pages 224–233, 2011.

[27] J. G. J. van Woudenberg, M. F. Witteman, and F. Menarini. Practical optical fault injection on secure microcontrollers. In *FDTC*, pages 91–99, 2011.

[28] J.L. Vasil'ev. On nongroup close-packed codes, Probl. Kibern., 8 (1962), 337–339. *English translation in Probleme der Kybernetik*, 8:92–95, 1965.

[29] Z. Wang and M. Karpovsky. Algebraic manipulation detection codes and their applications for design of secure cryptographic devices. In *IEEE Int'l On-Line Test Symp.*, pages 234–239, 2011.