Hardening AES Hardware Implementations Against Fault and Error Inject Attacks

Lake Bu and Michel A. Kinsy
Adaptive and Secure Computing Systems (ASCS) Laboratory
Department of Electrical and Computer Engineering
Boston University
{bulake, mkinsy}@bu.edu

ABSTRACT

The Advanced Encryption Standard (AES) enables secure transmission of confidential messages. Since its invention, there have been many proposed attacks against the scheme. For example, one can inject errors or faults to acquire the encryption keys. It has been shown that the AES algorithm itself does not provide a protection against these types of attacks. Therefore, additional techniques like error control codes (ECCs) have been proposed to detect active attacks. However, not all the proposed solutions show the adequate efficacy. For instance, linear ECCs have some critical limitations, especially when the injected errors are beyond their fault detection or tolerance capabilities. In this paper, we propose a new method based on a non-linear code to protect all four internal stages of the AES hardware implementation. With this method, the protected AES system is able to (a) detect all multiplicity of errors with a high probability and (b) correct them if the errors follow certain patterns or frequencies. Results shows that the proposed method provides much higher security and reliability to the AES hardware implementation with minimal overhead.

KEYWORDS

AES, Error detection, Error correction, Robust codes, Non-linearity.

ACM Reference Format:
Lake Bu and Michel A. Kinsy. 2018. Hardening AES Hardware Implementations Against Fault and Error Inject Attacks. In *Proceedings of Great Lakes Symposium on VLSI 2018 (GLSVLSI'18).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3194554.3194649

1 INTRODUCTION

In cryptography, the Advanced Encryption Standard (AES) [2] is widely used as a block cipher algorithm. It ensures the confidentiality of a plaintext message through encryption. The computational complexity of the AES algorithm makes it intractable to recover the plaintext message without the appropriate secret key. Since its introduction, there have been numerous proposed attacks against the AES algorithm, especially attacks exploiting potential hardware implementation vulnerabilities. Passive attacks, such as the Differential Power Analysis (DPA) [9], aim to steal the algorithm secret keys by analyzing the AES system runtime power behaviors.

For active attacks, a fault or an error is injected [10] during one or more stages of AES algorithm execution. These attacks infer important information (e.g., key length) about the AES system by observing and analyzing the erroneous output ciphers caused by the injected errors. Our work in this paper focuses on this class of attacks. Beyond attacks, an AES system is also vulnerable to random errors caused by the instability or aging of the cryptographic circuit. This can lead to errors propagating to the outputs of the AES system or its internal stages. Therefore, error detection and error tolerance capabilities are a critical design consideration in these systems. Hardware defense techniques have been explored for both injected and random errors [5]. Another approach is to use error control codes (ECCs) as a built-in self test (BIST) mechanism. ECC-based approaches tend to be more attractive because they (a) can be analyzed using precise mathematical models and (b) offer more cost efficient solutions [8].

Although these approaches have provided a certain level of reliability and security to AES systems, they are often limited in their error detection capabilities. If injected faults are beyond their detection or correction capability, then those errors could be (i) invisible to the detector and (ii) exploited by attackers. Therefore, in this paper, we propose a new method that protects each stage of the AES system with a non-linear code. The proposed approach (a) overcomes the weaknesses seen in linear error detection codes and (b) extends beyond the error correction capability range of previously established non-linear techniques. The key capabilities of the proposed method are:

(1) *Conditional All Error Detection:* all errors will be detected with a high probability in a multiplicity of settings;
(2) *Conditional All Error Correction:* the method is able to correct all errors under a variety of scenarios with a probability of 1 if the injected error is "lazy", i.e., an error that repeatedly appears in 3 or more cycles or rounds;
(3) *Customized Functionality:* the method can be efficiently customized for the four different stages of the AES algorithm/system.

Thus, we characterize the proposed method as "conditional all error detecting and all error correcting" (C-AED-AEC). The rest of the paper is organized as follows. Section 2 has a brief introduction of the four stages of the AES system. In Section 3, we present a widely used error control coding approach for AES systems and highlight some of its vulnerabilities. Section 4 contains the proposed method (C-AED-AEC). Section 5 has the hardware cost and performance results. Section 6 concludes the paper.

GLSVLSI'18, May 23–25, 2018, Chicago, IL, USA

ACM ISBN 978-1-4503-5724-1/18/05…$15.00
https://doi.org/10.1145/3194554.3194649

2 THE FOUR STAGES OF AES

In this section we briefly describe the four stages of the Advanced Encryption Standard (AES) algorithm. In each round of the ten or more rounds of the algorithm, the four stages or part of them will operate on a 4×4 matrix of data called the *state*. In order to facilitate the mathematical discussion in the following sections, we introduce the functions of the stages in the matrix form. For convenience some terms are defined as follows:

- v: the 4×4 state which serves as the input to each stage;
- $a_{i,j}$: the element located at the i^{th} row and j^{th} column of v;
- u: the 4×4 transformed state, which is the output of each stage;
- $b_{i,j}$: the element located at the i^{th} row and j^{th} column of u;
- b: the number of bits in a byte;
- $GF()$: the Galois finite field;
- $\otimes$: the finite field multiplication;
- $\oplus$: the finite field addition;
- $f_{\text{AddRoundKey}}$: the AddRoundKey function;
- f_{SubBytes}: the SubBytes function;
- $f_{\text{ShiftRows}}$: the ShiftRows function;
- $f_{\text{MixColumns}}$: the MixColumns function;
- e: the additive error injected by attackers;
- $\sim$: the distortion symbol, e.g., $\tilde{b}_{i,j} = b_{i,j} \oplus e_{b_{i,j}}$.

The mathematical representation of the four stages is as follows:

I. AddRoundKey:

$$b_{i,j} = f_{\text{AddRoundKey}}(a_{i,j}) = a_{i,j} \oplus k_{i,j}, \tag{1}$$

where $k_{i,j}$ is the key byte at the i^{th} row and j^{th} column of the key matrix.

II. SubBytes:

$$b_{ij} = f_{\text{SubBytes}}(a_{i,j}) = M_{inv} \otimes a_{i,j} \oplus M_{aff}, \tag{2}$$

where M_{inv} is the binary inversion matrix in $GF(2^8)$. With the binary vector M_{aff} it provides an affine function.

III. ShiftRows:

$$\begin{aligned} b_{0,*} &= f_{\text{ShiftRows0}}(a_{0,*}) = a_{0,*} \otimes M_{SR0}; \\ b_{1,*} &= f_{\text{ShiftRows1}}(a_{1,*}) = a_{1,*} \otimes M_{SR1}; \\ b_{2,*} &= f_{\text{ShiftRows2}}(a_{2,*}) = a_{2,*} \otimes M_{SR2}; \\ b_{3,*} &= f_{\text{ShiftRows3}}(a_{3,*}) = a_{3,*} \otimes M_{SR3}; \end{aligned} \tag{3}$$

where $M_{SR0}, M_{SR1}, M_{SR2}, M_{SR3}$ are binary matrices which shift the input by 0, 1, 2, 3 digits respectively.

IV. MixColumns:

$$b_{*,j} = f_{\text{MixColumns}}(a_{*,j}) = M_{MC} \otimes a_{*,j}, \tag{4}$$

where M_{MC} is a 4×4 Maximum Distance Separable (MDS) matrix.

Although there are many ways to describe the operations of the AES stages, the matrix form lends itself well to our BIST formulations and associated attacks in a clearer mathematical way.

3 RELATED WORKS ON THE ECC PROTECTED STAGES OF AES AND THEIR VULNERABILITIES

As previously mentioned, since the error injection attacks are often able to distort the AES internal stages and lead to the leakage of critical information of the secret keys, there is a strong demand of error detection, or even error correction, for the AES implementations. A popular and efficient approach is to use the error control codes (ECCs) to form a built-in self test (BIST) mechanism, which is often referred to as the self-checking checkers (SCC). Various codes including parity codes, cyclic codes, Hamming codes, and Reed-Solomon (RS) codes [1, 11] are adopted to detect different number of errors or correct some of them by SCC. In addition, codes are also used to protect the AES against non-invasive attacks such as differential power analysis (DPA) [4, 9].

3.1 Linear SCC Protected AES Stages

In a SCC, the input (state) of a stage goes through two functional modules:

- The specific AES stage:
$$b_{i,j} = f(a_{i,j})$$
- The corresponding redundancy generator (predictor):
$$R_{b_{i,j}} = g(f(a_{i,j}))$$

The outputs from the two modules are verified by the SCC's decoder for error detection:

$$H(b_{i,j}, R_{b_{i,j}}) \stackrel{?}{=} 0. \tag{5}$$

With those ECC codes, the SCC can achieve a limited error detection capability.

3.2 Vulnerabilities of the Linear SCC

Because of the linearity of the SCC induced by a linear ECC, there can be a large number of injected errors non-detectable, i.e., 'invisible" to the decoder. This attack scenario is illustrated below. For two arbitrary state elements $a_{i,j}$ and $a_{k,l}$, it is easy to see that [Eq. 5] satisfies the following if there is no error:

$$a_{i,j} \rightarrow H((b_{i,j}, R_{b_{i,j}})) = 0; \quad a_{h,l} \rightarrow H((b_{h,l}, R_{b_{h,l}})) = 0.$$

Since most of the ECC is linear and $g()$ and $H()$ are linear functions, we have the following relationship:

$$\begin{aligned} &H((b_{i,j}, R_{b_{i,j}}) \oplus (b_{h,l}, R_{b_{h,l}})) \\ =&H(b_{i,j}, R_{b_{i,j}}) \oplus H(b_{h,l}, R_{b_{h,l}}) = 0 + 0 = 0. \end{aligned}$$

This means that the linear combination of two (or more) legal entries is still a legal entry to $H()$. In another word, if an injected error e satisfies $H(e) = 0$, then when it is applied to this ECC protected AES stage, it will never be detected because

$$H((b_{i,j}, R_{b_{i,j}}) \oplus e) = H(b_{i,j}, R_{b_{i,j}}) \oplus H(e) = 0. \tag{6}$$

We call these invisible errors. In this situation, $b_{i,j}$ has been distorted by the injected error e, but it will never be detected by $H()$. For any given AES stage's output $b_{i,j}$, there exist 2^8 such invisible errors. Moreover, this fatal attack applies to all the AES implementations with linear ECCs, such as the ones mentioned earlier in this section.

4 THE CONDITIONAL ALL ERROR DETECTING AND ALL ERROR CORRECTION (C-AED-AEC) SCHEME

In this section, we introduce a new secure and reliable scheme to protect the AES stages from any errors, including ones invisible to the linear SCC. This scheme utilizes a non-linear ECC so [Eq. 6] will not always stand. The advantages of this proposed scheme are:

(1) It detects all errors with a probability close to 1;
(2) There are no errors non-detectable, i.e., invisible, to the decoder;
(3) If the injected error is lazy (i.e., the same error remains for three or more rounds or cycles of AES), it will be corrected with a probability of 1.

Based on the properties above, we name the new scheme *Conditional All Error Detecting and All Error Correction* (C-AED-AEC). The SCC diagram is given below.

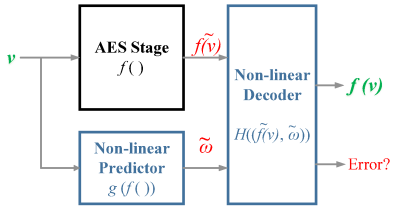

Figure 1: With the non-linear Robust SCC, there is no more $H(e) = 0 \rightarrow H(b \oplus e) = 0 \ \ \forall b$.

4.1 The Non-linear Robust Code

Before introducing the non-linear ECC code, we will first discuss the concept of Kernel of codes. This will help in understanding why linear ECC protections in AES implementations are vulnerable to certain injected errors.

Definition 4.1 Let $C \subseteq GF(2^N)$ be the set of N-bit codewords and M is an $(R \times N)$ matrix. C is defined by $C = \{c | M \cdot c = 0\}$. Set K_d is called the Kernel of C if:

$$K_d = \{e | e \oplus c \in C, \ \ \forall c \in C\}.$$

If C is linear, then $K_d = C$, and these errors can mask themselves in all verification by $H()$. As discussed at the end of Section 3, a linear ECC has a large invisible error set K_d, namely its legal codeword set. Therefore, it will be ideal to adopt a non-linear code [12] whose Kernel is $K_d = 0$. Thus, there will be no error capable of masking itself in all states under the check of $H()$. In this work, we use the Robust codes for their high error detection capability [3]. Although, the generalized form of the Robust codes has a very complicated construction and proof [7], we introduce a simplified version specially tailored to the four stages of AES.

Construction 4.1 For a vector x that can be equally partitioned into N pieces of b-bit symbols $\{x_0, x_1, \cdots, x_i, \cdots, x_{N-1}\}$, its corresponding b-bit signature symbol ω can be computed as:

$$\omega = \begin{cases} \bigoplus_{i=0}^{(N-2)/2} (x_{2i} \otimes x_{2i+1}), & N \text{ is even;} \\ x_0^3 \oplus \left[\bigoplus_{i=0}^{(N-3)/2} (x_{2i+1} \otimes x_{2i+2})\right], & N \text{ is odd.} \end{cases} \quad (7)$$

Note: all computations are carried out over $GF(2^b)$ finite field.

$c_R = (x, \omega) \in C_R$ is a Robust codeword of the Robust code C_R, where x is the information part, and ω the redundant part. ■

4.2 Conditional All Error Detection

A Robust code built by Construction 4.1 has $K_d = 0$. There exists no e which always makes $e \oplus c_R$ a legal Robust codeword $\forall c_R \in C_R$. In other words, any error can be detected with some probability by Robust codes. The lower bound of their error detection probability can be calculated by the error masking equation (EME).

For an error $e = \{e_x, e_\omega\}$ (that $\tilde{x}_i = x_i \oplus e_{x_i}$, $\tilde{\omega} = \omega \oplus e_\omega$) to be invisible to a given Robust codeword, it has to satisfy the EME, which is the equality of [Eq. 7] under $(\tilde{x}, \tilde{\omega})$, where $\tilde{x} = x \oplus e_x$, $\tilde{\omega} = \omega \oplus e_\omega$. Denoting the upper bound of the error masking probability as $\overline{P_{mask}}$, then the error detection probability of the Robust codes is $P_{det} \geq 1 - \overline{P_{mask}}$ for any given $e = \{e_x, e_\omega\}$ (proof in [6]):

$$P_{det} \geq 1 - \overline{P_{mask}} = \begin{cases} 1 - \frac{1}{2^b}, & N \text{ is even;} \\ 1 - \frac{2}{2^b}, & N \text{ is odd.} \end{cases} \quad (8)$$

Larger is b, higher error detection probability: $P_{det} \approx 1$. Furthermore, there are no errors completely invisible to the decoder.

4.3 Conditional All Error Correction

In the proposed approach, we also correct lazy inject errors. A lazy error will remain in place for several cycles or rounds. Here, we introduce two specific algorithms for lazy error correction for the Robust code tailored to the AES stages.

4.3.1 Lazy Error Correction when $N = 1$. Among the four stages of AES, the AddRoundKey and SubBytes operate over single bytes of the state. Therefore, for a Robust code predictor using [Eq. 7], we have the case of $N = 1$, which is an odd number. At this special case there is only one cubic term and [Eq. 7] becomes:

$$\omega = x^3 \quad (9)$$

A legal Robust codeword is $c = (x, x^3)$.

If we assume to have a lazy error $e = (e_x, e_\omega)$ that lasts for at least three different messages $(\tilde{x_0}, \tilde{\omega_0}), (\tilde{x_1}, \tilde{\omega_1}), (\tilde{x_2}, \tilde{\omega_2})$, then by substituting these terms in [Eq. 9], we can solve:

$$x_0 = \left[(\tilde{x_1}^2 \oplus \tilde{x_2}^2) \oplus \frac{\tilde{\omega_0} \oplus \tilde{\omega_2}}{\tilde{x_0} \oplus \tilde{x_2}} \oplus \frac{\tilde{\omega_0} \oplus \tilde{\omega_1}}{\tilde{x_0} \oplus \tilde{x_1}}\right] \Big/ (\tilde{x_1} \oplus \tilde{x_2}) \quad (10)$$

In a similar way, we can solve x_1 and x_2.

4.3.2 Lazy Error Correction when $N = 2$. Among the four stages of AES, the MixColumns and ShiftRows both operate over 4 bytes (a row or a column) of the state, which can be encoded as two Robust codewords with 2 bytes each. [Eq. 7] becomes:

$$\omega = x_0 \otimes x_1 \quad (11)$$

and a legal Robust codeword is $c = (x_0, x_1, \ x_0 \otimes x_1)$.

If we assume a lazy error $e = (e_{x_0}, e_{x_1}, e_\omega)$ which lasts for at least three different messages $(\tilde{x_{00}}, \tilde{x_{01}}, \tilde{\omega_0}), (\tilde{x_{10}}, \tilde{x_{11}}, \tilde{\omega_1}), (\tilde{x_{20}}, \tilde{x_{21}}, \tilde{\omega_2})$, then by substituting these terms in [Eq. 11], we can calculate each x, e.g.:

$$x_{00} = \tilde{x_{00}} \oplus \left[\tilde{\omega_1} \oplus \tilde{\omega_2} \oplus \tilde{x_{20}} \otimes \tilde{x_{21}} \oplus \tilde{x_{10}} \otimes \tilde{x_{11}} \oplus \frac{(\tilde{\omega_0} \oplus \tilde{\omega_2} \oplus \tilde{x_{00}} \otimes \tilde{x_{01}} \oplus \tilde{x_{10}} \otimes \tilde{x_{11}}) \otimes (\tilde{x_{20}} \oplus \tilde{x_{10}})}{\tilde{x_{00}} \oplus \tilde{x_{10}}}\right] \Big/ \left[\tilde{x_{11}} \oplus \tilde{x_{21}} \oplus \frac{(\tilde{x_{01}} \oplus \tilde{x_{11}}) \otimes (\tilde{x_{20}} \oplus \tilde{x_{10}})}{\tilde{x_{00}} \oplus \tilde{x_{10}}}\right] \quad (12)$$

Since all errors lasting for a few cycles can be corrected by the formulated SCC, we call this property *Conditional All-Error-Correction* (C-AEC).

4.4 The SCCs Functions for the Four Stages

By [Eq. 9, 11] and the four AES stages' functions [Eq. 1, 2, 3, 4], we have the SCC predictor functions as:

- AddRoundKey ($P_{det} \geq 1 - \frac{2}{2^8} = 99.2\%$):
$$\omega_{\text{AddRoundKey}} = (a_{i,j} \oplus k_{i,j})^3. \tag{13}$$
- SubBytes ($P_{det} \geq 1 - \frac{2}{2^8} = 99.2\%$):
$$\omega_{\text{SubByte}} = (M_{inv} \otimes a_{i,j} \oplus M_{aff})^3. \tag{14}$$
with $P_{det} \geq 1 - \frac{2}{2^8} = 99.2\%$.
- ShiftRows ($P_{det} \geq 1 - \frac{1}{2^8} = 99.6\%$), each row with two SCCs denoted by α, β:

$$\begin{aligned}
&\omega_{\text{ShiftRows0-}\alpha} = (a_{0,0} \otimes a_{0,1}), \quad \omega_{\text{ShiftRows0-}\beta} = (a_{0,2} \otimes a_{0,3});\\
&\omega_{\text{ShiftRows1-}\alpha} = (a_{1,1} \otimes a_{1,2}), \quad \omega_{\text{ShiftRows1-}\beta} = (a_{1,3} \otimes a_{1,0});\\
&\omega_{\text{ShiftRows2-}\alpha} = (a_{2,2} \otimes a_{2,3}), \quad \omega_{\text{ShiftRows2-}\beta} = (a_{2,0} \otimes a_{2,1});\\
&\omega_{\text{ShiftRows3-}\alpha} = (a_{3,3} \otimes a_{3,0}), \quad \omega_{\text{ShiftRows3-}\beta} = (a_{3,1} \otimes a_{3,2}).
\end{aligned} \tag{15}$$

- MixColumns ($P_{det} \geq 1 - \frac{1}{2^8} = 99.6\%$), each column with two SCCs denoted by α, β:

$$\begin{aligned}
\omega_{\text{MixColumns-}\alpha} =& (2a_{0,j} \oplus 3a_{1,j} \oplus a_{2,j} \oplus a_{3,j})\\
&\otimes (a_{0,j} \oplus 2a_{1,j} \oplus 3a_{2,j} \oplus a_{3,j});\\
\omega_{\text{MixColumns-}\beta} =& (a_{0,j} \oplus a_{1,j} \oplus 2a_{2,j} \oplus 3a_{3,j})\\
&\otimes (3a_{0,j} \oplus a_{1,j} \oplus a_{2,j} \oplus 2a_{3,j}).
\end{aligned} \tag{16}$$

The decoders are to verify [Eq. 13, 14, 15, 16] under the existence of errors. The error correction probability for lazy errors (lasts for at least 3 rounds) is 100%.

5 EVALUATION

We expanded and optimized the four predictor functions [Eq. 13, 14, 15, 16] over finite field $GF(2^8)$, as well as the decoder functions. Thus the overlay of $g(f())$ functions in the Robust predictors can be precomputed into a simpler function instead of applying $f()$ and $g()$ successively and separately. In our experimentation, we injected 4,019,798 errors of various complexities in the computation stages and measure the error detection rates. To test the lazy error correction capability, the errors are made to last for at least 3 rounds in a given stage, such that [Eq. 10 & 12] can be computed. We used the Xilinx Vertex 7 XC7VX330T FPGA board for our implementations and testing.

Table 1: Hardware and Timing Overhead

Stages	P_{det}	Overhead (C-AED)	Overhead (C-AED-AEC)
AddRoundKey	99.4%	56.3%	92.7%
SubBytes	99.3%	63.7%	101.3%
ShiftRows	99.6%	80.3%	144.0%
MixColumns	99.6%	77.9%	132.9%

[1] Hardware Overhead = $\frac{\text{Stage + Predictor + Decoder}}{\text{Stage}} - 1$.

The P_{det} for the AddRoundKey and SubBytes stages is below ($1 - \overline{P_{mask}} = 99.6\%$), because there exists no error satisfying their EMEs in cubic forms. In contrast, there is always a solution satisfying the EMEs equations for the ShiftRows and MixColumns stages. The implementation overhead associated with the error detection plus the error correction (C-AED-AEC) is larger than that of the error detection alone (C-AED). This increase is due to the fact that questions [Eq. 10 & 12] are more complex when compared to the error detection decoders. Altogether, the additional cost is justified since the implementation supports a stronger error tolerance capability.

6 CONCLUSION

In this paper, we have proposed a new technique to harden the reliability and security of AES hardware implementations using a non-linear code self-checking checkers. The proposed method shows key advantages over conventional linear approaches. Unlike the linear ECC approaches, the non-linear technique can (a) detect any injected error with a high probability and (b) guarantee the correction of all lazy errors, which commonly appear in high speed AES hardware implementations. In addition, since the new scheme is built upon error control codes, its performance and capability can be analyzed and estimated using mathematical models. The strong theoretical modeling foundation makes it possible for designers to evaluate and determine the cost/performance trade-off with high accuracy.

7 ACKNOWLEDGMENTS

This research is partially supported by the NSF grant (No. CNS-1745808).

REFERENCES

[1] Luca Breveglieri, Israel Koren, and Paolo Maistri. 2005. Incorporating error detection and online reconfiguration into a regular architecture for the advanced encryption standard. *Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on. IEEE* (2005).

[2] Joan Daemen and Vincent Rijmen. 2013. The design of Rijndael: AES-the advanced encryption standard. *Springer Science and Business Media* (2013).

[3] G. Gaubatz, B. Sunar, and M. G. Karpovsky. 2006. Non-linear residue codes for robust public-key arithme. *Fault Diagnosis and Tolerance in Cryptography* (2006).

[4] Marc Joye and Amir Moradi. 2015. Smart Card Research and Advanced Applications. *Springer International Publishing* (2015).

[5] Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim. 2001. Fault-Based Side-Channel Cryptanalysis Tolerant Rijndael Symmetric Block Cipher Architecture. *Defect and Fault Tolerance in VLSI Systems, 2001. Proceedings. 2001 IEEE International Symposium on. IEEE* (2001).

[6] Konrad Kulikowski, Mark Karpovsky, and Alexander Taubin. 2005. Robust codes for fault attack resistant cryptographic hardware. *Fault Diagnosis and Tolerance in Cryptography, 2nd International Workshop* (2005).

[7] K. Kulikowski, Z. Wang, and M. G. Karpovsky. 2008. Comparative analysis of robust fault attack resistant architectures for public and private cryptosystems. *IEEE 5th Workshop on Fault Diagnosis and Tolerance in Cryptography* (2008).

[8] Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha. 2003. Analyzing the energy consumption of security protocols. *Proceedings of the 2003 international symposium on Low power electronics and design. ACM* (2003).

[9] Emmanuel Prouff. 2005. DPA attacks and S-boxes. *nternational Workshop on Fast Software Encryption* (2005).

[10] Cyril Roscian, Jean-Max Dutertre, and Assia Tria. 2013. Frontside laser fault injection on cryptosystems-Application to the AES last round. *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on. IEEE* (2013).

[11] Chih-Hsu Yen and Bing-Fei Wu. 2006. Simple error detection methods for hardware implementation of advanced encryption standard. *IEEE transactions on computers* (2006).

[12] W. Zhen, M. Karpovsky, and K. J. Kulikowski. 2009. Replacing linear hamming codes by robust nonlinear codes results in a reliability improvement of memories. *IEEE/IFIP International Conference on Dependable Systems and Networks* (2009).