An Efficient Hardware-Based Fault Diagnosis Scheme for AES: Performances and Cost

Guido Bertoni[1], Luca Breveglieri[2], Israel Koren[3], Paolo Maistri[2]

[1]STMicroelectronics, Agrate Brianza, Milano, ITALY
[2]Department of Electronics and Information Technology
Politecnico di Milano, Milano, ITALY
[3]Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA, USA

guido.bertoni@st.com, Luca.Breveglieri@PoliMI.it,
koren@ecs.umass.edu, Paolo.Maistri@PoliMI.it

Abstract

Since standardization in 2001, the Advanced Encryption Standard has been the subject of many research efforts, aimed at developing efficient hardware implementations with reduced area and latency. So far, reliability has not been considered a primary objective. Recently, several error detecting schemes have been proposed in order to provide some defense against hardware faults in AES. The benefits of such schemes are twofold: avoiding wrong outputs when benign hardware faults occur, and preventing the collection of information about the secret key through malicious injection of faults. In this paper, we present a complete scheme for parity-based fault detection in a hardware implementation of the Advanced Encryption Standard which includes a key schedule unit. We also provide a preliminary evaluation of the hardware and latency overhead of the proposed scheme.

1 Introduction

The AES algorithm has been standardized by NIST in 2001 [15] with the adoption of the Rijndael algorithm [9], a Substitution-Permutation Network (SPN) cipher whose operations are based on binary extension fields.

Since then, many software and hardware implementations have been proposed. The former are targeting various platforms, from x86 architectures down to smart card devices, and are aimed at reducing the instruction count per encryption by exploiting the specific architecture instruction set. Hardware solutions have been proposed both for field-programmable devices (FPGAs) and custom devices (ASICs). The goal of FPGA-based implementations is minimizing the area consumed while maintaining an acceptable throughput. Most of the proposed FPGA-based implementations have focused on the substitution unit [17] or the permutation unit, and generally on sharing resources between the encryption and decryption routines. Other solutions have focused on careful placement of the functional units on the device [7].

In contrast, custom ASIC devices are less flexible, but can be optimized for inclusion in embedded systems. In this case, the focus is on exploiting the maximum parallelism allowed by the algorithm and increasing the throughput, e.g., by unrolling the round iterations and pipelining the operations. Such solutions allow the highest throughput, but often at the cost of a reduced compliance to the standard (see for instance [2], where the 2-Gbit throughput is obtained by sacrificing the support for keys longer than 128 bits or cipher modes other than ECB). Resource sharing is obviously not confined to FPGA implementation: in [16], the authors explored the sharing between the direct *MixColumns* and the *Inverse MixColumns* operations. Higher throughputs are possible, but the area requirements are much larger: see for instance [14] and [18].

Recently some schemes for fault detection and tolerance have been proposed. The motivation has been the increased likelihood of failures in devices as complex as current encryption units [9], and the development of advanced attack techniques based on the deliberate injection of faults [1, 3, 6, 8]. Preliminary proposals were based on exploiting the redundancy of functional units which is typical of encryption/decryption units [10]. Later research has studied the application of error detecting codes to symmetric block ciphers [4, 11]. Simple codes like parity allow obtaining very high detection capabilities at a reasonable cost. Using parity code for fault detection in generic Substitution-Permutation Networks (SPN) has also been shown in [12].

In this paper we further develop the model presented in [5] and extend the fault analysis to the Key Schedule unit. We show that the Key Schedule unit has a highly dispersive behavior that allows an error to propagate quickly, but this does not compromise the detection rate of the parity code. We also evaluate the hardware costs (area and latency) of some standard AES implementations when a fault detection capability is included in the device.

The paper is organized as follows. In Section 2 we provide a brief overview of the AES encryption algorithm. Section 3 describes the complete error model, extended to include the Key Schedule part, and presents the results of the software simulations of the model. The evaluation of the hardware costs and the performance impact are presented in Section 4. Section 5 concludes the paper.

2 The Rijndael algorithm

The Rijndael encryption algorithm [9] consists of three procedures, namely Encryption, Decryption (i.e., the inverse of Encryption) and Key Schedule. NIST imposed an input block size of 128 bits with the most common version using a 128-bit-long key, but longer keys are allowed. The ciphered output is 128-bit. In our analysis, we will always refer to the 128-bit-long-key version, but the results are still valid when longer keys are used.

The encryption process consists of 10 iterative rounds executed after a pre-processing key-mixing phase, where the initial key is added (modulo 2) to the initial input. The intermedi-

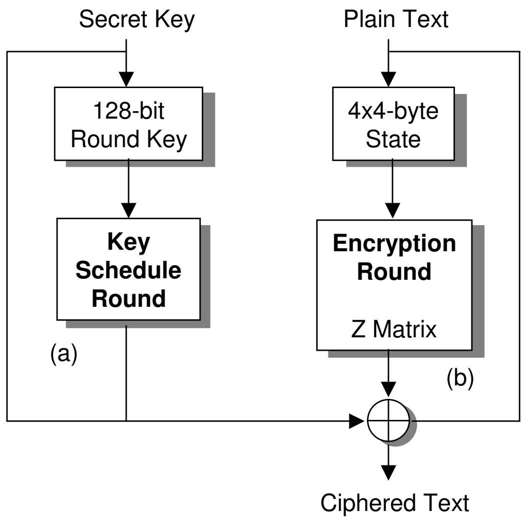

Figure 1. *Block diagram of Rijndael: Key Schedule (a) and Encryption (b).*

ate result of the encryption process is stored into a 16-byte square matrix S called *state*.The encryption round is a chain of four byte-based transformations, both linear and non-linear, defined over the Galois Field $GF(2^8)$ (see Figure 1). These transformations are:

SubBytes: a byte-wise non-linear substitution, it can be computed on-the-fly or implemented by means of a lookup table;

ShiftRows: a byte rotation of the rows of the state S using an offset that depends on the row itself;

MixColumns: a linear algebraic transformation of each column of the state, over the Galois Field;

AddRoundKey: a bit-wise addition (modulo 2) of the current round key, provided by the key schedule, to the state S.

Further details about AES can be found in [4] and [9]. Galois Fields are described in [13] and [15]. Here we want to focus on the key schedule only; we will consider 128-bit-long keys for simplicity. After the initial key has been loaded into the key schedule state, it is stored into a 16-byte square matrix which is organized into 4 32-bit words. At each round, the last word of the previous round key is rotated by 8 bits and processed through 4 Substitution Boxes. The first row is XOR'ed with the result from the S-Boxes and a precomputed constant; the following rows are XOR'ed with their preceding row. The pseudocode can be found in [9].

3 AES full error model

In this section we describe the generalized fault model of the AES encryption. This is an extension of the model presented in [5], which lacks the key schedule part. The implementation of the model is described and validated in Section 4.

3.1 The model

The model presented in [5] considered faults occurring in the datapath only. This was justified by the fact that the operations used in the key schedule are a subset of those used in the encryption process, thus the analysis may be replicated without much effort. Moreover, most fault attacks are aimed at the encryption datapath, since these are of most practical interest. Giraud [8] proposed a byte fault attack on the AES key schedule; still, attacking the key schedule is considered more difficult, since the key cache memory is usually tightly protected against intrusions or side channel information leakage.

We start with a brief review of the model presented in [5]. At round r, each byte $s_{i,j}^{(r)}$ of the state has an associated parity $p_{i,j}^{(r)}$, which is *predicted* from the state and the parity in the previous round $r-1$. The discrepancy between the parity matrix P and the actual computed parity $Parity(S)$ is the *Error Matrix* E:

$$E^{(r)} = P^{(r)} \oplus Parity(S^{(r)}) \tag{1}$$

Each element $e_{i,j}$ signals an inconsistency between the predicted parity and the current parity, computed from the actual data.

The above model proved to be very useful since each AES transformation updates the Error Matrix, increasing or decreasing the number of detectable errors, but never canceling them completely. The rules for propagating the elements of the Error Matrix through the various AES operations are:

SubBytes: the parity code bits can be re-generated from the current data, but this approach forces the validation of the code bits at the input, to avoid possible cancellations. On the other hand, the input parity code can be used to propagate the error to the output, for instance by including the parity code in the lookup table [4] and extending the table addressing with the additional parity bit. This is the approach we will consider and therefore the matrix E is propagated without any change.

ShiftRows: the rows of the matrix E are rotated according to the same rules used for the state. This step simply moves the errors, without cancellations or duplications.

MixColumns: each parity $e_{i,j}$ is updated using a linear combination of the elements in the same column; the exact rule can be found in [4].

AddRoundKey: provided that the key is error-free, this operation propagates E without any change, i.e., $E^{(r+1)} = E^{(r)}$.

Since the complete round transformation of the E matrix is linear, it can be modeled as $E \mapsto ZE$, where Z is a matrix of $GF(2)$ elements defined in [5]. The above model has been limited by the fact that it does not deal with key schedule errors. Although errors in the key schedule can be analyzed following the same approach used in the analysis of the encryption round, the question whether the key schedule demonstrates the same characteristics as the encryption round (in particular when injecting just a single fault), still remains unanswered. We will, therefore, extend in this paper the original model in order to include the error propagation within the key schedule component.

As was done for the datapath example, we associate a parity bit to each byte of the key state; hence, we have a matrix $K^{(r)}$, consisting of all the round keys and the matrix $P_K^{(r)}$ of the corresponding parity bits. We can thus define the Key Error Matrix E_K:

$$E_K^{(r)} = P_K^{(r)} \oplus Parity(K^{(r)}) \tag{2}$$

The operations used in the key schedule round are very simple: each column is updated adding the preceding column (modulo 2), except for the first column of each round. For this column, the operations involved are: (1) *substitution* of each byte of the 4^{th} column of the previous round using the same S-Boxes as in the *SubBytes* routine. This step, as in encryption, does not alter E_K. (2) *rotation* of the word just computed. Like *ShiftRows*, this step only moves a fault to a different position. (3) *exclusive OR* with a pre-computed constant, whose parity is hence known a priori. (4) *addition modulo* 2 with the original value of the word being updated. This step is the main cause of dispersion and collapse of the number of apparent faults.

It is clear that the whole Key Schedule round can be modeled using a linear transformation of the matrix E_K, i.e. $E_K \mapsto Z_K E_K$. The matrix Z_K is a 16×16 matrix on $GF(2)$:

$$Z_K = \begin{bmatrix} I_4 & 0 & 0 & A \\ I_4 & I_4 & 0 & A \\ I_4 & I_4 & I_4 & A \\ I_4 & I_4 & I_4 & A \oplus I_4 \end{bmatrix}, \qquad A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The matrix A takes its structure mainly from the rotation of the last row of the previous matrix. The substitution boxes and the addition of a constant do not alter the error; if these operations are faulty, they can be modelled by a fault at the beginning of the round. The lower triangular part, filled by 4-by-4 identity matrices, is due to the fact that each row is updated by XORing with the previous row. Z_K is obviously non singular, since the Key Schedule is invertible. Through computation, we have verified that Z_K^{15} is orthogonal (but note that AES specifies 14 rounds at most), while only Z^{60} is the Identity matrix. In contrast, the matrix Z is the generator of a much smaller cyclic group since $Z^8 = I$.

We can now merge the two sets of expressions and obtain the general model of an AES round, under the assumption that faults are injected at the beginning of a round:

$$E_K^{(r)} \mapsto E_K^{(r+1)} = Z_K E_K^{(r)} \tag{3}$$

$$E^{(r)} \mapsto E^{(r+1)} = Z E^{(r)} \oplus E_K^{(r)} \tag{4}$$

Under this new model, the constraint of a fault-free key schedule can be now relaxed. A fault in E_K now evolves in the round key as indicated in (3) and affects the state error as shown in (4), as the key addition is the last round operation after the application of Z (i.e., *ShiftRows* and *MixColumns*).

3.2 The Simulations

In [4], the error detection capabilities of the proposed parity code were studied using simulated fault injections into the actual AES encryption algorithm. In these simulations, a random pair of plain input and key was generated, the encryption procedure was started and a fault was injected during the encryption. The corrupted data was finally checked against the parity code bits. In [5], it was proved that the model is accurate enough to allow simulating the model rather than the entire encryption process, thus simplifying the analysis. Here, we extend this approach to our new generalized model, in order to understand the effect of combined faults both in the Encryption datapath and the Key Schedule.

Since analyzing the error propagation in the model is simpler (and less time consuming) than injecting faults into the encryption process (and comparing outputs and code bits), exhaustive simulation was used: up to 4 simultaneous faults were injected, and all the possible test cases were evaluated. The degrees of freedom for each injected fault were:

- The round;
- Injecting the fault in the round key or in the encryption state;
- The byte (out of the 16 bytes in the state or in the round key); note that in this model, the maximum granularity is at the byte level.

We focus on AES-128 since it is the most commonly used, but we expect our results to hold for longer keys as well. Moreover, it must be pointed out that since $Z^8 = I$, a state fault shrinks back to a single byte error after 8 rounds. This does not occur for key schedule faults since the period of Z_K is 60, as noted in Section 3.1. For each fault we have 320 possible injection spots, obtained from 10 rounds $\times 16$ bytes $\times 2$ possible matrices. Based on our simulations we have made the following important observations:

- All single faults are detected; this was already proved when the fault was injected in the state [4, 5]. It stems from the non-singularity of Z_K when the fault is injected in the Key Schedule.

Table 1. *Absolute results of exhaustive search of the simulation space.*

Injected faults	1 **fault**	2 **faults**	3 **faults**	4 **faults**
Search Space	320	51,040	5,410,240	428,761,520
Detected Faults	320	51,007	5,410,114	428,749,119
Undetected Faults	0	33	126	12,401
All faults in State	*0*	*32*	*0*	*6,040*
Faults in State and Key Sched.	*0*	*0*	*107*	*5,919*
All faults in Key Schedule	*0*	*1*	*19*	*442*

- An odd number of faults is always detected when all faults are injected only into the state. This was claimed in [5] but the new experiments revealed that this does not hold when some of the faults are injected into the key schedule as well (see Table 1).
- When two faults are injected, they are always detected when one is injected into the state and the second into the key schedule. They are not detected when they exploit the periodicity of Z^8, i.e., when they occur in the same byte of the state but separated by 8 rounds.
- The detection capability appears to be double that claimed in [4] (see Figure 2). This can be explained by the fact that the injection space is doubled (from 160 to 320 possible injection spots), thus halving the average density of the undetectable fault sets. Moreover, note that the matrix Z_K is highly dispersive, which means that any fault in the key schedule will quickly spread over a larger number of bytes and will later be added to the state, where it will be further processed by the encryption process.

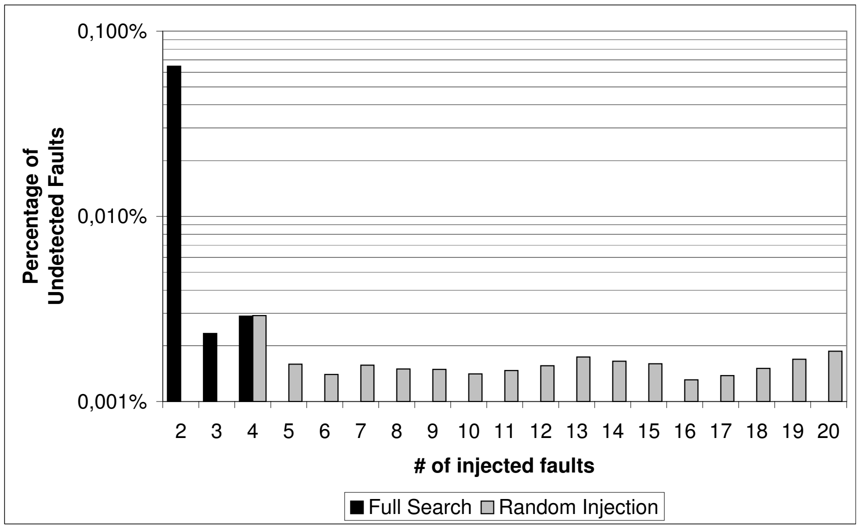

Figure 2. *Percentage of undetected faults with exhaustive (full search) and random injection in State and Key Schedule.*

4 Hardware Evaluation

In this section we evaluate the overhead, in terms of both area and latency, of the components needed to generate, predict and check the parity code with respect to a similar architecture without fault detection capabilities. A complete round of AES encryption has been described in VHDL and synthesized. To simplify the design, a 128-bit-wide datapath is assumed. This choice means that each single AES operation computes the whole block in parallel: hence, the *SubBytes* operation was implemented using 16 lookup tables operating on 8-bit inputs each, while the *MixColumns* was implemented using 4 different units, one for each column of the state. No particular constraints were set, allowing the synthesis tool to choose the suitable tradeoff between area and latency. Although this may not be the most efficient choice, it still allows us to evaluate the overhead introduced by the parity code.

The implementation of the prediction rules is quite straightforward for each round operation, requiring a few gates and proper routing of the signals. The only issue is the implementation of the *SubBytes*: the initial model [4] proposed to extend the original lookup tables to 9-bit-wide inputs and outputs. However, the additional input address bit would double the size of the table. Moreover, the extra output bit means an additional 12.5% overhead. This was partially overcome by merging all the invalid pairs of byte and parity into a single sequence with an incorrect parity (a null byte with a parity bit of '1'). This way, the overhead was reduced to just the additional output bit per byte. However, this means that a specific circuit had to be designed for address decoding. Our new solution does not store any "*wrong*" parity. Instead, it uses the 8-bit data input to compute both the S-Box output and the corresponding parity. Finally, the output parity is XOR'ed with each single input bit (input and parity). If the input parity was incorrect, this final step inverts the output parity and hence propagates the error; if there was no (apparent) error, then the parity is not altered (see Figure 3).

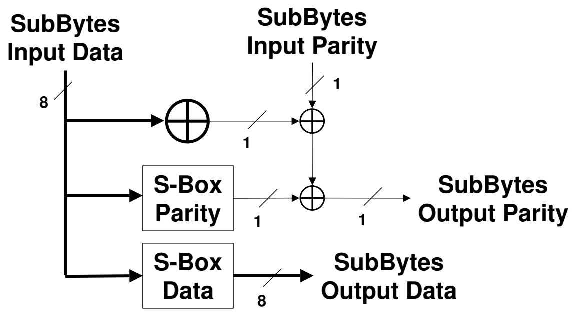

Figure 3. *Implementation of SubBytes and parity propagation by means of lookup tables.*

The basic architecture is a simple round implementation with 16 substitution tables, where each round is computed within a single clock cycle. Some logic is included to support the CBC encryption mode, which can be selected through a dedicated signal. Currently, the system computes only AES-128, mainly to keep the key schedule simple, in 11 clock cycles. A multiplexor controls the final output, setting the output to a null value until the final round is completed. The architecture has been synthesized in STMicro 0.18μm

technology using Synopsys tools: the critical path requires $8.88ns$ and about $233,095\mu m^2$; the library used did not include any wire model.

The architecture was extended to include error detection capabilities. A first implementation was obtained by including parity generators for the initial input and key, and by extending each component to include also parity prediction. A parity checker was appended at the output of the circuit, in order to validate the predicted parity and return the ciphered text. An error signal is returned when the parity checker detects any inconsistency. This solution requires about $276,467\mu m$ of silicon (18% overhead), but its main drawback is the length of the critical path. The presence of the parity checker at the end of the round increases the latency by up to $12.02ns$, which means an overhead of 35%. The number of required clock cycles is the same as that of the basic architecture.

A second version can be implemented to solve this issue, moving the parity checker out of the basic round architecture, being thus connected directly to the output of the state registers. The output will be still controlled by a multiplexor, which will give the correct result only at the end of the encryption process and if the parity checker validates the predicted parity. The main advantage would be that the parity checker is separated from the round datapath, thus shortening the critical path; on the other hand, an additional clock cycle would be required to validate the parity. This architecture is still under development and it will be completed soon. We expect an area requirement comparable to that of the version presented in this paper but a shorter latency.

Table 2. *Area and time evaluation of AES-128, with and without error detection capabilities.*

Architecture	**Area** (μm)	**Latency** (ns)	**Throughput** (Gbps)
AES-128 w/o EDC	233,095	8.88	1.31
AES-128 with EDC	276,467	12.02	0.97
Overhead	+18%	+35%	−26%

5 Conclusions

A complete parity code based scheme for fault detection in hardware implementations of AES has been described and analyzed, where single Faults in the encryption and in the key schedule units are detected. The required hardware for the fault detection has been synthesized and the resulting overheads in terms of area and delay have been evaluated. We think that the overall throughput can be increased moving the parity checker out of the critical path.

Acknowledgment: The work of Israel Koren has been supported in part by DARPA/AFRL NEST program under contract number F33615-02-C-4031.

References

[1] M. Akkar, C. Giraud, "An Implementation of DES and AES, Secure against some Attacks," *Proceedings of CHES '01*, pp. 315-325, 2001.

[2] Amphion, "High Performance AES (Rijndael) Encryption - Ultra Fast," Product Datasheet, http://www.amphion.com/capture.asp?doc=cs5240.

[3] F. Bao, R. Deng, Y. Han, A. Jeng, D. Narasimhalu, T. Nagir, "Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults," *The Second Workshop on Secure Protocols, (Pads)*, April, 1997, LNCS, Springer-Verlag, 1997.

[4] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "Error Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard," *IEEE Transactions on Computers*, Special Issue on Cryptographic Hardware and Embedded Software, pp. 492-505, April 2003.

[5] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "Detecting and locating faults in VLSI implementations of the Advanced Encryption Standard," *Proc. of the 2003 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 105-113, 2003.

[6] D. Boneh, R. DeMillo, R. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations," *Journal of Cryptology*, vol. 14, pp. 101-119, 2001.

[7] P. Chodowiec, K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm," Cryptographic Hardware and Embedded Systems - CHES 2003, *Lecture Notes in Computer Science*, vol. 2779, pp. 319-333, Springer-Verlag, 2003.

[8] C. Giraud, "DFA on AES," http://eprint.iacr.org/2003/008.ps.gz

[9] B. Gladman, "A Specification for Rijndael, the AES Algorithm," fp.gladman.plus.com/, 2001.

[10] R. Karri, W. Kaijie, P. Mishra, K. Yongkook, "Fault-based Side-Channel Cryptanalysis Tolerant Rijndael Symmetric Block Cipher Architecture," *Proc. of the 2001 Defect and Fault Tolerance in VLSI Systems*, pp. 418-426, 2001.

[11] R. Karri, G. Kuznetsov, M. Goessel, "Parity-based concurrent error detection in symmetric block ciphers," *Proceedings of International Test Conference 2003 - ITC 2003*, Volume 1, ISSN 1089-3539, pp. 919 - 926, 2003.

[12] R. Karri, G. Kuznetsov, M. Goessel, "Parity-Based Concurrent Error Detection of Substitution-Permutation Network Block Ciphers," Cryptographic Hardware and Embedded Systems - CHES 2003, *Lecture Notes in Computer Science*, vol. 2779, pp. 319-333, Springer-Verlag, 2003.

[13] R. Lidl, H. Niederreiter, *Introduction to Finite Fields and their Applications*, Cambridge University Press, 1986.

[14] S. Morioka, A. Satoh, "A 10 Gbps full-AES crypto design with a twisted-BDD S-Box architecture." *Proceedings of 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 98-103, 2002.

[15] National Institute of Standards and Technologies, "Announcing the Advanced Encryption Standard (AES)," *Federal Information Processing Standards Publication*, n. 197, November 26, 2001.

[16] A. Satoh, S. Morioka, K. Takano, S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," Advances in Cryptology - ASIACRYPT 2001, *Lecture Notes in Computer Science*, vol. 2248, Springer-Verlag, pp. 239-254, 2001.

[17] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, J.-D. Legat, "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs," Cryptographic Hardware and Embedded Systems - CHES 2003, *Lecture Notes in Computer Science*, vol. 2779, Springer-Verlag, pp. 334-350, 2003.

[18] I. Verbauwhede, P. Schaumont, H. Kuo, "Design and performance testing of a 2.29-GB/s Rijndael processor," *IEEE Journal of Solid-State Circuits*, Vol. 38, Issue 3, pp. 569-572, 2003.