Research Journal of Applied Sciences, Engineering and Technology 10(7): 824-830, 2015
DOI:10.19026/rjaset.10.2436
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: February 24, 2015 Accepted: April 2, 2015 Published: July 10, 2015

Research Article

Design and Implementation of Low Power AES SBOX with Error Detection Circuit

[1]V. Devendiran and [2]S. Letitia
[1]Faculty of Information and Communication Engineering, Anna University, Chennai, India
[2]Department of ECE, Thanthai Periyar Government Institute of Technology, Vellore, India

Abstract: Soft error is nowadays a serious problem when implementing AES SBOX algorithms in hardware. The objective of the study is to detect the error using parity bit for the AES SBOX and implementation in hardware with low area and power. This circuit can products the AES Encryption/Decryption process and systems against fault based attacks. It can also apply to any digital communication systems and security related applications.

Keywords: AES SBOX, error detection, low area

INTRODUCTION

Cryptographic algorithms play a crucial role in the information society. When we use teller machines, home banking services or credit cards, call someone on a mobile phone, get access to health care services, or buy something on the web, cryptographic algorithms are used to offer protection. These algorithms guarantee that nobody can steal our money, place a call at our expense, eavesdrop on our phone calls, or get unauthorized access to sensitive health data. Information technology keeps changing and will become increasingly pervasive, while disappearing from the eye of the user. However, this evolution keeps presenting new security challenges and there is no doubt that cryptographic algorithms and protocols will form an important part of the solution (Bertoni *et al*., 2003).

Fault detection: Fault detection and tolerance schemes for various implementations of cryptographic algorithm have recently been considered. Several motivations led to increase the reliability of these circuits. From one side the circuit implementation of cryptographic algorithms can be quite complex, increasing the probability of device failures. Fault detection is therefore helpful in finding faults during the production tests. In addition, fault tolerance schemes are very useful to on-line tolerate faults during mission time. From the other side, intentional intrusions and attacks based on the malicious injection of transient faults into the device are very efficient in order to extract the secret key (Boneh *et al*., 2001; Bertoni *et al*., 2003).

Fault based attacks:

Side channel attacks-power analysis attacks: Attacks based on fault injection are not the only way to gain access to sensitive information. In the last few years, new techniques for attacking implementations of cryptographic algorithms have been studied and developed. Unlike direct attacks on the algorithm, these techniques, called Side Channel Attacks, attempt to gather knowledge of sensitive data that may leak from a particular implementation of the cryptographic algorithm. One of the most successful side-channel attack exploits (Regazzoni *et al*., 2007; Brier *et al*., 2004) the correlation between the power consumption of a given device and the data being processed. These Power Analysis Attacks have particular relevance since for some of them; no knowledge regarding the implementation of the target device is needed in order to be effective. Several solutions have already been proposed and implemented, at different levels, to counteract these unconventional forms of attacks. However, almost all the proposed countermeasures so far have targeted a single type of side-channel attack and only very few studies have addressed the effects that one specific countermeasure can have on the resistance to a different type of attack (Kocher *et al*., 1999).

The Advanced Encryption Standard (AES) (2001) is a block cipher adopted as an encryption standard by the U.S. government. AES began immediately to replace the Data Encryption Standard (DES), which had been in use since 1976. AES outperforms DES in improved long-term security because of larger key sizes (128, 192 and 256 bits, respectively). Another major advantage of AES is the possibility of efficient implementation on various platforms. AES is suitable for small 8-bit microprocessor platforms and common 32-bit processors and it is appropriate for dedicated hardware implementations. Hardware implementations can reach throughput rates in the gigabit range.

Corresponding Author: V. Devendiran, Faculty of Information and Communication Engineering, Anna University, Chennai, India

A number of hardware implementations for the AES SBox circuit have been proposed (Zhang and Parhi, 2002) because AES Sbox was most expensive part of the circuit in terms of area. Next Section will describe in detail the algorithm of the AES and in particular the definition and the characteristics of the SBox. In this study, we focus on to detect the error using parity bit for the AES SBox.

The AES SBox performs nonlinear operation and variant with parity. The parity bit is not well-maintained after the transformation. This is the reason for developed to predict the value of the output parity bit starting from the input bit.

In this study, we focus on the design and implementation of low power AES SBox with error detection circuit Compared to previous works, our solution has produced low area and power.

MATERIALS AND METHODS

Advanced encryption standard: The Rijndael algorithm (Daemen and Rijmen, 1999) used for the AES standard implements a symmetric-key cryptographic function in which both the sender and receiver use a single key to encrypt and decrypt the information.

Although in the block length of Rijndael can be 128,192, or 256 bits, respectively the AES algorithm only adopted the block length of 128 bits. Meanwhile, the key length can be 128, 192, or 256 bits, respectively. The AES algorithm's internal operations are performed on a two dimensional array of bytes called State. The State consists of 4 rows of bytes and each row has N_b bytes. Each byte is denoted by $S_{i,j}$ ($0 \le i < 4$, $0 \le j < N_b$). Since the block length is 128 bits,

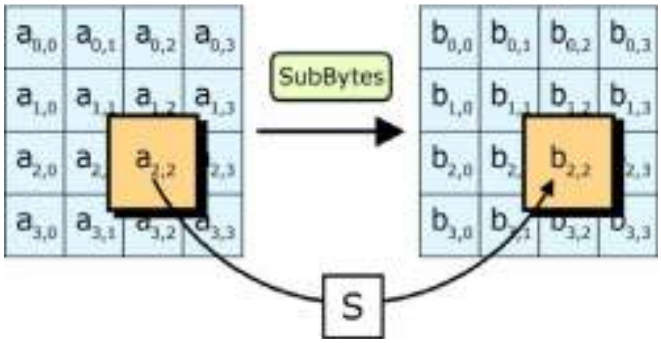


Fig. 1: Mapping of input bytes, state array and output bytes

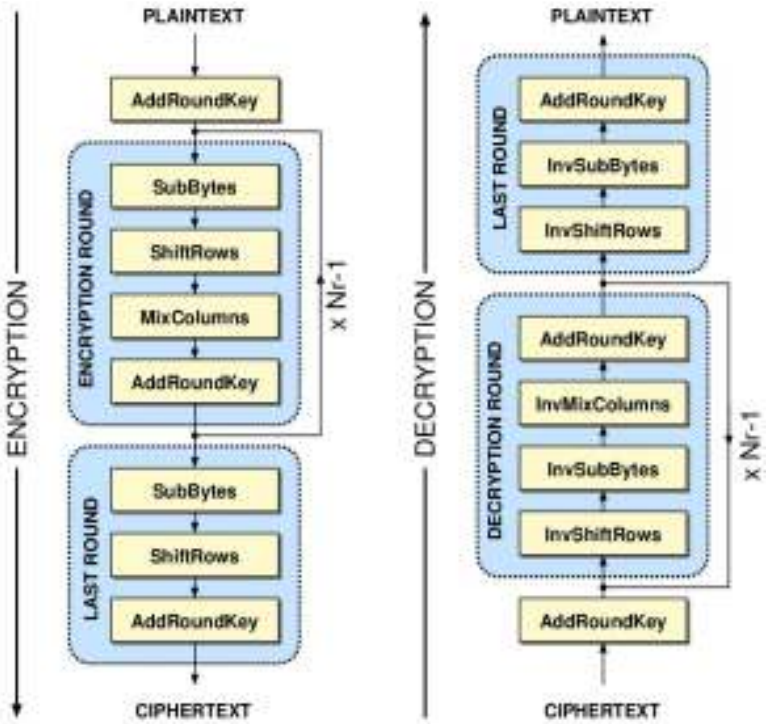


Fig. 2: AES algorithm (encryption and decryption)

each row of the State contains $N_b = 4$ bytes. For the sake of simplicity, we focus on key length equal to 128 bits. The four bytes in each column of the State array form a 32-bit word, with the row number as the index for the four bytes in each word. At the beginning of encryption or decryption, the array of input bytes is mapped to the State array as illustrated in Fig. 1. The 128-bit block can be expressed as 16 bytes: in0, in1, in 2, … in 15. Encryption and decryption processes are performed on the State, at the end of which the final value is mapped to the output bytes array out0, out1, out 2, … out15.

The AES algorithm is an iterative algorithm. Each iteration is called a round. The total number of rounds is 10. At the start of encryption, input is copied to the State array. After the initial round key addition, 10 rounds of encryption are performed. The first 9 rounds are the same, with small differences in the final round. As illustrated in Fig. 2, each of the first 9 rounds consists of 4 transformations: Sub Bytes, Shift Rows, Mix Columns and Add Round Key. The final round excludes the Mix Columns transformation. The encryption structure in Fig. 2 can be inverted to get a straightforward structure for decryption.

Sub bytes transformation: The Substitution-Box (S-box) is a basic component of symmetric key algorithms and should always be carefully chosen to create strong confusion and to resist certain kinds of cryptanalysis. The multiplicative inversion maps over Galois Field are frequently employed due to their ideal cryptographic characteristic.

This SBox is constructed by composing two transformations:

- Take the multiplicative inverse in the finite field GF (2^8)
  The element $(00000000)_2$ is mapped to itself
- Apply the following affine transformation (over GF (2)):

$$b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

For $0 \le i < 8$, where bi is the i^{th} bit of the byte and C_i is the i^{th} bit of a byte c whose value is fixed and is equal to {01100011}.

This transformation can be pre-calculated for each possible input value since it works on a single byte, therefore there are only 256 values.

Shift rows transformation: In this transformation, the bytes in the first row of the State do not change. The second, third and fourth rows shift cyclically to the left 1, 2 and 3 bytes, respectively.

Mix columns transformation: The Mix Columns transformation is performed on the State array column-by-column. Each column is considered as a four-term polynomial over GF (2^8) and multiplied by a (x) modulo x^{4+1}, where,

$$A(x) = (00000011)^2 x^3 + (00000001)^2 x^2 + (00000001)^2 x + (00000010)^2$$

Add round key transformation: In Add Round Key transformation, a round key is added to the State array by bitwise XOR operation. Each round key consists of 16 words generated from Key Expansion described below.

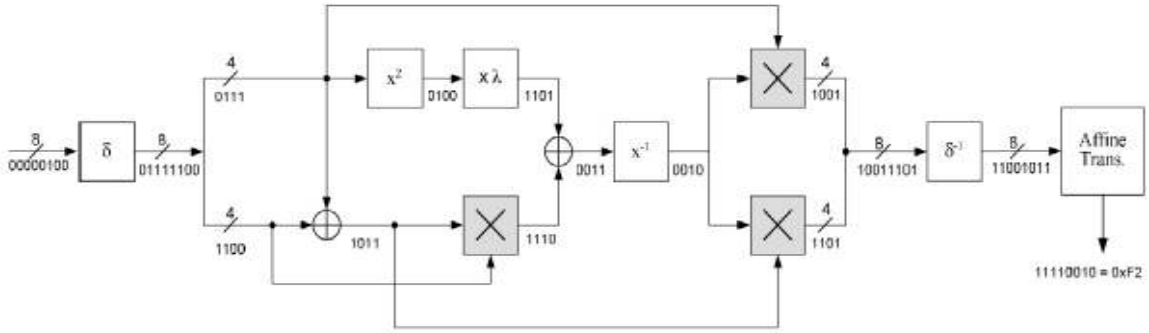
Key expansion: The key expansion routine, as part of the overall AES algorithm, takes the input key of 128 bits. The output is an expanded key of 11*128 bits, i.e., the expanded key is composed of the secret key and 10 round keys, one for each round.

Error detection circuit's overview: The AES (Rijndael) algorithm (Daemen and Rijmen, 1999) implements a block cipher for symmetric key cryptography. The block size is 128 bits, while the key size is 128, 192 or 256 bits, respectively. During the encryption process, four different transformations are iterated a number of times depending on the key size. The four basic transformations are: Shift Rows, Sub Bytes (using SBoxes), Mix Columns and finally Add Round Key. The added key is different in each round and these round keys are generated by a key schedule routine that takes the secret key and executes an expansion as specified in the standard. The same basic transformations are used during decryption, but they are applied in reverse order. For the AES S-box, we implemented four versions of the nonlinear function. The first circuit implements the nonlinear transformation as described in the standard, while in all the other three we added logic to provide error detection. In this section we summarize the solutions based on the parity bit for the SBox.

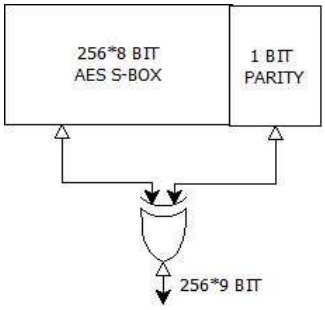
The parity check we used is the one proposed in a single odd parity bit is added to every byte. An AES S-box with an added error detection circuit (Bertoni *et al*., 2003).

The error detection circuit checks the correctness of the input and the output of the S-box. When new data enters the S-box, the check bits are separated from the data bits and error detection is performed. If no error is detected, the 8 data bits enter the S-box circuit. The S-box produces then the result of the non-linear transformation plus the corresponding check bits. At this point the second check is performed, again as described before. If no error is detected in both checks, the output of the S-box can be forwarded to the next round transformation, otherwise, a faulty output composed of all zeros except the rightmost bit is generated to signal the error.

Implementation of AES s-box error detection circuit: The SBox was implemented using 256*8 bit



(a)



(b)

Fig. 3: (a) AES S-box (b) error detection of AES S-box

memory. The Memory circuit consisting of a combination of data storage section and address decoding circuit. The input data bytes will have 8 bit data with even parity bits.

A solution to generate the outgoing parity bits is proposed in and sketched in Fig. 3a: an odd parity bit is either stored with each data byte in the SBox (memory implementation), or on-line generated by an ad-hoc combinational circuit (in the case of combinational logic implementation for the SBox). This solution is not very expensive and it guarantees acceptable fault coverage.

To increase the dependability and to detect additional input parity errors and some internal memory errors (data or decode), proposes replacing the original 8-bit decoder with a 9-bit one, yielding a 256×9 bit memory (Fig. 3b). If a 9-bit address with an even parity is decoded, the corresponding output byte with its associated even parity bit is produced. Otherwise, a constant word of 9 bits with a deliberately odd parity is output, e.g., "000000001". Thus, half of the entries in the SBox memory will be deliberately wrong (in the figure, all the rows marked with a '*'). In case of a single error in the input value, a wrong cell will be addressed. That cell will contain an erroneous parity bit that will be detected during the parity bit check. This solution guarantees higher fault coverage, but it's very expensive in terms of used area.

Proposed solution implementation of AES s-box error detection circuit: In this study, we focus on the use of the parity code for a single SBox. We propose a solution that is suitable for all the schemes where there is a parity check for each byte element of the matrix S. The main problem in implementing the parity bit for the SBox is related to the fact that the SBox transformation is not invariant with respect to the parity bit. Hence, it is necessary to implement a method to predict the output parity, given the input value.

In order to meet higher fault detection capability a code based fault detection approach has been adopted, consisting of information redundancy applied to both data and address of the memory storing the SBox values. With this solution we are able to target Address Faults as well. An Address Faults typically cause that during a read operation an unexpected cell is accessed by a given address. The use of detection codes based on both the address and the data allows the detection of Address Faults. One characteristic of the SBox is that it implements an invertible function. This feature allows calculating the input value starting from the output response. Therefore, it is possible to predict the parity bit of the input word starting from the output response of the SBox (without implementing the inverse function, see below for details). The main idea is that we do not add any parity bit in the memory that stores the SBox values (or into the combinational logic that implements it). On the contrary, we calculate the parity of the input value and we compare it with the parity bit predicted starting from the output value of the SBox. In addition, we calculate the parity bit of the output of the SBox and we compare it with the prediction of this bit

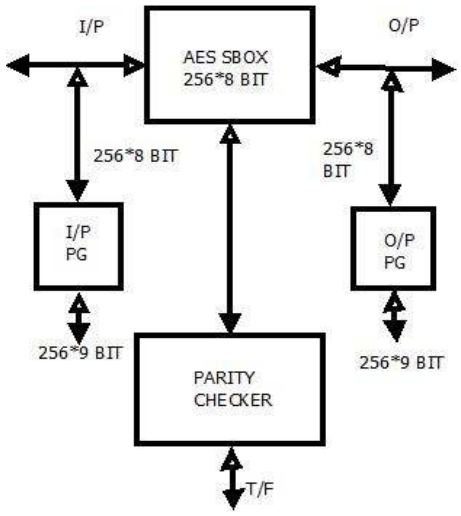


Fig. 4: Proposed architecture of AES SBOX

starting from the input value. All works presented in the past were based on the predictor of the output parity (Fig. 4).

We calculated the Output Parity Checker and the Input Parity Checker using the truth tables of the SBox and of the parity bits, calculated for both the input value and the output value. Table 1 shows the first elements of the truth tables. This scheme allows double protection of the SBox circuit and it should allow covering more faults than the architectures proposed in the literature.

This result is important when considering protecting circuits from side channel attacks the parity bit must be secured as well as the other bits.

RESULTS AND DISCUSSION

Experimental results: In this section we provide some result related to the area of the proposed approach (Karri *et al*., 2003). Analytical comparisons of the selected previous works are as summarized in Table 2.

The Proposed AES SBOX circuits have been developed in VHDL Language. It was simulated and synthesized on the Xilinx FPGA device VERTEX 2 XC2V80 as depicted in Fig. 4.

Table 1: Parity checker truth tables

Input	Parity	Output	Parity
00000100	1	11110010	1
00010011	1	01111101	0
00010001	0	10000010	0
00010101	1	01011001	0
Output parity checker		Input parity checker	
00000100	1	11110010	1
00010011	0	01111101	1
00010001	0	10000010	0
00010101	0	01011001	1

Table 2: Analytical comparisons of the selected previous works

	Used	Available	Utilization (%)
Existing work: normal SBOX			
No of slice	163	768	21
No of 4 i/p LUT	296	1536	19
No of bonded IOB	18	124	14
Proposed work: concurrent error detection SBOX			
No of slice	144	6144	2
No of 4 i/p LUT	265	12288	2
No of bonded IOB	17	320	5
Proposed work: non concurrent error detection SBOX			
No of slice	192	6144	3
No of 4 i/p LUT	297	12288	2
No of bonded IOB	18	320	2

At the different proposed AES SBOX results as listed below:

- Table 3 shows a Synthesis report of Normal AES SBOX.
- Table 4 shows a synthesis report of concurrent error detection AES SBOX.
- Table 5 shows a synthesis report of non-concurrent error detection AES SBOX.

The final simulation result was taken after replacement of Proposed AES SBOX using AES Algorithm. It was verified using MODELSIM XE III6.1E as depicted in Fig. 5.

The above said both proposed architectures have been performed in a minimized area and power.

Synthesis report of normal AES SBOX using XILINXISE 9.2I is shown in Table 3.

Synthesis report of concurrent error detection AES SBOX using XILINXISE 9.2I is shown in Table 4.

Table 3: AES normal SBOX

Device utilization summary

Logic utilization	Used	Available	Utilization (%)	Note (s)
Number of 4 input LUTs	264	12,288	2	
Logic distribution				
Number of occupied slices	167	6,144	2	
Number of slices containing only related logic	167	167	100	
Number of slices containing unrelated logic	0	167	0	
Totel number of 4 input LUTs	264	12,288	2	
Number of bonded IOBs	17	240	7	
Totel equivalent gate count for design	1,743			
Additional JTAG gate count for IOBs	816			

Table 4: AES concurrent error detection AES SBOX

CED partition summary				
No partition information was found				
Device utilization summary				
Logic utilization	Used	Available	Utilization (%)	Note (s)
Number of 4 input LUTs	264	12,288	2	
Logic distribution				
Number of occupied slices	167	6,144	2	
Number of slices containing only related logic	167	167	100	
Number of slices containing unrelated logic	0	167	0	
Totel number of 4 input LUTs	264	12,288	2	
Number of bonded IOBs	17	320	5	
Total equivalent gate count for design	1,743			
Additional JTAG gate count for IOBs	816			

Table 5: AES non concurrent error detection AES SBOX

NONCED partition summary				
No partition information was found				
Device utilization summary				
Logic utilization	Used	Available	Utilization (%)	Note (s)
Number of 4 input LUTs	297	12,288	2	
Logic distribution				
Number of occupied slices	192	6,144	3	
Number of slices containing only related logic	192	192	100	
Number of slices containing unrelated logic	0	192	0	
Totel number of 4 input LUTs	297	12,288	2	
Number of bonded IOBs	18	320	5	
Totel equivalent gate count for design	1,911			
Addition JTAG gate count for IOBs	864			

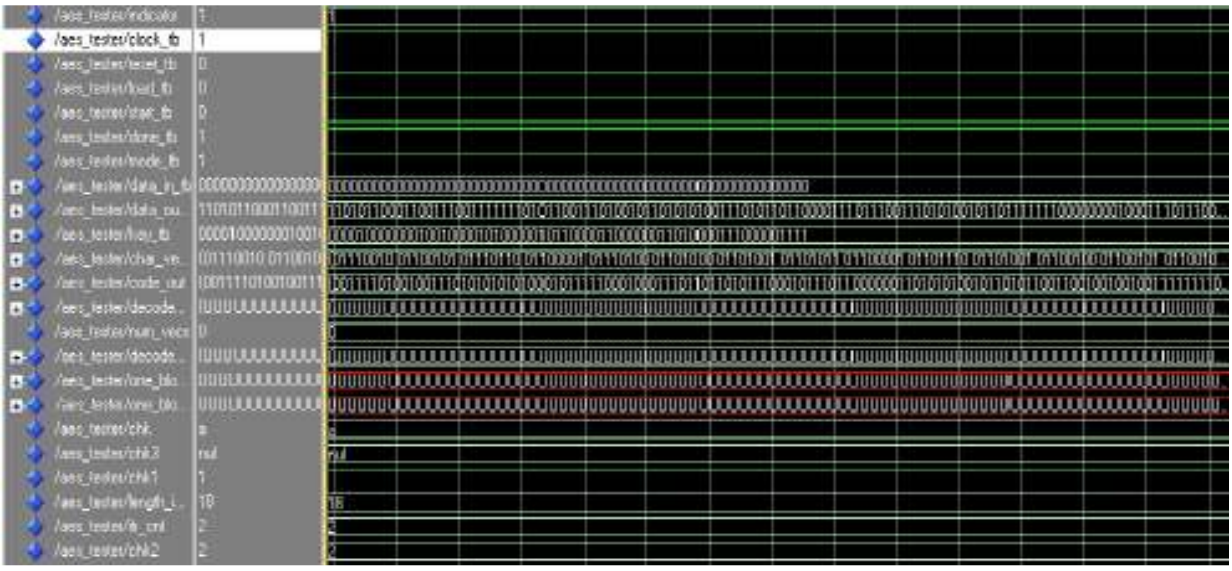


Fig. 5: AES encryption result

Synthesis report of non concurrent error detection AES SBOX using XILINXISE 9.2I is shown in Table 5.

Advanced encryption standard result using modelsim XE III6.1E is shown in Fig. 5.

CONCLUSION

In this study, the processor is designed using VHDL description language, the simulation is done using Modelsim XE III 6.1e and the design was synthesized on the Xilinx FPGA device VERTEX 2 XC2V80. The simulation results show that the processor provides a very good performance. Its Utilize a maximum 5% of area compare to the Normal AES SBOX area to a maximum of 21%.

The proposed future works to develop the circuit for error detection using sub pipeline architecture.

REFERENCES

Advanced Encryption Standard (AES), 2001. Federal Information Processing Standards Publication 197. Retrieved from: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

Bertoni, G., L. Breveglieri, I. Koren, P. Maistri and V. Piuri, 2003. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. IEEE T. Comput., 52: 492-505.

Boneh, D., R. DeMillo and R. Lipton, 2001. On the importance of eliminating errors in cryptographic computations. J. Cryptol., 14: 101-119.

Brier, E., C. Clavier and F. Olivier, 2004. Correlation power analysis with a leakage model. In: Joye, M. and J.J. Quisquater (Eds.), CHES, 2004, LNCS 3156, Springer-Verlag, Berlin, Heidelberg, pp: 16-29.

Daemen, J. and V. Rijmen, 1999. AES Proposal: Rijndael. Retrieved fron: http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf.

Karri, R., G. Kuznetsov and M. Goessel, 2003. Parity-based concurrent error detection of substitution-permutation network block ciphers. In: Walter, C.D. *et al*. (Eds.), CHES, 2003. LNCS 2779, Springer-Verlag, Berlin, Heidelberg, pp: 113-124.

Kocher, P.C., J. Jaffe and B. Jun, 1999. Differential power analysis: Leaking secrets. Proceeding of Crypto '99. LNCS 1666, Springer-Verlag, pp: 388-397.

Regazzoni, F., T. Eisenbarth, J. Großsch¨adl, L. Breveglieri, P. Ienne, I. Koren and C. Paar, 2007. Power attacks resistance of cryptographic S-boxes with added error detection circuits. Proceedings of the 21st IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems.

Zhang, X. and K.K. Parhi, 2002. Implementation approaches for the advanced encryption standard algorithm. IEEE Circ. Syst. Mag., 2(4): 24-46.