Ewa IDZIKOWSKA

Poznań University of Technology

doi:10.15199/48.2017.01.23

An operation-centered approach to fault detection in key scheduling module of cipher

Abstract. *In this paper a technique for fault detection in hardware implementation of the PP-1 symmetric block cipher has been studied. Simulations of the behaviour of fault propagation in the key scheduling process is reported. The simulation proves that both parts of the algorithm, data-path and control, should be protected. Previous studies [1, 2] have only considered the data-path, ignoring the key scheduling. A proposal for fault detection in key scheduling is presented, which require a limited amount of circuit overhead and does not require modification of the PP-1 algorithm.*

Streszczenie. *W pracy przedstawiono metodę wykrywania błędów w sprzętowej implementacji szyfru PP-1. Skupiono się na module generowania kluczy rundowych. Pokazano propagację błędów w tym module a tym samym to, że ważne jest wykrywanie błędów nie tylko w module przetwarzania danych ale także podczas wyznaczania kluczy rundowych. Zaproponowano metodę wykrywania błędów, która nie wymaga modyfikacji samego algorytmu PP-1 i nie wprowadza dużej nadmiarowości sprzętowej ani czasowej. (**Metoda wykrywanie błędów w module generowania kluczy szyfratora, skupiona na operacjach**).*

Keywords: PP-1 cipher, Concurrent Error Detection, round key scheduling, parity bits.
Słowa kluczowe: szyfr PP-1, współbieżne wykrywanie błędów, generowanie kluczy rundowych, bity parzystości.

1. Introduction

Secure implementations of cryptographic systems have received much attention. Conventional cryptanalysis deals with the mathematical properties of a system, and the physical cryptanalysis focuses on the physical behaviour of a system during operation. Basically, all assumptions for all kinds of physical attacks apply to all ciphers when they are implemented. Each attack can be different from the others, depending on the actual implementation and depending on the properties of the cipher.

Differential fault analysis (DFA) is a method of physical cryptanalysis and was originally proposed by Biham and Shamir in 1997 [3]. It assumes that an attacker can induce faults into a cipher and collect the correct as well as the faulty behaviours. Then the attacker compares the behaviors in order to retrieve the secret information embedded inside the cipher. It means that fault detection is a desirable property for preventing malicious attacks, aimed at extracting sensitive information, like the secret key, from the device.

There are different types of faults and methods of fault injection in encryption algorithms. The faults can be transient or permanent. Several transient and permanent faults and methods of fault injection such as varying supply voltage, external clocks, temperature or inducing faults using white light, laser and X-rays methods of fault injection are discussed in detail in [4].

Concurrent Error Detection (CED) techniques are widely used to enhance system dependability. The proposed solutions consist of using various forms of redundancy to obtain an attack-resistant architecture. These solutions have different area overhead, performance penalty, and fault detection latency [5, 6, 7].

This paper recommends possible countermeasure against the fault attacks. The countermeasure is, mainly, a parity check method to verify the correctness of the round key. Proposed method does not require modification of the PP-1 algorithm. We develop the model presented in [1, 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 provide simulation results related to the fault coverage of the proposed approach.

This paper is organized as follows. Sec. 2 and 3 present the PP-1 block cipher - processing path and key scheduling module respectively. In Sec. 4 error propagation in key scheduling module is shown. Possible faults and faults models are described in Sec. 5. In Sec. 6 we present CED schemes. Simulation results are presented in Sec. 7. Sec. 8 concludes the paper.

2. Processing path of PP-1 cipher

The scalable PP-1 cipher is a symmetric block cipher designed at the Institute of Control and Information Engineering, Poznań University of Technology. It was designed for platforms with limited resources, and it can be implemented for example in simple smart cards.

The PP-1 algorithm is an SP-network. It processes in r rounds data blocks of n bits, using cipher keys with lengths of n or $2n$ bits, where $n = t*64$, and t = 1, 2, 3, ... One round of the algorithm is presented in Fig. 1. It consists of $t=n/64$ parallel processing paths. In each path the 64-bit nonlinear operation NL is performed. Additionally the n-bit permutation P is used. In the last round, the permutation P is not performed. These algorithm is presented in [6].

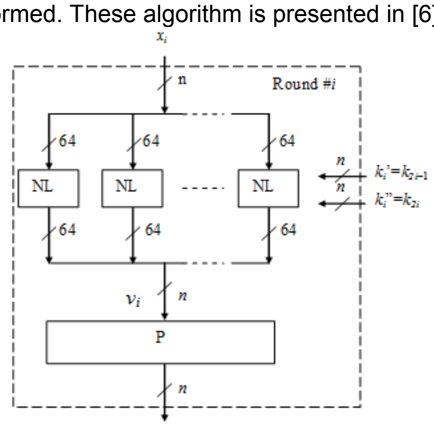

Fig. 1.One round of PP-1 (i = 1, 2, ..., r-1) [6]

Two n-bit round keys $k_i'=k_{2i-1}$ and $k_i''=k_{2i}$ are used in round i, where i = 1, 2, ..., r. Let j denote the number of the parallel processing paths from left to right, j = 1, 2, ..., t. Then $k_i' = k_{i,1}'|| k_{i,2}'||...||k_{i,t}'$, $k_i'' = k_{i,1}''|| k_{i,2}''||...||k_{i,t}''$.

The 64-bit round subkeys $k_{i,j}'$ and $k_{i,j}''$ used in the element NL #j, consist of eight 8-bit elementary keys $k_{i,j,l}$ (l = 1, 2, ..., 8), so $k_{i,j}' = k_{i,j,1}'|| k_{i,j,2}'||...|| k_{i,j,8}'$ and $k_{i,j}'' = k_{i,j,1}''|| k_{i,j,2}''||...|| k_{i,j,8}''$ [6].

The same algorithm is used for encryption and decryption. However, if in the encryption process we use the round keys k_1, k_2, ..., k_{2r} then in the decryption process these keys must be used in the reverse order, i.e. k_{2r}, k_{2r-1}, ..., k_1.

3. Round key scheduling

The round key scheduling is performed in 2*r*+1 iterations (*i* = 0, 1, ..., 2*r*), where *r* is the number of rounds. One iteration of key scheduling is presented in Fig. 2. The round keys k_1, k_2, ..., k_{2r} are produced on outputs of iterations #1 to #2*r* [6].

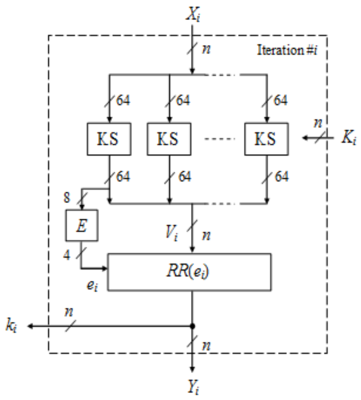

Fig. 2. One iteration of key scheduling (i = 0, 1, ..., 2r) [6]

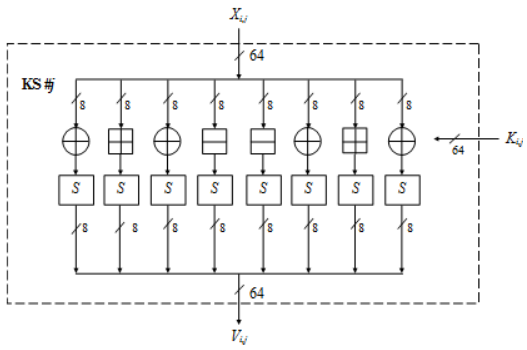

Fig. 3. KS − the main part of an iteration (j = 1, 2, ..., t) [6

The KS element of the iteration is shown in Fig.3. It is composed of substitution *S,* XOR, addition and subtraction modulo 256. The operation $RR(e_i)$ is the rotation of *n*-bit block V_i by e_i bits to the right. The 4-bit integer e_i is obtained as the result of the XOR operation for 4-bit arguments, which are the 4 most significant bits of the output of the two leftmost S-boxes. Thus for $V_i = v_1v_2...v_n$, where v_1 is the most significant bit, the value of e_i is calculated as follows:

$$e_i = E(v_1v_2...v_n) = (v_1 \oplus v_9)(v_2 \oplus v_{10})(v_3 \oplus v_{11})(v_4 \oplus v_{12}).$$

4. Error propagation in key scheduling

The error propagation behavior of the data path (i.e., the encryption or decryption process) was studied in [1]. Another part of the algorithm implementation that can be affected by faults is the key schedule. A single faulty bit injected during the round key computation process may cause a large number of erroneous bits in the next round keys. At Fig. 4 a single fault was injected in the first round and at Fig. 5 in the last round. Italic font indicates erroneous key bits in subsequent rounds (right side of the figure).

Error propagation analysis was carried out to understand the effect of an error injected into the round key computation. Experiments were conducted by injecting a single bit flip error at different bits randomly and the number of bits that were in error was computed. One faulty bit injected in one of the inputs of S-boxes in the first round causes about 52% faulty bits in the next rounds.

This analysis helps us in choosing suitable error detection schemes.

5. Faults models

Fault attack tries to modify the functioning of the computing device in order to retrieve the secret key. The attacker induces a fault during cryptographic computations. The feasibility of a fault attack or at least its efficiency depends on the exact capabilities of the attacker and the type of faults he can induce.

In our considerations we use a realistic fault model wherein either transient or permanent faults are induced randomly into the device. We consider single and multiple faults.

Faults are modelled as a *m* - bit error vectors $E=\{e_m,...,e_i,...,e_1\}$, where $m \in \{4,8,n\}$, $e_i \in \{0,1\}$ and $e_i=1$ indicates that bit *i* is faulty. The number of ones in this vector is equal the number of inserted faults. Fault simulations were performed for two kind of fault models. In one model the fault flips the bit, and the other model introduces bit stuck-at faults (stuck-at-1 and stuck-at-0).

Let $X=\{x_m,...,x_1\}$ be an error-free vector of bits. Vector $Xe=\{xe_m,...,xe_1\}$ is an erroneous input vector:

- for bit flip error— $xe_i = x_i \oplus e_i$,
- for stuck-at-1 fault — $xe_i = x_i + e_i$,
- for stuck-at-0 fault — $xe_i = x_i \times$ (not e_i),

where: ⊕ - xor,+ - or, × - and operations.

0: 0010111000101100010001001110001000011000011110110010100010010101	0: 0010111000101100010001001110001000011000011110110010100010010101
1: 1100111110010010001001010110100011101101001101100100100100101101	1: 1110010010001110010010100011101101001101100100100100101101110011
2: 1100101101111011100111101010101001100011111000010101111111010010	2: 0110011100110111110111100101010100010010001100111110011110101000
3: 1101101101111101001101110000000110101001001001000000111000000110	3: 0000011111111011011110010000001001010010010011001110101011011011000
4: 1011010011010111000110001111000110111010110100101110110100110000	4: 0111100110010010010001011010001111001101100000001000101001010110
5: 1011110000111100101010111000010100000010110110100110100011101100	5: 1011000010111110111101000101000100111000010101101101001011101100
6: 0101111101010101110111110111111111000011001000011010100000000001	6: 1100000101011111001101100101110100000100101000000001001001001001
7: 0100111010010001111011011001000100110100110111111111101100011000	7: 0101110001011001100111111001000011010010001000111100101111011110
8: 1110010100100011010001100000100101011101011000011010110001100110	8: 0110110110100110011110011000000100101100001000001101001010011000
9: 1011100100001100100100110010101111000100010110110111000010101001	9: 1011111010111110111101011111010000010101010010001010010110010011 1
10: 0101110001100000111010100101101111011111001110100111011111100010	10: 0001110111111101100111100100001101010010111001010010000010100 1101
11: 0101110010100101111100110011010111100101110100011011100001011100	11: 1010100110110011010111011001011101010000000011110111000011001011
12: 1010000111000100000011000100001000110000000100111010100000001001000	12: 1000010110000000111010001000011001111110100101011100010011100110
13: 1100011110001100101100001011101111001111000011100011100010000001	13: 0111101011111100111100101110000011111110110111100101011101000110
14: 1001010111010011001110010010111000111100100000001001111110001101 0	14: 0010001101100100101010101000111000001010101100110110010010100111
15: 0110111010001110011111011010001000100011000100000001010100111001 0	15: 0000010111110101110010101011011101111100100011010100100010101111 1
16: 1011001000111001100101011100000101101100001001000111110010111101	16: 1110010111101100101011111011011111100110110011100010011001101010
17: 0110011101100000100101000011010011110110101110101100110100011010	17: 0101000000110000001001101000101100100000101000101001110001100001 0
18: 0000010101111001001011111010010010000000100111001101001111011011	18: 0001111001100001101101100000101110011010001001010001110000000100
19: 0101101011001011011101010000101010001101101100101100010001000001	19: 0111100111000100000010010111010101000111010011010001000001100011 11
20: 0101000000110110010111101100100011000110100000010110010100110001 0	20: 0011100111001101001101111101110100111000011100011000101011010100
21: 0101100011011001010101010110011000111010011011010010000100010100	21: 0001010111011111111111110000110100110101010001000011011110000110 1
22: 1111010010101111111010010101011010101011000111100101001001111101	22: 0110101001000100100011110010010101001111111010100011010010011000

Fig. 4. A single fault injected in the first round of key scheduling

0: 0010111000101100010001001110001000011000011110110010100010010101	0: 0010111000101100010001001110001000011000011110110010100010010101
1: 1100111110010010001001010110100011101101001101100100100100101101	1: 1100111110010010001001010110100011101101001101100100100100101101
2: 1100101101111011100111110101010100110001111100001010111111010010	2: 1100101101111011100111110101010100110001111100001010111111010010
3: 110110110111111010011011100000001101010010010010000000111000000110	3: 110110110111111010011011100000001101010010010010000000111000000110
4: 1011010011010111000110001111000110111010110100101110110100110000	4: 1011010011010111000110001111000110111010110100101110110100110000
5: 10111100001111001010101110000101000000010110110100110100011101100	5: 10111100001111001010101110000101000000010110110100110100011101100
6: 01011111010101011110111110111111111100001100100001101010000000001	6: 01011111010101011110111110111111111100001100100001101010000000001
7: 0100111010010001111011011001000100110100110111111111101100011000	7: 0100111010010001111011011001000100110100110111111111101100011000
8: 1110010100100011010001100000100101011101011000011010110001100110	8: 1110010100100011010001100000100101011101011000011010110001100110
9: 1011100100001100100100110010101111000100010110110111000010101001	9: 1011100100001100100100110010101111000100010110110111000010101001
10: 01011100011000001110101001011011110111110011101001110111111100010	10: 01011100011000001110101001011011110111110011101001110111111100010
11: 0101110010100101111100110011010111100101110100011011100001011100	11: 0101110010100101111100110011010111100101110100011011100001011100
12: 1010000111000100000110001000010001100000010011101010000001001000	12: 1010000111000100000110001000010001100000010011101010000001001000
13: 1100011110001100101100001011101111001111000011100011100010000001	13: 1100011110001100101100001011101111001111000011100011100010000001
14: 10010101110100110011100100101110001111001000000010011111100011010	14: 10010101110100110011100100101110001111001000000010011111100011010
15: 01101110100011100111110110100010001000110001000000010101001110010	15: 01101110100011100111110110100010001000110001000000010101001110010
16: 1011001000111001100101011100000101101100001001000111110010111101	16: 1011001000111001100101011100000101101100001001000111110010111101
17: 0110011101100000100101000011010011110110101110101100110100011010	17: 0110011101100000100101000011010011110110101110101100110100011010
18: 00000101011110010010111110100100100000001001110011010011110111011	18: 00000101011110010010111110100100100000001001110011010011110111011
19: 0101101011001011011101010000101010001101101100101100010001000001	19: 0101101011001011011101010000101010001101101100101100010001000001
20: 01010000001101100101111011001000110001101000000101100101001100010	20: 01010000001101100101111011001000110001101000000101100101001100010
21: 0101100011011001010101010110011000111010011011010010000100010100	21: 0101100011011001010101010110011000111010011011010010000100010100
22: 1111010010101111111010010101011010101011000111100101001001111101	22: 1010001011100101010110010101010001111001010010011111011111010010

Fig. 5. A single fault injected in the last round of key scheduling

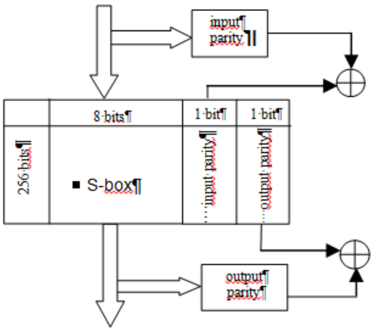

Fig. 6. Parity based CED with input and output parity bits

CED

When the data-path is assumed to be fault-free and the key scheduling is affected by the injection of a single faulty bit at some round, it has been verified that a faulty bit injected in the early rounds causes a high number of erroneous bits. If the erroneous round key is used for decryption, it is not possible to detect the presence of a faulty bit in the key material. The sender will be unable to realize that the transmitted encrypted data is corrupted and the receiver will decrypt useless data. Consequently, special attention must be paid to the fault management of the round key. The operations are the same as in the case of the data processing path.

A proposal for error detection in the data-path of PP-1 was described in [5]. The goal there was to prevent an attacker from breaking the cipher system by injecting one or more incorrect bits.

In this paper we will analyse the possibilities of errors detection in the part of key schedule. As it mentions above, the operations are the same as in the case of the data processing path, it means substitution box *S*, XOR, addition and subtraction modulo 256. Besides the operation $RR(e_i)$ is used. It is the rotation of *n*-bit block V_i by e_i bits to the right. Each of these operations is protected.

In [5] has been proposed, implemented and tested a parity based method of concurrent error detection in S-boxes. The S-box is usually implemented as a 256x8 bits memory, consisting of a data storage section and an address decoding circuit. To increase the dependability and detect input, output and internal memory errors of the S-box we propose replacing the 256x8 bits memory that stores the S-box values with 256 x10 bits memory. One of these two additional bits is parity bit generated for incoming data bytes, the other one is parity bit generated for outgoing data (Fig. 6).

In our experiments we focused on transient and permanent, single and multiple stuck-at faults and bit flips faults. Single, transient stuck-at-0/1 errors are detected in 50%, but permanent errors are detected in 100%. Detection percentage for single bit flip errors is close to 100%. The same is observable for permanent and transient errors.

Two another operations - XOR and $RR(e_i)$ (rotation of *n*-bit block V_i by e_i bits to the right) we protect using parity code. Parity bits are capable of detecting all single bit errors and those multiple bit errors where the number of errors is odd. We cannot, however, employ just a single parity bit for fault detection. As it shown in Sections 4, errors spread quickly throughout the key scheduling block and, on the average, about half of the state bits become corrupt. Hence, the fault coverage of the parity bits would be at best around 50%, which is unacceptable in practice.

To circumvent these problems, we propose to associate one parity bit with each input/output byte of XOR element (Fig. 7).

$$p(A + K) = p(A) \oplus p(K)$$

where: A – data byte, K – key byte.

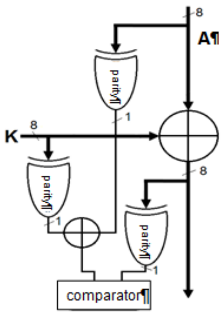

Fig. 7. Parity based CED for XOR operation

In this way each parity bit will depend only on a limited portion of the data (8 bits).

Rotation $RR(e_i)$ we protect using only one parity bit for input data, and one for output data and we detect only single bit errors and those multiple bit errors where the number of errors is odd.

A method of CED for two successive operations, addition and subtraction modulo 256 is shown in the Fig. 8. We use an inverse operation for each data byte. In this case area overhead is more as 100% but all errors are detected.

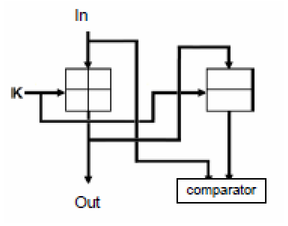

Fig. 8. CED for addition operation

6. Simulation results

In this section, we provide simulation results related to the fault coverage of the proposed approach. We present simulation results on the vulnerability of these techniques for fault models from Section 5. The faults were introduced on inputs, outputs of all operations and into internal memory of the S-box.

In order to measure the detection capability we used VHDL hardware description language and the VHDL simulator provided by Aldec, Active-HDL. The VHDL model of the key scheduling module of the PP-1 cipher has been modified with the faults. In our considerations we used a realistic fault model wherein faults are induced randomly into the device at the beginning of the rounds, i.e., faults are not injected between the round operations. In this experiment we focused on transient and permanent, single and multiple stuck-at faults and bit flips faults.

We perform a check at the output of each round operations (Figs. 9 and 10) and at the end of every round (Fig. 11). In the first case it is determined the probability of detecting all injected faults. Each security module operates independently of the others and detect errors only in its area.

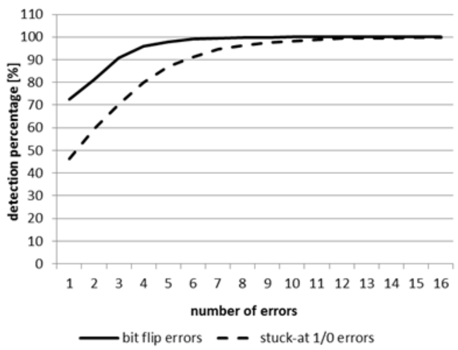

Fig. 9. Probability of permanent errors detection

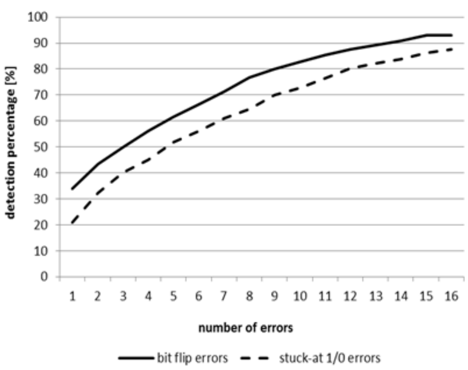

Fig. 10. Probability of transient errors detection

In the second case we determine the probability of detecting only those faults that changed the round keys. In this case all single, permanent errors are detected. In the Fig. 11 we can see, that the percentage of undetected multiple, permanent errors is small (less than 0.15%) and decreases with the number of bit errors. We can say that according to an exponential law.

Percentage of undetected transient errors is greater and is maximum 1.2%.

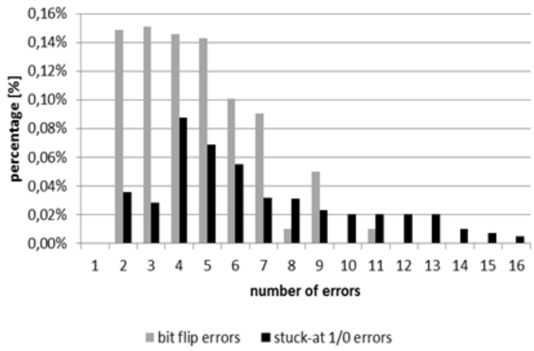

Fig. 11. Permanent faults undetected at the end of round

7. Conclusion

Fault attacks are becoming a serious threat to hardware implementations of ciphers and proper countermeasures must be adopted to foil them. The simulation proves that both parts of the algorithm, data-path and control, should be protected. Previous studies [1, 2] have only considered the data-path, ignoring the key scheduling. In this paper we have presented an operation-centered approach to the incorporation of fault detection into cryptographic device implementations with the small hardware overhead. This method of error detecting can provide a useful protection against fault attacks and, in general, against errors occurring during the encryption process. It provide full coverage of single-bit errors and high coverage of multiple-bit errors, which are the most common in fault attacks. A proposed fault detection method in key scheduling module required a limited amount of circuit overhead and does not require modification of the PP-1 algorithm.

***Authors**: dr. inż. Ewa Idzikowska, Politechnika Poznańska, Instytut Automatyki i Inżynierii Informatycznej, ul. Piotrowo 3a, 60-965 Poznań, e-mail: ewa.idzikowska@put.poznan.pl*

The correspondence address is:
e-mail: *ewa.idzikowska@put.poznan.pl*

REFERENCES

[1] Idzikowska E., Bucholc K., Error detection schemes for CED in block ciphers, *Proc. of the 5th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing EUC, Shanghai,* (2008, 22-27
[2] Idzikowska E., CED for involutional functions of PP-1 cipher, *Proceedings of the 5th International Conference on Future Information Technology*, Busan, (2010)
[3] Biham E., Shamir A.: Differential fault analysis of secret key cryptosystems, *Proc of Crypto*, (1997)
[4] Bar-El H., Choukri H., Naccache D., Tunstall M., Whelan C.: The Sorcerer's Apprentice Guide to Fault Attacks. *Proc. IEEE,* vol. 94, (2006), 370-382
[5] Idzikowska E., CED for S-boxes of symmetric block ciphers, *PAK* vol. 56, No. 10, (2010), 1179-1183
[6] Bucholc K., Chmiel K., Grocholewska-Czuryło A., Stokłosa J., PP-1 block cipher, *Polish Journal of Environmental Studies*, vol. 16, No. 5B, (2007), 315−320
[7[Bertoni G., Breveglieri L., Koren I., Maistri P., and Piuri V., On the Propagation of Faults and Their Detection in a Hardware Implementation of the Advanced Encryption Standard, *Proc. Conf. Application-Specific Systems, Architectures, and Processors* (ASAP '02), (2002) 303-312