Ewa IDZIKOWSKA
POZNAŃ UNIVERSITY OF TECHNOLOGY,
pl. M. Skłodowskiej-Curie 5, 60-965 Poznań, Poland

Analysis and detection of errors in hash function HaF-256

Dr inż. Ewa IDZIKOWSKA

She received the MSc degree in computer science from Wroclaw University of Technology and the PhD degree in computer science from AGH University of Science and Technology, Cracow. She is currently a researcher at Poznan University of Technology. Her research interests include reliability and diagnosis of logical circuits, test generation, fault diagnosis, and concurrent error detection, especially in hardware implementations of crypto¬systems.

e-mail: ewa.idzikowska@put.poznan.pl

Abstract

HaF-256 (Hash Function) is a dedicated cryptographic hash function considered for verification of the data integrity. It is suitable for both software and hardware implementation. HaF has an iterative structure. This implies that even a single transient error at any stage of the hash value computation results in a large number of errors in the final hash value. Hence, detection of errors becomes a key design issue. Concurrent checking of cryptographic chips has also a great potential for detecting faults injected into a cryptographic chip to break the key. In this paper the propagation of errors in the VHDL model of HaF-256 is shown, and then the error detection scheme based on hardware duplication is proposed and analysed. There was achieved 100% fault coverage in the case of single and multiple, transient and permanent bit flip faults.

Keywords: hash function, HaF-256, concurrent error detection, DWC.

Analiza i detekcja błędów w funkcji skrótu HaF-256

Streszczenie

Funkcja skrótu HaF-256 jest funkcją kryptograficzną używaną do kontroli integralności danych. Jej implementacja może być zarówno programowa jak i sprzętowa. HaF ma strukturę iteracyjną. Sprawia to, że nawet pojedynczy, przemijający błąd wprowadzony w dowolnym miejscu cyklu obliczeniowego skutkuje dużą liczbą błędów w wyznaczonej wartości skrótu. Celowe wprowadzanie błędów to jeden z możliwych ataków na funkcje kryptograficzne, stąd współbieżne wykrywanie błędów to jeden ze sposobów przeciwstawiania się tym atakom. W pracy pokazana została propagacja błędów w modelu VHDL funkcji skrótu HaF-256. Następnie zaproponowano zabezpieczenie elementarnych operacji tej funkcji poprzez ich duplikację i porównywanie wyników. Badania symulacyjne zaproponowanego zabezpieczenia wykonano za pomocą symulatora Active-HDL firmy Aldec. Badano skuteczność zabezpieczeń dla błędów stałych i przemijających a także dla błędów pojedynczych i wielokrotnych. W badaniach uwzględniano dwa modele błędów. Jeden polegający na przyjmowaniu przez ścieżkę stałej wartości 1 lub 0 (stuck-at-0/1 fault) drugi na przyjmowaniu przez ścieżkę wartości przeciwnej do zadanej (bit flip fault). Uzyskane wyniki pozwoliły stwierdzić, że błędy polegające zmianie wartości na przeciwną zostały wykryte w 100%. Dotyczy to zarówno błędów pojedynczych jak i wielokrotnych a także stałych i przemijających. Wykrywalność błędów typu sklejenie ze stałą wartością 0 lub 1 jest niższa i została przedstawiona w pracy.

Słowa kluczowe: funkcja skrótu, HaF-256, współbieżne wykrywanie błędów, DWC.

1. Introduction

A hash function H is a transformation that takes an input m and returns a fixed-size string, which is called the hash value h (that is, $h = H(m)$). Hash functions with just this property have a variety of general computational uses, but when employed in cryptography, the hash functions are usually chosen to have some additional properties, e.g. $H(m)$ must be relatively easy to compute for any given m, one-way and collision-free [10]. A cryptographic hash function plays very important role in the provision of message integrity checks, digital signatures, password storage and verification etc.

Hush functions are computationally complex, and in order to satisfy the high throughput requirements of many applications, they are often implemented by means of VLSI (Very Large Scale Integration) devices. The high complexity of such implementations raises concerns regarding their reliability. There is a need to develop methodologies and techniques for designing robust cryptographic systems, and to protect them against both accidental faults and intentional intrusions and attacks, in particular those based on the malicious injection of faults into the device for the purpose of extracting the secret information [3] and [4]. If an attacker deliberately generates a glitch attack, causing a flip-flop state to change or corrupt data values when they are transferred from one digest operation to another, even a single fault can result in multiple errors in the hash value computed. The severity of the problem necessitates detection of errors a key design issue. A digest round consists of several operations. Errors can creep in at any of these operations and can affect one or several bits at any of the operations in a digest round. As HaF is considered for use in security services, concurrent error detection is important. This necessitates an analysis on the propagation of error from the point of origin to the output.

CED has certain associated penalties such as hardware cost and the performance degradation due to interaction between the circuit and the detection logic, which need to be considered while designing the error detection circuit. The design goal of the CED is to achieve 100% error detection with minimal penalty. CED techniques involve redundancy in the form of hardware [8], time [9] or information. A CED circuit based on hardware redundancy can for example duplicate the complete circuit. It means that hardware overhead is more than 100%. In time redundancy, the same hardware is used to perform both the normal computation and re-computation using the same input data. The advantage of this technique is that it uses minimum hardware. The drawbacks of this technique are that it entails ≥100% time overhead and it can only detect transient faults. In information redundancy technique, data are appended with additional bits and a coding scheme is used to detect errors. Coding techniques marginally increase the hardware as well as performance overhead. Combinations of the above techniques are also employed to minimize the overhead for CED [1] and [5].

In this paper the propagation of errors in HaF-256 is studied. There are taken into consideration transient as well as permanent faults injected at different stages of hash value computation. It is found that even a single error injected resulted in half the bits of hash value being in error and the errors are spread across the computed hash value. Next the author focuses on hardware redundancy CED techniques and proposed error detection schemes based on hardware duplication. To protect basic operations such as multiplication mod (2^n+1), addition modulo 2, addition modulo 2^n we used DWC (Duplication With Comparison) scheme. The proposed approach is tested and the results are presented.

This paper is organized as follows. Section 2 presents hash function HaF-256. There is shown a method of one block processing, round function, operations of step function. In Section 3 faults models are presented. In Section 4 the error analysis is carried out to understand the effect of an error injected into the hash computation circuit. The error detection scheme is described in Section 5. Simulation results are given in Section 6. The concluding remarks are contained in Section 7.

2. HaF-256

The general model for HaF-256 is shown in Fig. 1 [2].

The original message m has to be formatted before hash value computation begins. After formatting the message m we have the message M and this massage is divided on blocks M_0, M_1, …, M_{k-1}. Each block M_i is processed with the salt s by the iterative compression function φ. After processing all blocks we receive the hash value $h(m) = H_k$ as the result.

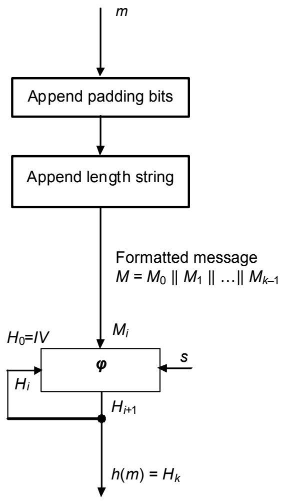

Fig. 1. Model of hash function HaF-256 [2]
Rys. 1. Model funkcji sktótu HaF-256 [2]

The length of formatted message M should be a multiple of 256 bits. It means, that the length of the input block equals 256 bits.

The block M_i is processed in two rounds.

The round function (Fig. 2) has two inputs N_i, H_i, ($H_0=IV$ is an initial value) and two outputs N_i^*, H_i^*. Before processing in round #*1*, the block M_i is modified, $N_i = M_i \oplus s$, where s is a salt, $|s| = 256$. In the round #l ($l \in \{1, 2\}$) four least significant bits of N_i indicate the number of bits the string N_i is rotated to the left ($<<lsb_4(N_i)$) and added (mod 2 of respective bits) to H_i. Next the block $H_i \oplus (N_i << lsb_4(N_i))$ is divided into 16 subblocks of equal length: A_0, A_1, …, A_{15}. They are processed by a step function and after processing they are concatenated giving H_i^*. The output $N_i^* = N_i << lsb_4(N_i)$. Before processing in the round #2 the blocks are permuted: $N_i = H_i^*$ and $H_i = N_i^*$.

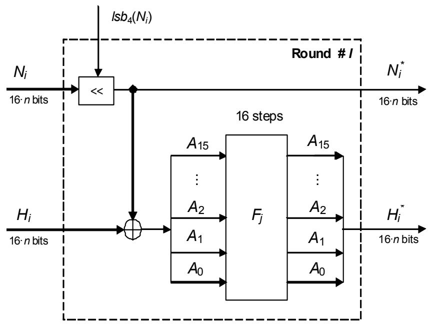

Fig. 2. Round function [2]
Rys. 2. Funkcja rundowa [2]

Each round consists of 16 steps. The step #j is indicated by the integer $j \in \{0, 1, \ldots, 15\}$. Operations creating a function F_j are performed in each step (Fig. 3).

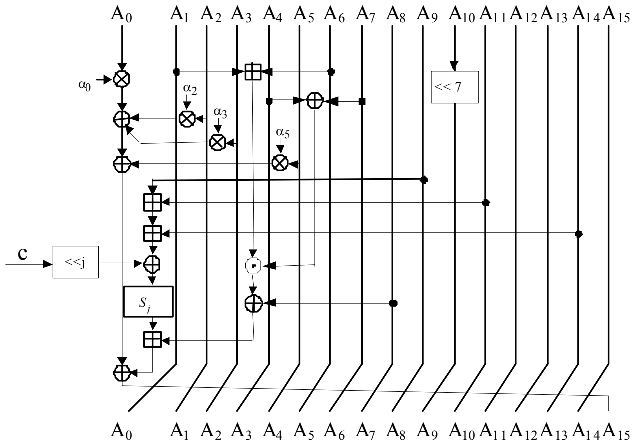

Fig. 3. Step function F_j [2]
Rys. 3. Funkcja F_j [2]

In Fig. 3 the following notations are used:

$a \odot b$ – multiplication mod (2^n+1) of n-bit non-zero integers a and b,

$v << t$ – t-bit left rotation of a string v, $|v| = 16n$,

$|v|$ – the length in bits of a string v,

$v \oplus w$ – bitwise XOR of strings v and w, $|v| = |w|$,

$v \boxplus w$ – addition mod 2^n of integers represented (in base 2) by strings v and w,

$p_1(x) \otimes p_2(x)$ – multiplication of polynomials p_1 and p_2 modulo an irreducible polynomial $R(x)$,

A_r – working variable, $r = 0,1,\ldots,15$,

c – masking constant,

α_0, α_2, α_3, α_5 – the polynomials,

S_j – substitution function; it consists of four S-boxes S_0, S_1, S_2 and S_3, each of dimension 16×16.

After processing in two rounds the value H_i^* of chaining variable is splitted into 16 subblocks A_0, A_1, …, A_{15} of equal lengths and each of them is modified by adding (mod 2^n) to it the respective input subblocks of H_i being inputs to the round #1. Next, all subblocks A_0, A_1, …, A_{15} are concatenated giving $H_{i+1} = A_0 \,||\, A_1 \,||\, \ldots \,||\, A_{15}$.

3. Fault models

In the considerations there is used a fault model wherein either transient or permanent faults are induced randomly into the device. We consider single and multiple faults. Faults are modeled as a 16-bit error vector $E=\{e_{15},\ldots,e_i,\ldots,e_1,e_0\}$, where $e_i \in \{0,1\}$ and $e_i = 1$ indicates that bit i is faulty. Their number in this vector is equal to the number of inserted faults. Fault simulations were performed for two kinds of fault models. In one model a fault flips the bit, and in the other model (stuck-at-0/1) the bit takes a constant value 0 or 1.

Let $X=\{x_{15},\ldots,x_1,x_0\}$ be an error-free vector of bits.

Vector $Xe=\{xe_{15},\ldots,xe_1,xe_0\}$ is an erroneous vector [6] and [7]:

- $xe_i = x_i \oplus e_i$ — if the fault flips the bit,
- $xe_i = x_i + e_i$ — for stuck-at-1 fault,
- $xe_i = x_i \times \bar{e}_i$ — for stuck-at-0 fault,

where: ⊕ - xor, + - or, × - and.

4. Error propagation

Error propagation analysis is carried out to understand the effect of an error injected into the hash computation circuit. The error was injected in:

- the inputs of a round function,
- the inputs of step function,

Experiments were conducted by injecting a single bit flip error in one of the inputs of a round function or a step function and obtaining the number of erroneous bits at the output of the same block. The errors were introduced at different bits randomly in every step and the number of bits that were in error was computed. As an example we show the propagation of the error injected in the first step of round #1 in subblock A_{13}. Instead of the value *F9DD* there is the faulty value *F95D*. Output values $A_0, A_1, \ldots, A_{15}$ after processing every of 16 steps of the round function are given in Table 1. The faulty values are depicted by bold, italic font.

One faulty bit in the step #0 of the round function causes about 43% faulty bits in the last, #15 step. A faulty bit in the input of round #1 of round function causes 49% faulty bits in the output of round #2.

This analysis helps us choose suitable error detection schemes.

Tab. 1. Error propagation in round function of HaF-256
Tab. 1. Propagacja błędów podczas realizacji funkcji rundowej w HaF-256

	Faulty free input value A0, A1, …, A15; A13=F9DD
	076A3663D2541F389E94CA4E1A759C92ED3282E55F31FC1514A6F9DDBA029ABE
	Faulty input value A0, A1, …, A15; A13=F95D
	076A3663D2541F389E94CA4E1A759C92ED3282E55F31FC1514A6F95DBA029ABE
Step	**Faulty outputs**
0	3663D2541F389E94CA4E1A759C92ED3282E598AFFC1514A6***F95D***BA029ABEDFA8
1	D2541F389E94CA4E1A759C92ED3282E598AF0AFE14A6***F95D***BA029ABEDFA8C321
2	1F389E94CA4E1A759C92ED3282E598AF0AFE530A***F95D***BA029ABEDFA8C321***0741***
3	9E94CA4E1A759C92ED3282E598AF0AFE530A***AEFC***BA029ABEDFA8C321***0741***A47E
4	CA4E1A759C92ED3282E598AF0AFE530A***AEFC***015D9ABEDFA8C321***0741***A47E***1059***
5	1A759C92ED3282E598AF0AFE530A***AEFC***015D5F4DDFA8C321***0741***A47E***105943F0***
6	9C92ED3282E598AF0AFE530A***AEFC***015D5F4DD46FC321***0741***A47E***105943F0D7D2***
7	ED3282E598AF0AFE530A***AEFC***015D5F4DD46F90E1***0741***A47E***105943F0D7D2C084***
8	82E598AF0AFE530A***AEFC***015D5F4DD46F90E1***A083***A47E***105943F0D7D2C084A3FF***
9	98AF0AFE530A***AEFC***015D5F4DD46F90E1***A083***3F52***105943F0D7D2C084A3FF9977***
10	0AFE530A***AEFC***015D5F4DD46F90E1***A083***3F52***2C8843F0D7D2C084A3FF9977CA78***
11	530A***AEFC***015D5F4DD46F90E1***A083***3F52***2C88F821D7D2C084A3FF9977CA785AAE***
12	***AEFC***015D5F4DD46F90E1***A083***3F52***2C88F821E96BC084A3FF9977CA785AAE875B***
13	015D5F4DD46F90E1***A083***3F52***2C88F821E96B4260A3FF9977CA785AAE875B8AF7***
14	5F4DD46F90E1***A083***3F52***2C88F821E96B4260FFD19977CA785AAE875B8AF7D530***
15	D46F90E1***A083***3F52***2C88F821E96B4260FFD1BBCCCA785AAE875B8AF7D5307F1C***

5. Error detection in basic operations

To protect basic operations such as multiplication mod (2^n+1) ($\odot$), addition modulo 2 ($\oplus$) and addition modulo 2^n ($\boxplus$) in step function of HaF-256 (Fig. 3) we used DWC (Duplication With Comparison) (Fig. 4).

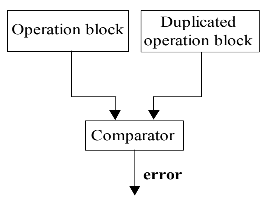

Fig. 4. DWC scheme
Rys. 4. Schemat DWC

Step function F_j with DWC scheme elements for one of addition modulo 2^n ($\boxplus$) operation is presented in Fig. 5. In this figure gray boxes are an extra elements to detect errors in $A_1 \boxplus A_6$ operation. The outputs of the addition block and the duplicated addition block are compared (box *errcheck*) and if they are different, an error signal is generated.

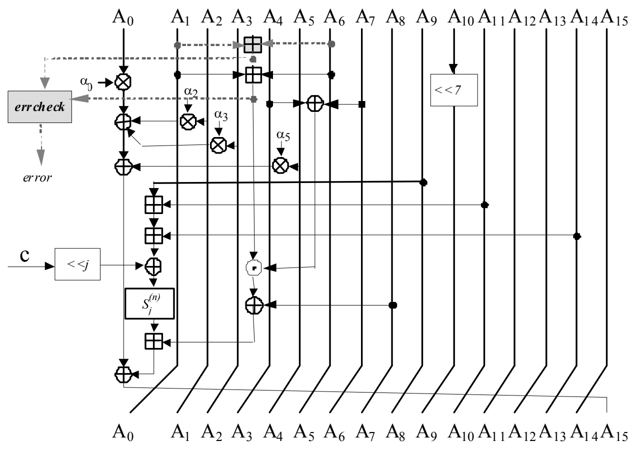
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15
errcheck
error
α0
α2
α3
α5
<<7
C
<<j
S(n) j
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15

Fig. 5. Step function F_j with DWC for one of addition modulo 2^n ($\boxplus$) operation
Rys. 5. Funkcja F_j z DWC dla jednej z operacji dodawania modulo 2^n ($\boxplus$)

Experiments were conducted by injecting errors in the input of an operation block and observe if the error is detected. Capability of single and multiple, transient and permanent fault detection using this fault detection scheme is presented in Section 6.

6. Simulation results

In order to measure the detection capability of the proposed in Section 5 error detection schema, there were used VHDL (*Very High Speed Integrated Circuits Hardware Description Language*) hardware description language and the VHDL simulator, Active-HDL by Aldec. In this section we provide simulation results related to the fault coverage of the proposed approach. We present simulation results on the vulnerability of this schema for fault models from Section 3. The faults were injected into inputs of an operation blocks and observe if the error is detected. We consider random faults, in the sense that the faulty value is assumed to be random and uniformly distributed. The VHDL model of the HaF-256 function was modified by injected faults. The output signals were compared with correct signals. In this way, the obtained fault coverage gives a measure of the error detection capability.

In this experiment we focused on transient and permanent, single and multiple stuck-at faults and bit flips faults. Errors were injected in the input of an operation block and observe if the error is detected. The obtained faults coverage for addition modulo 2^n operation ($\boxplus$) is shown in Fig. 6 for permanent faults and in Fig. 7 for transient faults. Single permanent stuck-at-0/1 faults are detected by proposed CED in 57.9%, transient faults in 57.2%. All single bit flip faults, permanent and transient, are detected.

Figs. 6 and 7 show also the dependence of the error detection probability on the number of injected faults. Not only single, but also multiple bit flip faults are always detected.

Percentage of permanent and transient stuck-at-0/1 error detection for basic operations of the step function of HaF-256 are presented in Tables 2 and 3, respectively.

DWC scheme detects all bit flip errors.

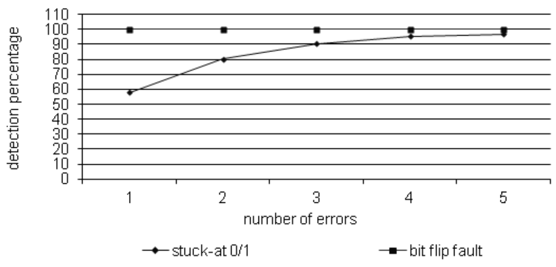

Fig. 6. Probability of permanent error detection using DWC for addition modulo 2^n operation (⊞)
Rys. 6. Prawdopodobieństwo wykrycia błędów stałych w operacji dodawania modulo 2^n (⊞) za pomocą schematu DWC

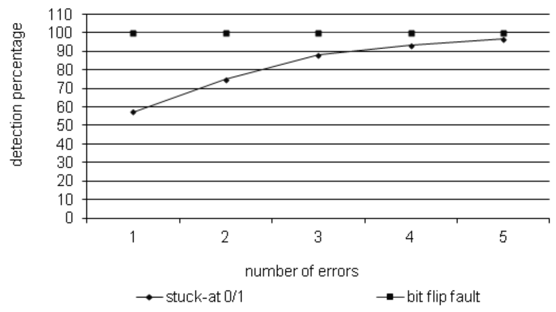

Fig. 7. Probability of transient error detection using DWC for addition modulo 2n operation (⊞)
Rys. 7. Prawdopodobieństwo wykrycia błędów przejściowych w operacji dodawania modulo 2^n (⊞) za pomocą schematu DWC

Tab. 2. Probability of permanent error detection for operations of step function of HaF-256
Tab. 2. Prawdopodobieństwo wykrycia błędów stałych w podstawowych operacjach funkcji HaF-256

	Stuck-at-0/1				
Number of errors	1	2	3	4	5
$a \odot b$	57.2	74.9	88.1	93.1	96.6
$v \oplus w$	61.1	82.1	85.4	93.3	96.5
$v \boxplus w$	57.9	80.3	90.3	95.3	96.8
$p_1(x) \otimes p_2(x)$	60.9	78.2	85.8	93.2	96.1

Tab. 3. Probability of transient error detection for operations of step function of HaF-256
Tab. 3. Prawdopodobieństwo wykrycia błędów przemijających w podstawowych operacjach funkcji HaF-256

	Stuck-at-0/1				
Number of errors	1	2	3	4	5
$a \odot b$	56.7	73.5	86.9	91.1	95.5
$v \oplus w$	60.2	80.9	83.9	92.1	95.6
$v \boxplus w$	57.1	76.4	88.3	93.5	95.9
$p_1(x) \otimes p_2(x)$	59.8	76.9	84.7	91.9	95.3

7. Concluding remarks

Fault attacks are becoming a serious threat to hardware implementations of cryptographic systems. Proper countermeasures must be adopted to foil them.

In this paper the propagation of errors in HaF-256 is studied. We take into consideration single, transient as well as permanent faults injected at different stages of hash value computation. It is found that even a single injected error resulted in half the bits of hash value being in error and the errors are spread across the computed hash value. To protect basic operations such as multiplication mod (2^n +1), addition modulo 2, addition modulo 2^n we used DWC (Duplication With Comparison) scheme. In this way we can detect both transient and permanent faults that affect the output of the operations. We can also protect against single and multiple errors. This method can provide high coverage for multiple-bit errors, which are the most common fault attacks. The coverage also depends heavily on the fault model. Simulation experiments conducted on a large number of test cases show that our scheme has 100% fault coverage in the case of bit flip errors. This solution can be useful for concurrent checking cryptographic chips.

The research project is scheduled for years 2010–2013 and supported partly by scientific grant from the Polish Ministry of Education and Science.

8. References

[1] Bertoni G., Breveglieri L., Koren I., Paolo Maistri, Piuri V.: Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard. IEEE Transactions on Computers, vol. 52, No. 4, pp. 492--505 (2003).
[2] Bilski T., Bucholc K., Grocholewska-Czuryło A., Stokłosa J.: Hash Function Concept. Report No. 596, Poznań University of Technology, Poznań (2011).
[3] Boneh D., DeMillo R., Lipton R.: On the Importance of Eliminating Errors in Cryptographic Computations. In: Journal of Cryptology, vol. 14, pp. 101--119 (2001).
[4] Boneh D., DeMillo R., Lipton R.: On the importance of checking cryptographic protocols for faults. Proceedings of Eurocrypt, pp. 37-51. Springer-Verlag LNCS 1233 (1997).
[5] Breveglieri L., Koren I., Maistri P.: An Operation-Centered Approach to Fault Detection in Symmetric Cryptography Ciphers. IEEE Transactions on Computers, vol. 56, No. 5, 635--649 (2007).
[6] Idzikowska E., Bucholc K.: Concurrent Error Detection in S-boxes. International Journal of Computer Science & Ap-plications, vol. 4, No. 1, pp. 27--32 (2007).
[7] Idzikowska E., Bucholc K.: Error detection schemes for CED in block ciphers. Proceedings of the 5th IEEE/IFIP In-ternational Conference on Embedded and Ubiquitous Com-puting, pp. 22--27. Shanghai (2008).
[8] Idzikowska E.: CED for S-boxes of symmetric block ciphers. Pomiary Automatyka Kontrola, vol.56, No. 10, pp. 1179–1183 (2010).
[9] Idzikowska E.: CED for involutional functions of PP-1 cipher. Proceedings of the 5th International Conference on Future Information Technology. Busan (2010).
[10] Stokłosa J., Bilski T., Pankowski T.: Bezpieczeństwo danych w systemach informatycznych. PWN, Warszawa-Poznań (2001).

otrzymano / received: 03.09.2011
przyjęto do druku / accepted: 03.10.2011 *artykuł recenzowany*