Applied Mathematical Sciences, Vol. 7, 2013, no. 144, 7161 - 7171
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ams.2013.38482

Replace AES Key Expansion Algorithm By Modified Genetic Algorithm

Sliman Arrag[1], Abdellatif Hamdoun[2], Abderrahim Tragha [3] and Salah eddine Khamlich[4]

[1, 2, 4] Laboratory of Information Processing
Hassan II-Mohammedia University
Casablanca, Morocco

[3] Laboratory of Information Technology and Modeling
Hassan II-Mohammedia University
Casablanca, Morocco

Abstract

This paper presents new algorithms that simplify the creation and expansion process of the encryption key of the AES algorithm, which is considered one of the most important elements in the process of encryption, by creating new key generator architectures that allows us to have 10 different keys. These changes, based on genetic algorithm and the linear feedback shift register algorithm, improve the performances and the efficiency of the encryption algorithm.
The implementations of the hardware and software architectures are studied in Altera Cyclone II devices by using the VHDL language.

Keywords: AES, Genetic algorithm, LFSR, VHDL, FPGA

1 Introduction

Schedule key is very important phase in ciphering algorithms, as a strong schedule key means a strong cipher that would be more resistant to various forms of attacks, such as differential and linear cryptanalysis [1].
Carter, Dawsony and Nielseny [1] classified AES candidates according to key schedules as the authors thought that key expansion is very important as strong schedule key means stronger algorithm against both linear and differential

cryptanalysis. They recommended that the schedule key of AES candidates should be upgraded.

Genetic Algorithms (GAs) have played a strong role in data security systems, Yaseen et.al. [2] used genetic algorithm for the cryptanalysis, Spillman and et al. used GA to cryptanalysis a simple substitution [3], Mathews used GA in transposition ciphers [4] and Spillman used GA in knapsack based systems [5], Bagnall used GA to crack difficult systems such as block cipher (Data Encryption Standard DES) [6], Grundlingh et. Al. used GA to attack mono-alphabetic substitution but their approach not seemed effective against transposition [7]. Bagnall et al. used Genetic Algorithm as cryptanalysis of a three rotor machine using genetic algorithm and their results showed that an unknown three rotor machine can be cryptanalysed with about 4000 letters of cipher text [8], Dimovski et al. [9] presented an automated attack on the polyalphabetic substitution cipher whereas Rashed [10] used Limited Genetic Algorithm to generate a pool of cipher keys and schedule keys that will be used in ciphering and deciphering AES processes. However it was suggested having a pool of AES keys and a schedule key would be taken from this pool then the ciphered block and index of the start location [11]. It is useful to avoid the normal key scheduling process, and specify the cipher keys (which should be random and independent) directly. In this research the genetic algorithm process will be modified to assist in the process of generating ciphering and scheduling keys.

2 Description of key scheduling

Is a critical process in AES that generates (Nr+1) round keys based on an external single key. The Key expansion process of AES algorithm uses a Cipher Key K to generate a key schedule.
Simply follow the steps below to have several key generated, as shown in (Fig. 1).

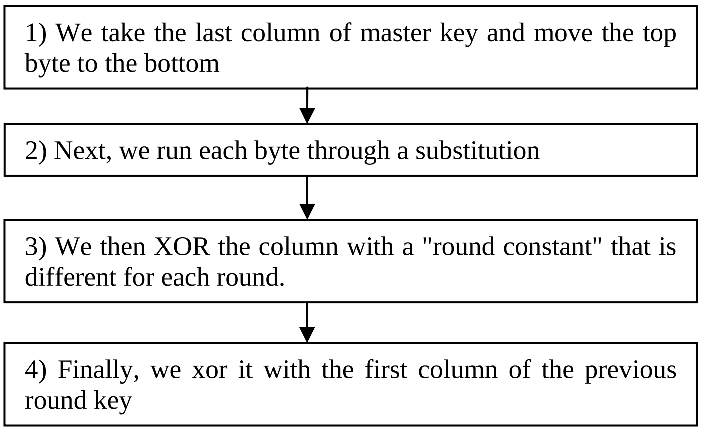
1) We take the last column of master key and move the top byte to the bottom
2) Next, we run each byte through a substitution
3) We then XOR the column with a "round constant" that is different for each round.
4) Finally, we xor it with the first column of the previous round key

Figure 1: round of Key Schedule

In this research the modified genetic algorithm process will be modified to assist in the key ciphering and scheduling process.

3 Basic Idea of Genetic algorithms [10]

Genetic algorithms consist of three phases as following:
1- Reproduction Operation: The old string is carried through into a new population depending on the performance index values. The fitness values are calculated for each candidate string using a fitness function, which depends on a goal for optimization problems. According to the fitness values, string with larger fitness values give rise to a larger number of copies in the next generation.
2- Crossover operation: The strings are randomly mated using the crossover operation. Each pair of candidate strings will undergo crossover with the probability cross. This operation provides randomized information exchange among the strings.
3- Mutation operation: Mutation is simply an occasional random alteration of the value of a string position. In a binary code, this involves changing a 1 to 0 and vice versa. The sequence of successive stages of genetic algorithms is shown in figure [6].

4 Proposed Algorithm (MGA)

To date there has been no reported research using genetic algorithm in any form or shape within a key expansion. The work introduced in this paper will show the use of the first modified genetic algorithm in key expansion algorithm. The modified genetic algorithm will only use some of the conventional genetic algorithm process; this will include random initialization of the first population of cipher keys, and then apply the process of random cross over and mutation to produce further sets of cipher keys. In this research all cipher keys will be considered acceptable and hence, there will be no need to search for a best cipher key, since all keys will be used to cipher data in this system. The proposed algorithm in this case will only be used to generate cipher keys and it will not involve any search or optimization techniques.

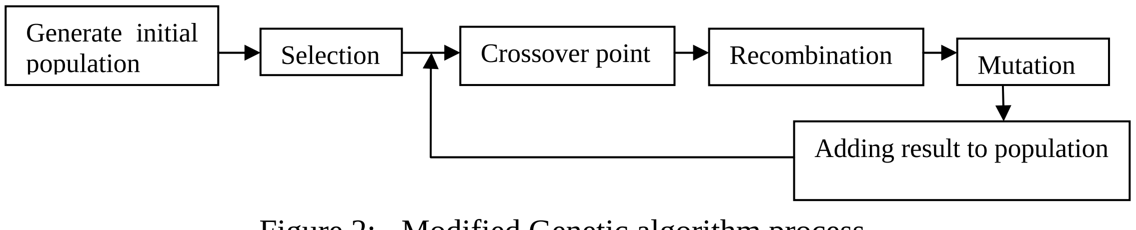

Figure 2: Modified Genetic algorithm process

In the modified genetic algorithm there is no evaluation of the population as the all elements are considered fit. The algorithm works as shown in (Fig. 2).

4.1 Initialization of the first generation

The first generation of sixteen random must be generated, which are considered as the cipher keys (individuals or genotype; as shown in Table 1). Each individual is one octet. Suppose that we work in AES-128, so the cipher key with 16 octets must be generated randomly.

Cipher Key (127 to 0)
$=CK_{15}CK_{14}CK_{13}CK_{12}CK_{11}CK_{10}CK_{9}CK_{8}CK_{7}CK_{6}CK_{5}CK_{4}CK_{3}CK_{2}CK_{1}CK_{0}$

Table 1: Cipher Key with 16 bytes

CK_{15}	CK_{11}	CK_{7}	CK_{3}
CK_{14}	CK_{10}	CK_{6}	CK_{2}
CK_{13}	CK_{9}	CK_{5}	CK1
CK_{12}	CK_{8}	CK_{4}	CK_{0}

4.2 Selection Operator

All individuals in the population are considered best and be chosen as well. All elements (known as elite name) must survive to the next generation and copy it into the new generation. New elements with the same size (16 elements) must be generated. For a child, two individuals from the current generation (parents) are crossed. A parent is any element of the population. So that the population would be: $CK_{15}CK_{14}CK_{13}CK_{12}CK_{11}CK_{10}CK_{9}CK_{8}CK_{7}CK_{6}CK_{5}CK_{4}CK_{3}CK_{2}CK_{1}CK_{0}$

4.3 Crossover operator

During this phase, 16 children are generated to fill the new generation. For each child, the two adjacent parents could be selected as C_i and C_i +1. Then both parents are crossed together.

4.4 Point Crossover

The crossover is a process that aims to produce children who combine the best characteristics of their parents.

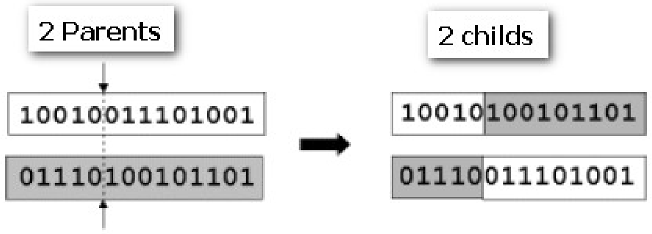

Figure 3: Crossover one point

4.5 Mutations Operator

The operation of mutation is an alteration in the genetic composition of some chromosomes to ensure genetic diversity within the population, as shown in (Fig. 4).

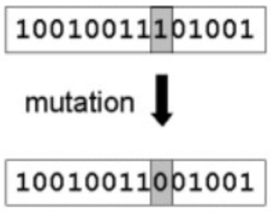

Figure 4: mutation operator

Note: The same procedure will be repeated in all parts of the population.

4.6 Illustrative Example

At first we generated random numbers as cipher key, as shown in Table 2.

Table 2: Cipher key

61	67	6D	6D
72	73	61	69
72	6C	6E	74
61	69	5F	69

We used in Cipher Key: arragsliman_miti (ascii code)

= 6172726167736C696D616E5F6D697469 (Hex code).

To choose the crossover point to a point and the point of mutation we suggested that this be based on the original key used for encryption and decryption, the crossover point and the mutation found in the following manner:

1- Either considering the numerical value of the first three bits of the original 128-bit key is used as the crossover point mutation, as these three bits will be between 0 and 7, as shown in the following table:

Table 3: Point of crossover and mutation

The first three bits of the original key ($C_2C_1C_0$)	Point of crossover and mutation
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Note: in this case the first three bits represent the encryption key crossing point and transfer for all parents (16 individuals or 16 bytes of the original key of AES 128-bit algorithm) adjacent selected (C_i and ci +1) to produce the new population of children.
- First we divided the 128-bit key of 8 for 16 bytes, each byte as a parent is to us as we have already shown.
- Then the arithmetic summation of 8 bits each parent (and individual byte of the 128-bit key used), as follows:

$$p1=\sum_{i=0}^{7}C_i \quad , \quad p2=\sum_{i=8}^{15}C_i \text{ ,...} \text{ and } \quad p16=\sum_{i=120}^{127}C_i$$

Note: P1, P2, P3 ... and P16 represent the crossover and mutation of each byte of 16 bytes of the original keyword.
- The result gives us the value of crossover and the mutation of each single parent.
- To the crossing of parent 1 and parent 2 to compare the first result with the result of P1 to P2, if the two results are equal in this case will be passing in a single point (), if not the crossing will be in two different points between the parent and adjacent selected (C_i and C_i +1), below an illustrative example:
In Table (6), which is already covered in the previous example, it takes two parents (both bytes)
parent1: 61 (in hex) = 01100001 (binary)
parent2: 72 (in hex) = 01110010 (binary)
Then we calculate p1 (the arithmetic summation of 8 bits parent1)
p1 = 0 +1 +1 +0 +0 +0 +0 +1 = 3.
Then we calculate p2 (the arithmetic summation of 8 bits parent2)
p2 = 0 +1 +1 +1 +0 +0 +1 +0 = 4.
In our example we find that p1 and p2 different so we have a cross to two points (point 3 and point 4), and point of mutation to the parent is 1 point 3 and the parent 2 and point 4.
So the two children will reproduce as follows (see table below):

Table 4: Describe the products of the new bytes

Parent 1	Parent 2	Parent1 after the crossover	Parent2 after the crossover	Child1 after the mutation	Child2 after the mutation
01100001	01110010	01110001	01100010	01010001	01110010

And will follow the same steps for every 14 people who remain.

2 - The second and final proposal is to extract the points of the cross and mutation randomly without using the original key, our new method is based on the use of a shift register with linear feedback 64 bits such that the first three bits ($b_2b_1b_0$) output shift register 64-bit linear feedback will be crossing points and transfer for all 16 bytes of the key used for encryption and decryption as well (the 16 individuals or 16 bytes of the original AES key of 128 bits) adjacent selected (C_i

and C_i +1) to produce the new population of children.

4.6 Description of LFSR

The most widely used technique for generating a pseudo-random sequence is that the shift registers LFSR linear feedback (Linear Feedback Shift Register) (Nguyen, 2005). This technique generates a pseudo random bit manner. The bit sequence is random, but is repeated in time. This depends on the length of the LFSR is to say the number of flip-flops used. Indeed, an LFSR is a set of flip in cascade (register) and returns a set of outputs of these latches on doors or exclusive (XOR or XNOR) to the first flip-flop. If we put N the number of scales used, the number of states or possible sequences is 2N-1. Thus, at each clock pulse, the sequence is shifted one bit to the right. Moreover, the clock signal is solely responsible for the generation of signal bits. And a new bit sequence is obtained in each clock pulse.
An LFSR is characterized by:
• The number of flip N: represents the degree of the shift register, which sets the number of possible sequences 2^N-1 which characterizes the period of the generation of sequences before repetition.
• The number of returns from the outputs of flip-flops and their positions in the registry.
• The initial condition is the sequence of bits from which the shift register begins. The sequence in which all bits are 1 is prohibited in the case of the use of XNOR, while the sequence in which all bits are 0 is prohibited in the case of using XOR. In effect, this causes the lock register in its original state.
In our case we used Galois-type architecture. We chose to implement a 64-bit LFSR to generate possible sequences 2^{64}-1 equivalent 18446744073709551615 possible states. This structure is shown in Figure 5

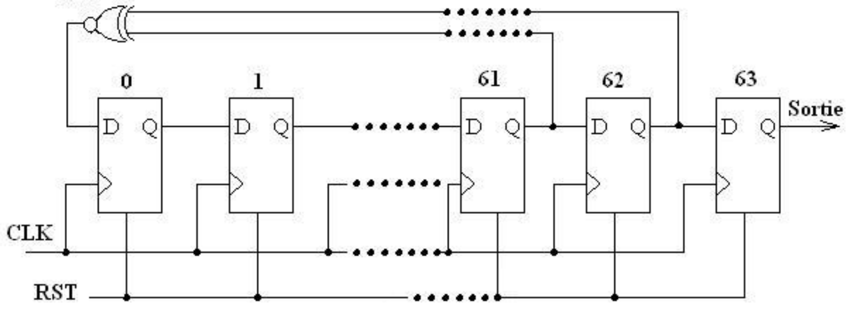

Figure 5: Linear Feedback Shift Register a 64 bit

The control signals are Clock and Reset. The signal output (Sortie) is set in the bits generated in a random manner. The system generates a bit at each clock.

Thus, the transmission bit rate of our transmitter or the bit rate will initially depend on the clock frequency used.

5 Proposed hardware architecture for MGA

In this paper, we have design and develop a hardware architecture of the MGA (Modified Genetic Algorithm) process, that concluding: Selection a population (Cipher key) module, crossover module, and mutation module, to obtain Schedule key algorithm based a genetic algorithm, to attain to software and hardware implementation of the AES algorithm more optimized and over secured.
The VHDL simulations were carried out on the EP2C70F896C6 device from Altera supplied Cyclone II family and Quartus v 9.1 was used to view the output waveforms. The various modules of the genetic algorithm system are described here. The schedule key and AES-128 based a modified genetic algorithm are implemented and designed in FPGA by using VHDL language as shown in (fig 6)

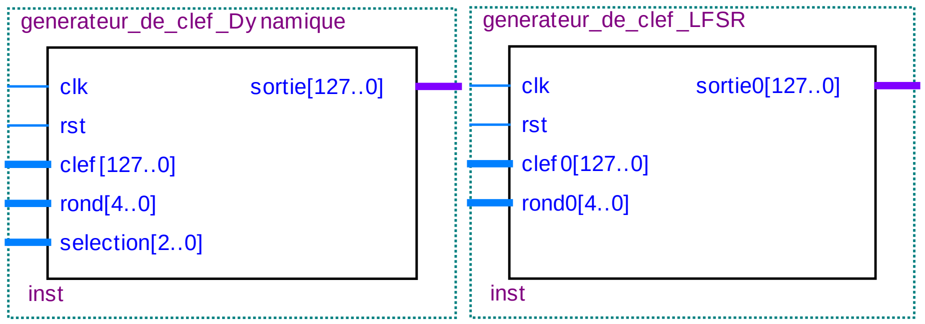

Figure 6: block of AES key expansion dynamic and LFSR

Pin name	I/O	Function description
CLK	I	System frequency
Rst	I	System reset
Donner	I	Plaintext bits
Clef	I	Key for Encryption or decryption
rond	I	Execution of 10 round
Sortie	O	Cypher text

Table 5: I/O functional descriptions of proposed and modified AES-128

Implementation	FPGA Device					response time
	Total pins	Total logic elements	Peak virtual memory Megabyte	Total register	Total memory bits	
Generateur_de_clef	263	992	351	128	0	15. 54
Generateur_de_clef dynamique	266	2136	312	128	0	15.5
Generateur_de_clef LFSR	263	2131	312	63	0	141.19

Table 6: Comparative table between different implementation constitute AES algorithm

From Comparative Table 6, we can see that the architecture of the implementation of the first architecture generateur_de_clef occupies only (992 logic elements) with a response time (15.54 ns) the other the second architecture Generateur_de_clef _dynamique requires only (2136 logic elements) with a response time equal (15.5 ns) and Generateur_de_clef_LFSR (and register with linear feedback shift 64 bits) needs (2131 logic elements) with a response time equal (141.19 nm).

6 Simulation and interpretation

After compiling the overall circuit is created a file for the simulation of its operation. The simulation allows you to try the file described in VHDL, to control the outcome in time. Before describing the simulation we note that the clock used is 50 MHz The following figures describe the temporal simulation of different architectures generator key (dynamic shift register with linear feedback 64 bits) that have already discussed in this chapter to verify the effectiveness of our source code in effect while respecting criteria already mentioned too.

Plaintext: hamdoun_&_tragha
Key: arragsliman_miti
Cyphertext:[154][224][139]\][253][147][231]4[144][242][127]k[253][155][253][2]
Time of temporal simulation: 04 s

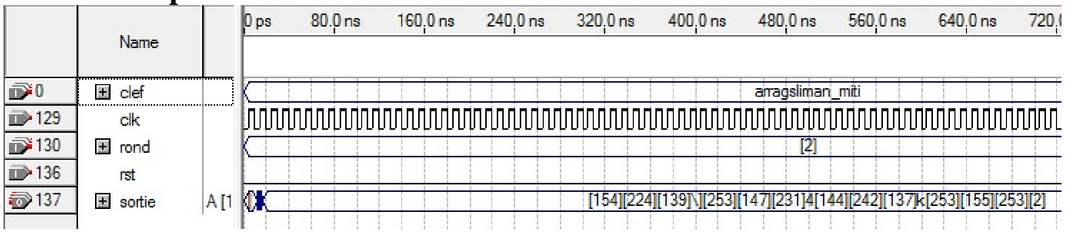

Figure 7: Simulation of cipher key by using simple AES key expansion

Plaintext: hamdoun_&_tragha
Clé : arragsliman_miti
Cypher text: m~~mk[127]'eambSaexe
Time of temporal simulation: 04 s

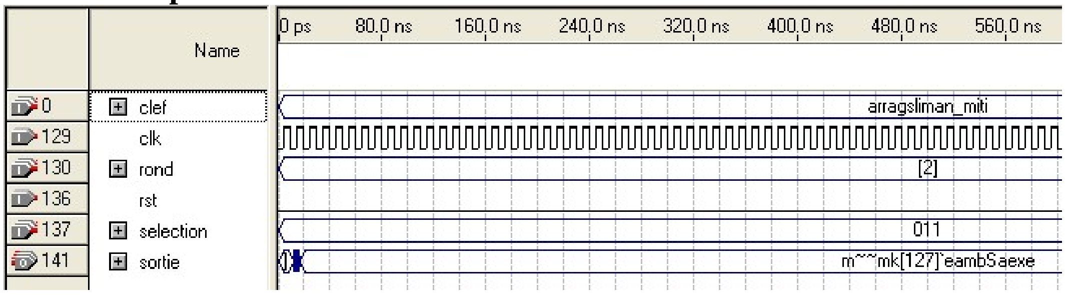

Figure 8: Simulation of cipher key by using MGA (Dynamic)

Plaintext: hamdoun_&_tragha
Key: arragsliman_miti
cyphertext:[161][178][178][161][167][179][172][169][173][161][174][159][173][169][180][169]
Time of temporal simulation: 05 s

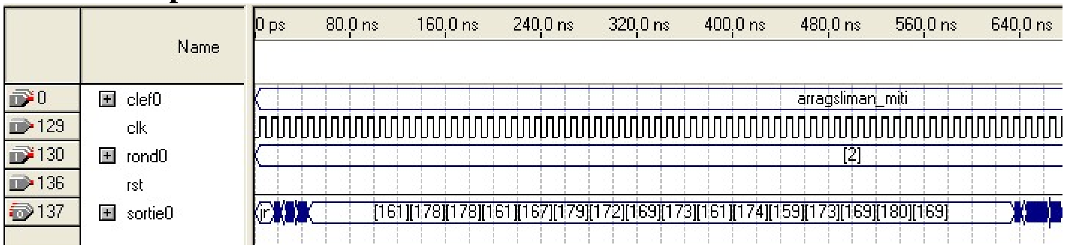

Figure 9: Simulation of cipher key by using MGA (LFSR)

6.1 Advantages of our MGA

Many operations and methods related to schedule key are eliminated as they will not needed.
No need to expand the ciphering key as the proposed method uses ready schedule key.
It is difficult to predict the numbers in the key, since this MGA.
The proposed method can be used in all AES key (128, 196 and 256).

7 Conclusion

This paper showed how modified genetic algorithms can be used to produce a ciphering or schedule key. However the Modified Genetic algorithm can be used to expand the cipher key to schedule key in any ciphering algorithm. The results Obtained using this method has showed a highly secured and efficient algorithm and it decreased the complexity of the original AES algorithm by more than 50%.
As future work the algorithm can be enhanced to have an inverse algorithm.
MGA can be used in generating schedule key for AES algorithm in different size

(128, 196 and 256), that will be used in both ciphering and deciphering phases. As future work, it is believed that MGA can be used to replace Key expansion in private key cryptosystems.

References

[1] Carter G., Dawsony E. and Nielseny L., Key Schedule Classification of the AES Candidates, Proceedings of Second AES Candidate Conference (AES2), Rome, Italy, March 1999.

[2] Yaseen I. and Sahasrabuddhe H., A Genetic Algorithm for the Cryptanalysis of Chor-Rivest Knapsack Public Key Crypro System (PKC), Third International Conference on Computational Intelligence and Multimedia Application, September 23-26, 1999, New Delhi, India.

[3] Spillman R. et al, The Use a Genetic Algorithm in the Cryptanalysis of Simple Substitution Ciphers, Cryptologia, Vol. 17, No. 1, 1993.

[4] Mathews R., The use of Genetic Algorithm in Cryptanalysis, Cryptologia, Vol. 17 No. 4, 1993.

[5] Spillman R., Cryptanalysis of Knapsack using Genetic Algorithms, Cryptologia, Volume 17 No. 1, 1993.

[6] Bagnall A., The Application of Genetic Algorithm Cryptanalysis, Mater Degree Thesis, 1996, School of Information Systems, University of East Anglia.

[7] Grundlingh W. and Vuuren J., Using Genetic Algorithm to Break a Simple Cryptographic Cipher, Available via http://dip.sun.ac.za/~vuuren/papers/genetic.ps

[8] Bagnall A., McKeon G. and Rayward-Smith V., The Cryptanalysis of a Three Rotor Machine Using Genetic Algorithm, available at http://citeseer.nj.nec.com/162166.html

[9] Dimovski A., Gligoroski D., Attack on the Polyalphabetic Substitution Cipher Using a Parallel Genetic Algorithm, Swiss-Macedonian scientific cooperation trought SCOPES project, March 2003, Ohrid, Macedonia.

[10] Rashed A., Intelligent Encryption Decryption System Using Partial Genetic Algorithm and Rijndael Algorithm, Ph.D. Thesis, Arab Academy for Banking and Financial Sciences, Computer Information System Department, 2004.

[11] Ajlouni N. A. El-Sheikh and A.Abdali Rashed, New Approach in Key Generation and Expansion in Rijndael Algorithm, International Arab Journal of Information Technology, vol. 3, no. 1, January 2006, www.IAJIT.org.

Received: August 30, 2013