Modified AES with Key dependent S-box, rotor dependent subkeys and frequency analysis resistance

(AES-KSF)

Fatma Ahmed, Dalia Elkamchouchi

***Abstract*— With the widespread use of computers and Internet as well as electronic commerce, computer security becomes more and more important. In this paper, we introduce AES with key dependent S-box, rotor dependent subkeys and frequency analysis resistance (AES-KSF). The AES-KSF algorithm is compared with AES and gives excellent results from the viewpoint of the security characteristics and the statistics of the ciphertext. Also, we apply the randomness test to the AES-KSF algorithm and the results shown that the new design passes all tests which proven its security.**

***Keywords*— Advanced Encryption Algorithm (AES), key dependent S-box, rotor, frequency analysis, inverse power function.**

I. Introduction

AES is short for Advanced Encryption Standard and is a United States encryption standard defined in Federal Information Processing Standard (FIPS) 192, published in November 2001 [1]. It was approved as a federal standard in May 2002. AES is the most recent of the four current algorithms ratified for federal us in the United States. Rijndael submitted by Joan Daemen and Vincent Rijmen, is a symmetric key, iterated block cipher based on the arithmetic in the Galois Field $GF(2^8)$. AES Input and Output consists of 128 bit sequences. The cipher key is 128, 192, or 256 bits. Byte is the unit of processing. Input blocks are 16 bytes each. AES operations are Conducted on a two dimensional array of bytes called the state. The state consists of four rows of bytes each containing N_b bytes where N_b is the block length divided by 32. Rijndael round function acts on a state N_r times, where N_r is equal to the number of rounds that can be 10, 12 or 14 rounds, depending on N_k , where N_k is equal to the number of 32-bit words comprising the Cipher Key [2]. Rijndael round is consists of 4 transformations:

SubBytes: Transformation in the Cipher that processes the State using a nonlinear byte substitution table (S-box) that operates on each of the State bytes independently which provides nonlinearity and confusion.

ShiftRows: Transformation in the Cipher that processes the State by cyclically shifting the last three rows of the State by different offsets to provide inter-column diffusion.

MixColumns: Transformation in the Cipher that takes all of the columns of the State and mixes their data (independently of one another) to produce new columns which provides inter-Byte diffusion.

AddRoundKey: Transformation in the Cipher and Inverse Cipher in which a RoundKey is added to the State using an XOR operation which provides confusion.

This paper introduces a new modification on AES algorithm to exhibit a substantial avalanche effect, to ensure that no trapdoor is present in the cipher, to make the key schedule so strong that the knowledge of one round key does not help in finding the cipher key or other round keys, and to resist the frequency analysis on ciphertext.

II. AES-KSF

AES-KSF is block cipher; it can encrypt block of plaintext of length 128 byte into blocks of the same length. The key length can be 128, 192, or 256 bytes. The total number of rounds depends on the key length that can be 10, 12 or 14 respectively. We assume a key length of 128 byte, which is likely to be the one most commonly implemented. The input to the encryption and decryption algorithms is block of length 128 byte. This block is copied into the 8×16 matrix of bytes, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix. Similarly the 128 byte key is depicted as 8×16 matrix of bytes. This key is then expanded into an array of key schedule words; each word is four bytes and the total key schedule is 32×11 words. The encryption and decryption process of AES-KSF resembles that of AES. Fig.1 shows the overall structure of AES-KSF.

A. *Substitute Bytes Transformation*

AES-KSF is a technique seeks to make AES depends on key. In this paper, key dependent S-box is introduced. The S-box construction attempt in the following way:

1- Initialize the S-box: The first column contains 0x00, 0x01,……, 0x0F. The second column contains 0x10, 0x11,…0x1F etc, and so on. Thus, the value of the Byte at column y and row x is [xy].

2- Merge the S-box with the user key: at the first XORing, the first 128 elements in the S-box with the user key. The output of this process is replaced by the first 128

Fatma Ahmed[1]

[1]Dept. of Electrical engineering, Alexandria Higher Institute of Engineering and Technology (AIET)
Alexandria, Egypt
moonyally@yahoo.com

Dalia Elkamchouchi[2]

[2]Dept of Electrical engineering, Faculty of Engineering, Alexandria University
Alexandria, Egypt
Daliakamsh@yahoo.com

elements in the S-box then the user key is to be shifted by one bit and XORed with next 128 elements in the S-box.

3- Map each byte in the S-box to its multiplicative inverse in the finite field $GF(2^8)$; the value {00} is mapped to itself.

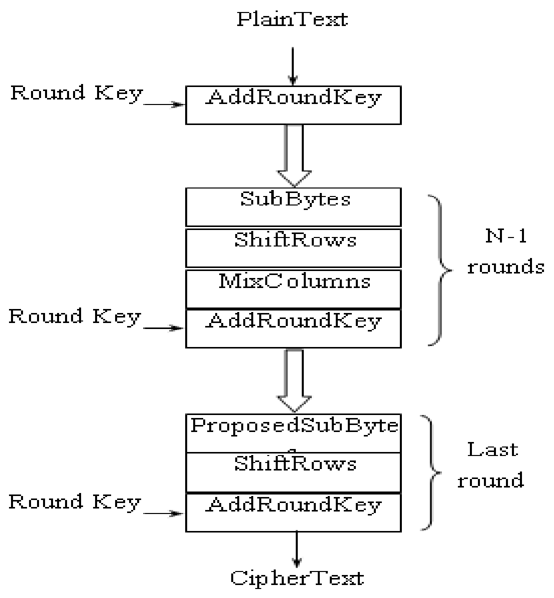
4- Consider that each byte in the S-box consists of 8 bits labeled $(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$. Apply the following transformation to each bit of each byte in the S-box:

$$b_i' = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i \quad (1)$$

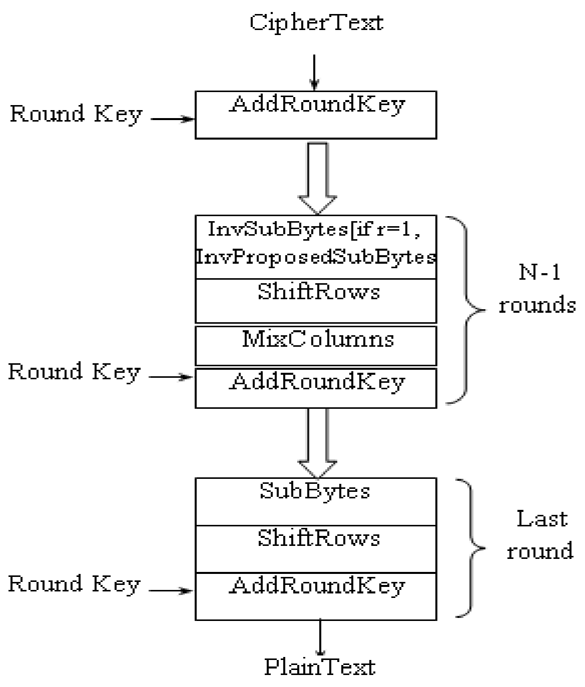
Where c_i is the ith bit of byte c; that is, $(c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0)$.

The new S-box is designed to be resistant to known cryptanalytic attacks since we use the mapping of inverse exponent, one of APN (Almost Perfect Nonlinear) function [3], in each element of the new S-box. Also, the design of new S-box has a low correlation between input bits and output bits, and the property that the output cannot be described as a simple mathematical function of the input. In addition, the constant c was chosen so that the new S-box has no fixed points [S-box (a) = a] and no "opposite fixed points" [S-box (a) = $\bar{a}$], where $\bar{a}$ is the bitwise complement of a. The new S-box is designed to be depend on user key, so if we change the user key or even few bits this will produce new substitution in S-box. The inverse S-box is constructed by applying the inverse of the transformation in Equation1 followed by taking the multiplicative inverse in $GF(2^8)$. The inverse transformation is:

$$b_i' = b_{(i+2)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus d_i \quad (2)$$



(a) Encryption structure



(b) Decryption structure
Figure.1 AES-KSF algorithm

Where d_i is the ith bit of byte c ; that is,

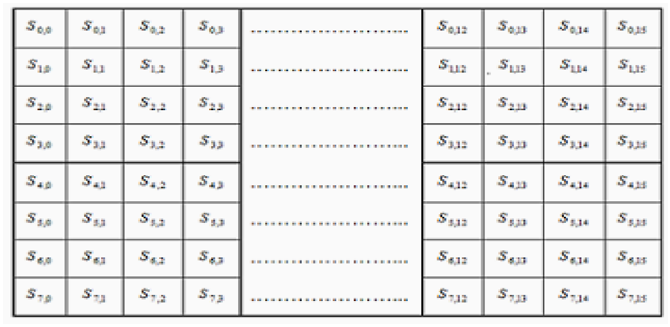
$$d_i' = c_{(i+2)\bmod 8} \oplus c_{(i+5)\bmod 8} \oplus c_{(i+7)\bmod 8} \quad (3)$$

B. *ShiftRows Transformation*

The forward shift row transformation is performed in following way: The first row of State is not altered. For the second row, a 2-byte circular left shift is performed. For the third row, a 4-byte circular left shift is performed. And so on until the eighth row, a 14-byte circular left shift is performed [4]. The inverse shift row transformation performs the circular shifts in the opposite direction for each of the last seven rows, with a two-byte circular right shift for the second row, and so on.

C. *MixColumns Transformation*

The forward mix column transformation, in AES, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. The transformation can be defined by the MDS (Maximum Distance Separable) matrix multiplication on the State [5].In AES-KSF, The data is copied into the 8×16 matrix of bytes. First we divide the state of data into sub states each one is 4×4 matrix of bytes. So we have eight sub states. Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in $GF(2^8)$.



$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$	....	$S_{0,12}$	$S_{0,13}$	$S_{0,14}$	$S_{0,15}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	....	$S_{1,12}$	$S_{1,13}$	$S_{1,14}$	$S_{1,15}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$	....	$S_{2,12}$	$S_{2,13}$	$S_{2,14}$	$S_{2,15}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	....	$S_{3,12}$	$S_{3,13}$	$S_{3,14}$	$S_{3,15}$
$S_{4,0}$	$S_{4,1}$	$S_{4,2}$	$S_{4,3}$	....	$S_{4,12}$	$S_{4,13}$	$S_{4,14}$	$S_{4,15}$
$S_{5,0}$	$S_{5,1}$	$S_{5,2}$	$S_{5,3}$	....	$S_{5,12}$	$S_{5,13}$	$S_{5,14}$	$S_{5,15}$
$S_{6,0}$	$S_{6,1}$	$S_{6,2}$	$S_{6,3}$	....	$S_{6,12}$	$S_{6,13}$	$S_{6,14}$	$S_{6,15}$
$S_{7,0}$	$S_{7,1}$	$S_{7,2}$	$S_{7,3}$	....	$S_{7,12}$	$S_{7,13}$	$S_{7,14}$	$S_{7,15}$

Figure.2 AES data structures

Then we apply the mix columns transformation on each sub sate (ex first sub state).

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,2} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \quad (4)$$

The inverse mix column transformation is defined by the following matrix multiplication:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,2} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \quad (5)$$

In order to make sure that every bit in data block will effect in whole ciphertext we perform XOR operation between rows. The fourth row is XORing with the sixth row and the output is replaced by the old content of the fourth row. Also, the fifth row is XORing with the third row and the output is replaced by the old content of the fifth row.

D. *AddRoundKey Transformation*

The 128 bytes of State are bitwise XORed with the 128 bytes of the round key. The inverse add round key transformation is identical to the forward add round key transformation, because the XOR operation is its own inverse.

E. *Frequency Analysis Resistance*

In order to resist the frequency analysis attack and the well known methods of brute-force attack, the cipher must hide ciphertext statistics [6]. In order to make AES-KSF provides highly hide ciphertext statistics, we rotate it's S-box in the last round by the output of itself. The output from the S-box in our system at the last round is block cipher of length 128 byte, at each byte we divide it into 4 bit left and 4 bit right. The first 4 bit left represents the number of row and the other half represents the number of column. Depending on the number of data block entered the encryption process, we rotate the represented row or column of the S-box. If the number of data block is odd, then we rotate the represented row of S-box one step, else we rotate the represented column one step. In decryption, we don't need to rotate the S-box, we only get the number of row or column which rotated in encryption algorithm from input data into decryption algorithm then subtract the number of rotation of specified row or column with respect to number of data block from the output data of the S-box. So if we change only one bit in plaintext, it will cause different rotation in the S-box, then more than half of the ciphertext will change which provide the avalanche effect.

F. *AES-KSF Key Expansion*

The AES-KSF key expansion algorithm takes as input a 32-word (128-byte) key and produces a linear array of 32×11 words. This is sufficient to provide a 32-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher. The key expansion procedure of AES-KSF is like the expansion procedure of AES except in the step of S-box we use rotor instead [7]. We have one bank of rotor contains six cylinders. All cylinders are relatively prime in length. At first step we find the 128 bit which represented the MD2 hash digits of the user key. These 128 bits divided into two halves each half give us one of the cylinder's numbers in the rotor bank. In the two cylinders, we update its contents using the inverse exponent function. In key expansion process, the input byte, to the rotor, is considered as an address in the first cylinder and the output is the contents of that address. The output of the first cylinder is considered as an address in the second cylinder and the output is the contents of that address. After a byte comes out of the rotor, the first cylinder rotates one position. This will create new substitution. That is if the last byte is applied to the rotor again, the output will be another different byte. Using the rotor improves the avalanche effect because if only one bit change, this leads to change more than half of the output.

III. Security Analysis

A. *Avalanche Effect*

In cryptography, the **avalanche effect** refers to a desirable property of cryptographic algorithms. The avalanche effect is evident when an input is changed slightly (for example, flipping a single bit) the output changes significantly (e.g. half the output bits flip). In the case of quality block ciphers, such a small change in either the key or the plaintext should cause a drastic change in the ciphertext. Constructing a cipher to exhibit a substantial avalanche effect is one of the primary design objectives. The avalanche effect is calculated as:

$$\text{Av Effect}=\frac{\text{No. of flipped in the ciphered text}}{\text{No. of bits in the ciphered text}}\times 100\% \qquad (6)$$

In our case, we take two plaintexts and two blocks of data flipping one bit from everyone in different positions and calculate the avalanche effect. Then we flip the user key in different positions and calculate the avalanche effect [8]. The following results are obtained after calculating the respective Avalanche Effects.

TABLE.1 AV EFFECT FOR 1 BIT CHANGE IN THE PLAINTEXT

Plaintext	Length of plaintext in bits	Change first bit in plaintext		Change last bit in plaintext		Change middle bit in plaintext	
		AES	AES - KSF	AES	AES-KSF	AES	AES-KSF
Case 1	20480	0.03%	49.8%	0.04%	50.3%	0.03%	50.2%
Case 2	158720	0.04%	49.9%	0.04%	50%	0.04%	50.1%
Case 3	1024	6.5%	51.8%	6.5%	50%	6.2%	50.1%
Case 4	1024	6.5%	52.9%	6.5%	50.4%	6.6%	51.2%

TABLE.2 AV EFFECT FOR 1 BIT CHANGE IN THE USER KEY

Plaintext	Length of plaintext in bits	Change first bit in key		Change middle bit in key		Change last bit in key	
		AES	AES-KSF	AES	AES-KSF	AES	AES-KSF
Case 1	20480	45.3%	49.4%	51.5%	51.4%	51.7%	51.7%
Case 2	158720	50.2%	50.1%	50.1%	50.3%	49.9%	50.1%
Case 3	1024	47.7%	53.3%	47.7%	51.8%	48.4%	52.3%
Case 4	1024	48.8%	50.1%	49.9%	51.7%	49.8%	51%

The avalanche effect of the proposed algorithm is producing very high as comparison AES because in AES if only one bit changes, it effects on its data block not all the blocks, while in AES-KSF because we rotate S-box using the output ciphertext so if only one bit changes it produces different rotation in S-box.

B. *Secret Data Groups*

Considering the secret data used in AES, the brute force attack for the key in the case of 128 bit block is $(2^{128} = 3.4 \times 10^{38})$. Considering the secret data used in AES-KSF, the brute force attack for the key in the case of 128 bytes block is $2^{128\times8} = 1.8 \times 10^{308}$. The S-box in AES-KSF will produce different substitution in the last round because it shifted by every output byte, so after encrypt every data block, the S-box shifted by 64 times for row and 64 times for column. To calculate the brute force attack in this case we will find a huge number of trails to break the system.

C. *Language Statistics*

Language redundancy [9] is the greatest problem for any cryptosystem. The cryptanalyst uses the language redundancy to attack cryptosystems ciphertext. If the message is long enough, the cryptanalyst computes the frequency of each of the characters and consider different number of combinations up to the length of the cryptosystem block. The cryptanalyst will then try to estimate the plaintext from this statistical result. A cryptosystem is considered unbreakable against statistical analysis if its ciphertext has flat distribution. To implement the strength of new AES-KSF, Figs 3&4 show the plaintext statistics of the used file. The ciphertext statistics of AES and new AES-KSF are plotted in Figures 5 to 8.

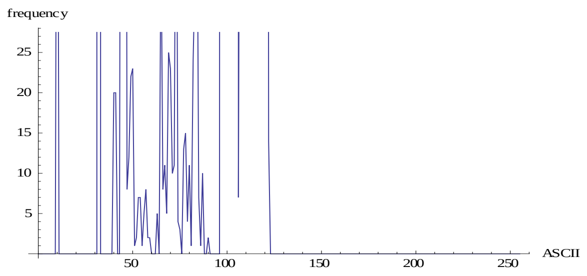


Figure.3 Plaintext statistics of a text file

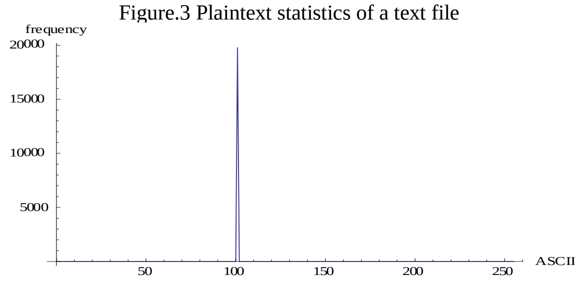


Figure.4 Plaintext statistics of repeated text file

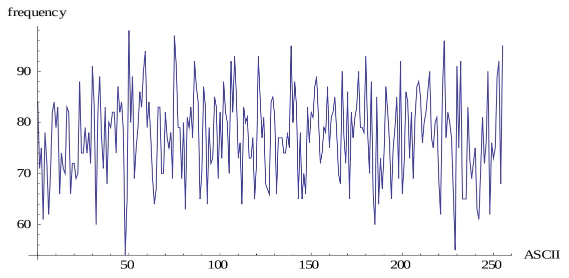


Figure.5 AES-KSF ciphertext statistics

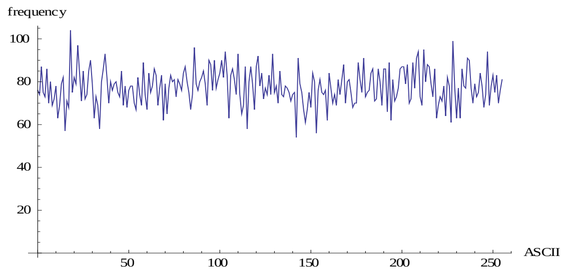


Figure.6 AES ciphertext statistics

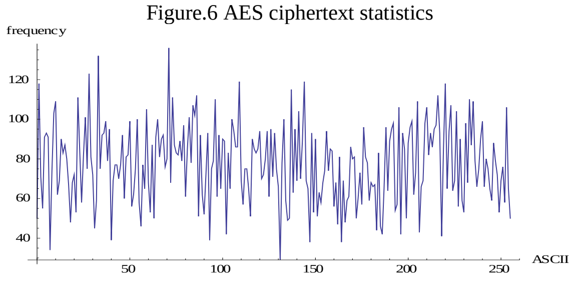


Figure.7 AES-KEF ciphertext statistics of message consisting of 20 Kbytes of character "e"

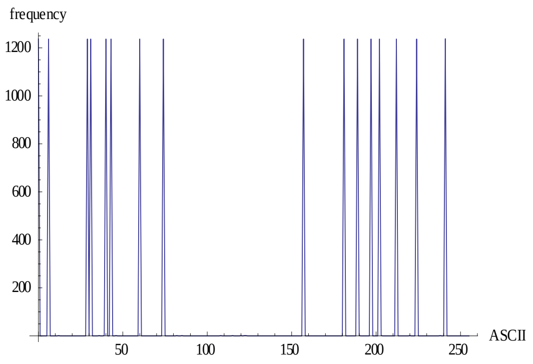


Figure.8 AES ciphertext statistics of a message consisting of 20 Kbytes of character "e"

D. *NIST Statistical Suite*

The National Institute of Standards and Technology (NIST) [10] develops a Test Suite as a statistical package consisting of 16 tests that were developed to test the randomness of (arbitrarily long) binary sequences produced by either hardware or software based Cryptographic random or pseudorandom number generators. These tests focus on a variety of different types of non randomness that could exist in a sequence. Some tests are decomposable into a variety of subtests. The average values of the statistical tests for both algorithms were given in Table 3.

TABLE.3. AES-KSF VS. AES STATISTICAL TESTS

Test name \ Algorithm	AES	AES-KSF
Frequency (Monobit) Test	99%	100%
Frequency Test within a Block	100%	100%
Runs Test	100%	100%
Longest Run of 1's in a Block Test	99%	99%
Binary Matrix Rank Test	98%	100%
Discrete Fourier Transform Test	100%	100%
Non-overlap Template Match Test	100%	100%
Overlap Template Matching Test	98%	99%
Maurer's Universal Statistical Test	100%	100%
Lempel-Ziv Compression Test	99%	100%
Linear Complexity Test	97%	99%
Serial Test	98%	99%
Approximate Entropy Test	99%	100%
Cumulative Sums (Cusum) Test	99%	100%
Random Excursions Test	98%	100%
Random Excursions Variant Test ($\alpha = 0.05$)	91%	98%

IV. Conclusion

In this paper a new improved version of AES has been proposed. AES-KSF doesn't contradict the security and simplicity of the AES algorithm. We tried to keep all the mathematical criteria for AES without change. We have improved the security of AES increase the size of data block to 128 bytes and the size of key to 128, 192 and 256 bytes. Also we make its S-box to be key dependent and rotate depend on data to extend its period. We use rotor cryptosystem in the key expansion procedure so we garanturee that even weak key is used, the subkeys will not effect because the rotor system provides strong cryptosystem. In AES-KSF system if only one bit change in the plaintext or the user key it cause more than half of the ciphertext to be change. Finally, our proposal is rigid to withstand the well-known methods of brute-force.

References

[1] J. Daemen and V. Rijmen, "The Design of Rijndael: AES–The Advanced Encryption Standard." Springer-Verlag, 2002.

[2] William Stallings, Cryptography and Network Security Principles and Practices, Fourth Edition, 2005.

[3] GÖLO˘G LU, F., AND POTT, A.; Almost perfect nonlinear functions: a possible geometric approach. In Proceedings of Contactforum, Coding Theory and Cryptography II (2007), S.Nikova, B. Preneel, L. Storme, and J. A. Thas, Eds.,KVAB, pp. 75–100, (2007).

[4] J. Daemen and V. Rijmen, AES Proposal: Rijndael, AES Algorithm Submission, September 3, 1999, available from the US National Institute of Standards and Technology (NIST)

[5] Junod, P., Vaudenay, S.: Perfect Diffusion Primitives for Block Ciphers: Building Efficient MDS Matrices. In: Selected Areas in Cryptography (2004)

[6] Mohd Zaid Waqiyuddin Mohd Zulkifli "Attacks on Cryptography"April 2008.

[7] W. O. Chan, Cryptanalysis of SIGABA, Master's Thesis, Department of Computer Science, San Jose State University, May 2007.

[8] Amish Kumar, "effective implementation and avalanche effect of AES", International Journal of Security, Privacy and Trust Management (IJSPTM), Vol. 1, No 3/4, August 2012.

[9] Bruce Schneier, "Applied Cryptography, Protocols, Algorithms, and Source Code in C" Wiley Computer Publishing, Second Edition, John Wiley & Sons, Inc.

[10] NIST, "A Statistical Test Suite for Random and Pseudorandom Generators for Cryptographic Applications", NIST Special Publication 800-22, 2003.

Fatma Ahmed Held a Masters' of science in Electrical Engineering from Faculty of Engineering, Alexandria University. She works on Alexandria Higher Institute of Engineering and Technology. She studies for Ph.D. in Electrical Engineering from Faculty of Engineering, Alexandria University.

Dalia Elkamchouchi Held a Masters' of science in Electrical Engineering from Faculty of Engineering, Alexandria University. She works on Alexandria Higher Institute of Engineering and Technology. She Held a Ph.D. in Electrical Engineering from Faculty of Engineering, Alexandria University.