Virtual Implementation of Simplified Advanced Encryption Standard (SAES)

Abdul Atisam Farooq, Waqas Ahmed, Zeeshan Ahmed

***Abstract*— Secure information transmission has always been of immense importance in all sorts of communication systems. In this area, cryptographic algorithms have been evolved and employed over decades. After the release of Advanced Encryption Standard (AES) most of the people have made its animations in 128-bit, which is very difficult to understand by the beginners. This paper is about complete structure of Simplified Advanced Encryption Standard (SAES) and its step by step implementation in graphics form using Flash. Aim of this paper is to represent SAES in such an easy way that it helps beginners to understand each step of key generation, encryption and decryption easily using 16-bit plain text as well as keys. Adding to the complete description of SAES, the authors also discuss few important points, which are needed in addition to understand for the implementation of Advanced Encryption Standard (AES) from SAES.**

In this paper authors also have proposed a general formula to calculate the length of the key for more than one round's in AES and SAES.

***Index Terms*—AES, cryptography, decryption, encryption, SAES, security.**

I. Introduction

SECURE communication is the major requirement of any communication system. A good security algorithm should not be very complex rather it should be easy to implement, fast and unbreakable. AES is considered to be the most secure algorithm so for.

SAES is a simplified form of Advance Encryption Standard (AES). AES accepts 128/192/256-bit plain text and 128-bit key for encryption and decryption and 10/12/14 different rounds, each round involve the use of new key of 128-bit in length [1]. In this paper the main focus is on SAES, authors use 16-bit plain text (which is converted into a 16-bit cipher text), 16-bit key and two rounds for encryption and decryption each, in which each round uses an expanded 16-bit key.

Abdul Atisam Farooq, Lecturer, was with Electrical Engineering Department, HITEC University, Taxila, Pakistan. He is now with the Lingua Nova AG, Kasinostr. 32, 5000 Aarau, Switzerland (phone: +41-79-622-2749; e-mail: atisamnajm@yahoo.com).

Waqas Ahmed, Jr. Lecturer, is with the Electrical Engineering Department, HITEC University, Taxila, Pakistan. (e-mail: imwaqasahmed@live.com).

Zeeshan Ahmed, Jr. Lectured, is with the Electrical Engineering Department, HITEC University, Taxila, Pakistan. (e-mail: zeeshan_ehtram@yahoo.com).

II. Key Expansion

AES is a symmetric algorithm as it uses same key for encryption and decryption. Since there are three rounds in

SAES and each round uses 16-bit key, therefore 16-bit input key is further expended into two more 16-bit keys which makes total key length of (16+2*16)=48-bit.

Equation (1) is proposed to calculate the length of the key for more than one round's in AES and SAES:

$$L = Ki + [(N\text{-}1) * Ki] \quad (1)$$

Where L = Total key length

Ki= Input key in bits

N= No of rounds

Input key used in AES = 128/192/256bit

In this paper input key length used in SAES is 16-bit.

The block diagram of the algorithm, used for key expansion in SAES, is shown in Fig. 1.

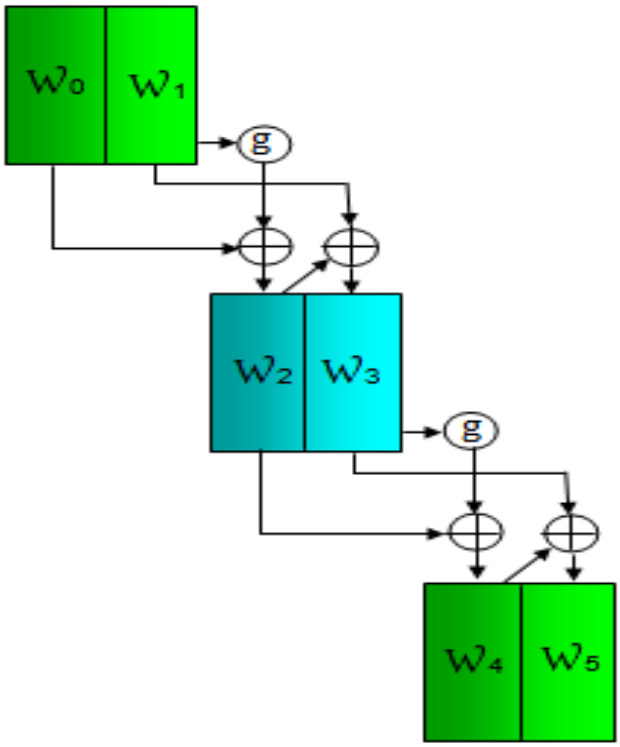


Fig. 1. Key expansion algorithm

W_0 & W_1 each accommodate 8-bits of input key (also known as KEY-01); KEY-02 is stored in W_2 & W_3, KEY-03

is stored in W_4 & W_5. KEY-01, KEY-02 and KEY-03 are used in round-0, round-1 and round-2 during encryption and decryption respectively.

Mathematically

$W_2 = g(W_1) + W_0$

$W_3 = W_2 + W_1$

$W_4 = g(W_3) + W_2$

$W_5 = W_2 + W_1$

Fig. 2 presents internal structure of "Function g" which involves following steps:

8-bit data input into function g is treated as two nibbles named N_0 and N_1.

N_0 and N_1 are swapped.

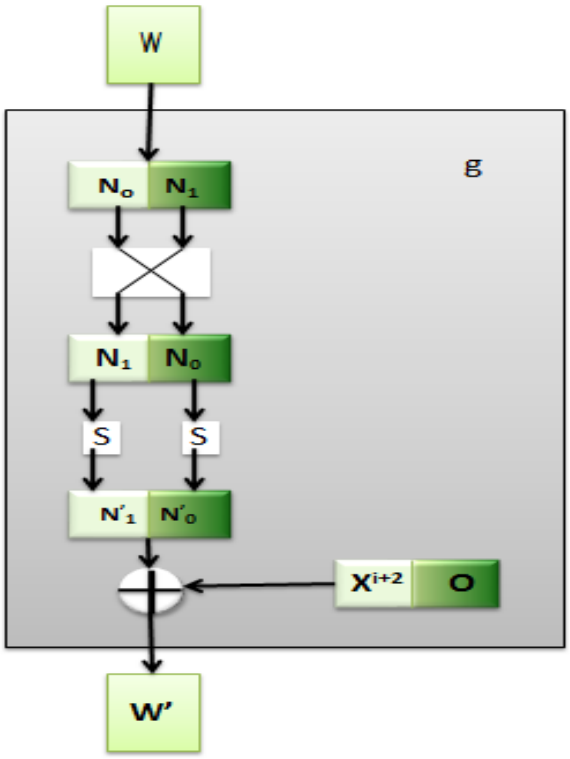


Fig. 2. Function g

Values of N_0 and N_1 are replaced with specific values chosen from S-BOX of Fig. 5. First two bits of each nibble (N_0, N_1) indicate a row and last two bits indicate column of S-BOX. New values are labeled as N_0' and N_1' in function g.

N_0' and N_1' are then combined with x^(i+2) and '0000' via XOR operation to produce the output of function g.

x^(i+2) is a polynomial, in which the value of 'i' depend on round number. Therefore i=1 while generating first byte of KEY-02 i-e. W2, this is specified as round-1, i=2 for generating W4 during round-2 and so on.

III. Encryption

One of important features of AES is that it works on matrices, which make its implementation easy. Since mathematics of AES makes use of 'Galois Field Algebra', therefore this fact makes it more fast and secure. Fig. 3 and onwards show step by step implementation of encryption and decryption using SAES in flash.

The only difference between this SAES and AES is that, AES includes 10/12/14 no of rounds rather than three rounds as shown in Fig. 3 in the case of SAES.

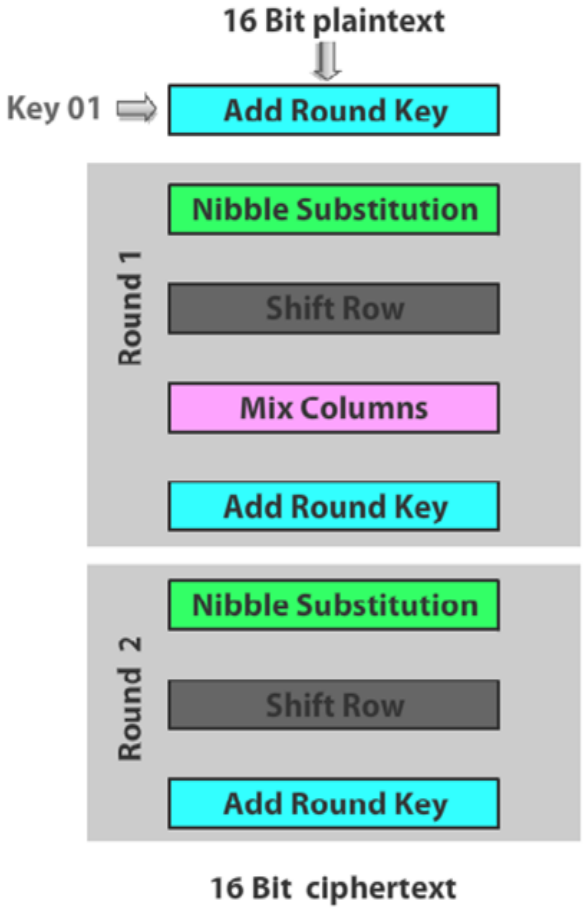


Fig. 3. Encryption overview

Because AES works on matrices, therefore plain text is placed in a matrix column wise known as state matrix. Encryption process performs following four operations in multiple rounds on state matrix.

A. Add Round Key

Round-0 includes only one operation that is "Add Round Key" in which 16-bit data in the state matrix is combined with 16-bit key via XOR operation. "A749" is used as plain text and "2D55" as KEY-01.

The result is shown in Fig. 4 "8A1C". Round-1 and round-2 also involve Add Round Key, therefore in AES whatever are the number of rounds, each round includes the step of Add Round Key.

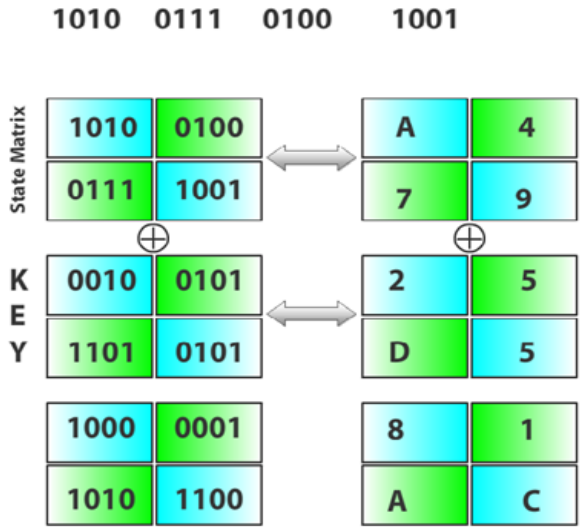
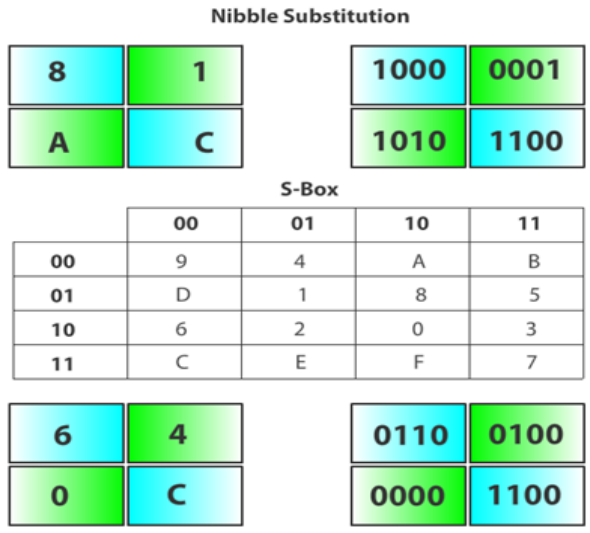


Fig. 4. Add Round Key

B. *Nibble Substitution*

Instead of dividing the block into a four by four array of bytes, SAES divide this into a two by two array of nibbles. This is called the state array [2].



	00	01	10	11
00	9	4	A	B
01	D	1	8	5
10	6	2	0	3
11	C	E	F	7

Fig. 5. Nibble Substitution

Values of state matrix are replaced with specific values chosen from S-BOX of Figure-5. As $(8)_{10}$ is equal to $(1000)_2$ therefore first two bits of each nibble in state matrix indicate a row and last two bits indicate column of S-BOX. The resultant values in the state matrix are written in Hexadecimal form i-e "604C".

C. *Shift Row*

Each row in the state matrix except first one is left shifted as shown in Fig. 6.

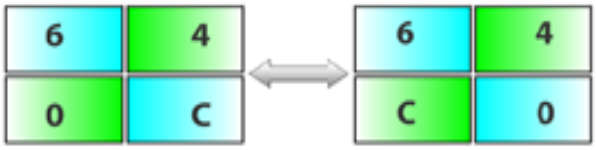


Fig. 6. Shift row

If state matrix has dimensions more than 2*2 as in case of AES then last row is left shifted by one, second last row is left shifted by two and so on, but first row remains unchanged.

D. *Mix Column*

The resultant state matrix of shift row is then multiplied with a specific matrix whose inverse must exist. If the inverse of this matrix does not exist then decryption of cipher text is no more possible.

We have used $\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix}$ matrix for the mixing purpose [1].

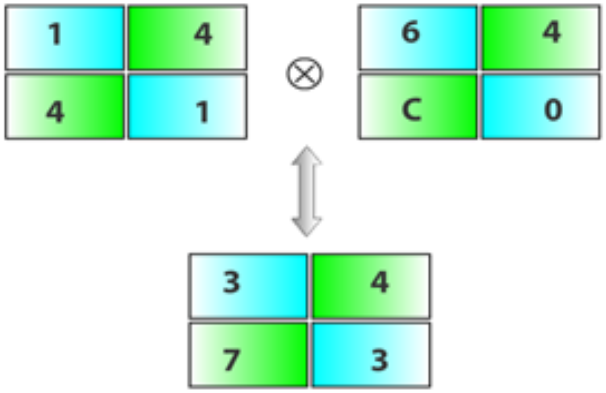


Fig. 7. Mix column

The result is of mix column is 3743 as given in Fig. 7. It is important to note that all the results shown in this paper are derived using Galois field algebra.

IV. Decryption

AES is a symmetric cipher therefore same key is used for encryption as well as decryption.

Implementation of decryption is illustrated in Fig. 8, 2D55, BCE9, A34A are used as keys in round-0, round-1 and round-2 respectively.

Since decryption is reverse process of encryption therefore it involves same steps as encryption but in reverse order with respect to their sequence as well as their operation. Decryption of SAES includes three rounds in which each round involve 'Add Round Key'. Similar to encryption four fundamental steps of decryption in SAES are repeated in subsequent rounds therefore author discuss them in detail as follow.

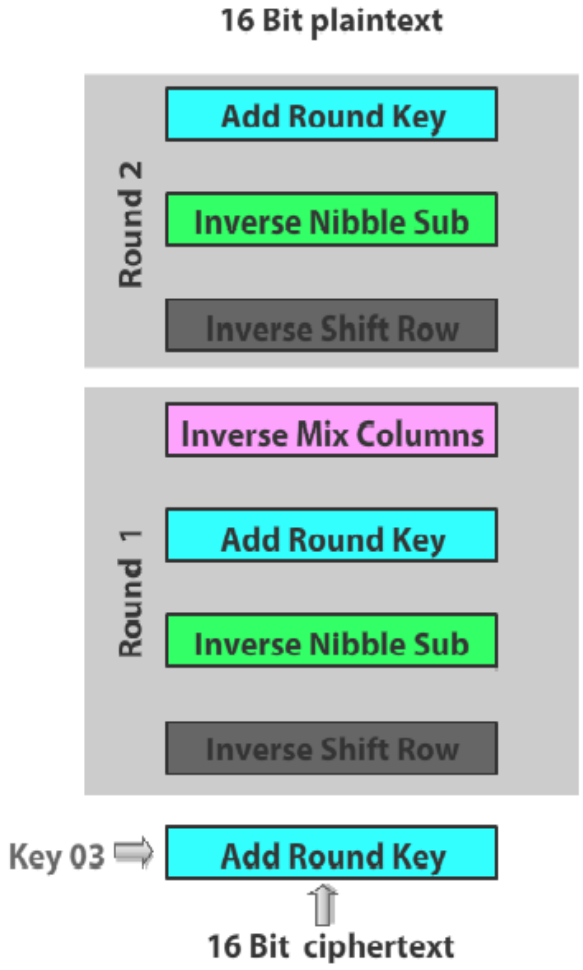


Fig.8. Decryption overview

A. *Inverse Mix Column*

Input matrix to inverse mix column is multiplied with a matrix which is inverse matrix of $\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix}$ used in Mix Column during encryption process.

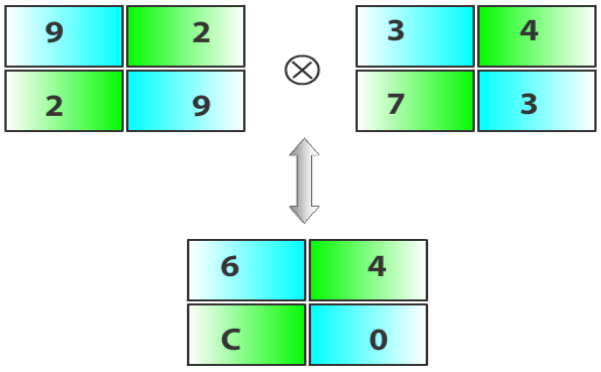


Fig. 09. Inverse Mix Column

Since

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix}\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Therefore

$\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix}$ is the inverse of $\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix}$

B. *Inverse Shift Row*

Each row in the state matrix except first one is right shifted as shown in Fig. 10.

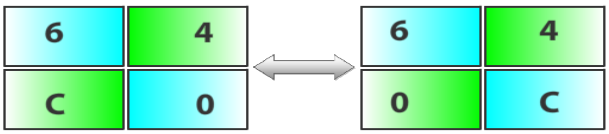


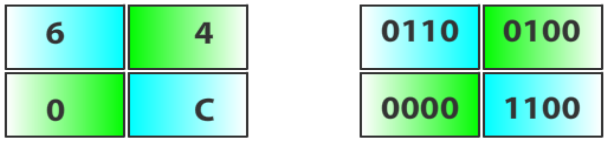
Fig. 10. Inverse Shift Row

If state matrix has dimensions more than 2*2 as in case of AES then last row is right shifted by one, second last row is right shifted by two and so on, but first row remains unchanged.

C. *Inverse Nibble Substitution*

Values of state matrix are replaced with specific values chosen from Inverse S-BOX.

Fig. 11 shows implementation of inverse nibble substitution.



Inverse S-Box

	00	01	10	11
00	A	5	9	B
01	1	7	8	F
10	6	0	2	3
11	C	4	D	E

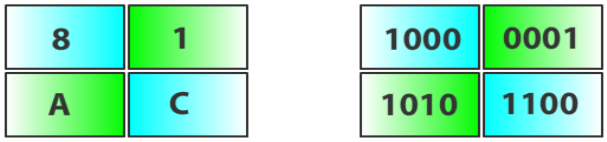


Fig. 11. Inverse Nibble Substitution

Since $(6)_{10}$ is equal to $(0110)_2$ therefore first two bits of each nibble in state matrix indicate a row and last two bits indicate column of inverse S-BOX. The new values in the state matrix are written in Hexadecimal form i-e "8A1C".

D. *Add Round Key*

Operation of Add Round Key in encryption as well as decryption is same as illustrated in Fig. 4 and as discussed

above same keys are used in each round of both encryption and decryption.

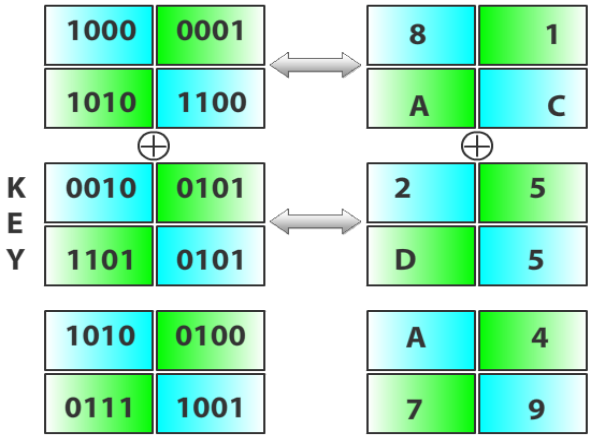


Fig. 12. Add Round Key

Fig. 12 illustrates recovered data after Add Round Key as A749 which is same plain text as encrypted at transmitter side as shown in Fig. 4.

V. Implementation

Since SAES deals with 16-bit data, therefore each entry in the state matrix is a nibble. Addition in 'Add Round Key', substitution from SBOX during 'Nibble Substitution' and 'Shift Row' operations does not result in a value that require more number of bits than state matrix holds, but the problem arises with 'Mix Columns', where multiplication of two nibbles result in a number that require more than 4-bits to store.

The Mix Columns transformation is defined as [1].

$$\begin{bmatrix} s'_{0,0} & s'_{0,1} \\ s'_{1,0} & s'_{1,1} \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix}$$

$$s'_{0,0} = s_{0,0} \oplus (4 * s_{0,1})$$
$$s'_{0,1} = s_{0,1} \oplus (4 * s_{1,1})$$
$$s'_{1,0} = (4 * s_{0,0}) \oplus s_{1,0}$$
$$s'_{1,1} = (4 * s_{0,1}) \oplus s_{1,1}$$

Inverse Mix Columns transformation is defined as [1].

$$\begin{bmatrix} s'_{0,0} & s'_{0,1} \\ s'_{1,0} & s'_{1,1} \end{bmatrix} = \begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix}$$

Since multiplication of 4-bit numbers result in a number which needs more than 4-bits to store but the problem is that each entry of the state matrix is limited to a 4-bit number in 16-bit SAES, therefore the number must be normalized to 4-bit in finite field GF(2^4) using polynomial arithmetic. In this paper the polynomial used to normalize a number to 4-bits is [1].

$$m(x) = x^4 + x + 1$$

$m(x)$ is a monic and irreducible polynomial, [5] and is derived from $(19)_{10} = (10011)_2$

VI. Results

SAES is a 16-bit block cipher which includes steps of Add Round Key, Nibble Substitution, Shift Row and Mix Columns repeatedly appearing in each round of encryption as shown in Fig. 3. Same is case with decryption.

Each of the four blocks for encryption and decryption are discussed individually in the previous sections of this paper. Table I and Table II is showing the step by step input and results of encryption and decryption process respectively.

TABLE I
STEP BY STEP ENCRYPTION RESULTS

Operation	Input	Key	Output
Add Round Key	A749	2D55	8A1C
Nibble Substitution	8A1C	××××	604C
Shift Row	604C	××××	6C40
Mix Columns	6C40	××××	3743
Add Round Key	3743	BCE9	8BAA
Nibble Substitution	8BAA	××××	6300
Shift Row	6300	××××	6003
Add Round Key	6003	A34A	C349

TABLE II
STEP BY STEP DECRYPTION RESULTS

Operation	Input	Key	Output
Add Round Key	C349	A34A	6003
Inverse Shift Row	6003	××××	6300
Inverse Nibble Sub	6300	××××	8BAA
Add Round Key	8BAA	BCE9	3743
Inverse Mix Columns	3743	××××	6C40
Inverse Shift Row	6C40	××××	604C
Inverse Nibble Sub	604C	××××	8A1C
Add Round Key	8A1C	2D55	A749

VII. Conclusion

After the release of Advanced Encryption Standard (AES) most of the people have made its animations in 128-bit, which is very difficult to understand by the beginners [3], [4]. This paper is to represent SAES in such an easy way that it helps beginners to understand each step of key generation, encryption and decryption easily using 16-bit plain text and keys. In this paper the authors have also discussed some of the key points used to implement 128-bit AES. After going through this paper one can easily understand and implement the 128-bit AES algorithm.

A formula is also proposed in this paper to calculate the length of key, therefore by using this formula it is easy to calculate the total number of bits in key, required for all rounds.

References

[1] William Stallings, "Cryptography and network security" 4th edition, pp. 134-173, Nov. 2005.

[2] Mohammad Musa, Edward Schaefer, and Stephen Wedig, A simplified AES algorithm and its linear and differential cryptanalyses, Cryptologia 27, pp. 148–177, Apr. 2003.
[3] Rijndael 128-bit cipher animation [online]. Available:http://www.formaestudio.com/rijndaelinspector/
[4] AES encryption process animation, 2009 [online]. Available:http://blog.ultrassecreto.com/wp-content/uploads/2009/06/projetofinal.html
[5] Ranjan Bose, "Information Theory, Coding and Cryptography" 2nd edition, 2008.

Abdul Atisam Farooq has received MS degree in electrical engineering from HITEC University, Taxila, Pakistan, in 2011 and B.S. degree in electrical engineering from Bahria University, Islamabad, Pakistan, in 2009 with an honor of Gold Medal. In 2009, he has joined the Department of Electrical Engineering at HITEC University, Taxila, Pakistan as a Lab Engineer and has been promoted at the position of lecturer in 2011. In August 2012 he joined Lingua Nova AG, Aarau, Switzerland. His major research interests are in the area of Cryptography, wireless communication, Instrumentation and Electromechanical systems.

Waqas Ahmed attained his degree of Bachelors in Communication Engineering in the year 2008 from FAST University Islamabad, Pakistan. For his Masters in Communication Engineering, he joined HITEC University, Taxila, Pakistan in 2009. During his course-work, he was appointed as a Lab-Engineer at the Department of Electrical Engineering of same University and right now he is serving as a junior lecturer. His area of interest involves Signal Processing, Image Processing, Speech Processing, Wireless Communication, and Secure Communication.

Zeeshan Ahmad got his degree of Bachelors in Electronics Engineering in 2008 from International Islamic University Islamabad, Pakistan. He joined HITEC University, Taxila, Pakistan as a Lab Engineer and also started his MS there in 2009. Now a day he is serving as a Junior Lecturer in the Department of Electrical Engineering of same University. His area of research involves Control Systems, Wireless Communication, and Secure Communication.