"AES Implementation And Designing To Perform Encryption And Encoding"

Dr.S.K.Dubey1, Charu Tyagi2
[1] Head of the Department AIMT, Gr.Noida
[2]Astt. Prof. RKGIT (W), Ghaziabad

Abstract

This paper proposes an efficient FPGA implementation of advanced encryption standard (AES) .Advanced Encryption Standard (AES), a Federal Information Processing Standard (FIPS), is an approved cryptographic algorithm that can be used to protect electronic data. The AES can be programmed in software or built with pure hardware. However Field Programmable Gate Arrays (FPGAs) offer a quicker and more customizable solution. Its implementation here used the mixing of VHDL and Handel-C and also used partial and dynamic reconfiguration in order to reach a very high performance.

1. Introduction

Today most of our information is transmitted digitally. This transmission occurs through many insecure networks as wireless networks. The unauthorized access to such information is very easy, so we transmit it using certain algorithms so that it appears unintelligent sequence of bytes to others. The algorithm originates from the initiative of the National Institute of Standards and Technology (NIST) in 1997 to select a new symmetric key encryption algorithm. The Rijndael algorithm is a symmetric block cipher that can process data blocks of 128 bits through the use of cipher keys with lengths of 128, 192, and 256 bits. Rijndael algorithm was selected as the Advanced Encryption Standard (AES) due to the combination of security, performance, efficiency, ease of implementation and flexibility.AES is based on a design principle known as a substitution-permutation network. It is fast in both software and hardware. The input and output for the AES algorithm consists of sequences of 128 bits. These sequences are referred to as blocks and the numbers of bits they contain are referred to as their length.

2. AES ENCRYPTION

The encryption consists of a number of different transformations applied consecutively over the data block bits, in a fixed number of iterations, called rounds. The number of rounds depends on the length of the key used for the encryption process. The encryption is achieved by passing the plain text through an initial round, nine equal rounds and a final round..In all the phases of each round, the algorithm operates on 4×4 array of bytes called the state. The AES algorithm is divided into four different phases which are executed in a sequential way forming rounds.

2.1 Algorithm Specifications

For the AES algorithm, the length of the input block, the output block and the State is 128 bits. This is represented by *Nb* = 4, which reflects the number of 32-bit words (number of columns) in the State. At the start of the Cipher, the input is copied to the State array. After an initial Round Key addition, the State array is transformed by implementing a round function 10, 12, or 14 times (depending on the key length), with the final round differing slightly from the first ***Nr*** -1 rounds. The final State is then copied to the output. The round function is parameterized using a key schedule that consists of a one-dimensional array of four-byte words derived using the Key Expansion routine. Each

round function contains uniform and parallel four steps: Sub Bytes, Shift Rows, Mix Column and Add Round Key transformation and each step has its own particular functionality. Here the round key is derived from the initial key and repeatedly applied to transform the block of plain text into cipher text blocks. The block and the key lengths can be independently specified to any multiple of 32 bits, with a minimum of 128 and a maximum of 256 bits. The repeated application of a round transformation state depends on the block length and the key length. For various block length and key length variable's value are given in table1. The number of rounds of AES algorithm to be performed during the execution of the algorithm is dependent on the key size. The number of rounds, Key length and Block Size in the AES standard is summarized in Table 1.

Table 1: Margin specifications Key-Block-Round Combinations for AES

	Key length(Nk round)	Key length (bits)	Number of rounds(Nr)
AES-128	4	128	10
AES-192	6	192	12
AES-256	8	256	14

The encryption/decryption process runs as follows in figure 1.

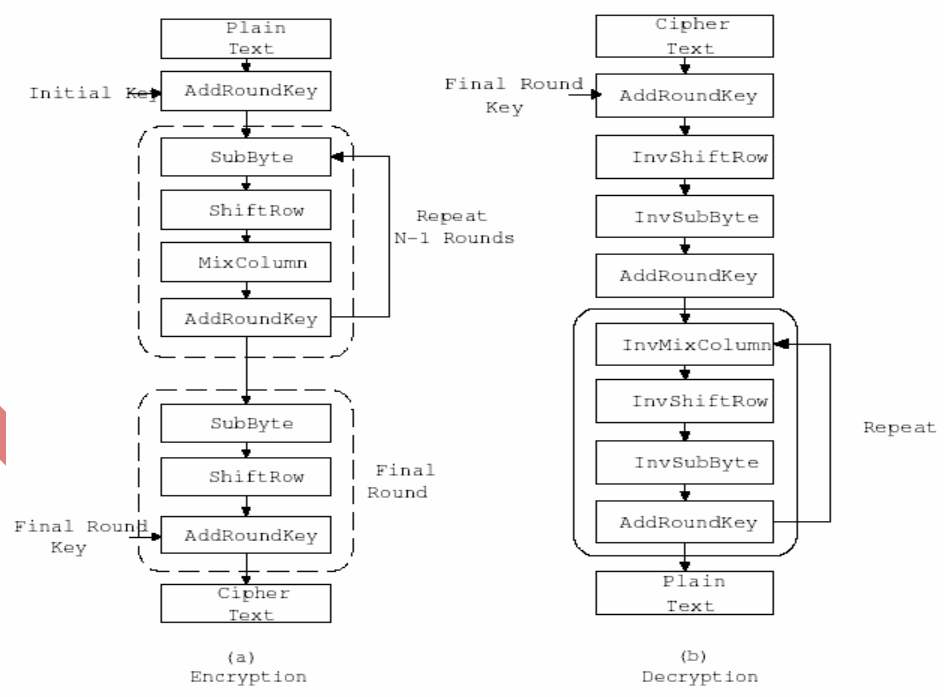


Fig.1: AES algorithm

(a) Encryption Structure (b) Decryption Structure

The various round functions are defined as follows-

2.1.1 The Sub Bytes Transformation

The bytes substitution transformation Byte sub (state) is a non-linear substitution of bytes that operates independently on each byte of the State using a substitution table(S box) This S-box which is invertible, is constructed by composing two transformations.

1. Take the multiplicative inverse in the finite field GF (28), The element {00} is mapped to itself.
2. Apply the following affine transformation (over GF (2))

$$b_i' = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$$

for $0 \leq i \leq 8$, where bi is the ith bit of the byte, and ci is the *i*th bit of a byte *c* with the value {63} or {01100011}. Here and elsewhere, a prime on a variable (e.g., b') indicates that the variable is to be updated with the value on the right. In matrix form, the affine transformation element of the S-box can be expressed as

$$\begin{bmatrix} b0' \\ b1' \\ b2' \\ b3' \\ b4' \\ b5' \\ b6' \\ b7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b0 \\ b1 \\ b2 \\ b3 \\ b4 \\ b5 \\ b6 \\ b7 \end{bmatrix}$$

1.1.2 .The Shift Row Transformation

In the Shift Rows transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, *r* =0, is not shifted.

This has the effect of moving bytes to "lower" positions in the row (i.e., lower values of *c* in a given row), while the "lowest" bytes wrap around into the "top" of the row (i.e., higher values of *c* in a given row). Figure 2 illustrates the transformation

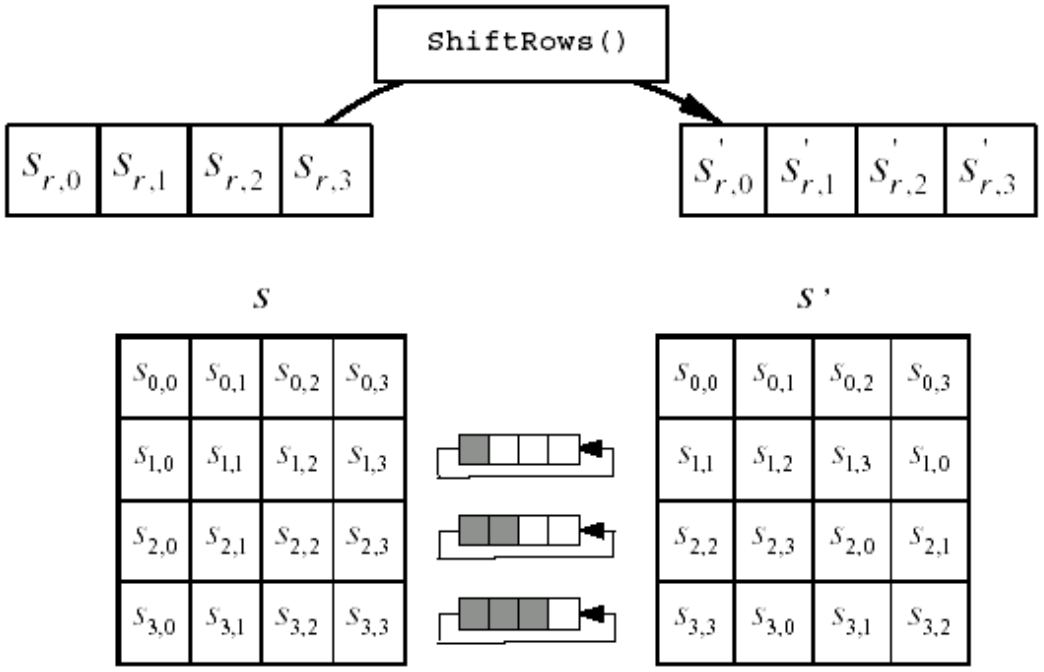


Figure2.Cyclic Shift of the Last Three Rows of the State

2.1.3. Mix Column Transformation

This transformation is based on Galois Field multiplication. Each byte of a column is replaced with another value that is a function of all four bytes in the given column. The Mix Columns transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over GF (28) and multiplied modulo $x4 + 1$ with a fixed polynomial $a(x)$, given by the following equation

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}.$$

This can be written as a matrix multiplication

$S'(x) = a(x) \otimes S(x)$:

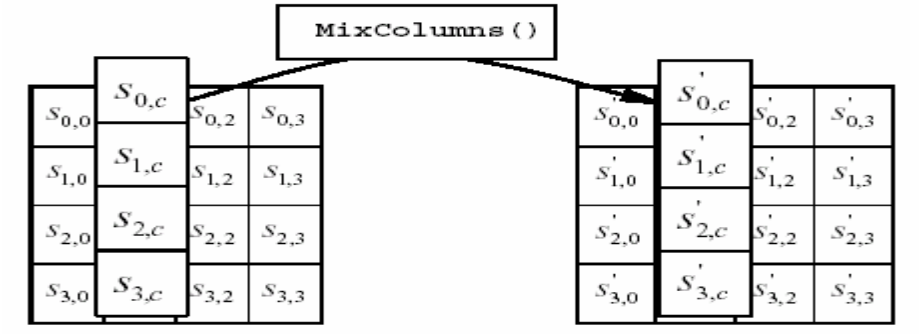


Figure3 Mixing of columns of state.

2.1.4. Addition of Round Key Transformation

In the Addition of Round Key transformation, a Round Key is added to the State by a simple bitwise XOR operation. Each Round Key consists of Nb words from the key schedule generation. Those Nb words are each added into the columns of the State, such that

$$[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] = [S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}] \otimes [W_{round*Nb}] \quad \text{for } 0 \leq c < Nb,$$

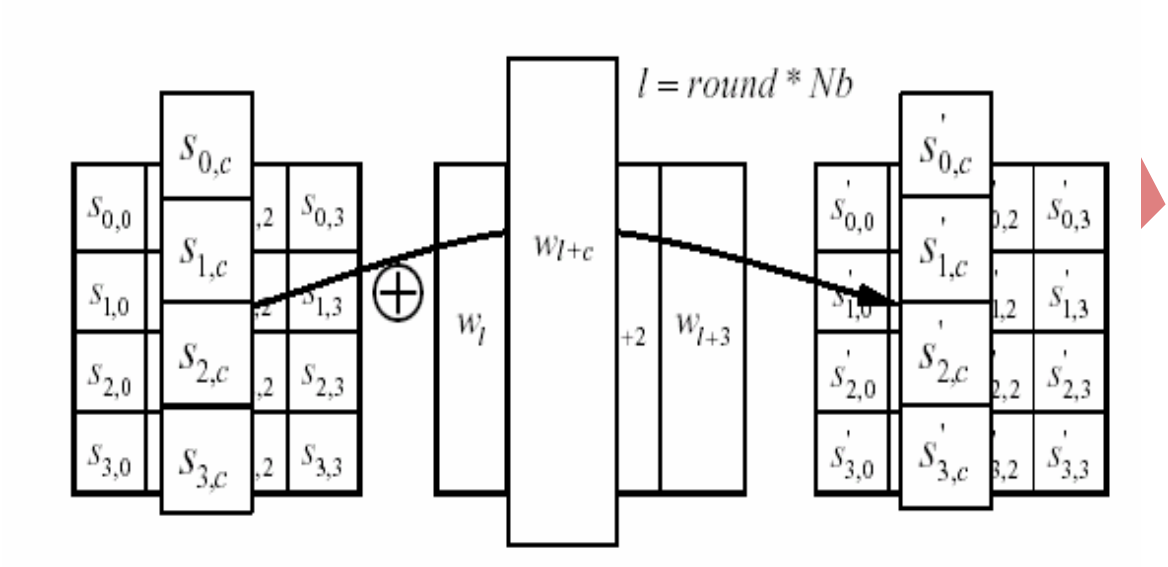


Figure 4.Exclusive-OR Operation of State and Cipher Key Words

where [wi] are the key generation words, and round is a value in the range in the Encryption, the initial Round Key addition occurs when round = 0, prior to the first application of the round function. The application of the Add Round Key transformation to the Nr rounds of the encryption occurs when $1 \leq round \leq Nr$. The action of this transformation is illustrated in figure 4, where l = round * Nb.

3. KEY Generation

Each round key is a 4-word (128-bit) array generated as a product of the previous round key, a constant that changes each round, and a series of S-Box lookups for each 32-bit word of the key. The first round key is the same as the original user input. Each byte (w0 - w3) of initial key is XOR'd with a constant that depends on the current round, and the result of the S-Box lookup for , to form the next round key. The Key schedule Expansion generates a total of Nb(Nr + 1) words: the algorithm requires an initial set of Nb words, and each of the Nr rounds requires Nb words of key data. The resulting key schedule consists of a linear array of 4-byte words, with *i* in the range $0 \leq i < Nb(Nr + 1)$.

4. Methodology

Our approach is based on mix of two hardware description language. On the one hand we have VHDL, a hardware description language which will allow us to implement the components that will be reconfigured in run time by means of JBits, a java library developed to make this task. In order to do this we have to locate manually the components and o synthesize them.

On the other hand, the final implementation of both algorithms has been made by means of Handel-C (the other language used).This language is very close to the classical programmer (it is similar to C language) and allows us to reduce the final development time thanks to its ability to implement and manage several elements of our design, like pipelining registers, memories, the host FPGA communication system etc. Finally, as we use two different languages, we have to join both codes. To do this we synthesize the VHDL components by means of XST tool of Xilinx ISE 8.1i and we reference them in handle –C by means of interfaces. Besides the languages used, we have followed a pipeline and parallel methodology. In both algorithms we have pipeline as much as possible. This pipelining has been made in two ways: an operational pipelining and inner pipelining In the operational pipelining all the operations of both algorithms are executed in a pipeline way, that is when the first operation finishes the second block goes into that operation while the first goes into the next operation. In the inner operation the most complex operations will be pipelined in order to increment the clock frequency. This other pipelining stage is only used in the IDEA algorithm .Since the AES algorithm has no complex operation.Finally, we have used parallelism as much as possible, calculating for example, the result of all bytes of the state in a parallel way in all phases of AES algorithm. Later we will see the pipelining and parallelism in detail.

4.1. *The union between VHDL and Handel-C*

We have mixed VHDL and Handel-C in our design. This mix is made in two phases: a first step in which all the VHDL components must be synthesized, a second step in which these components will be used in Handel –C by means of interfaces. The second step, we will employ the synthesized VHDL elements in our Handel –C code by means of interfaces. This utilization is made in two parts:

1. Interface definition: In this first phase, the interface of all elements must be set up. Besides we will establish the inputs to and the outputs from the interfaces. The name of each element interface
   must be same as the synthesized VHDL entity name of that element.
2. Interface access; After the interface is defined, we can access it by means of its instance name. It is important to emphasize that two instances of the same interface are different and they will be implemented separately. In order to access to the output data of an instance we only have to write the name of the instance followed by the output port, the two separated by a dot.

5. Performance comparison between hardware and software implementation

The hardware implementation of AES algorithm was realized in a way, that it was possible to perform each function of algorithm independently. An event command was transmitted (using status and control register of RC1000) from host to FPGA, which defined a particular function to be executed. Therefore we were capable to determine the executing time of individual function. In order to demonstrate the advantage of parallel implementation of AES algorithm, we implemented AES algorithm once using the parallelism (PAR) and once without using it (SEQ).

The average execution times of different functions by hardware implementation of AES algorithm (by encoding/decoding one block-128bit of data) are presented in Table 2. The average execution times of different functions by software implementation (reference implementation by Brian Glad man in Visual C++) of AES algorithm (AMD Athlon 1.3 GHz processor was used) are presented in Table 3

The data in Table 2 show the benefit of using parallelism. The Cipher is executed 110% faster using parallel implementation then using sequential implementation. The parallel Inverse Cipher execution is also much faster (1.38

times) in comparison by sequential Inverse Cipher. Comparison of data in Table 2 with data in Table 3 leads to some further conclusions:

• The Cipher, implemented in hardware (parallel implementation) is executed approximately 23 times faster than the Cipher implemented in software.

• The Inverse Cipher, implemented in hardware (parallel implementation) is executed approximately 29 times faster than the Inverse Cipher implemented in software.

• The Key Expansion function implemented in hardware is executed 1.88 times slower than the Key Expansion function implemented in software.

• The DMA transfer, which is used to transfer data to and from the SRAM banks respectively, consumes a large amount of time.

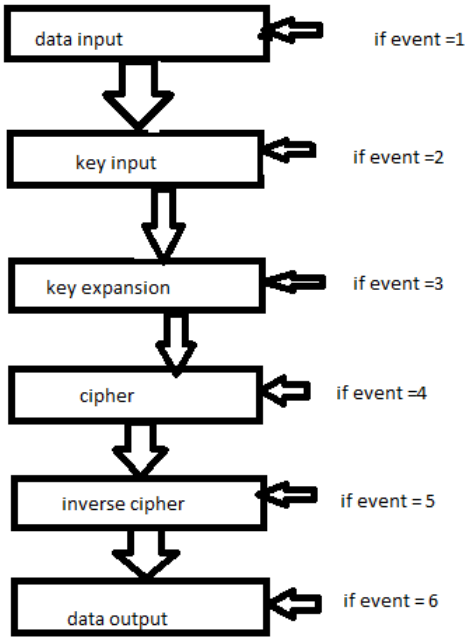


Figure 5.Structure of hardware implementation of AES algorithm

As described above, the Key Expansion function generates the Key Schedule using the Input Cipher Key. In hardware implementation the Key Schedule was stored in a block RAM (BRAB) of FPGA. Because of such implementation, parallel execution of Key Expansion function was limited (this is the reason for slow execution in hardware implementation in comparison with software implementation). Notice, however, that the Key Schedule in the case of encrypting a large amount of data is generated only once. On the other hand, the Cipher and Inverse Cipher function are performed on separate blocks (128 bit) of data until the whole amount of data is encrypted. The final performance of the hardware implementation of AES algorithm is therefore practically not affected by the speed of the Key Expansion function execution

Table 2:Execution times by hardware implementation.

function	average execution time (microsecond)	Throughput (Mbits/sec)
Sending control and status message	64.70	-
DMA transfer	256.65	-
Key expansion	8.86	14.44
Cipher SEQ	22.91	5.59
Cipher PAR	.70	182.86
Inverse Cipher SEQ	22.91	5.59
Inverse Cipher PAR	.70	182.86

.

Table 3:Execution times of different functions by software implementation.

function	average execution times (μs)	throughput (Mbit/s)
Key Expansion	4.71	27.81
Cipher	16.50	7.76
Inverse Cipher	20.56	6.23

As we can see from Table 2, the DMA transfer consumes large amount of time. Similar to the Key Expansion function, the DMA transfer is also performed only once for fixed amount of data. So in the case of encrypting small amount of data (< 100 Kbytes) the final performance of software implementation is better than hardware implementation. The main reason for that is the low speed of DMA transfer. In the case of encrypting large amount of data (> 100 Kbytes) the speed of Cipher and Inverse Cipher execution (that are performed many times) contribute the main part to the final performance of algorithm. In this case, the hardware implementation of AES algorithm is much faster than the software implementation. With timing analysis and optimization of Handel-C design of the AES algorithm hardware implementation can be further improved. Currently our hardware implementation of the AES algorithm works with a clock rate
of 74.4 MHz which meets the needs of the target application.

6. Application of AES in image encryption

Due to the increasing use of images in industrial process, it is essential to protect the confidential image data from unauthorized access .Many image protection techniques had been used till now.But all the techniques had one common defect that made them ineffective. They posses the policy of centralized storage, in that an entire protected image is usually maintained in a single information carrier. If a cracker detects an abnormality in the information carrier in which the protected image resides, he or she may intercept it, attempt to decipher the secret inside or simply

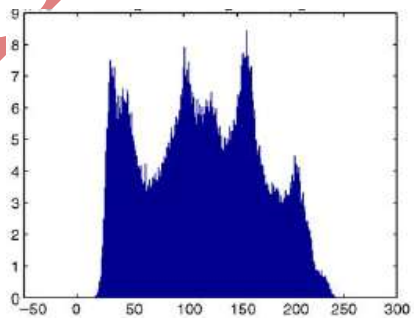
ruin the entire information carrier(and once the information carrier is destroyed, the secret image is also lost forever). Another method is to encrypt image data used DES (Data Encryption Standard). But DES is very complicated and involves large computations. A software DES implementation is not fast enough to process the vast amount of data generated by multimedia applications and a hardware DES implementation (a set-top box) adds extra costs both to broadcasters and to receivers. In order to tackle these problems AES was used. AES is very fast symmetric block algorithm by hardware implementation in comparision to other algorithms. Applications that require fast processing such as smart cards, cellular phones and image-video encryption use AES algorithm. However, a central consideration for any cryptographic system is its susceptibility to possible attacks against the encryption algorithm such as statistical attack, differential attack, and various brute attacks.

There are two levels of security for digital image encryption: low-level security encryption and high level security encryption. In low-level security encryption, the encrypted image has degraded visual quality compared to that of the original one, but the content of the image is still visible and understandable to the viewers. In the high-level security case, the content is completely scrambled and the image just looks like random noise. In this case, the image is not understandable at all to the viewers. A modified version of AES algorithm is used for encrypting image. The modification is done by adding a key stream generator, such as (A5/1, W7), to the AES image encryption algorithm in order to increase the image security and in turn the encryption performance.

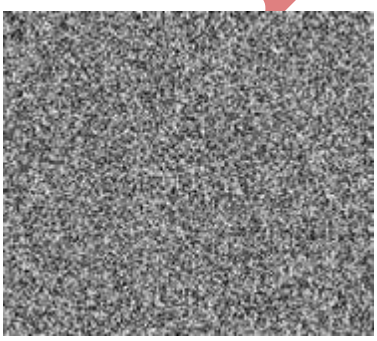
6.1 Security analysis by statistical method

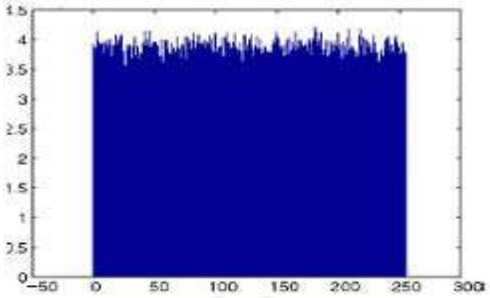
To confuse the powerful attackers Shannon pointed two methods of confusion and diffusion based on statistical analysis. Statistical analysis has been performed on the AES, demonstrating its superior confusion and diffusion properties which strongly defend against statistical attacks. This is shown by a test of histogram. The histogram of the ciphered image is fairly uniform and is significantly different from that of the original image. Therefore, it does not provide any indication to employ any statistical attack on the image under consideration. Moreover, there is no loss of image quality after performing the encryption/decryption steps.

original image



histogram of original image





Encrypted image histogram of encrypted algorithm

Fig. 6 Histograms of the plain image and ciphered image

Also we test the correlation between two vertically adjacent pixels, and two horizontally adjacent pixels respectively, in a ciphered image. First, we randomly select n pairs of two adjacent pixels from an image. Then, we calculate the correlation coefficient of each pair by using the following formula.

$$cov(x,y) = E(x - E(x))(y - E(y))$$

Where x and y are grey-scale values of two adjacent pixels in the image. Figs. 7 show the correlation distribution of two horizontally adjacent pixels in the plain-image and in the ciphered image, respectively

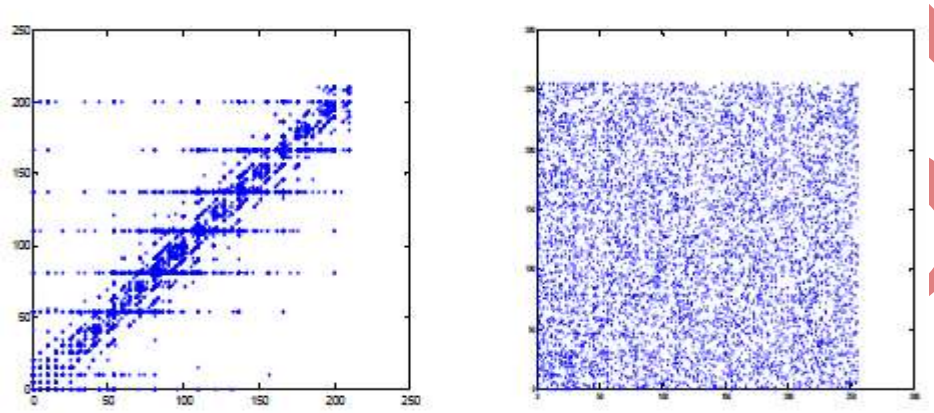


Fig 8. Correlation of two horizontally adjacent pixels; (a) in the plain-image, and (b) in the ciphered-image

Table 4.

CORRELATION COEFFICIENTS OF TWO ADJACENT PIXELS IN TWO IMAGES

(LENA TEST IMAGE ENCRYPTED USING DIFFERENT K_i)

correlation	horizontal	vertical
Plain image	0,93	0,95
Image encrypted by k_1	0,058	0,051
Image encrypted by k_2	0,036	0,035
Image encrypted by k_3	0,047	0,044
Image encrypted by k_4	0,06	0,054

References

1. SCHNEIER, B. (1996), Applied Cryptography: Protocols, Algorithms, and Source Code in C, John Wiley & Sons .

2. William Stallings,(2006), Cryptography and Network Security, Principles and Practices, 4th ed. Pearson Education, pp. 134-161.
3. Ming-Haw Jing, Zih-Heng Chen, Jian-Hong Chen, and Yan-Haw Chen, (2007)"Reconfigurable system for high speed and diversified AES using FPGA", **Microprocessors and Microsystems,** vol. 31, Issue 2, , pp. 94-102.
4. HELION Technology Limited, "High performance AES (Rijndael) cores for FPGA," available at **http://www.heliontech.com/core2.htm.**
5. Chih-Chung Lu and Shau-Yin Tseng,(2002) "Integrated Design of AES (Advanced Encryption Standard) Encrypted and Decrypted," *Proc. IEEE Int. Conf. on Application-Specific Systems, Architectures, and Processors, (ASAP'02)*, pp. 277-285.
6. H. Cheng, L. Xiaobo, (2000)Partial encryption of compressed images and videos. *IEEE Trans. Signal Process*. 48 (8), 2439–2451.