AES Algorithm's Architectural Implementation For Secure Data Transmission

Jyotirmoy Pathak

School of Electronics and Electrical Engineering
Lovely Professional University, Punjab

Abstract

Advanced Encryption Standard (AES) calculation is one of the most ordinarily utilized symmetric square figure calculations around the world. The calculation has its own specific structure to scramble and unscramble delicate information in equipment and programming. It is very hard for programmers to get genuine information while scrambling by the AES calculation. It utilizes a mix of Octet Substitution, S-box, section pivots, Exclusive-OR(XOR) and a blended segment. There is no such proof of anybody split this calculation to date. AES can manage three diverse key sizes, for example, AES 256,192,128 piece and every one of these figures has 128-piece square size. This undertaking will give a review of the AES calculation and clarify a few critical highlights of this calculation in subtleties

1. Introduction

AES is subject to an arrangement standard known as a substitution–change masterminds and is viable in both programming and equipment [1]. AES is a variety of Rijndael which has a fixed square size of 128 bits, and a key size of 128, 192, or 256 bits. It is specific for the encryption of electronic data set up by the U.S. National Institute of Standards and Technology (NIST) in 2001. It is a web gadget to scramble and interpret content utilizing the AES encryption calculation. You can pick 128, 192 or 256-piece long key size for encryption and unscrambling. The eventual outcome of the method is downloadable in a substance record. The Advanced Encryption Standard, or AES, is a symmetric square figure picked by the U.S. government to make sure about organized information and is completed in programming and hardware all through the world to encode fragile data [2-4].

The National Institute of Standards and Technology (NIST) began the headway of AES in 1997 [5] when it proclaimed the prerequisite for a successor computation for the Data Encryption Standard (DES), which was starting to end up being unprotected to animal constrain attacks. This new, propelled encryption calculation would be unclassified and must be "fit for guaranteeing unstable government information well into the next century," as showed by the NIST presentation of the strategy for development of a propelled encryption standard count. It was proposed to be anything other than hard to realize in hardware and programming, similarly as in restricted circumstances (for example, in shrewd cards) and offer incredible watchmen against various attack methodology. The more unmistakable and comprehensively grasped symmetric

encryption computation obligated to be encountered nowadays is the Advanced Encryption Standard (AES). It is discovered something like multiple times speedier than triple DES [6].

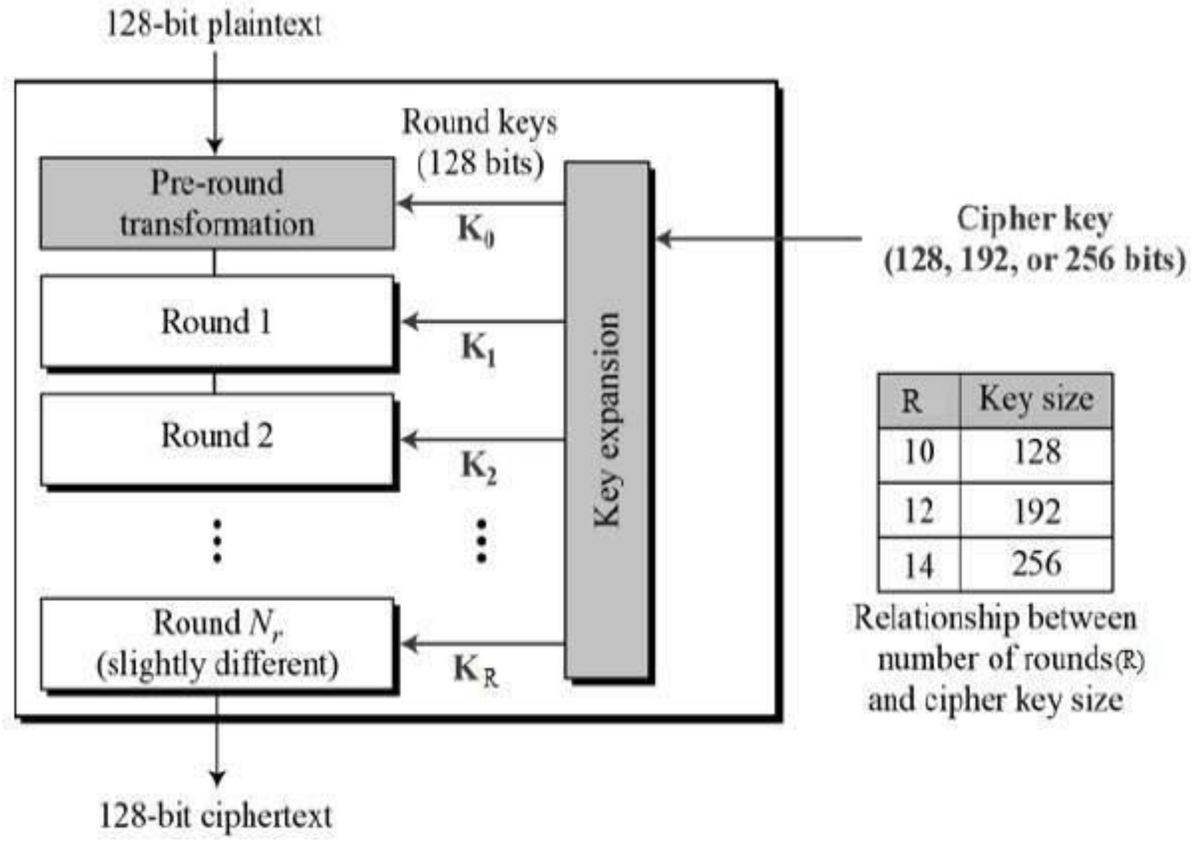
A swap for DES was required as its key size was too pretty much nothing. With growing preparing power, it was seen as defenseless against careful key interest attack. Triple DES was expected to crush this impediment yet it was found moderate.

The highlights of AES are as per the following –

- Symmetric key symmetric square figure
- 128-bit information, 128/192/256-piece keys
- Software was implementable in C and Java.
- Provide full determination and configuration subtleties
- Stronger and quicker than Triple-DES.

AES is an iterative rather than Feistel's figure. It relies upon 'substitution–change mastermind'. It contains a movement of associated assignments, some of which incorporate displacing commitments by unequivocal yields (substitutions) and others incorporate reworking bits around (changes). Inquisitively, AES plays out the total of its counts on bytes rather than bits. Thusly [7], AES treats the 128 bits of a plaintext hinder as 16 bytes. These 16 bytes are planned in four portions and four segments for taking care of as a cross section.

Rather than DES, the amount of rounds in AES is variable and depends upon the length of the key. AES uses 10 rounds for 128-piece keys, 12 rounds for 192-piece keys and 14 rounds for 256-piece keys appeared in figure 1. All of these rounds uses an other 128-piece round key, which is resolved from the first AES key.



R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

Table 1 AES vs DES

	DES	**AES**
Year	1976	1999
Block size	64	128
Key length	56	128, 192, 256
Number of rounds	16	9, 11, 13
Encryption primitives	Substitution, permutation	Substitution, shift, bit mixing
Cryptographic primitives	Confusion, diffusion	Confusion, diffusion
Design	Open	Open
Design rationale	Closed	Open
Selection process	Secret	The secret, but accept open public comment
Source	IBM, enhanced by NSA	Independent cryptographers

2. AES Features

The assurance strategy for this new symmetric key calculation was totally open to open examination and comment; this ensured a thorough, clear assessment of the structures submitted. NIST demonstrated the new propelled encryption standard computation must be a square figure furnished for managing 128-piece squares, using keys assessed at 128, 192, and 256 bits; other models for being picked as the accompanying propelled encryption standard count included:

•Security: Competing calculations were to be decided on their capacity to oppose assault, when contrasted with other submitted figures, however security quality was to be viewed as the most significant factor in the opposition.

•Cost: Intended to be discharged under a worldwide, nonexclusive and eminence free premise, the up-and-comer calculations were to be assessed on computational and memory proficiency.

Implementation: Algorithm and usage attributes to be assessed incorporated the adaptability of the calculation; reasonableness of the calculation to be actualized in equipment or programming; and generally speaking, the overall effortlessness of execution.

3. Architecture of AES

The Advanced Encryption Standard (AES) calculation has distributed by NIST as a draft FIPS197 in 2001. There are various equipment executions were proposed for it, among all the usage generally, they have focused on the AES with 128-bits key size [8]. This key size is viewed as proper for a large portion of the business applications, where utilizing higher key sizes is considered as an abundance of assets. It includes higher territory usage with longer preparing time and difficult to execute for little scale gadgets. Figure 2 presents, AES is an iterative process, repeat internal operation (a) AddRoundKey (b) SubByte (c) ShiftRow and (d) MixColumn for N number of times. In the last round mix column operation is not executed.

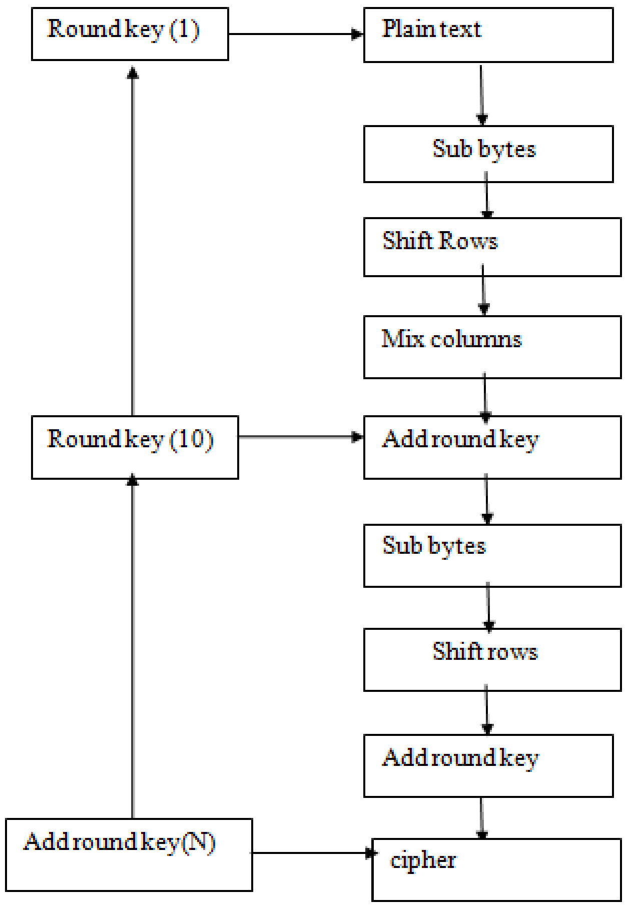
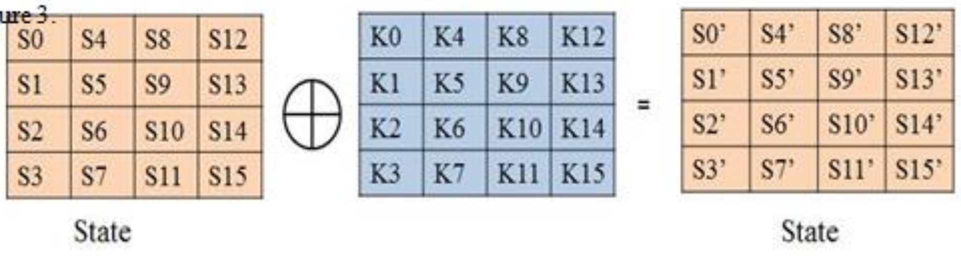


Figure 2 Rounds of AES Encryption algorithm

3.1 AddRoundKey Transformation

The AddRoundKey adds the round keyword with each column of the state matrix. It is similar to MixColumns; the AddRoundKeyproceeds one column at a time. The most important in this transfomation, that it includes the cipher key[9]. The state column will get XOR with the key which is generated by a key generator and create another state as shown in figure 3.



In the AddRoundKey() 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 Those Nb words are each added into the columns of the State, such that

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] \text{ xor } [w_{round*Nb+c}] \text{ for } 0 \le c < Nb$$

where $[w_i]$ are the key schedule words, and round is a value in the range $0 \le round \le Nr$. In the Cipher, the initial Round Key addition occurs when round = 0, prior to the first application of the round function. The application of the AddRoundKey() transformation to the N_r rounds of the Cipher occurs when $1 \le round \le Nr$.

3.1.1 Key Expansion

The key development term depicts the activity of creating all Round Keys from the first info key. The underlying round key is a unique key in the event of encryption and if there should arise an occurrence of decoding the last gathering of the created by key development keys will be unique keys. As referenced before this underlying round key will be added to the info before beginning the encryption or unscrambling emphases. The 128 bits key size, 10 groups of round keys will be produced with 16 bytes size. The round keys are created word by word. There are some comparative encryption changes used to create the round key.

3.2 ShiftRow

The transformation is called Shift Rows performs in encryption, in which rows are cyclic shifting to the left. The number of shifting depends upon the row number of the state matrix. First row no shifting, second-row one byte, third row two bytes and fourth-row three-byte shifting left. In the decryption [10], InvShiftRows transformation performs the right cyclic shifting operation inverse of ShiftRows; a number of shifting depends on a number of row numbers. Figure 4 shows the Cyclic ShiftRows transformation for the AES algorithm. In the ShiftRows () 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. Specifically, the ShiftRows () transformation proceeds as follows:

$$s' = s \text{ for } 0 \le r < 4 \text{ and } 0 \le c < Nb.$$

Where the shift value shift (r, Nb) depends on the row number, r, as follows shift (1, 4) = 1; shift (2, 4) = 2; shift (3, 4) = 3

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).

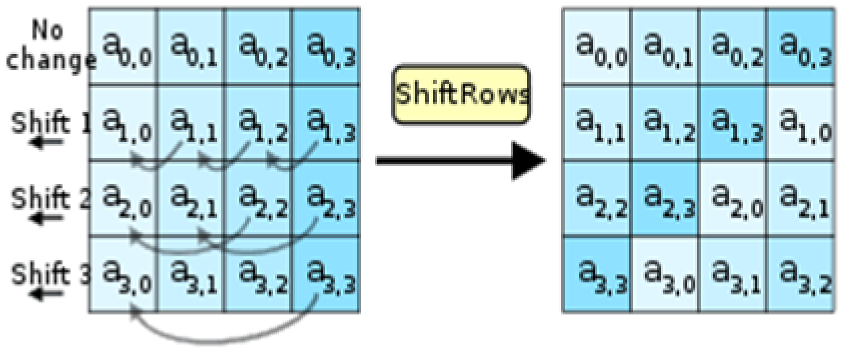

No change
Shift 1
Shift 2
Shift 3
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Figure 4 ShiftRow stage

3.3 Mix column /inverse mix column transformation

The MixColumns transformation operates on the State column-by-column, treating each column as a four-term polynomial. The MixColumns transformation functions after the ShiftRows on the State column-by-column, considering each column as a four-term polynomial. Inverse MixColumns are the inverse process of MixColumns which is used in the decryption of ciphertext. The columns are considered as polynomials over GF (2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial A (x).

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

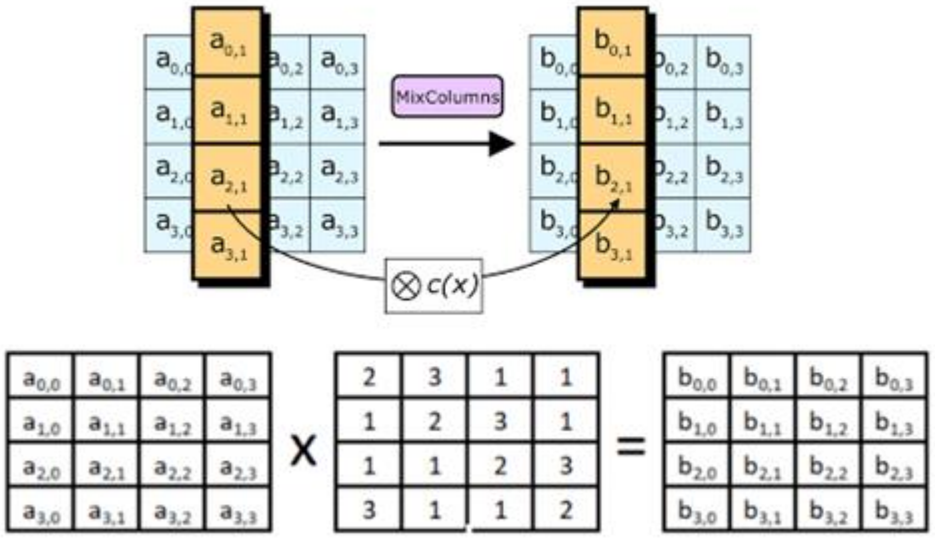

MixColumns
⊗ c(x)
$a_{0,1}$ $a_{1,1}$ $a_{2,1}$ $a_{3,1}$
$b_{0,1}$ $b_{1,1}$ $b_{2,1}$ $b_{3,1}$
$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$

Figure 5 MixColumn stage

The algorithm for MixColumns and Inverse MixColumns involves multiplication and addition in GF (2^8). The MixColumns multiplies the rows of the constant matrix by a column in the state. Figure 5 describes the operation of this transformation; key addition is the next transformation of the encryption.

3.4 SubBytes (substitution bytes)

The first transformation, SubBytes, is used for encryption and inverse SubBytes used for decryption. The SubBytes substitution is a nonlinear byte substitution that operates independently on each byte of the State using a substitution table (S-box). Take the multiplicative inverse in the finite field GF (2^8) and affine transform to do the SubBytes transformation. Inverse affine transform has to find for inverse SubBytes transformation then the multiplicative inverse of that byte.

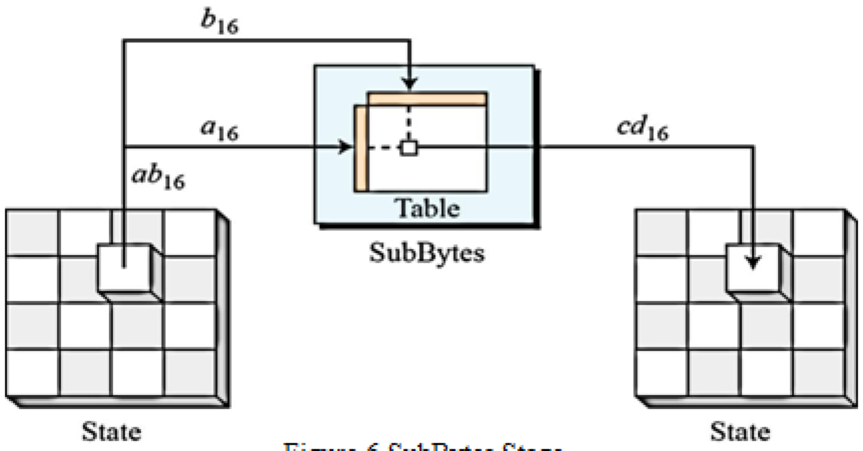
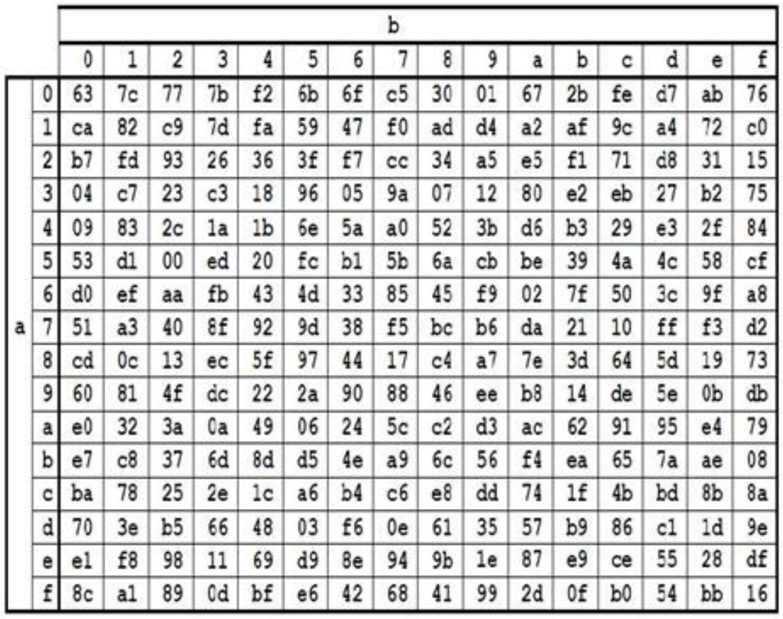


Figure 6 SubBytes Stage

Figure 6 indicates how the transformation can be done. There are two hexadecimal digits a and b in one state element, the left digit (a) defines the row and the right digit (b) defines the column of the substitution table. The junction of these two digits is the new bytes. Inverse SubBytes transformation is inverse of SubBytes transformation. It can find in a similar way the only table which is used for mapping the byte is different. The SubBytes transformation is done through S-box. There are two techniques to perform substitutions, (i)using the S-box table, and (ii) using composite field arithmetic. There are separate tables for SubBytes and its inverse; Table II is used for SubBytes transformation and Table III used for its inverse. It can be found using S-box architecture in composite field arithmetic.SubBytes table is also called as S-box and the inverse SubBytes table is an Inverse S - box. There are two parts of affine transformation and its inverse; a constant matrix will be multiplied with the data in the multiplication part, then the addition part, where a constant vector is added to multiplication result.

The substitute byte transformation, called SubBytes. AES defines a matrix of byte values, called an S-box that contains all possible 256 8-bit values. Each individual byte of State is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values provide as indexes into the S-box to select an 8-bit output value for the next process.

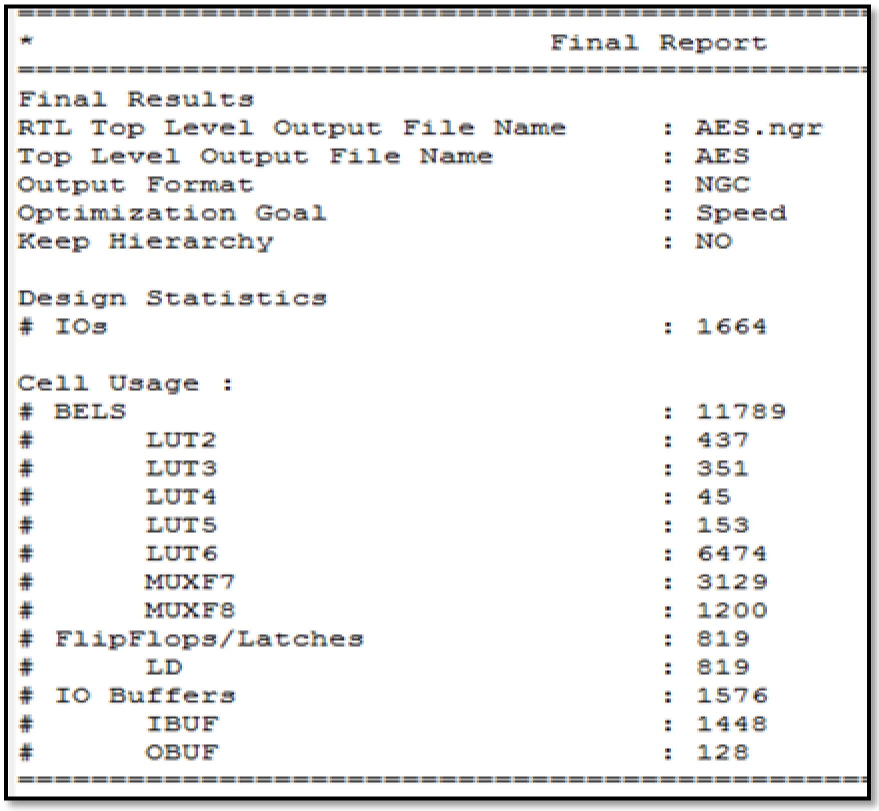
Table 2 Subbytes Transformation Table



		b															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
a	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

4. Implementation Result

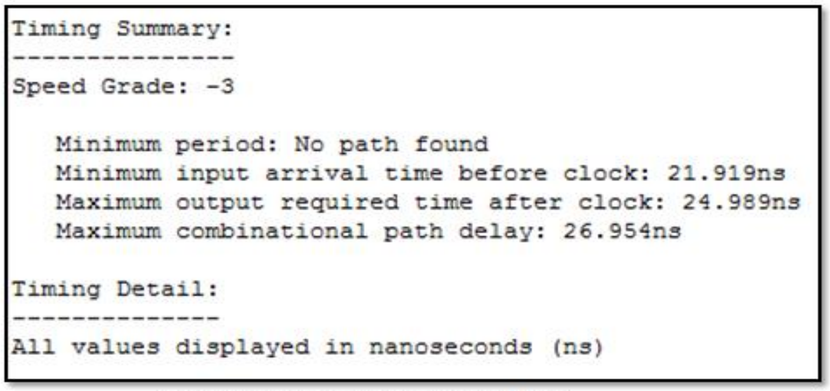
In this work, Xilinx ISE9.1i has been used to implement AES with Verilog HDL language. The synthesis result is obtained with XST synthesizer-based in Spartern3E FPGA. The Spartan-3E FPGA family offers the minimal effort and stage highlights you're searching for, making it perfect for the entryway is driven programmable rational structures. Spartan-3E is the seventh family in the earth-shattering low-cost Spartan Series and the third Xilinx family fabricated with cutting edge 90nm procedure innovation.



```
===============================================
*                                  Final Report
===============================================
Final Results
RTL Top Level Output File Name      : AES.ngr
Top Level Output File Name          : AES
Output Format                       : NGC
Optimization Goal                   : Speed
Keep Hierarchy                      : NO

Design Statistics
# IOs                               : 1664

Cell Usage :
# BELS                              : 11789
#      LUT2                         : 437
#      LUT3                         : 351
#      LUT4                         : 45
#      LUT5                         : 153
#      LUT6                         : 6474
#      MUXF7                        : 3129
#      MUXF8                        : 1200
# FlipFlops/Latches                 : 819
#      LD                           : 819
# IO Buffers                        : 1576
#      IBUF                         : 1448
#      OBUF                         : 128
===============================================
```



```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: No path found
   Minimum input arrival time before clock: 21.919ns
   Maximum output required time after clock: 24.989ns
   Maximum combinational path delay: 26.954ns

Timing Detail:
--------------
All values displayed in nanoseconds (ns)
```

Figure 9 Timing summary report of AES

Every outcome depends on re-enactments from the Xilinx ISE Simulator instruments, utilizing Test Bench Waveform Generator. All the individual changes of both encryption and unscrambling are recreated utilizing the FPGA SPARTAN-3E family. The waveforms produced by the 128-piece byte substitution change. The data sources are the clock of 1000ns timespan and128-bit state as a standard rational vector, whose yield is128-bit S-box query table substitution task is finished 1 clock cycles. Every one of the outcomes depends on reproductions from the Xilinx ISE Simulator devices, utilizing Test Bench Waveform Generator. All the individual change of encryption is re-enacted utilizing the FPGA SPARTAN-3E family. The waveforms plotted by the 128-piece total encryption Process.

(Cipher): AES square length/Plane Text = 128bits (Nb=4) Key length = 28 bits (Nk =4);

No. of Rounds = 10(Nr =10)

5. Conculusion

VLSI design of the S-box for AES with Verilog HDL is presented in this chapter. Bigger key lengths convert into an exponential increment in the multifaceted nature of a comprehensive inquiry. Side-channel assaults, be that as it may, utilize a partition and-overcome approach and henceforth it is, for the most part, accepted that expanding the key length can't be utilized as moderation. Quantity of rounds might be expanded to improve the quality of the AES. The quality of the AES calculation may be upgraded by expanding the key length from 128 bits to 512 bits and along these lines, the quantity of rounds is expanded so as to give a more grounded encryption technique to verify correspondence

Reference

[1] Kuo, Henry, and Ingrid Verbauwhede. "Architectural optimization for a 1.82 Gbits/sec VLSI implementation of the AES Rijndael algorithm." In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 51-64. Springer, Berlin, Heidelberg, 2001.

[2] Zhao, Jia, Xiaoyang Zeng, Jun Han, and Jun Chen. "Very low-cost VLSI implementation of AES algorithm." In *2006 IEEE Asian Solid-State Circuits Conference*, pp. 223-226. IEEE, 2006.

[3] Li, Zhen-rong, Yi-qi Zhuang, Chao Zhang, and J. I. N. Gang. "Low-power and area-optimized VLSI implementation of AES coprocessor for Zigbee system." *The Journal of China Universities of Posts and Telecommunications* 16, no. 3 (2009): 89-94.

[4] Deng, Liang, and Hongyi Chen. "A new VLSI implementation of the AES algorithm." In *IEEE 2002 International Conference on Communications, Circuits, and Systems and West Sino Expositions*, vol. 2, pp. 1500-1504. IEEE, 2002.

[5] Kumar, Saurabh. "VLSI implementation of AES algorithm." Ph.D. diss., 2013.

[6] Zhang, Xinmiao, and Keshab K. Parhi. "High-speed VLSI architectures for the AES algorithm." *IEEE transactions on very large scale integration (VLSI) systems* 12, no. 9 (2004): 957-967.

[7] Jyrwa, Banraplang, and Roy Paily. "An area-throughput efficient FPGA implementation of the block cipher AES algorithm." In *2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies*, pp. 328-332. IEEE, 2009.

[8] Zhao, Jia, Jun Han, Xiaoyang Zeng, and Jun Chen. "VLSI implementation of an AES algorithm resistant to Differential Power Analysis attack." In *2007 7th International Conference on ASIC*, pp. 838-841. IEEE, 2007.

[9] Haldankar, Chaitali, and Sonia Kuwelkar. "Implementation of AES and blowfish algorithm." *International Journal of Research in Engineering and Technology* 3, no. 3 (2014): 143-146.

[10] Sangwan, Chetna, Chetna Bhardwaj, and Taruna Sikka. "VLSI Implementation of Advanced Encryption Standard." In *2012 Second International Conference on Advanced Computing & Communication Technologies*, pp. 412-418. IEEE, 2012.