IMPROVING AES ALGORITHM

Minor Project 2

Enrol. No. (s) - 20104013,20104025,20104059

Name of Student (s)- Harshit Chopra, Kartik Gupta, Ayush Sharma

Name of supervisor - Dr. Shardha Porwal

May - 2023

Submitted in partial fulfilment of the Degree of

Bachelor of Technology

in

Information Technology

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY UNIVERSITY, NOIDA

ACKNOWLEDGEMENT

We would like to place on record my deep sense of gratitude to Dr. Shardha Porwal, Designation, Jaypee Institute of Information Technology, India for her generous guidance, help and useful suggestions.

Name of Students (Enrollment) **Signature(s) of Students**

Harshit Chopra(20104013)

Kartik Gupta (20104025)

Ayush Sharma(20104059)

DECLARATION

We hereby declare that this submission is our own work and that, to the best of our knowledge and beliefs, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma from a university or other institute of higher learning, except where due acknowledgment has been made in the text.

Place: Jaypee Institute of Information Technology, Sector 62, Noida, Uttar Pradesh

Date:

Name: Harshit Chopra

Enrolment No.: 20104013

Name: Kartik Gupta

Enrolment No.: 20104025

Name: Ayush Sharma

Enrolment No.: 20104059

CERTIFICATE

This is to certify that the work titled “Improving AES encryption” submitted by Harshit Chopra, Kartik Gupta, and Ayush Sharma of B.Tech of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of any other degree or diploma.

Digital Signature of Supervisor :

Name of Supervisor : Dr Shardha Porwal

Designation : Asst. Prof. (Sr. Grade)

Date :

ABSTRACT

The most popular and commonly used symmetric cryptosystem is called Advanced Encryption Standard (AES). It was replaced by DES when NIST standardised it in 2000.

Four steps will be taken to process the key:

1. S.box lookup table-based transformation approach for a non-linear byte substitution
2. Cyclical shifting method for each row's key matrix in shift rows
3. Using the Galois Field and the matrix finite field GF(2^8), mix Column - Dot matrix operation with XOR.
4. Add Round Key - Round Key with State Data XOR Addition Operation

The shift rows step's circular motion moves slowly. Using array shift row mapping, which has an index value that is directly mapped to the index of the key state that should be placed, could make these circular operations faster. These actions will lessen the amount of manual circular movement.

Mix Column is a linear transformation technique. Each element of the state characters 28 is multiplied by the components of the multiplication matrix that were produced by the two four-term polynomials with the coefficient element of GF(2^8). It is possible to combine several transformations into one to cut down on superfluous stages and boost efficiency. Combining the sub bytes, move rows, and mix column steps could speed up this operation.

AES has some flaws when it comes to a range of cryptanalytic criteria, despite being immune to linear and differential assaults. An S-box written as a polynomial, for instance, should have a high algebraic degree. The AES S-box is a simple structure with a 254 algebraic degree and just 9 monomials. The fact that some parts of F2^8 have short iteration durations is another lax criteria for the AES S-box. In comparison to the ideal value of 16, the AES S-box transformation period is equivalent to 4, which is a relatively low value.

TABLE OF CONTENTS

ABBREVIATIONS AND NOMENCLATURE

AES: Advanced Encryption Standard

S-Box: Substitution box

IDE: Integrated development environment

INTRODUCTION

Rijndael, a contestant from the US National Institute of Standards and Technology, introduced the Advanced Encryption Standard (AES) in 2001. Data Encryption Standard (DES) ageing issue is resolved with AES encryption. Standard Rijndanel symmetric block cypher that can encrypt and decrypt plaintext blocks of 128 bits utilising keys of 128 bits, 192 bits, or 256 bits in size. The Rijndael cipher is appropriate for 8-bit and 32-bit processing and has a straightforward construction. There are several plaintext transformations in the cypher. The number of rounds to be executed depends on the key length. 128-bit keys require 10 rounds, 192-bit keys require 12 rounds, and 256-bit keys require 14 rounds. On finite fields GF(2^8), the AES arithmetic operations are addition, subtraction, multiplication, and division. Performance of AES operations depends on the length of the key. Sub Bytes, Shift Rows, Mix Column, and Add Round Key are the four different transformations that are used in each round. Every transformation generates a matrix output with the same dimension by taking any 16-byte block with a 4 x 4 matrix as input.

BACKGROUND STUDY

The authors show that a single random byte fault at the eighth round's AES input is sufficient to determine the block cypher key. Simulations show that the key can be precisely identified without the use of a brute-force search in the event of two incorrect ciphertext combinations. Additionally, the least mistake against AES has been used. The authors show that AES-192 can only be broken using two pairs of correct and defective ciphertexts, in contrast to AES-256, which may be broken using three pairs of accurate and flawed ciphertexts.

Wang claims that the AES S-box affine transformation period is 4, and it falls short of the maximum value of 16. According to study by Wang et al., the iterative period of the AES S-box demonstrates a short-period phenomenon, and all of the periods are smaller than 88. Murphy and Robshaw determined that there are just 9 terms in the algebraic expression of the AES S-box, making it a fairly simple algebraic expression.

Satoh improved AES by utilising a library of circuit CMOS standard cells. Ahmad employed a truth table-based combinational logic implementation on a Virtex II FPGA processor. Daemen et al. claim that adopting 32-bit data as the basic data unit improves the execution performance of AES. The AES-NI enhanced instruction set is used by Intel to significantly improve the AES algorithm. There are many alternative hardware implementations for AES optimization.

Shift row steps involve a circular movement procedure that moves slowly. We can make it better and speed up the execution. A linear transformation procedure is Mix Column. Different transformations can be combined into one to reduce extra steps and improve performance. S-box should be written as a polynomial with a high algebraic degree. Distribution of elements should be more balanced for the periodicity criterion.

REQUIREMENT ANALYSIS

An element of a state key matrix that executes a circular shift per row is arranged in a shift row. In each row, the circular shift length varies. These circling movements happen slowly.

Mix Column is a linear transformation technique. the elements of the multiplication matrix produced by the two polynomials with four terms each and a coefficient element of GF(2^8). are multiplied by 28 state characters, one for each element. It is possible to combine several transformations into one to cut down on superfluous stages and boost efficiency.

AES has some flaws when it comes to a range of cryptanalytic criteria, despite being immune to linear and differential assaults. An S-box written as a polynomial, for instance, should have a high algebraic degree. The AES S-box is a simple structure with a 254 algebraic degree and just 9 monomials. The fact that some parts of F(2^8) have short iteration durations is another lax criteria for the AES S-box. In comparison to the ideal value of 16, the AES S-box transformation period is equivalent to 4, which is a relatively low value.

DETAILED DESIGN

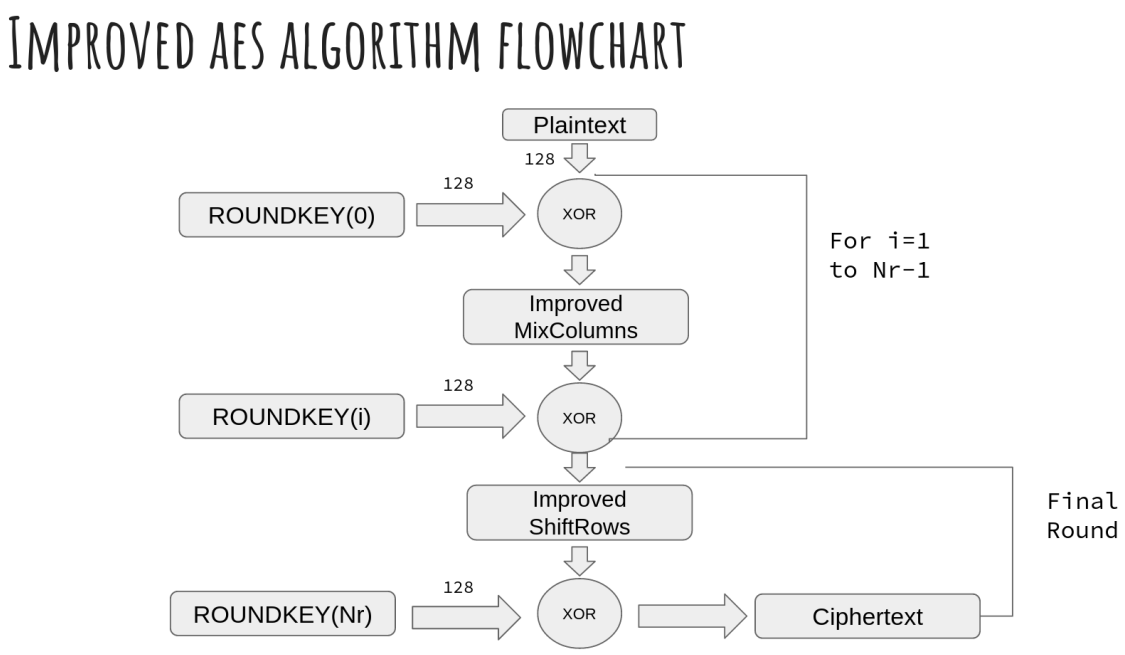


Fig. 1 (Flowchart)

1.Sub Bytes -Transformation process for a non-linear byte substitution using S.box lookup table

2. Shift Rows - Cyclical shifting process for key matrix in each row

3. Mix Column - Dot matrix operation combined with XOR using matrix finite field GF(2^8) and Galois Field.

4. Add Round Key - XOR addition operation for round key with state data

SubBytes are steps of byte-to-byte substitution. The table is 16x16 in size and includes hexadecimal characters in it. A given input key state byte is replaced by the hexadecimal.The key and plaintext characters must be converted to hexadecimal for the AES encryption and decryption processes to work properly.

A **Shift Row** is a way to arrange state key matrix components so that each row executes a circular shift. In each row, the circular shift length varies. Never is the front row relocated. The last element in the second row shifts the first element one space to the right. The last row moves three first components, and the third row moves two first elements to the right at the last element.

A linear transformation procedure is Mix Column. Each state character element was multiplied by the coefficient element of GF(28) in the multiplication matrix that was created from two four-term polynomials. A finite field matrix and a key state matrix are multiplied in the first section of MixColumn.

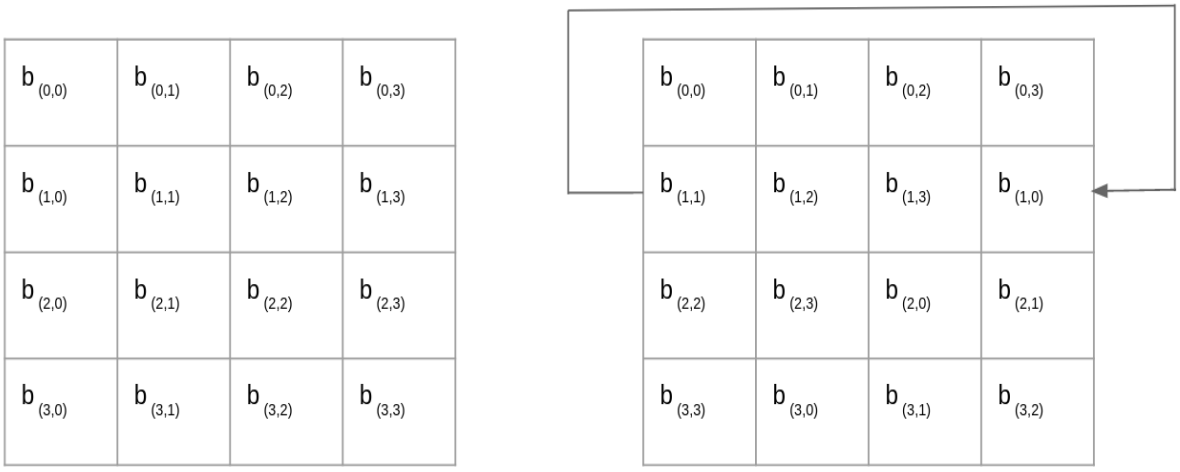
Instead of moving and rotating each element, array shift mapping improves the shift row transformation.

Mix improvement By consolidating multiple transformations into one, the column transformation procedure eliminates the need for the Sub Byte phase.

The AES S-box is fairly straightforward and has an algebraic degree of 254 with only 9 monomials. We developed a new affine transformation that improved S-box security. Since 256 is the only period, the distribution of F28's elements is more evenly distributed for the periodicity criteria. The new S-box is more resistant to potential algebraic attacks than the AES S-box since it has an ideal algebraic complexity of 255.

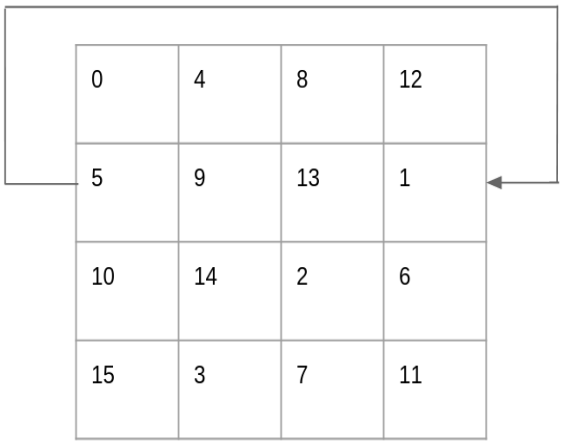
IMPLEMENTATION

An element of a state key matrix that executes a circular shift per row is arranged in a shift row. In each row, the circular shift length varies. Never is the front row relocated. Second row: at the last element, move the first element one space to the right. The last row moves three first items, and the third row moves two first elements to the right at the last element.



Using array shift row mapping, which has an index value that is directly mapped to the index of the key state that should be placed, could make these circular operations faster. These actions will lessen the amount of manual circular movement.

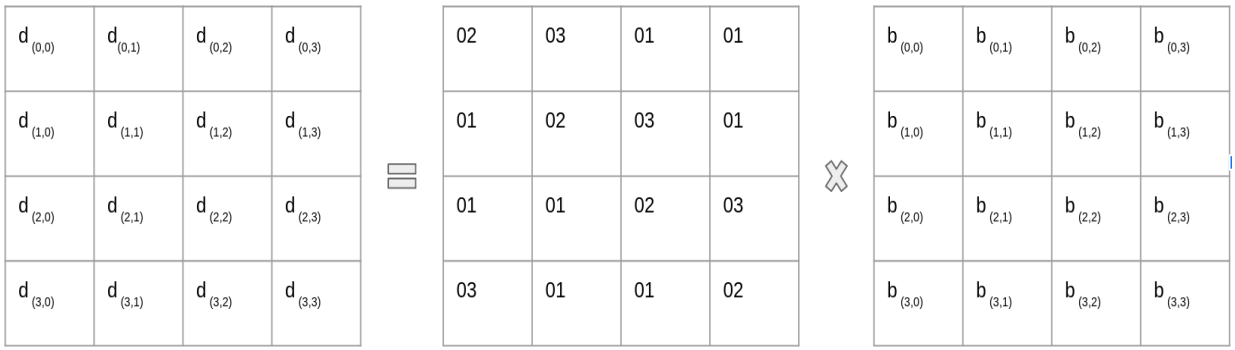
0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15



Array shift row value index for circular movement:

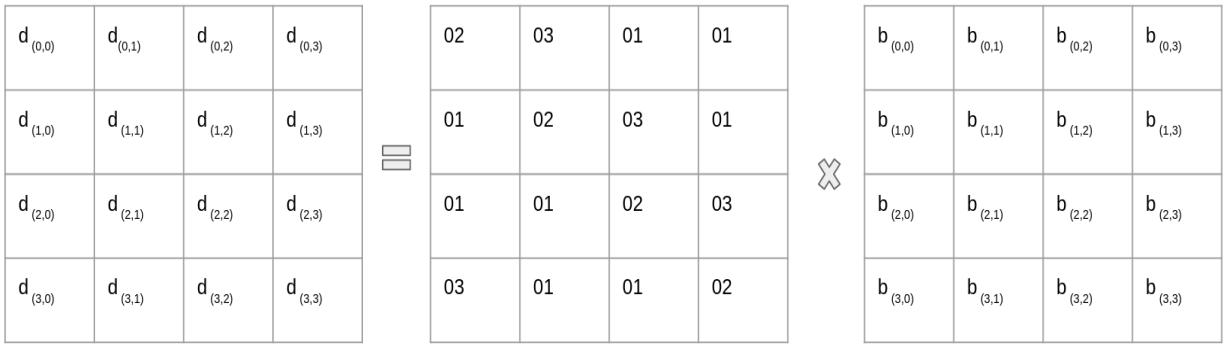
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	0	5	10	15	4	9	14	3	8	13	2	7	12	1	6	11

A linear transformation procedure is Mix Column. MixColumns operations are performed by the Rijndael cipher, along with the ShiftRows step, which is the primary source of diffusion in Rijndael. Each state character element was multiplied by the coefficient element of GF(2^8) in the multiplication matrix that was created from two four-term polynomials which have the coefficient element of GF(2^8).



This process could be made faster by combining the sub bytes, shift rows and mix column step.

Mix Columns:



Shift Rows :

$b_{(0,0)}$	$b_{(0,1)}$	$b_{(0,2)}$	$b_{(0,3)}$
$b_{(1,0)}$	$b_{(1,1)}$	$b_{(1,2)}$	$b_{(1,3)}$
$b_{(2,0)}$	$b_{(2,1)}$	$b_{(2,2)}$	$b_{(2,3)}$
$b_{(3,0)}$	$b_{(3,1)}$	$b_{(3,2)}$	$b_{(3,3)}$

=

$a_{(0,0)}$	$a_{(0,1)}$	$a_{(0,2)}$	$a_{(0,3)}$
$a_{(1,0)}$	$a_{(1,1)}$	$a_{(1,2)}$	$a_{(1,3)}$
$a_{(2,0)}$	$a_{(2,1)}$	$a_{(2,2)}$	$a_{(2,3)}$
$a_{(3,0)}$	$a_{(3,1)}$	$a_{(3,2)}$	$a_{(3,3)}$

Sub Bytes:

$a_{(0,0)}$	$a_{(0,1)}$	$a_{(0,2)}$	$a_{(0,3)}$
$a_{(1,0)}$	$a_{(1,1)}$	$a_{(1,2)}$	$a_{(1,3)}$
$a_{(2,0)}$	$a_{(2,1)}$	$a_{(2,2)}$	$a_{(2,3)}$
$a_{(3,0)}$	$a_{(3,1)}$	$a_{(3,2)}$	$a_{(3,3)}$

=

$S[t_{(0,0)}]$	$S[t_{(0,1)}]$	$S[t_{(0,2)}]$	$S[t_{(0,3)}]$
$S[t_{(1,0)}]$	$S[t_{(1,1)}]$	$S[t_{(1,2)}]$	$S[t_{(1,3)}]$
$S[t_{(2,0)}]$	$S[t_{(2,1)}]$	$S[t_{(2,2)}]$	$S[t_{(2,3)}]$
$S[t_{(3,0)}]$	$S[t_{(3,1)}]$	$S[t_{(3,2)}]$	$S[t_{(3,3)}]$

(where $S[t_{i,j}]$ is substitution of bytes

of a state with a byte from lookup table)

AES S-box

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

This process could be made faster by combining the sub bytes, shift rows and mix column step.

Mix Columns:

$d_{(0,0)}$	$d_{(0,1)}$	$d_{(0,2)}$	$d_{(0,3)}$	$02 * S[t_{(0,0)}]$	$03 * S[t_{(0,1)}]$	$01 * S[t_{(0,2)}]$	$01 * S[t_{(0,3)}]$
$d_{(1,0)}$	$d_{(1,1)}$	$d_{(1,2)}$	$d_{(1,3)}$	$01 * S[t_{(1,1)}]$	$02 * S[t_{(1,2)}]$	$03 * S[t_{(1,3)}]$	$01 * S[t_{(1,0)}]$
$d_{(2,0)}$	$d_{(2,1)}$	$d_{(2,2)}$	$d_{(2,3)}$	$01 * S[t_{(2,2)}]$	$01 * S[t_{(2,3)}]$	$02 * S[t_{(2,0)}]$	$03 * S[t_{(2,1)}]$
$d_{(3,0)}$	$d_{(3,1)}$	$d_{(3,2)}$	$d_{(3,3)}$	$03 * S[t_{(3,3)}]$	$01 * S[t_{(3,0)}]$	$01 * S[t_{(3,1)}]$	$02 * S[t_{(3,2)}]$

Multiplication of a Galois field by the Galois field for 1 results in the input not being changed at all. For multiplication by the Galois field for 2 the results are any one of 256 values and the same for multiplication by the Galois field 3.

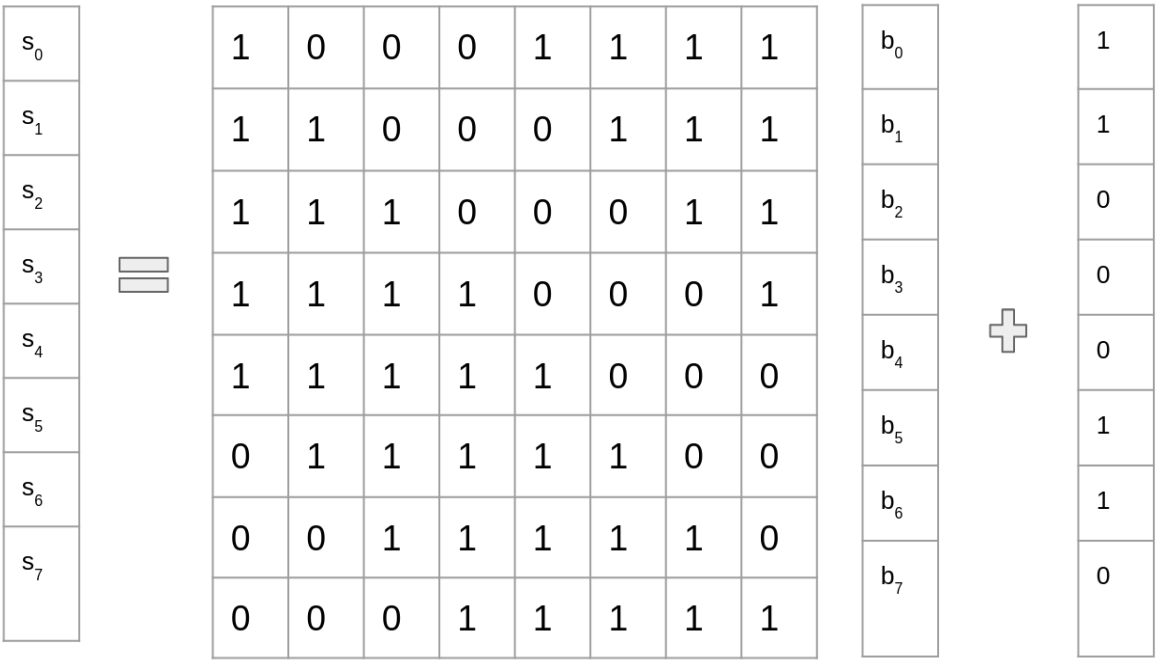
Multiply by 2:

```
0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18,0x1a,0x1c,0x1e,
0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0x3e,
0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58,0x5a,0x5c,0x5e,
0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78,0x7a,0x7c,0x7e,
0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98,0x9a,0x9c,0x9e,
0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8,0xba,0xbc,0xbe,
0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8,0xda,0xdc,0xde,
0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8,0xfa,0xfc,0xfe,
0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03,0x01,0x07,0x05,
0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23,0x21,0x27,0x25,
0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43,0x41,0x47,0x45,
0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63,0x61,0x67,0x65,
0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83,0x81,0x87,0x85,
0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3,0xa1,0xa7,0xa5,
0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3,0xc1,0xc7,0xc5,
0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3,0xe1,0xe7,0xe5
```

Multiply by 3:

```
0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14,0x17,0x12,0x11,
0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27,0x22,0x21,
0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74,0x77,0x72,0x71,
0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44,0x47,0x42,0x41,
0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4,0xd7,0xd2,0xd1,
0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4,0xe7,0xe2,0xe1,
0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4,0xb7,0xb2,0xb1,
0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84,0x87,0x82,0x81,
0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f,0x8c,0x89,0x8a,
0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf,0xbc,0xb9,0xba,
0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef,0xec,0xe9,0xea,
0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf,0xdc,0xd9,0xda,
0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f,0x4c,0x49,0x4a,
0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f,0x7c,0x79,0x7a,
0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f,0x2c,0x29,0x2a,
0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f,0x1c,0x19,0x1a
```

An 8-bit input, c, is translated into an 8-bit output, s = S(c), by the S-box. Polynomials over GF(2) are used to interpret both the input and the output. First, the input is transformed into its multiplicative inverse in Rijndael's finite field, $GF(2^8) = GF(2)[x]/(x8 + x4 + x3 + x + 1)$. The identity of zero is mapped to itself. The affine transformation shown below is then applied to the multiplicative inverse:



An 8-bit input, c, is mapped by the S-box to an 8-bit output, s = S(c). It is understood that the input and output are both polynomials over GF(2). In Rijndael's finite field, $GF(2^8) = GF(2)[x]/(x8 + x4 + x3 + x + 1)$, the input is first translated to its multiplicative inverse. Zero is mapped to itself as the identity. The multiplicative inverse is then changed using the affine transformation shown below:

Let F_q be a finite field with q elements. For $n \geq 2$, let GL(n, Fq) be the group of invertible $n \times n$ matrices with entries in Fq .

The order of GL(n, Fq) is : $|GL(n, Fq)| = \prod (qn - qk)$

Let A =

1	0	0	0	1	1	0	1
1	1	0	0	1	0	0	1
0	1	1	1	0	0	0	1
0	0	0	0	1	1	0	1
0	0	1	0	0	0	1	0
1	0	0	0	1	0	1	1
0	1	1	1	0	0	0	0
1	1	0	1	0	1	1	0

and,

α = 0xfe = (1, 1, 1, 1, 1, 1, 1, 0),

β = 0x3f = (0, 0, 1, 1, 1, 1, 1, 1).

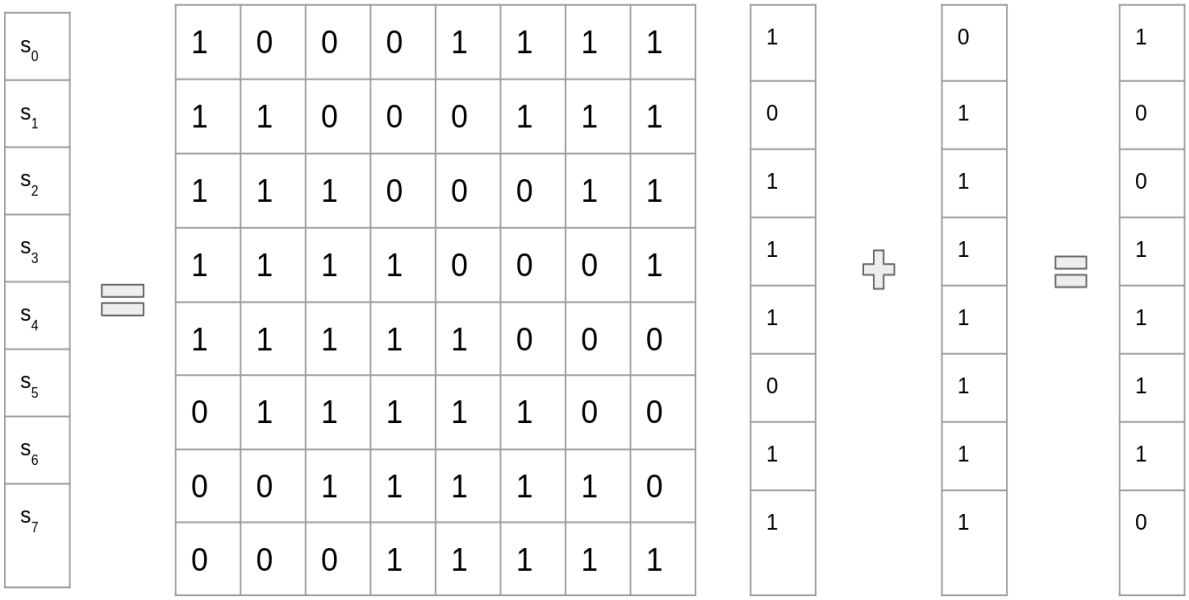
The new S-box is generated by the multivariate Boolean function S_N defined for x $\in$ G.F(2^8)

$$S_N = \begin{cases} (Ax+\alpha/Ax+\beta) & If \quad Ax+\beta \neq 0 \\ 0x01 & If \quad Ax+\beta = 0 \end{cases}$$

Example : SN (0xdd) = 0xed

0xdd = (1, 1, 0, 1, 1, 1, 0, 1) = (x7 , x6 , x5 , x4 , x3 , x2 , x1 , x0) .

Applying affine transformation: $Ax+\alpha$

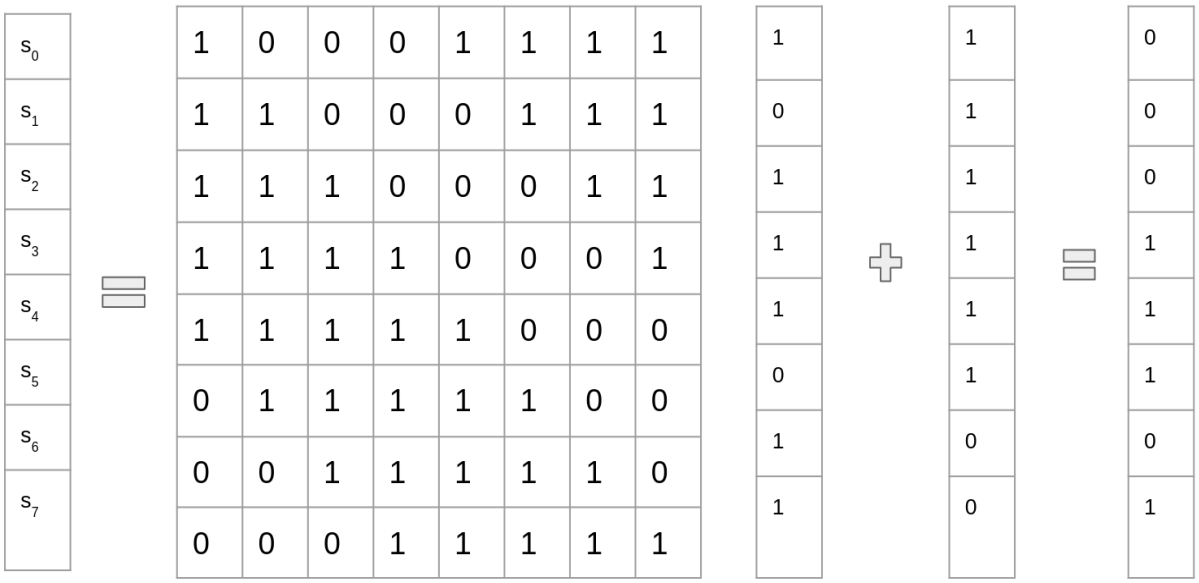


so Ax + α = (0, 1, 1, 1, 1, 0, 0, 1) = 0x79

Example : SN (0xdd) = 0xed

0xdd = (1, 1, 0, 1, 1, 1, 0, 1) = (x7 , x6 , x5 , x4 , x3 , x2 , x1 , x0) .

Applying affine transformation: $Ax+\beta$



so Ax + β = (1, 0, 1, 1, 1, 0, 0, 0) = 0xb8

S-box value:

S_N (0xdd) = (Ax + α)/(Ax + β)

=0x79/0xb8

= $(t^6 + t^5 + t^4 + t^3 + 1)/ (t^7 + t^5 + t^4 + t^3)$

= $t^7 + t^6 + t^5 + t^3 + t^2 + 1 \pmod{t^8 + t^4 + t^3 + t + 1}$

= (1, 1, 1, 0, 1, 1, 0, 1)

= 0xed

New S-Box :

```
unsigned s_box[256] =
    {
     0x36 ,0x94 ,0x89 ,0xcb ,0x77 ,0x96 ,0xd2 ,0x4b ,0x05 ,0xf7 ,0xab ,0xc5 ,0x6d ,0xa1 ,0xd6 ,0x5b
    ,0x61 ,0x91 ,0xe7 ,0xd0 ,0x1f ,0xa9 ,0x43 ,0x1d ,0x9b ,0xbe ,0xf4 ,0xb8 ,0x42 ,0x63 ,0x87 ,0xbb
    ,0x02 ,0x58 ,0xc3 ,0xac ,0xe4 ,0xe5 ,0xeb ,0xb3 ,0x83 ,0x70 ,0x64 ,0x20 ,0x57 ,0x08 ,0x60 ,0x85
    ,0x2f ,0x90 ,0x07 ,0xee ,0x23 ,0x33 ,0x81 ,0x12 ,0x14 ,0xea ,0x39 ,0x21 ,0x62 ,0xcd ,0x28 ,0x2e
    ,0x2c ,0xf6 ,0xdd ,0x25 ,0xbc ,0x11 ,0xa7 ,0xe6 ,0xfd ,0x53 ,0x98 ,0x9c ,0x38 ,0x1b ,0x5c ,0x54
    ,0x75 ,0x95 ,0x26 ,0x00 ,0x09 ,0x3b ,0x44 ,0x9d ,0x15 ,0x5d ,0x1c ,0x9a ,0x5f ,0xc9 ,0xa4 ,0x78
    ,0x5a ,0xf3 ,0x0b ,0x0c ,0xe9 ,0x0a ,0x06 ,0x3e ,0x71 ,0xe1 ,0xfa ,0xf5 ,0x7f ,0x65 ,0x19 ,0xdf
    ,0x8e ,0x32 ,0xfb ,0x74 ,0x50 ,0xd9 ,0x72 ,0x24 ,0x45 ,0x0f ,0x69 ,0x76 ,0xda ,0x41 ,0xb1 ,0xdb
    ,0x79 ,0x80 ,0x3a ,0x49 ,0xe8 ,0xbf ,0x73 ,0x16 ,0x18 ,0x8d ,0xce ,0xa3 ,0x0e ,0xc6 ,0xef ,0xe3
    ,0xd7 ,0x99 ,0x6e ,0x35 ,0xfc ,0xaf ,0xa2 ,0xc1 ,0xde ,0xc2 ,0x1e ,0xd1 ,0x6c ,0xf1 ,0xaa ,0x7e
    ,0x8c ,0x52 ,0xd4 ,0x4a ,0x7c ,0x93 ,0xf0 ,0xe2 ,0xd8 ,0x66 ,0x04 ,0x9e ,0x84 ,0x3c ,0x13 ,0xae
    ,0x86 ,0x88 ,0xa5 ,0x68 ,0xd3 ,0x37 ,0x3d ,0x56 ,0x6a ,0x5e ,0x7a ,0xad ,0xc8 ,0xb2 ,0x40 ,0x67
    ,0x0d ,0xb7 ,0x46 ,0x7d ,0xa6 ,0x82 ,0x6b ,0x3f ,0x34 ,0x22 ,0xb0 ,0xc0 ,0x29 ,0x4e ,0x59 ,0x7b
    ,0xc7 ,0x31 ,0xba ,0x47 ,0xfe ,0xc4 ,0xd5 ,0xe0 ,0x92 ,0xb9 ,0x10 ,0xa0 ,0x8b ,0xed ,0x55 ,0x97
    ,0xca ,0x1a ,0xf9 ,0x2a ,0xcc ,0xf2 ,0x4c ,0x51 ,0x03 ,0x30 ,0x4d ,0xf8 ,0xb4 ,0xbd ,0xcf ,0x48
    ,0xec ,0x2b ,0x9f ,0xff ,0x27 ,0x17 ,0xb6 ,0x8f ,0x8a ,0xb5 ,0x01 ,0xa8 ,0x6f ,0x4f ,0xdc ,0x2d
    };
```

Source Code of AES:

```
#include <iostream>

#include <string.h>

#include <chrono>

using namespace std;

using namespace std::chrono;


unsigned s_box[256] =

   {

    0x63 ,0x7c ,0x77 ,0x7b ,0xf2 ,0x6b ,0x6f ,0xc5 ,0x30 ,0x01 ,0x67 ,0x2b ,0xfe ,0xd7 ,0xab ,0x76

   ,0xca ,0x82 ,0xc9 ,0x7d ,0xfa ,0x59 ,0x47 ,0xf0 ,0xad ,0xd4 ,0xa2 ,0xaf ,0x9c ,0xa4 ,0x72 ,0xc0

   ,0xb7 ,0xfd ,0x93 ,0x26 ,0x36 ,0x3f ,0xf7 ,0xcc ,0x34 ,0xa5 ,0xe5 ,0xf1 ,0x71 ,0xd8 ,0x31 ,0x15

   ,0x04 ,0xc7 ,0x23 ,0xc3 ,0x18 ,0x96 ,0x05 ,0x9a ,0x07 ,0x12 ,0x80 ,0xe2 ,0xeb ,0x27 ,0xb2 ,0x75

   ,0x09 ,0x83 ,0x2c ,0x1a ,0x1b ,0x6e ,0x5a ,0xa0 ,0x52 ,0x3b ,0xd6 ,0xb3 ,0x29 ,0xe3 ,0x2f ,0x84

   ,0x53 ,0xd1 ,0x00 ,0xed ,0x20 ,0xfc ,0xb1 ,0x5b ,0x6a ,0xcb ,0xbe ,0x39 ,0x4a ,0x4c ,0x58 ,0xcf

   ,0xd0 ,0xef ,0xaa ,0xfb ,0x43 ,0x4d ,0x33 ,0x85 ,0x45 ,0xf9 ,0x02 ,0x7f ,0x50 ,0x3c ,0x9f ,0xa8

   ,0x51 ,0xa3 ,0x40 ,0x8f ,0x92 ,0x9d ,0x38 ,0xf5 ,0xbc ,0xb6 ,0xda ,0x21 ,0x10 ,0xff ,0xf3 ,0xd2

   ,0xcd ,0x0c ,0x13 ,0xec ,0x5f ,0x97 ,0x44 ,0x17 ,0xc4 ,0xa7 ,0x7e ,0x3d ,0x64 ,0x5d ,0x19 ,0x73

   ,0x60 ,0x81 ,0x4f ,0xdc ,0x22 ,0x2a ,0x90 ,0x88 ,0x46 ,0xee ,0xb8 ,0x14 ,0xde ,0x5e ,0x0b ,0xdb

   ,0xe0 ,0x32 ,0x3a ,0x0a ,0x49 ,0x06 ,0x24 ,0x5c ,0xc2 ,0xd3 ,0xac ,0x62 ,0x91 ,0x95 ,0xe4 ,0x79

   ,0xe7 ,0xc8 ,0x37 ,0x6d ,0x8d ,0xd5 ,0x4e ,0xa9 ,0x6c ,0x56 ,0xf4 ,0xea ,0x65 ,0x7a ,0xae ,0x08

   ,0xba ,0x78 ,0x25 ,0x2e ,0x1c ,0xa6 ,0xb4 ,0xc6 ,0xe8 ,0xdd ,0x74 ,0x1f ,0x4b ,0xbd ,0x8b ,0x8a
```

```
,0x70 ,0x3e ,0xb5 ,0x66 ,0x48 ,0x03 ,0xf6 ,0x0e ,0x61 ,0x35 ,0x57 ,0xb9 ,0x86 ,0xc1 ,0x1d ,0x9e
,0xe1 ,0xf8 ,0x98 ,0x11 ,0x69 ,0xd9 ,0x8e ,0x94 ,0x9b ,0x1e ,0x87 ,0xe9 ,0xce ,0x55 ,0x28 ,0xdf
,0x8c ,0xa1 ,0x89 ,0x0d ,0xbf ,0xe6 ,0x42 ,0x68 ,0x41 ,0x99 ,0x2d ,0x0f ,0xb0 ,0x54 ,0xbb ,0x16
};

unsigned mul2[]{
    0x00, 0x02, 0x04, 0x06, 0x08, 0x0a, 0x0c, 0x0e, 0x10, 0x12, 0x14, 0x16, 0x18, 0x1a, 0x1c, 0x1e,
    0x20, 0x22, 0x24, 0x26, 0x28, 0x2a, 0x2c, 0x2e, 0x30, 0x32, 0x34, 0x36, 0x38, 0x3a, 0x3c, 0x3e,
    0x40, 0x42, 0x44, 0x46, 0x48, 0x4a, 0x4c, 0x4e, 0x50, 0x52, 0x54, 0x56, 0x58, 0x5a, 0x5c, 0x5e,
    0x60, 0x62, 0x64, 0x66, 0x68, 0x6a, 0x6c, 0x6e, 0x70, 0x72, 0x74, 0x76, 0x78, 0x7a, 0x7c, 0x7e,
    0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c, 0x8e, 0x90, 0x92, 0x94, 0x96, 0x98, 0x9a, 0x9c, 0x9e,
    0xa0, 0xa2, 0xa4, 0xa6, 0xa8, 0xaa, 0xac, 0xae, 0xb0, 0xb2, 0xb4, 0xb6, 0xb8, 0xba, 0xbc, 0xbe,
    0xc0, 0xc2, 0xc4, 0xc6, 0xc8, 0xca, 0xcc, 0xce, 0xd0, 0xd2, 0xd4, 0xd6, 0xd8, 0xda, 0xdc, 0xde,
    0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf0, 0xf2, 0xf4, 0xf6, 0xf8, 0xfa, 0xfc, 0xfe,
    0x1b, 0x19, 0x1f, 0x1d, 0x13, 0x11, 0x17, 0x15, 0x0b, 0x09, 0x0f, 0x0d, 0x03, 0x01, 0x07, 0x05,
    0x3b, 0x39, 0x3f, 0x3d, 0x33, 0x31, 0x37, 0x35, 0x2b, 0x29, 0x2f, 0x2d, 0x23, 0x21, 0x27, 0x25,
    0x5b, 0x59, 0x5f, 0x5d, 0x53, 0x51, 0x57, 0x55, 0x4b, 0x49, 0x4f, 0x4d, 0x43, 0x41, 0x47, 0x45,
    0x7b, 0x79, 0x7f, 0x7d, 0x73, 0x71, 0x77, 0x75, 0x6b, 0x69, 0x6f, 0x6d, 0x63, 0x61, 0x67, 0x65,
    0x9b, 0x99, 0x9f, 0x9d, 0x93, 0x91, 0x97, 0x95, 0x8b, 0x89, 0x8f, 0x8d, 0x83, 0x81, 0x87, 0x85,
    0xbb, 0xb9, 0xbf, 0xbd, 0xb3, 0xb1, 0xb7, 0xb5, 0xab, 0xa9, 0xaf, 0xad, 0xa3, 0xa1, 0xa7, 0xa5,
    0xdb, 0xd9, 0xdf, 0xdd, 0xd3, 0xd1, 0xd7, 0xd5, 0xcb, 0xc9, 0xcf, 0xcd, 0xc3, 0xc1, 0xc7, 0xc5,
    0xfb, 0xf9, 0xff, 0xfd, 0xf3, 0xf1, 0xf7, 0xf5, 0xeb, 0xe9, 0xef, 0xed, 0xe3, 0xe1, 0xe7, 0xe5};
```

```
unsigned char mul3[]{

    0x00, 0x03, 0x06, 0x05, 0x0c, 0x0f, 0x0a, 0x09, 0x18, 0x1b, 0x1e, 0x1d, 0x14, 0x17, 0x12, 0x11,
    0x30, 0x33, 0x36, 0x35, 0x3c, 0x3f, 0x3a, 0x39, 0x28, 0x2b, 0x2e, 0x2d, 0x24, 0x27, 0x22, 0x21,
    0x60, 0x63, 0x66, 0x65, 0x6c, 0x6f, 0x6a, 0x69, 0x78, 0x7b, 0x7e, 0x7d, 0x74, 0x77, 0x72, 0x71,
    0x50, 0x53, 0x56, 0x55, 0x5c, 0x5f, 0x5a, 0x59, 0x48, 0x4b, 0x4e, 0x4d, 0x44, 0x47, 0x42, 0x41,
    0xc0, 0xc3, 0xc6, 0xc5, 0xcc, 0xcf, 0xca, 0xc9, 0xd8, 0xdb, 0xde, 0xdd, 0xd4, 0xd7, 0xd2, 0xd1,
    0xf0, 0xf3, 0xf6, 0xf5, 0xfc, 0xff, 0xfa, 0xf9, 0xe8, 0xeb, 0xee, 0xed, 0xe4, 0xe7, 0xe2, 0xe1,
    0xa0, 0xa3, 0xa6, 0xa5, 0xac, 0xaf, 0xaa, 0xa9, 0xb8, 0xbb, 0xbe, 0xbd, 0xb4, 0xb7, 0xb2, 0xb1,
    0x90, 0x93, 0x96, 0x95, 0x9c, 0x9f, 0x9a, 0x99, 0x88, 0x8b, 0x8e, 0x8d, 0x84, 0x87, 0x82, 0x81,
    0x9b, 0x98, 0x9d, 0x9e, 0x97, 0x94, 0x91, 0x92, 0x83, 0x80, 0x85, 0x86, 0x8f, 0x8c, 0x89, 0x8a,
    0xab, 0xa8, 0xad, 0xae, 0xa7, 0xa4, 0xa1, 0xa2, 0xb3, 0xb0, 0xb5, 0xb6, 0xbf, 0xbc, 0xb9, 0xba,
    0xfb, 0xf8, 0xfd, 0xfe, 0xf7, 0xf4, 0xf1, 0xf2, 0xe3, 0xe0, 0xe5, 0xe6, 0xef, 0xec, 0xe9, 0xea,
    0xcb, 0xc8, 0xcd, 0xce, 0xc7, 0xc4, 0xc1, 0xc2, 0xd3, 0xd0, 0xd5, 0xd6, 0xdf, 0xdc, 0xd9, 0xda,
    0x5b, 0x58, 0x5d, 0x5e, 0x57, 0x54, 0x51, 0x52, 0x43, 0x40, 0x45, 0x46, 0x4f, 0x4c, 0x49, 0x4a,
    0x6b, 0x68, 0x6d, 0x6e, 0x67, 0x64, 0x61, 0x62, 0x73, 0x70, 0x75, 0x76, 0x7f, 0x7c, 0x79, 0x7a,
    0x3b, 0x38, 0x3d, 0x3e, 0x37, 0x34, 0x31, 0x32, 0x23, 0x20, 0x25, 0x26, 0x2f, 0x2c, 0x29, 0x2a,
    0x0b, 0x08, 0x0d, 0x0e, 0x07, 0x04, 0x01, 0x02, 0x13, 0x10, 0x15, 0x16, 0x1f, 0x1c, 0x19, 0x1a};

unsigned char rcon[256] = {
0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
```

```
0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d
};
void KeyExpansionCore(unsigned char *in, unsigned char i) {
unsigned int *q = (unsigned int *)in;
*q = (*q >> 8) | ((*q & 0xff) <<24);          // left shift operator "<<" and Right shift operator ">>"
in[0] = s_box[in[0]]; in[1] = s_box[in[1]];
in[2] = s_box[in[2]]; in[3] = s_box[in[3]];
in[0] ^=rcon[i];
```

```
}
void KeyExpansion(unsigned char *inputKey, unsigned char *expandedKeys) {

for (int i = 0; i < 16; i++)
{
   expandedKeys[i] = inputKey[i];
}

int bytesGenerated = 16;     // We've generated 16bytes so far (by copying the original key)
int rconIteration = 1;       // Rcon Iteration begins at 1
unsigned char temp[4];       // Temporary storage for core

while (bytesGenerated < 176)
{
   for (int i = 0; i < 4; i++)
   {
      temp[i] = expandedKeys[i+ bytesGenerated -4];
   }

   if(bytesGenerated % 16 == 0){
      KeyExpansionCore(temp,rconIteration);
```

```
            rconIteration++;
        }

        for (unsigned char a = 0; a < 4; a++)
        {
            expandedKeys[bytesGenerated] = expandedKeys[bytesGenerated - 16] ^ temp[a];
            bytesGenerated++;
        }

    }

}

void SubBytes(unsigned char *state)
{
    for (int i = 0; i < 16; i++)
    {
        state[i] = s_box[state[i]];
    }
}
```

```
void ShiftRows(unsigned char *state)
{
   int a[4];
   int b[4];
   int c[4];
   int d[4];
   int count = 1;
   for (int i = 0; i < 4; i++)
   {
      switch(count)
      {       case 1 :
         for (int j = 0; j < 4; j++)
         {
             a[j] = state[i+(j*4)];
         }
      break;
      case 2 :
         for (int j = 0; j < 4; j++)
         {
             b[j] = state[i+(j*4)];
         }
       break;
```

```
        case 3 :
            for (int j = 0; j < 4; j++)
            {
                c[j] = state[i+(j*4)];
            }

        break;
         case 4 :
            for (int j = 0; j < 4; j++)
            {
                d[j] = state[i+(j*4)];
            }

        break;
        default :
            cout<<"Default executed";
         break;

        }
    count++;

}
```

```
int length_b = sizeof(b)/sizeof(b[0]);
     for(int i = 0; i < 1; i++){
    int j, first;

first = b[0];

for(j = 0; j < length_b-1; j++)
{

    b[j] = b[j+1];
 }              b[j] = first;
}

int length_c = sizeof(c)/sizeof(c[0]);
for(int i = 0; i < 2; i++){
int j, first;
first = c[0];

for(j = 0; j < length_c-1; j++)
{
     c[j] = c[j+1];
 }
```

```
    c[j] = first;
}

int length_d = sizeof(d)/sizeof(d[0]);

for(int i = 0; i < 3; i++){

int j, first;
first = d[0];

for(j = 0; j < length_d-1; j++)
{
     d[j] = d[j+1];
 }
     d[j] = first;
}

unsigned char tmp[16];

tmp[0] = a[0];
tmp[1] = b[0];
tmp[2] = c[0];
```

```
tmp[3] = d[0];


tmp[4] = a[1];

tmp[5] = b[1];

tmp[6] = c[1];

tmp[7] = d[1];


tmp[8] =  a[2];

tmp[9] =  b[2];

tmp[10] = c[2];

tmp[11] = d[2];


tmp[12] = a[3];

tmp[13] = b[3];

tmp[14] = c[3];

tmp[15] = d[3];


for (int i = 0; i < 16; i++)

{

   state[i] = tmp[i];

}
```

```
}

void MixColumns(unsigned char *state) {
 unsigned char tmp[16];

 tmp[0] = (unsigned char)(mul2[state[0]] ^ mul3[state[1]] ^ state[2] ^ state[3]);
 tmp[1] = (unsigned char)(state[0] ^ mul2[state[1]] ^ mul3[state[2]] ^ state[3]);
 tmp[2] = (unsigned char)(state[0] ^ state[1] ^ mul2[state[2]] ^ mul3[state[3]]);
 tmp[3] = (unsigned char)(mul3[state[0]] ^ state[1] ^ state[2] ^ mul2[state[3]]);

 tmp[4] = (unsigned char)(mul2[state[4]] ^ mul3[state[5]] ^ state[6] ^ state[7]);
 tmp[5] = (unsigned char)(state[4] ^ mul2[state[5]] ^ mul3[state[6]] ^ state[7]);
 tmp[6] = (unsigned char)(state[4] ^ state[5] ^ mul2[state[6]] ^ mul3[state[7]]);
 tmp[7] = (unsigned char)(mul3[state[4]] ^ state[5] ^ state[6] ^ mul2[state[7]]);

 tmp[8] = (unsigned char)(mul2[state[8]] ^ mul3[state[9]] ^ state[10] ^ state[11]);
 tmp[9] = (unsigned char)(state[8] ^ mul2[state[9]] ^ mul3[state[10]] ^ state[11]);
 tmp[10] = (unsigned char)(state[8] ^ state[9] ^ mul2[state[10]] ^ mul3[state[11]]);
 tmp[11] = (unsigned char)(mul3[state[8]] ^ state[9] ^ state[10] ^ mul2[state[11]]);

 tmp[12] = (unsigned char)(mul2[state[12]] ^ mul3[state[13]] ^ state[14] ^ state[15]);
 tmp[13] = (unsigned char)(state[12] ^ mul2[state[13]] ^ mul3[state[14]] ^ state[15]);
```

```
 tmp[14] = (unsigned char)(state[12] ^ state[13] ^ mul2[state[14]] ^ mul3[state[15]]);
 tmp[15] = (unsigned char)(mul3[state[12]] ^ state[13] ^ state[14] ^ mul2[state[15]]);

 for (int i = 0; i < 16; i++)
 {
   state[i] = tmp[i];
 }

}
void AddRoundKey(unsigned char *state, unsigned char *roundKey)
{

   for (int i = 0; i < 16; i++)
   {
      state[i] ^= roundKey[i];
   }
}

void AES_Encrypt(unsigned char *message, unsigned char *key)
{
   unsigned char state[16];
   for (int i = 0; i < 16; i++)
```

```
{
    state[i] = message[i];
}

int numberOfRounds = 9;


unsigned char expandedKey[176];
KeyExpansion(key, expandedKey);


AddRoundKey(state, key); // Whitening/AddRoundKey

for (int i = 0; i < numberOfRounds; i++)
{
    SubBytes(state);
    ShiftRows(state);
    MixColumns(state);
    AddRoundKey(state, expandedKey + (16 * (i+1)));
}
```

```
    SubBytes(state);
    ShiftRows(state);
    AddRoundKey(state, expandedKey + 160);



    for (int i = 0; i < 16; i++)
    {
        message[i] = state[i];
    }

}

void PrintHex(unsigned char x){
    if(x /16 <10) cout<<(char)((x/16) + '0');
    if(x /16 >=10) cout<<(char)((x/16-10) + 'A');

    if(x % 16 <10) cout<<(char)((x%16) + '0');
    if(x % 16 >=10) cout<<(char)((x%16 -10) + 'A');
}

int main()
```

{

int timetime=0;

for(int count=0;count<9999;count++){

auto start = high_resolution_clock::now();

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

```
2a b0 81 f4 32 f4 72 0e f5 8c 18 97 39 56 ca 3a c3 01 8a 76 c6 43 42 81 08 36 27 31 bf 1c ec 17 ab 09
17 b3 5b 6a a0 1e 98 0a 30 a2 99 f7 3c c9 7e be 38 d4 28 5f 05 13 c8 51 16 15 ec 50 91 29 c1 f6 fc 4c
37 48 24 56 61 23 66 37 08 b0 42 17 42 14 89 57 73 7a 60 1a ee a6 9e e3 e8 9d e5 ea 39 02 00 8c 02 f3
c8 18 ca d7 d0 65 f6 c9 6a 6c 46 f4 7b c2 ";

    unsigned char key[16] =

      {

         1, 2, 3, 4,

         5, 6, 7, 8,

         9, 10, 11, 12,

         13, 14, 15, 16};



  int originalLen = strlen((const char *)message);

  int lenOfPaddedMessage = originalLen;


  if(lenOfPaddedMessage % 16 !=0){

     lenOfPaddedMessage = (lenOfPaddedMessage / 16 + 1) * 16;

  }


  unsigned char *paddedMessage = new unsigned char[lenOfPaddedMessage];

  for (int i = 0; i < lenOfPaddedMessage; i++)

  {

     if(i >= originalLen) paddedMessage[i] = 0;
```

```
        else paddedMessage[i] = message[i];
    }

for (int i = 0; i < lenOfPaddedMessage; i+=16)
{
    AES_Encrypt(paddedMessage+i, key);

}

delete[] paddedMessage;
auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    timetime=timetime+int(duration.count());
    }

    timetime=timetime/9999;
    cout<<"\n\nAverage Time: "<<timetime<<endl;
    return 0;
}
```

Source code of Improved AES:

```
#include <iostream>

#include <string.h>

#include <chrono>

using namespace std;

using namespace std::chrono;


unsigned s_box[256] =

   {

    0x63 ,0x7c ,0x77 ,0x7b ,0xf2 ,0x6b ,0x6f ,0xc5 ,0x30 ,0x01 ,0x67 ,0x2b ,0xfe ,0xd7 ,0xab ,0x76

   ,0xca ,0x82 ,0xc9 ,0x7d ,0xfa ,0x59 ,0x47 ,0xf0 ,0xad ,0xd4 ,0xa2 ,0xaf ,0x9c ,0xa4 ,0x72 ,0xc0

   ,0xb7 ,0xfd ,0x93 ,0x26 ,0x36 ,0x3f ,0xf7 ,0xcc ,0x34 ,0xa5 ,0xe5 ,0xf1 ,0x71 ,0xd8 ,0x31 ,0x15

   ,0x04 ,0xc7 ,0x23 ,0xc3 ,0x18 ,0x96 ,0x05 ,0x9a ,0x07 ,0x12 ,0x80 ,0xe2 ,0xeb ,0x27 ,0xb2 ,0x75

   ,0x09 ,0x83 ,0x2c ,0x1a ,0x1b ,0x6e ,0x5a ,0xa0 ,0x52 ,0x3b ,0xd6 ,0xb3 ,0x29 ,0xe3 ,0x2f ,0x84

   ,0x53 ,0xd1 ,0x00 ,0xed ,0x20 ,0xfc ,0xb1 ,0x5b ,0x6a ,0xcb ,0xbe ,0x39 ,0x4a ,0x4c ,0x58 ,0xcf

   ,0xd0 ,0xef ,0xaa ,0xfb ,0x43 ,0x4d ,0x33 ,0x85 ,0x45 ,0xf9 ,0x02 ,0x7f ,0x50 ,0x3c ,0x9f ,0xa8

   ,0x51 ,0xa3 ,0x40 ,0x8f ,0x92 ,0x9d ,0x38 ,0xf5 ,0xbc ,0xb6 ,0xda ,0x21 ,0x10 ,0xff ,0xf3 ,0xd2

   ,0xcd ,0x0c ,0x13 ,0xec ,0x5f ,0x97 ,0x44 ,0x17 ,0xc4 ,0xa7 ,0x7e ,0x3d ,0x64 ,0x5d ,0x19 ,0x73

   ,0x60 ,0x81 ,0x4f ,0xdc ,0x22 ,0x2a ,0x90 ,0x88 ,0x46 ,0xee ,0xb8 ,0x14 ,0xde ,0x5e ,0x0b ,0xdb

   ,0xe0 ,0x32 ,0x3a ,0x0a ,0x49 ,0x06 ,0x24 ,0x5c ,0xc2 ,0xd3 ,0xac ,0x62 ,0x91 ,0x95 ,0xe4 ,0x79

   ,0xe7 ,0xc8 ,0x37 ,0x6d ,0x8d ,0xd5 ,0x4e ,0xa9 ,0x6c ,0x56 ,0xf4 ,0xea ,0x65 ,0x7a ,0xae ,0x08

   ,0xba ,0x78 ,0x25 ,0x2e ,0x1c ,0xa6 ,0xb4 ,0xc6 ,0xe8 ,0xdd ,0x74 ,0x1f ,0x4b ,0xbd ,0x8b ,0x8a
```

```
,0x70 ,0x3e ,0xb5 ,0x66 ,0x48 ,0x03 ,0xf6 ,0x0e ,0x61 ,0x35 ,0x57 ,0xb9 ,0x86 ,0xc1 ,0x1d ,0x9e
,0xe1 ,0xf8 ,0x98 ,0x11 ,0x69 ,0xd9 ,0x8e ,0x94 ,0x9b ,0x1e ,0x87 ,0xe9 ,0xce ,0x55 ,0x28 ,0xdf
,0x8c ,0xa1 ,0x89 ,0x0d ,0xbf ,0xe6 ,0x42 ,0x68 ,0x41 ,0x99 ,0x2d ,0x0f ,0xb0 ,0x54 ,0xbb ,0x16
};

unsigned mul2[]{
0x00, 0x02, 0x04, 0x06, 0x08, 0x0a, 0x0c, 0x0e, 0x10, 0x12, 0x14, 0x16, 0x18, 0x1a, 0x1c, 0x1e,
0x20, 0x22, 0x24, 0x26, 0x28, 0x2a, 0x2c, 0x2e, 0x30, 0x32, 0x34, 0x36, 0x38, 0x3a, 0x3c, 0x3e,
0x40, 0x42, 0x44, 0x46, 0x48, 0x4a, 0x4c, 0x4e, 0x50, 0x52, 0x54, 0x56, 0x58, 0x5a, 0x5c, 0x5e,
0x60, 0x62, 0x64, 0x66, 0x68, 0x6a, 0x6c, 0x6e, 0x70, 0x72, 0x74, 0x76, 0x78, 0x7a, 0x7c, 0x7e,
0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c, 0x8e, 0x90, 0x92, 0x94, 0x96, 0x98, 0x9a, 0x9c, 0x9e,
0xa0, 0xa2, 0xa4, 0xa6, 0xa8, 0xaa, 0xac, 0xae, 0xb0, 0xb2, 0xb4, 0xb6, 0xb8, 0xba, 0xbc, 0xbe,
0xc0, 0xc2, 0xc4, 0xc6, 0xc8, 0xca, 0xcc, 0xce, 0xd0, 0xd2, 0xd4, 0xd6, 0xd8, 0xda, 0xdc, 0xde,
0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf0, 0xf2, 0xf4, 0xf6, 0xf8, 0xfa, 0xfc, 0xfe,
0x1b, 0x19, 0x1f, 0x1d, 0x13, 0x11, 0x17, 0x15, 0x0b, 0x09, 0x0f, 0x0d, 0x03, 0x01, 0x07, 0x05,
0x3b, 0x39, 0x3f, 0x3d, 0x33, 0x31, 0x37, 0x35, 0x2b, 0x29, 0x2f, 0x2d, 0x23, 0x21, 0x27, 0x25,
0x5b, 0x59, 0x5f, 0x5d, 0x53, 0x51, 0x57, 0x55, 0x4b, 0x49, 0x4f, 0x4d, 0x43, 0x41, 0x47, 0x45,
0x7b, 0x79, 0x7f, 0x7d, 0x73, 0x71, 0x77, 0x75, 0x6b, 0x69, 0x6f, 0x6d, 0x63, 0x61, 0x67, 0x65,
0x9b, 0x99, 0x9f, 0x9d, 0x93, 0x91, 0x97, 0x95, 0x8b, 0x89, 0x8f, 0x8d, 0x83, 0x81, 0x87, 0x85,
0xbb, 0xb9, 0xbf, 0xbd, 0xb3, 0xb1, 0xb7, 0xb5, 0xab, 0xa9, 0xaf, 0xad, 0xa3, 0xa1, 0xa7, 0xa5,
0xdb, 0xd9, 0xdf, 0xdd, 0xd3, 0xd1, 0xd7, 0xd5, 0xcb, 0xc9, 0xcf, 0xcd, 0xc3, 0xc1, 0xc7, 0xc5,
0xfb, 0xf9, 0xff, 0xfd, 0xf3, 0xf1, 0xf7, 0xf5, 0xeb, 0xe9, 0xef, 0xed, 0xe3, 0xe1, 0xe7, 0xe5};
```

```
unsigned char mul3[]{

    0x00, 0x03, 0x06, 0x05, 0x0c, 0x0f, 0x0a, 0x09, 0x18, 0x1b, 0x1e, 0x1d, 0x14, 0x17, 0x12, 0x11,
    0x30, 0x33, 0x36, 0x35, 0x3c, 0x3f, 0x3a, 0x39, 0x28, 0x2b, 0x2e, 0x2d, 0x24, 0x27, 0x22, 0x21,
    0x60, 0x63, 0x66, 0x65, 0x6c, 0x6f, 0x6a, 0x69, 0x78, 0x7b, 0x7e, 0x7d, 0x74, 0x77, 0x72, 0x71,
    0x50, 0x53, 0x56, 0x55, 0x5c, 0x5f, 0x5a, 0x59, 0x48, 0x4b, 0x4e, 0x4d, 0x44, 0x47, 0x42, 0x41,
    0xc0, 0xc3, 0xc6, 0xc5, 0xcc, 0xcf, 0xca, 0xc9, 0xd8, 0xdb, 0xde, 0xdd, 0xd4, 0xd7, 0xd2, 0xd1,
    0xf0, 0xf3, 0xf6, 0xf5, 0xfc, 0xff, 0xfa, 0xf9, 0xe8, 0xeb, 0xee, 0xed, 0xe4, 0xe7, 0xe2, 0xe1,
    0xa0, 0xa3, 0xa6, 0xa5, 0xac, 0xaf, 0xaa, 0xa9, 0xb8, 0xbb, 0xbe, 0xbd, 0xb4, 0xb7, 0xb2, 0xb1,
    0x90, 0x93, 0x96, 0x95, 0x9c, 0x9f, 0x9a, 0x99, 0x88, 0x8b, 0x8e, 0x8d, 0x84, 0x87, 0x82, 0x81,
    0x9b, 0x98, 0x9d, 0x9e, 0x97, 0x94, 0x91, 0x92, 0x83, 0x80, 0x85, 0x86, 0x8f, 0x8c, 0x89, 0x8a,
    0xab, 0xa8, 0xad, 0xae, 0xa7, 0xa4, 0xa1, 0xa2, 0xb3, 0xb0, 0xb5, 0xb6, 0xbf, 0xbc, 0xb9, 0xba,
    0xfb, 0xf8, 0xfd, 0xfe, 0xf7, 0xf4, 0xf1, 0xf2, 0xe3, 0xe0, 0xe5, 0xe6, 0xef, 0xec, 0xe9, 0xea,
    0xcb, 0xc8, 0xcd, 0xce, 0xc7, 0xc4, 0xc1, 0xc2, 0xd3, 0xd0, 0xd5, 0xd6, 0xdf, 0xdc, 0xd9, 0xda,
    0x5b, 0x58, 0x5d, 0x5e, 0x57, 0x54, 0x51, 0x52, 0x43, 0x40, 0x45, 0x46, 0x4f, 0x4c, 0x49, 0x4a,
    0x6b, 0x68, 0x6d, 0x6e, 0x67, 0x64, 0x61, 0x62, 0x73, 0x70, 0x75, 0x76, 0x7f, 0x7c, 0x79, 0x7a,
    0x3b, 0x38, 0x3d, 0x3e, 0x37, 0x34, 0x31, 0x32, 0x23, 0x20, 0x25, 0x26, 0x2f, 0x2c, 0x29, 0x2a,
    0x0b, 0x08, 0x0d, 0x0e, 0x07, 0x04, 0x01, 0x02, 0x13, 0x10, 0x15, 0x16, 0x1f, 0x1c, 0x19, 0x1a};

unsigned char rcon[256] = {
0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
```

```
0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d
};

void KeyExpansionCore(unsigned char *in, unsigned char i) {

unsigned int *q = (unsigned int *)in;
*q = (*q >> 8) | ((*q & 0xff) <<24);          // left shift operator "<<" and Right shift operator ">>"
```

```
in[0] = s_box[in[0]]; in[1] = s_box[in[1]];

in[2] = s_box[in[2]]; in[3] = s_box[in[3]];


in[0] ^=rcon[i];


}

void KeyExpansion(unsigned char *inputKey, unsigned char *expandedKeys) {

for (int i = 0; i < 16; i++)

{

   expandedKeys[i] = inputKey[i];

}


int bytesGenerated = 16;     // We've generated 16bytes so far (by copying the original key)

int rconIteration = 1;       // Rcon Iteration begins at 1

unsigned char temp[4];       // Temporary storage for core


while (bytesGenerated < 176)

{

   for (int i = 0; i < 4; i++)

   {

      temp[i] = expandedKeys[i+ bytesGenerated -4];

   }
```

```
    if(bytesGenerated % 16 == 0){
        KeyExpansionCore(temp,rconIteration);
        rconIteration++;
    }
    for (unsigned char a = 0; a < 4; a++)
    {
        expandedKeys[bytesGenerated] = expandedKeys[bytesGenerated - 16] ^ temp[a];

        //increased by 4 then in next iteration it is again increased by 4 and so on.....
        bytesGenerated++;
    }

}

}

void SubBytes(unsigned char *state)
{

    for (int i = 0; i < 16; i++)
```

```
    {
        state[i] = s_box[state[i]];
    }
}
void ShiftRows(unsigned char *state)
{
    unsigned char tmp[16];

    tmp[0] = state[0];
    tmp[1] = state[5];
    tmp[2] = state[10];
    tmp[3] = state[15];

    tmp[4] = state[4];
    tmp[5] = state[9];
    tmp[6] = state[14];
    tmp[7] = state[3];

    tmp[8] = state[8];
    tmp[9] = state[13];
    tmp[10] = state[2];
    tmp[11] = state[7];
```

```
    tmp[12] = state[12];
    tmp[13] = state[1];
    tmp[14] = state[6];
    tmp[15] = state[11];


      for (int i = 0; i < 16; i++)
    {
        state[i] = tmp[i];
    }
}
void MixColumns(unsigned char *state) {
 unsigned char tmp[16];

 tmp[0] = (unsigned char)(mul2[state[0]] ^ mul3[state[1]] ^ state[2] ^ state[3]);
 tmp[1] = (unsigned char)(state[0] ^ mul2[state[1]] ^ mul3[state[2]] ^ state[3]);
 tmp[2] = (unsigned char)(state[0] ^ state[1] ^ mul2[state[2]] ^ mul3[state[3]]);
 tmp[3] = (unsigned char)(mul3[state[0]] ^ state[1] ^ state[2] ^ mul2[state[3]]);

 tmp[4] = (unsigned char)(mul2[state[4]] ^ mul3[state[5]] ^ state[6] ^ state[7]);
 tmp[5] = (unsigned char)(state[4] ^ mul2[state[5]] ^ mul3[state[6]] ^ state[7]);
```

```
 tmp[6] = (unsigned char)(state[4] ^ state[5] ^ mul2[state[6]] ^ mul3[state[7]]);
 tmp[7] = (unsigned char)(mul3[state[4]] ^ state[5] ^ state[6] ^ mul2[state[7]]);

 tmp[8] = (unsigned char)(mul2[state[8]] ^ mul3[state[9]] ^ state[10] ^ state[11]);
 tmp[9] = (unsigned char)(state[8] ^ mul2[state[9]] ^ mul3[state[10]] ^ state[11]);
 tmp[10] = (unsigned char)(state[8] ^ state[9] ^ mul2[state[10]] ^ mul3[state[11]]);
 tmp[11] = (unsigned char)(mul3[state[8]] ^ state[9] ^ state[10] ^ mul2[state[11]]);

 tmp[12] = (unsigned char)(mul2[state[12]] ^ mul3[state[13]] ^ state[14] ^ state[15]);
 tmp[13] = (unsigned char)(state[12] ^ mul2[state[13]] ^ mul3[state[14]] ^ state[15]);
 tmp[14] = (unsigned char)(state[12] ^ state[13] ^ mul2[state[14]] ^ mul3[state[15]]);
 tmp[15] = (unsigned char)(mul3[state[12]] ^ state[13] ^ state[14] ^ mul2[state[15]]);

 for (int i = 0; i < 16; i++)
 {
   state[i] = tmp[i];
 }

}
void AddRoundKey(unsigned char *state, unsigned char *roundKey)
{
```

```
    for (int i = 0; i < 16; i++)
    {
        state[i] ^= roundKey[i];
    }
}

void AES_Encrypt(unsigned char *message, unsigned char *key)
{
    unsigned char state[16];
    for (int i = 0; i < 16; i++)
    {
        state[i] = message[i];
    }
    int numberOfRounds = 9;
    unsigned char expandedKey[176];
    KeyExpansion(key, expandedKey);
    AddRoundKey(state, key); // Whitening/AddRoundKey

    for (int i = 0; i < numberOfRounds; i++)
    {

        SubBytes(state);
```

```
        ShiftRows(state);

        MixColumns(state);

        AddRoundKey(state, expandedKey + (16 * (i+1)));

    }

    SubBytes(state);
    ShiftRows(state);
    AddRoundKey(state, expandedKey + 160);
    for (int i = 0; i < 16; i++)
    {
        message[i] = state[i];
    }

}

void PrintHex(unsigned char x){
    if(x /16 <10) cout<<(char)((x/16) + '0');
    if(x /16 >=10) cout<<(char)((x/16-10) + 'A');
```

```
    if(x % 16 <10) cout<<(char)((x%16) + '0');
    if(x % 16 >=10) cout<<(char)((x%16 -10) + 'A');
}

int main()

{
    int timetime=0;
    for(int count=0;count<9999;count++){
    auto start = high_resolution_clock::now();
    unsigned char message[] = "20 4c 56 6b c2 28 1a d5 48 a0 6f f7 8c aa f9 82 48 6c c7 07 a2 a7 3d 9f
50 41 7a 35 31 3d b5 f5 a8 96 94 be 22 8e f6 d9 ec 85 5f 03 40 f8 77 97 bc 3f c6 41 0f 0d ba 98 58 fd
4e 9e 30 0d 0c 5a e8 69 52 1c 2e e0 4b 23 39 3a 53 67 0c 60 ee 02 b4 4f c9 0b 37 9a 0f fe a5 a5 fa 0a
0c 28 51 2e 98 ff c8 dc 6b e9 d4 bb da 93 bc 16 a5 74 1e bc 86 65 47 9f c9 ed b4 60 48 28 12 0e a5 f2
44 d3 74 55 34 bc 08 7b aa be b5 10 df b9 7c 62 08 b4 8b 03 19 04 26 14 14 9d 3e 5b ac 2d b0 1d 44
42 49 e5 54 a1 45 d6 db e0 f2 49 ae 04 8f e0 57 c9 c3 b3 31 19 0e d3 4f 0d b0 88 db 47 94 71 82 54 c2
47 fd d3 18 51 b0 c9 a9 21 db 8c ea a2 8c 8c 56 ec 85 fc 76 a7 5c ac ca 5e 86 d4 9d ba 9c a0 1a 89 c3
b2 bd 9a 8e 79 49 8e f5 64 66 b7 df fb 27 b9 0b ff bb 23 6d 49 69 f8 ec 88 c0 3f 18 2a 57 c1 da 04 b9
60 43 35 45 39 17 2b 51 57 39 ee 6b db 55 76 54 61 1c 01 4e ef 78 06 5d f1 c9 75 18 67 fb 08 bf 28 4c
fb ec a6 a1 c6 9b ad 6a 32 30 70 d9 4d 38 7e 22 ef f4 15 dd 34 57 16 39 fa 04 27 64 47 54 64 34 e2 94
bc 64 0c d4 74 56 8d 22 72 ab 3c 8a a0 5e 5d 7f 8a a7 f8 e7 31 9c 47 d5 f9 a3 f6 75 57 47 87 37 27 ea
bf 10 10 99 10 22 06 7b 53 b7 f7 71 48 23 e4 ee 68 ee 5d 25 34 9e 55 d3 6c fa 67 c5 71 06 e1 a7 5a d5
ac 1b 75 90 25 b3 7d 26 f1 83 f4 c6 21 5d 59 03 67 16 74 4e 51 39 9d 3e 70 8b 39 27 c2 10 c6 64 fa d4
3f 23 d2 ee 05 dd ff 94 85 a2 28 e2 09 f5 7b 34 7d b5 79 8c 26 e1 ca 3c 30 6f d0 f8 04 b3 d5 02 04 c4
7e 8d 29 83 e8 9c da c2 d5 cf b3 d6 b1 aa 37 1a 56 71 81 6c 4e c3 89 2b c6 3c 28 89 e9 54 08 21 84 ac
1b 69 55 e3 47 8f a3 0f 0b 05 c3 f4 71 ae 13 35 84 21 7d 04 d5 d7 91 c5 84 6d f3 06 ee c8 b8 68 74 6a
```

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

```
unsigned char key[16] =
    {
        1, 2, 3, 4,
        5, 6, 7, 8,
        9, 10, 11, 12,
        13, 14, 15, 16};

int originalLen = strlen((const char *)message);
int lenOfPaddedMessage = originalLen;

if(lenOfPaddedMessage % 16 !=0){
    lenOfPaddedMessage = (lenOfPaddedMessage / 16 + 1) * 16;
```

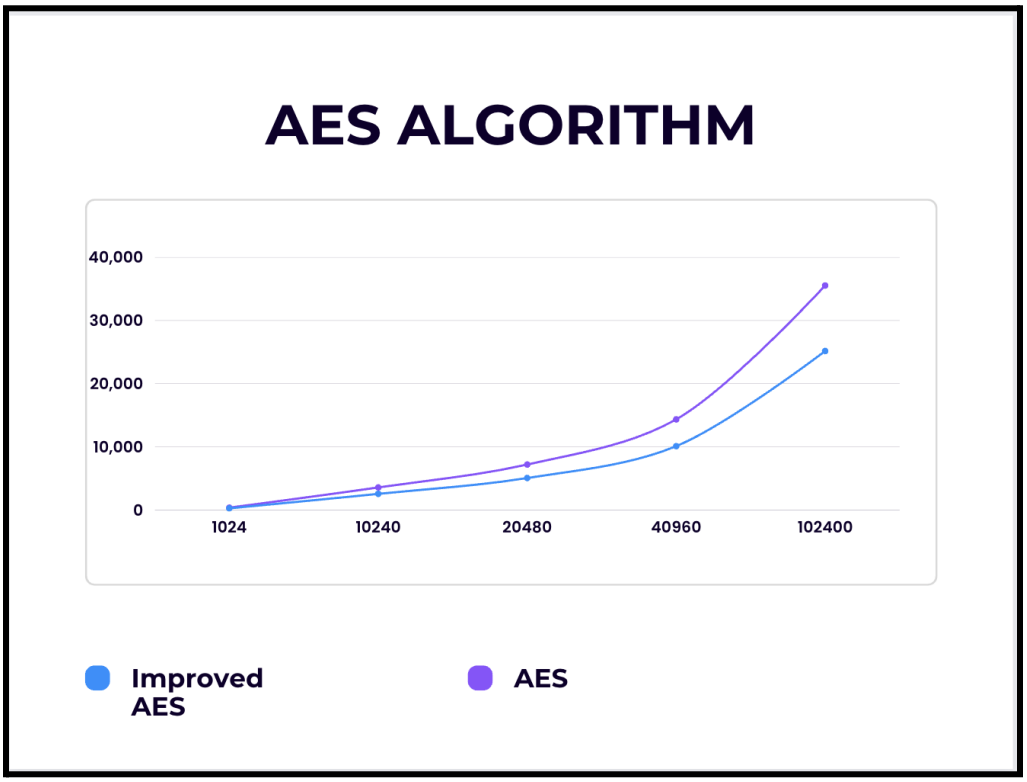
```
    }
    unsigned char *paddedMessage = new unsigned char[lenOfPaddedMessage];
    for (int i = 0; i < lenOfPaddedMessage; i++)
    {
        if(i >= originalLen) paddedMessage[i] = 0;
        else paddedMessage[i] = message[i];
    }
for (int i = 0; i < lenOfPaddedMessage; i+=16)
{
    AES_Encrypt(paddedMessage+i, key);


}
delete[] paddedMessage;
auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
     timetime=timetime+int(duration.count());
     }
    timetime=timetime/9999;
    cout<<"\n\nAverage Time: "<<timetime<<endl;


    return 0;
}
```

EXPERIMENTAL RESULTS AND ANALYSIS

Bytes	Normal AES(microsecond)	Improved AES(microsecond)
1024	360	253
10240	3565	2546
20480	7192	5053
40960	14340	10099
102400	35546	25168



Cryptographic Criteria of the New S-box:

Periodicity of the New S-box : Minimum compositions to get the identity function.

Let S : $F2^n \rightarrow F2^n$ be the function dening an S-box. For x $\in$ $F2^n$, the period of x under S is the smallest positive integer r such that

$S^r(x) = x$. AES, there are 5 possible periods, namely 2, 27, 59, 81 and 87.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	59	81	59	59	87	59	59	59	87	81	87	27	81	81	81	59
1	81	81	81	81	27	87	81	81	87	59	81	87	87	87	81	87
2	59	59	87	27	59	59	27	81	87	59	87	27	87	27	59	87
3	87	59	27	59	87	87	59	87	59	81	81	87	81	81	87	59
4	81	81	87	81	87	27	87	81	59	87	87	81	59	81	87	81
5	87	87	59	87	59	87	27	81	59	87	87	81	87	59	59	81
6	87	27	81	59	81	81	59	87	27	87	59	59	87	81	27	59
7	87	87	81	2	81	59	59	59	81	87	81	59	81	81	81	59
8	81	81	81	81	81	87	87	81	87	87	81	81	81	59	59	2
9	87	81	81	87	87	87	87	87	87	87	87	27	87	59	27	27
a	81	27	81	87	87	59	59	87	59	59	81	81	81	87	87	87
b	87	27	87	81	59	59	87	59	87	27	87	81	81	81	87	87
c	87	81	59	59	87	59	59	59	27	81	81	87	81	81	81	81
d	87	87	59	59	59	59	87	81	27	87	81	27	87	81	87	27
e	81	81	87	81	87	87	59	87	27	81	81	81	81	87	87	27
f	81	27	87	81	87	59	87	27	81	87	27	59	87	59	81	81

For the new S-box, 256 is the unique period so that the distribution of elements of $F2^8$ is more balanced for the periodicity criterion.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
1	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
2	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
3	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
4	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
5	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
6	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
7	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
8	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
9	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
a	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
b	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
c	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
d	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
e	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
f	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256

Algebraic Complexity of the New S-box:

Let S be an S-box over $F2^n$. Then S is completely defined by the set $\{(x_i , y_i) \mid x_i \in F2^n , y_i = S(x_i)\}$. A polynomial expression for S is determined by Lagrange's interpolation polynomial

$$P(x) = \sum y_i L_i(x), \qquad L_i(x) = \prod (x - xi) / \prod (x - xj)$$

The polynomial P (x) is of degree at most $2^n - 1$ and the number of its non-zero monomials is called the algebraic complexity. For AES, the polynomial is:

$$P(x) = 05x^{254} + 09x^{253} + f9x^{251} + 25x^{257} + f4x^{239} + 01x^{223} + b5x^{191} + 8fx^{127} + 63,$$

$$P(x) = \sum a_i x^i$$

where the list of the coefficient a_i is listed. The algebraic complexity of the new S-box is 255, which is optimal and makes it more resistant to possible algebraic attacks than the AES S-box.

	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0
f	00	b6	6c	30	3e	32	e5	06	68	b2	9c	8e	54	b9	0d	c8
e	01	c0	6d	aa	3a	0c	1a	7e	eb	52	48	4e	b5	cf	8a	5c
d	56	5b	1d	0b	42	43	4d	06	5c	15	37	49	02	ea	e9	d6
c	c4	35	b7	f2	ca	d0	0c	9a	28	ba	1c	8a	d7	ef	31	be
b	2e	ac	b5	6e	b1	6c	18	61	a3	06	8f	c4	10	0e	3b	c1
a	ff	55	f8	60	99	0c	b8	3a	88	90	ad	c6	61	83	a7	16
9	a4	48	5a	1b	a4	1f	b8	c4	3c	af	d5	33	4d	90	7d	60
8	cf	65	7e	5d	bb	43	b4	41	95	6c	0c	86	e0	02	b2	93
7	a2	6f	c6	e1	1d	71	6a	93	9d	12	c6	9f	d4	5e	c7	84
6	c3	84	1f	38	6e	a9	52	ea	98	97	ec	1f	bd	12	c4	32
5	49	ae	1a	63	b4	fe	7b	b4	e7	f4	04	2b	f8	e4	f2	47
4	fa	e3	04	c6	72	f8	fb	2c	bf	c8	e6	e1	0c	2a	2d	4a
3	e5	c3	73	0c	99	8a	8d	a9	25	39	16	c1	1b	3f	c0	19
2	5d	fd	9b	5d	fb	1d	f9	c7	a8	c4	03	48	63	63	15	83
1	f6	50	18	50	3c	57	96	0b	dc	dd	41	a0	fd	05	e7	50
0	13	66	d8	f8	fa	ea	93	72	a7	1d	5b	5e	0b	75	45	36

CONCLUSION OF THE REPORT

The optimization of the Advanced Encryption Standard was successful. Array shift mapping has enhanced shift row transformation. The SubByte transformation step was eliminated by combining the Mix Column transformation stages into one. A new affine transformation improved the S-box's security. Since 256 is the only period, the distribution of $F2^8$ elements is more evenly distributed for the periodicity criteria. The new S-box is more resistant to potential algebraic attacks than the AES S-box since it has an ideal algebraic complexity of 255. According to our findings, the encryption process has improved by **34.4406%.**

FUTURE SCOPE

Using specialist hardware, such as Field Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs), the AES algorithm can be implemented.

Utilise parallel processing techniques to implement the AES algorithm, which allows you to encrypt and decrypt data at the same time across several processor cores. The speed of computation can be considerably boosted by utilising more processing power.

SIMD Use Instructions:Utilising SIMD (Single Instruction Multiple Data) instructions, the AES algorithm can be improved. Multiple data elements can be processed simultaneously thanks to these instructions, which can speed up computation.

REFERENCES

[1] Selent, D.. Advanced encryption standard. Rivier Academic Journal 2010.

[2] Sarker, M.Z.H., Parvez, M.S.. A cost effective symmetric key cryptographic algorithm for small amounts of data. In: 9th International Multitopic Conference, IEEE INMIC 2005. IEEE; 2005.

[3] Stallings, W., Tahiliani, M.P.. Cryptography and network security: principles and practice; vol. 6. Pearson London; 2014

[4] Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems, Journal of Cryptology, vol.4, no.1, pp. 3–72.

[5] Carlet, C.: Vectorial Boolean Functions for Cryptography. In: Y. Crama & P.Hammer (Eds.), Boolean Models and Methods in Mathematics, Computer Science, and Engineering (Encyclopedia of Mathematics and its Applications, pp. 398–470. Cambridge: Cambridge University Press (2010)

IMPROVING AES ALGORITHM

Harshit Chopra : 20104013
Kartik Gupta : 20104025
Ayush Sharma : 20104059

Introduction

Information security includes cryptography, which deals with the study of algorithms and processes for safe data. The design of cryptographic algorithms is frequently improved as technology advances to ensure the security of data.

The standard Rijndael symmetric block cypher can encrypt and decrypt plaintext blocks of 128 bits using keys that are 128 bits, 192 bits, or 256 bits in size. The Rijndael cypher is appropriate for 8-bit and 32-bit processing and has a straightforward layout. The number of rounds to be executed depends on the key length. 128-bit keys require 10 rounds, 192-bit keys require 12 rounds, and 256-bit keys require 14 rounds.

.Addition, subtraction, multiplication, and division are the Basic arithmetic operations on a finite field $GF(2^8)$. Performance of AES operations depends on the length of the key. Sub Bytes, Shift Rows, Mix Column, and Add Round Key are the four different transformations that are used in each round. Every transformation starts with a 16-byte block that is a 4 x 4 matrix and outputs a matrix of the same dimensions.

Problem Statement

A shift row is arranging elements of state key matrix which performs a circular shift each row. The circular shift length is different every each row. These circular movement process are slow.

A linear transformation procedure is Mix Column. The components of the multiplication matrix that resulted from the two four-term polynomials with the coefficient element of GF(2^8). are multiplied by each element of the state characters 2^8 . Several transformations can be combined into one to reduce extra steps and improve performance.

Although AES is resistant to linear and differential attacks, it presents some weaknesses in regards with a variety of cryptanalytic criteria. A typical example is that an S-box should have high algebraic degree when expressed as a polynomial. The AES S-box is fairly straightforward and has an algebraic degree of 254 with only 9 monomials. Another weak criterion for the AES S-box is that some elements of $F2^8$ have short iterative periods. AES S-box transformation period is equal to 4 which is very low in comparison with the optimal value 16.

State-of-the-art

1.

The authors demonstrate that the block cypher key can be figured out with just one random byte defect at the eighth round's AES input. Simulations demonstrate that, in the case of two erroneous ciphertext pairs, the key may be precisely determined without the use of a brute-force search. The smallest error against AES has also been employed. The authors demonstrate that whereas AES-256 can be cracked using three pairs of correct and flawed ciphertexts, AES-192 can only be cracked using two pairs of correct and flawed ciphertexts.

D. Mukhopadhyay, "An improved fault based attack of the advanced encryption standard," *Lect. Notes Comput. Sci. (including Subser. Lect.Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5580, pp. 421–434, 2009.

M. Tunstall, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced encryption standard using a single fault," *Inf. Secur. Theory Pract. Secur. Priv. Mob. Devices Wirel. Commun.*, pp. 224–233, 2011.

2.

Using a library of circuit CMOS standard cells, Satoh optimised AES. Ahmad used combinational logic that is implemented on a Virtex II FPGA processor utilising a truth table. Daemen et al. enhance the execution performance of AES by using 32-bit data as the foundation data unit. Intel uses the AES-NI expanded instruction set to dramatically enhance the AES algorithm. For AES optimization, there are numerous different hardware implementations.

Satoh, A., Morioka, S., Takano, K., Munetoh, S.. A compact rijndael hardware architecture with s-box optimization. In: Asiacrypt; vol.2248. Springer; 2001, p. 239–254.

3.

On the entire AES-256, Biryukov et al. suggested a distinguisher and related-key attack. AES S-box affine transformation period is 4, according to Wang, and it does not attain the maximum 16. The iterative period of the AES S-box exhibits a short-period phenomena, and all of the periods are smaller than 88, according to research by Wang et al. Murphy and Robshaw examined the algebraic expression of the AES S-box and found that there are just 9 terms involved, making it a very straightforward algebraic expression. There has been a lot of work done on the AES S-box, including the most recent significant advancement that explained why the algebraic expression of the AES S-box is so simple and proposed an improved S-box. The S-algebraic box's expression includes 255 terms, making its complexity 2291. But with just 9 terms, the algebraic formulation of the inverse S-box is relatively straightforward. However, the algebraic expression of the inverse S-box becomes very simple and only 9 terms are involved.

Y. B. Wang, Property of affine transformation in S-box of AES, Journal of PLA University: Science and Technology, vol.4, no.2, pp.5-9, 2002

J. M. Liu, B. D. Wei and X. M. Wang, One AES S-box to increase complexity and its cryptanalysis, Journal of Systems Engineering and Electronics, vol.18, no.2, pp.427-433, 2007.

OBJECTIVES AND WORK DISTRIBUTION

Information security includes cryptography, which deals with the study of algorithms and processes for safe data. The most noticeable and significant components to offer high level security are block ciphers. Our goal is to make the AES algorithm better.

- The Circular movement process in shift row steps is slow. We can improve it and reduce the execution time..
- Mix Column is a linear transformation process. Combining different transformation into one can remove unnecessary steps and increase the performance.
- S-box should have high algebraic degree when expressed as a polynomial. Distribution of elements of should be more balanced for the periodicity criterion.

WORK DISTRIBUTION

Harshit Chopra: Shift Rows

Kartik Gupta: S-Box

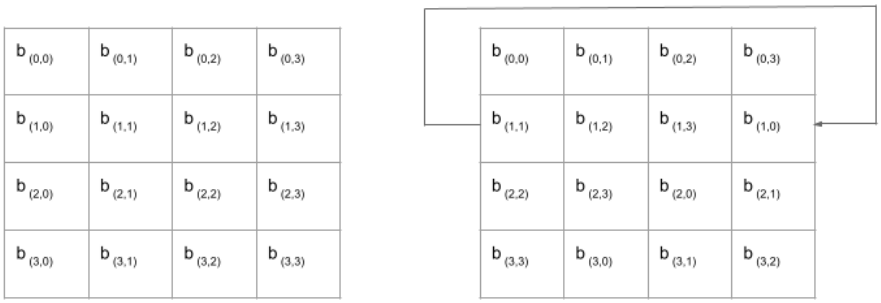
Ayush Sharma: Mix Columns

Proposed Design

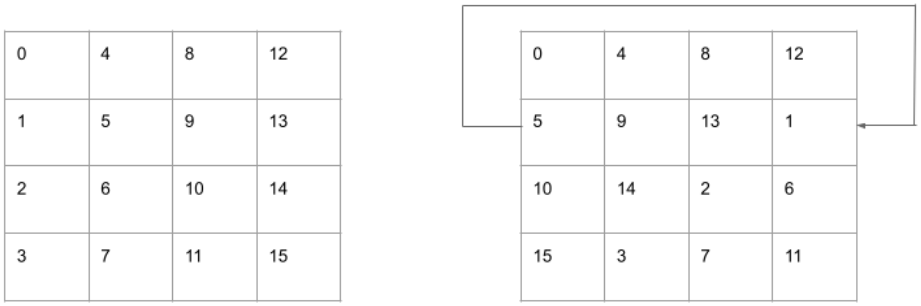
- Improve the shift row transformation by using array shift mapping instead of moving and rotating each element.
- Improvement in Mix Column transformation process by combining different transformation into one and making Sub Byte step unnecessary.
- The AES S-box has algebraic degree 254 with only 9 monomials which is very simple. We created new affine transformation which increased the security of S-box. 256 is the unique period so that the distribution of elements of $F2^8$ is more balanced for the periodicity criterion. The algebraic complexity of the new S-box is 255, which is optimal and makes it more resistant to possible algebraic attacks than the AES S-box.

Implementation

A **shift row** is arranging elements of state key matrix which performs a circular shift each row. The circular shift length is different every each row. The first row is never moved over. Second row move one first element to the right at last element. Third row move two first elements to the right at last element and the last row move three first elements.



These circular process could be faster by using array shift row mapping which has index value that mapped directly to index of key state that should be placing. These steps will reduce the circular movement process that executed manually.



Array shift row value index for circular movement:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	0	5	10	15	4	9	14	3	8	13	2	7	12	1	6	11

Mix Column is a linear transformation process. MixColumns operation performed by the Rijndael cipher, along with the ShiftRows step, is the primary source of diffusion in Rijndael. Each element of state characters multiplied against elements of multiplication matrix that came from two four-term polynomial which has the coefficient element of $GF(2^8)$.

$d_{(0,0)}$	$d_{(0,1)}$	$d_{(0,2)}$	$d_{(0,3)}$
$d_{(1,0)}$	$d_{(1,1)}$	$d_{(1,2)}$	$d_{(1,3)}$
$d_{(2,0)}$	$d_{(2,1)}$	$d_{(2,2)}$	$d_{(2,3)}$
$d_{(3,0)}$	$d_{(3,1)}$	$d_{(3,2)}$	$d_{(3,3)}$

=

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

×

$b_{(0,0)}$	$b_{(0,1)}$	$b_{(0,2)}$	$b_{(0,3)}$
$b_{(1,0)}$	$b_{(1,1)}$	$b_{(1,2)}$	$b_{(1,3)}$
$b_{(2,0)}$	$b_{(2,1)}$	$b_{(2,2)}$	$b_{(2,3)}$
$b_{(3,0)}$	$b_{(3,1)}$	$b_{(3,2)}$	$b_{(3,3)}$

(where b matrix is a result of shift rows)

This process could be made faster by combining the sub bytes, shift rows and mix column step.

Mix Columns:

$d_{(0,0)}$	$d_{(0,1)}$	$d_{(0,2)}$	$d_{(0,3)}$
$d_{(1,0)}$	$d_{(1,1)}$	$d_{(1,2)}$	$d_{(1,3)}$
$d_{(2,0)}$	$d_{(2,1)}$	$d_{(2,2)}$	$d_{(2,3)}$
$d_{(3,0)}$	$d_{(3,1)}$	$d_{(3,2)}$	$d_{(3,3)}$

=

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

×

$b_{(0,0)}$	$b_{(0,1)}$	$b_{(0,2)}$	$b_{(0,3)}$
$b_{(1,0)}$	$b_{(1,1)}$	$b_{(1,2)}$	$b_{(1,3)}$
$b_{(2,0)}$	$b_{(2,1)}$	$b_{(2,2)}$	$b_{(2,3)}$
$b_{(3,0)}$	$b_{(3,1)}$	$b_{(3,2)}$	$b_{(3,3)}$

(where b matrix is a result of Shift Rows)

Shift Rows :

$b_{(0,0)}$	$b_{(0,1)}$	$b_{(0,2)}$	$b_{(0,3)}$
$b_{(1,0)}$	$b_{(1,1)}$	$b_{(1,2)}$	$b_{(1,3)}$
$b_{(2,0)}$	$b_{(2,1)}$	$b_{(2,2)}$	$b_{(2,3)}$
$b_{(3,0)}$	$b_{(3,1)}$	$b_{(3,2)}$	$b_{(3,3)}$

=

$a_{(0,0)}$	$a_{(0,1)}$	$a_{(0,2)}$	$a_{(0,3)}$
$a_{(1,0)}$	$a_{(1,1)}$	$a_{(1,2)}$	$a_{(1,3)}$
$a_{(2,0)}$	$a_{(2,1)}$	$a_{(2,2)}$	$a_{(2,3)}$
$a_{(3,0)}$	$a_{(3,1)}$	$a_{(3,2)}$	$a_{(3,3)}$

(where a is a result of Sub Bytes)

Sub Bytes:

$a_{(0,0)}$	$a_{(0,1)}$	$a_{(0,2)}$	$a_{(0,3)}$
$a_{(1,0)}$	$a_{(1,1)}$	$a_{(1,2)}$	$a_{(1,3)}$
$a_{(2,0)}$	$a_{(2,1)}$	$a_{(2,2)}$	$a_{(2,3)}$
$a_{(3,0)}$	$a_{(3,1)}$	$a_{(3,2)}$	$a_{(3,3)}$

=

$S[t_{(0,0)}]$	$S[t_{(0,1)}]$	$S[t_{(0,2)}]$	$S[t_{(0,3)}]$
$S[t_{(1,0)}]$	$S[t_{(1,1)}]$	$S[t_{(1,2)}]$	$S[t_{(1,3)}]$
$S[t_{(2,0)}]$	$S[t_{(2,1)}]$	$S[t_{(2,2)}]$	$S[t_{(2,3)}]$
$S[t_{(3,0)}]$	$S[t_{(3,1)}]$	$S[t_{(3,2)}]$	$S[t_{(3,3)}]$

(where $S[t_{i,j}]$ is substitution of bytes

of a state with a byte from lookup table)

AES S-box

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

This process could be made faster by combining the sub bytes, shift rows and mix column step.

Mix Columns:

$d_{(0,0)}$	$d_{(0,1)}$	$d_{(0,2)}$	$d_{(0,3)}$
$d_{(1,0)}$	$d_{(1,1)}$	$d_{(1,2)}$	$d_{(1,3)}$
$d_{(2,0)}$	$d_{(2,1)}$	$d_{(2,2)}$	$d_{(2,3)}$
$d_{(3,0)}$	$d_{(3,1)}$	$d_{(3,2)}$	$d_{(3,3)}$

$=$

$02 * S[t_{(0,0)}]$	$03 * S[t_{(0,1)}]$	$01 * S[t_{(0,2)}]$	$01 * S[t_{(0,3)}]$
$01 * S[t_{(1,1)}]$	$02 * S[t_{(1,2)}]$	$03 * S[t_{(1,3)}]$	$01 * S[t_{(1,0)}]$
$01 * S[t_{(2,2)}]$	$01 * S[t_{(2,3)}]$	$02 * S[t_{(2,0)}]$	$03 * S[t_{(2,1)}]$
$03 * S[t_{(3,3)}]$	$01 * S[t_{(3,0)}]$	$01 * S[t_{(3,1)}]$	$02 * S[t_{(3,2)}]$

Multiplication of a Galois field by the Galois field for 1 results in the input not being changed at all. For multiplication by the Galois field for 2 the results are any one of 256 values and the same for multiplication by the Galois field 3.

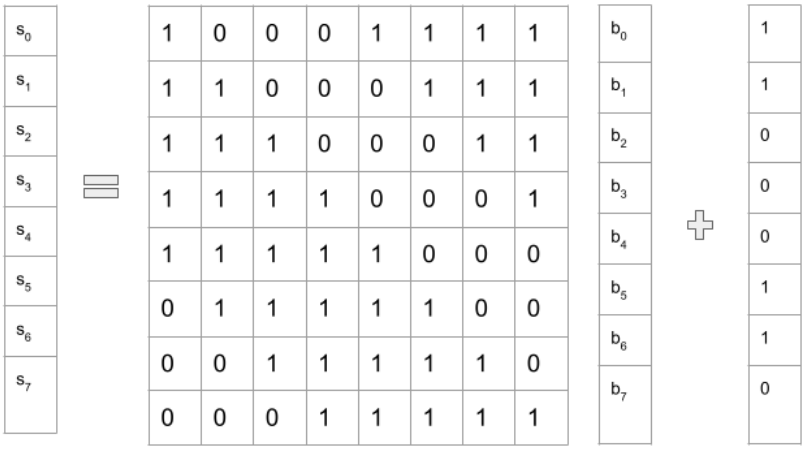
Multiply by 2:

```
0x00,0x02,0x04,0x06,0x08,0x0a,0x0c,0x0e,0x10,0x12,0x14,0x16,0x18,0x1a,0x1c,0x1e,
0x20,0x22,0x24,0x26,0x28,0x2a,0x2c,0x2e,0x30,0x32,0x34,0x36,0x38,0x3a,0x3c,0x3e,
0x40,0x42,0x44,0x46,0x48,0x4a,0x4c,0x4e,0x50,0x52,0x54,0x56,0x58,0x5a,0x5c,0x5e,
0x60,0x62,0x64,0x66,0x68,0x6a,0x6c,0x6e,0x70,0x72,0x74,0x76,0x78,0x7a,0x7c,0x7e,
0x80,0x82,0x84,0x86,0x88,0x8a,0x8c,0x8e,0x90,0x92,0x94,0x96,0x98,0x9a,0x9c,0x9e,
0xa0,0xa2,0xa4,0xa6,0xa8,0xaa,0xac,0xae,0xb0,0xb2,0xb4,0xb6,0xb8,0xba,0xbc,0xbe,
0xc0,0xc2,0xc4,0xc6,0xc8,0xca,0xcc,0xce,0xd0,0xd2,0xd4,0xd6,0xd8,0xda,0xdc,0xde,
0xe0,0xe2,0xe4,0xe6,0xe8,0xea,0xec,0xee,0xf0,0xf2,0xf4,0xf6,0xf8,0xfa,0xfc,0xfe,
0x1b,0x19,0x1f,0x1d,0x13,0x11,0x17,0x15,0x0b,0x09,0x0f,0x0d,0x03,0x01,0x07,0x05,
0x3b,0x39,0x3f,0x3d,0x33,0x31,0x37,0x35,0x2b,0x29,0x2f,0x2d,0x23,0x21,0x27,0x25,
0x5b,0x59,0x5f,0x5d,0x53,0x51,0x57,0x55,0x4b,0x49,0x4f,0x4d,0x43,0x41,0x47,0x45,
0x7b,0x79,0x7f,0x7d,0x73,0x71,0x77,0x75,0x6b,0x69,0x6f,0x6d,0x63,0x61,0x67,0x65,
0x9b,0x99,0x9f,0x9d,0x93,0x91,0x97,0x95,0x8b,0x89,0x8f,0x8d,0x83,0x81,0x87,0x85,
0xbb,0xb9,0xbf,0xbd,0xb3,0xb1,0xb7,0xb5,0xab,0xa9,0xaf,0xad,0xa3,0xa1,0xa7,0xa5,
0xdb,0xd9,0xdf,0xdd,0xd3,0xd1,0xd7,0xd5,0xcb,0xc9,0xcf,0xcd,0xc3,0xc1,0xc7,0xc5,
0xfb,0xf9,0xff,0xfd,0xf3,0xf1,0xf7,0xf5,0xeb,0xe9,0xef,0xed,0xe3,0xe1,0xe7,0xe5
```

Multiply by 3:

```
0x00,0x03,0x06,0x05,0x0c,0x0f,0x0a,0x09,0x18,0x1b,0x1e,0x1d,0x14,0x17,0x12,0x11,
0x30,0x33,0x36,0x35,0x3c,0x3f,0x3a,0x39,0x28,0x2b,0x2e,0x2d,0x24,0x27,0x22,0x21,
0x60,0x63,0x66,0x65,0x6c,0x6f,0x6a,0x69,0x78,0x7b,0x7e,0x7d,0x74,0x77,0x72,0x71,
0x50,0x53,0x56,0x55,0x5c,0x5f,0x5a,0x59,0x48,0x4b,0x4e,0x4d,0x44,0x47,0x42,0x41,
0xc0,0xc3,0xc6,0xc5,0xcc,0xcf,0xca,0xc9,0xd8,0xdb,0xde,0xdd,0xd4,0xd7,0xd2,0xd1,
0xf0,0xf3,0xf6,0xf5,0xfc,0xff,0xfa,0xf9,0xe8,0xeb,0xee,0xed,0xe4,0xe7,0xe2,0xe1,
0xa0,0xa3,0xa6,0xa5,0xac,0xaf,0xaa,0xa9,0xb8,0xbb,0xbe,0xbd,0xb4,0xb7,0xb2,0xb1,
0x90,0x93,0x96,0x95,0x9c,0x9f,0x9a,0x99,0x88,0x8b,0x8e,0x8d,0x84,0x87,0x82,0x81,
0x9b,0x98,0x9d,0x9e,0x97,0x94,0x91,0x92,0x83,0x80,0x85,0x86,0x8f,0x8c,0x89,0x8a,
0xab,0xa8,0xad,0xae,0xa7,0xa4,0xa1,0xa2,0xb3,0xb0,0xb5,0xb6,0xbf,0xbc,0xb9,0xba,
0xfb,0xf8,0xfd,0xfe,0xf7,0xf4,0xf1,0xf2,0xe3,0xe0,0xe5,0xe6,0xef,0xec,0xe9,0xea,
0xcb,0xc8,0xcd,0xce,0xc7,0xc4,0xc1,0xc2,0xd3,0xd0,0xd5,0xd6,0xdf,0xdc,0xd9,0xda,
0x5b,0x58,0x5d,0x5e,0x57,0x54,0x51,0x52,0x43,0x40,0x45,0x46,0x4f,0x4c,0x49,0x4a,
0x6b,0x68,0x6d,0x6e,0x67,0x64,0x61,0x62,0x73,0x70,0x75,0x76,0x7f,0x7c,0x79,0x7a,
0x3b,0x38,0x3d,0x3e,0x37,0x34,0x31,0x32,0x23,0x20,0x25,0x26,0x2f,0x2c,0x29,0x2a,
0x0b,0x08,0x0d,0x0e,0x07,0x04,0x01,0x02,0x13,0x10,0x15,0x16,0x1f,0x1c,0x19,0x1a
```

The S-box maps an 8-bit input, c, to an 8-bit output, $s = S(c)$. Both the input and output are interpreted as polynomials over GF(2). First, the input is mapped to its multiplicative inverse in $GF(2^8) = GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$, Rijndael's finite field. Zero, as the identity, is mapped to itself. The multiplicative inverse is then transformed using the following affine transformation:



s_0		1	0	0	0	1	1	1	1	b_0		1
s_1		1	1	0	0	0	1	1	1	b_1		1
s_2		1	1	1	0	0	0	1	1	b_2		0
s_3	=	1	1	1	1	0	0	0	1	b_3		0
s_4		1	1	1	1	1	0	0	0	b_4		0
s_5		0	1	1	1	1	1	0	0	b_5		1
s_6		0	0	1	1	1	1	1	0	b_6		1
s_7		0	0	0	1	1	1	1	1	b_7		0

Since the whole transformation of AES S-box is S-box(x) = La ×x^{-1}+63, it does not matter which affine transformation matrix and irreducible polynomial are selected and the finial algebraic expression of AES S-box only involves 9 terms. The security of AES S-box is questioned because of its simplicity. AES S-box needs to be improved in order to get rid of the vulnerability of basic algebraic expressions.

Let F_q be a finite field with q elements. For n ≥ 2, let GL(n, Fq) be the group of invertible n × n matrices with entries in Fq .

The order of GL(n, Fq) is : $|GL(n, Fq)| = \prod (q^n - q^k)$

Let A =

1	0	0	0	1	1	0	1
1	1	0	0	1	0	0	1
0	1	1	1	0	0	0	1
0	0	0	0	1	1	0	1
0	0	1	0	0	0	1	0
1	0	0	0	1	0	1	1
0	1	1	1	0	0	0	0
1	1	0	1	0	1	1	0

and,

α = 0xfe = (1, 1, 1, 1, 1, 1, 1, 0),

β = 0x3f = (0, 0, 1, 1, 1, 1, 1, 1).

The new S-box is generated by the multivariate Boolean function S_N defined for $x \in G.F(2^8)$

$$S_N = \begin{cases} (Ax+\alpha / Ax+\beta) & If \quad Ax+\beta \neq 0 \\ 0x01 & If \quad Ax+\beta = 0 \end{cases}$$

Example : SN (0xdd) = 0xed
0xdd = (1, 1, 0, 1, 1, 1, 0, 1) = (x7 , x6 , x5 , x4 , x3 , x2 , x1 , x0) .
Applying affine transformation: $Ax+\alpha$

s_0	=	1	0	0	0	1	1	1	1	1	+	0	1
s_1		1	1	0	0	0	1	1	1	0		1	0
s_2		1	1	1	0	0	0	1	1	1		1	0
s_3		1	1	1	1	0	0	0	1	1		1	1
s_4		1	1	1	1	1	0	0	0	1		1	1
s_5		0	1	1	1	1	1	0	0	0		1	1
s_6		0	0	1	1	1	1	1	0	1		1	1
s_7		0	0	0	1	1	1	1	1	1		1	0

so Ax + α = (0, 1, 1, 1, 1, 0, 0, 1) = 0x79

Example : SN (0xdd) = 0xed
0xdd = (1, 1, 0, 1, 1, 1, 0, 1) = (x7 , x6 , x5 , x4 , x3 , x2 , x1 , x0) .
Applying affine transformation: $Ax+\beta$

s_0	=	1	0	0	0	1	1	1	1	1	1	0
s_1		1	1	0	0	0	1	1	1	0	1	0
s_2		1	1	1	0	0	0	1	1	1	0	0
s_3		1	1	1	1	0	0	0	1	1	1	1
s_4		1	1	1	1	1	0	0	0	1	1	1
s_5		0	1	1	1	1	1	0	0	0	1	1
s_6		0	0	1	1	1	1	1	0	1	0	0
s_7		0	0	0	1	1	1	1	1	0	0	1

so Ax + β = (1, 0, 1, 1, 1, 0, 0, 0) = 0xb8

S-box value:

S_N (0xdd) = (Ax + α)/(Ax + β)

=0x79/0xb8

= $(t^6 + t^5 + t^4 + t^3 + 1)/ (t^7 + t^5 + t^4 + t^3)$

= $t^7 + t^6 + t^5 + t^3 + t^2 + 1 \pmod{t^8 + t^4 + t^3 + t + 1}$

= (1, 1, 1, 0, 1, 1, 0, 1)

= 0xed

New S-Box :

```
unsigned s_box[256] =
    {
     0x36 ,0x94 ,0x89 ,0xcb ,0x77 ,0x96 ,0xd2 ,0x4b ,0x05 ,0xf7 ,0xab ,0xc5 ,0x6d ,0xa1 ,0xd6 ,0x5b
    ,0x61 ,0x91 ,0xe7 ,0xd0 ,0x1f ,0xa9 ,0x43 ,0x1d ,0x9b ,0xbe ,0xf4 ,0xb8 ,0x42 ,0x63 ,0x87 ,0xbb
    ,0x02 ,0x58 ,0xc3 ,0xac ,0xe4 ,0xe5 ,0xeb ,0xb3 ,0x83 ,0x70 ,0x64 ,0x20 ,0x57 ,0x08 ,0x60 ,0x85
    ,0x2f ,0x90 ,0x07 ,0xee ,0x23 ,0x33 ,0x81 ,0x12 ,0x14 ,0xea ,0x39 ,0x21 ,0x62 ,0xcd ,0x28 ,0x2e
    ,0x2c ,0xf6 ,0xdd ,0x25 ,0xbc ,0x11 ,0xa7 ,0xe6 ,0xfd ,0x53 ,0x98 ,0x9c ,0x38 ,0x1b ,0x5c ,0x54
    ,0x75 ,0x95 ,0x26 ,0x00 ,0x09 ,0x3b ,0x44 ,0x9d ,0x15 ,0x5d ,0x1c ,0x9a ,0x5f ,0xc9 ,0xa4 ,0x78
    ,0x5a ,0xf3 ,0x0b ,0x0c ,0xe9 ,0x0a ,0x06 ,0x3e ,0x71 ,0xe1 ,0xfa ,0xf5 ,0x7f ,0x65 ,0x19 ,0xdf
    ,0x8e ,0x32 ,0xfb ,0x74 ,0x50 ,0xd9 ,0x72 ,0x24 ,0x45 ,0x0f ,0x69 ,0x76 ,0xda ,0x41 ,0xb1 ,0xdb
    ,0x79 ,0x80 ,0x3a ,0x49 ,0xe8 ,0xbf ,0x73 ,0x16 ,0x18 ,0x8d ,0xce ,0xa3 ,0x0e ,0xc6 ,0xef ,0xe3
    ,0xd7 ,0x99 ,0x6e ,0x35 ,0xfc ,0xaf ,0xa2 ,0xc1 ,0xde ,0xc2 ,0x1e ,0xd1 ,0x6c ,0xf1 ,0xaa ,0x7e
    ,0x8c ,0x52 ,0xd4 ,0x4a ,0x7c ,0x93 ,0xf0 ,0xe2 ,0xd8 ,0x66 ,0x04 ,0x9e ,0x84 ,0x3c ,0x13 ,0xae
    ,0x86 ,0x88 ,0xa5 ,0x68 ,0xd3 ,0x37 ,0x3d ,0x56 ,0x6a ,0x5e ,0x7a ,0xad ,0xc8 ,0xb2 ,0x40 ,0x67
    ,0x0d ,0xb7 ,0x46 ,0x7d ,0xa6 ,0x82 ,0x6b ,0x3f ,0x34 ,0x22 ,0xb0 ,0xc0 ,0x29 ,0x4e ,0x59 ,0x7b
    ,0xc7 ,0x31 ,0xba ,0x47 ,0xfe ,0xc4 ,0xd5 ,0xe0 ,0x92 ,0xb9 ,0x10 ,0xa0 ,0x8b ,0xed ,0x55 ,0x97
    ,0xca ,0x1a ,0xf9 ,0x2a ,0xcc ,0xf2 ,0x4c ,0x51 ,0x03 ,0x30 ,0x4d ,0xf8 ,0xb4 ,0xbd ,0xcf ,0x48
    ,0xec ,0x2b ,0x9f ,0xff ,0x27 ,0x17 ,0xb6 ,0x8f ,0x8a ,0xb5 ,0x01 ,0xa8 ,0x6f ,0x4f ,0xdc ,0x2d
    };
```

Cryptographic Criteria of the New S-box

Periodicity of the New S-box : Minimum compositions to get the identity function.

Let S : $F2^n \rightarrow F2^n$ be the function dening an S-box. For x $\in$ $F2^n$, the period of x under S is the smallest positive integer r such that $S^r(x) = x$. AES, there are 5 possible periods, namely 2, 27, 59, 81 and 87.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	59	81	59	59	87	59	59	59	87	81	87	27	81	81	81	59
1	81	81	81	81	27	87	81	81	87	59	81	87	87	87	81	87
2	59	59	87	27	59	59	27	81	87	59	87	27	87	27	59	87
3	87	59	27	59	87	87	59	87	59	81	81	87	81	81	87	59
4	81	81	87	81	87	27	87	81	59	87	87	81	59	81	87	81
5	87	87	59	87	59	87	27	81	59	87	87	81	87	59	59	81
6	87	27	81	59	81	81	59	87	27	87	59	59	87	81	27	59
7	87	87	81	2	81	59	59	59	81	87	81	59	81	81	81	59
8	81	81	81	81	81	87	87	81	87	87	81	81	81	59	59	2
9	87	81	81	87	87	87	87	87	87	87	87	27	87	59	27	27
a	81	27	81	87	87	59	59	87	59	59	81	81	81	87	87	87
b	87	27	87	81	59	59	87	59	87	27	87	81	81	81	87	87
c	87	81	59	59	87	59	59	59	27	81	81	87	81	81	81	81
d	87	87	59	59	59	59	87	81	27	87	81	27	87	81	87	27
e	81	81	87	81	87	87	59	87	27	81	81	81	81	87	87	27
f	81	27	87	81	87	59	87	27	81	87	27	59	87	59	81	81

For the new S-box, 256 is the unique period so that the distribution of elements of $F2^8$ is more balanced for the periodicity criterion.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
1	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
2	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
3	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
4	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
5	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
6	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
7	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
8	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
9	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
a	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
b	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
c	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
d	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
e	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256
f	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256	256

Algebraic Complexity of the New S-box:

Let S be an S-box over $F2^n$. Then S is completely defined by the set $\{(x_i , y_i) \mid x_i \in F2^n , y_i = S(x_i)\}$. A polynomial expression for S is determined by Lagrange's interpolation polynomial

$$P(x) = \sum y_i L_i(x), \qquad L_i(x) = \prod (x - xi) / \prod (x - xj)$$

The polynomial P (x) is of degree of at most $2^n - 1$ and the number of its non-zero monomials is called the algebraic complexity. For AES, the polynomial is:

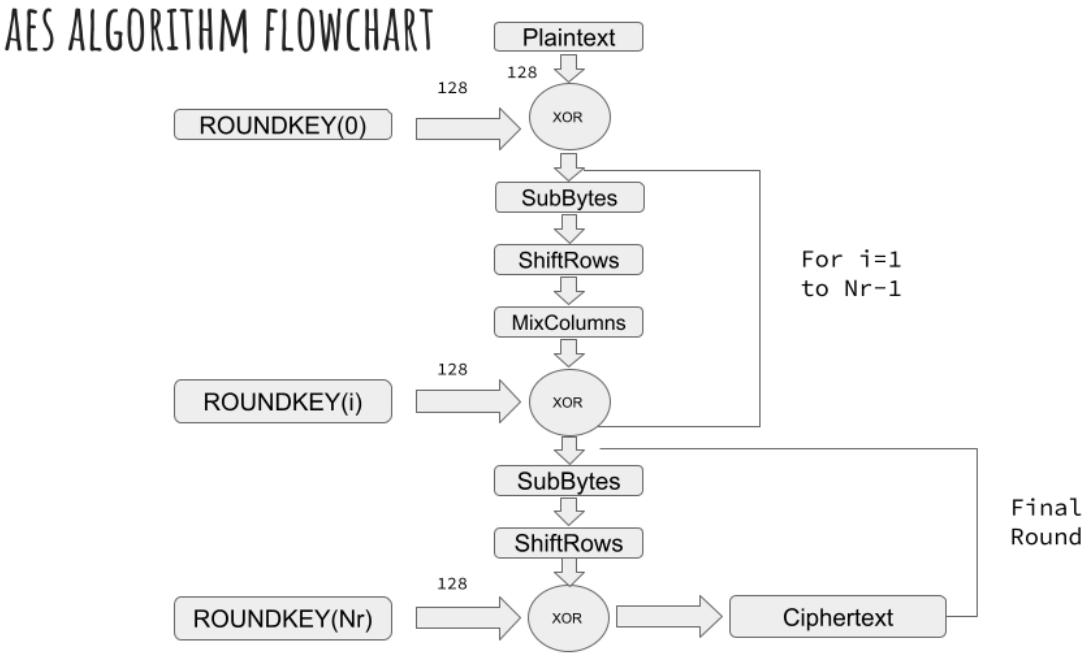
$$P(x) = 05x^{254} + 09x^{253} + f9x^{251} + 25x^{247} + f4x^{239} + 01x^{223} + b5x^{191} + 8fx^{127} + 63,$$

$$P(x) = \sum a_j x^j$$

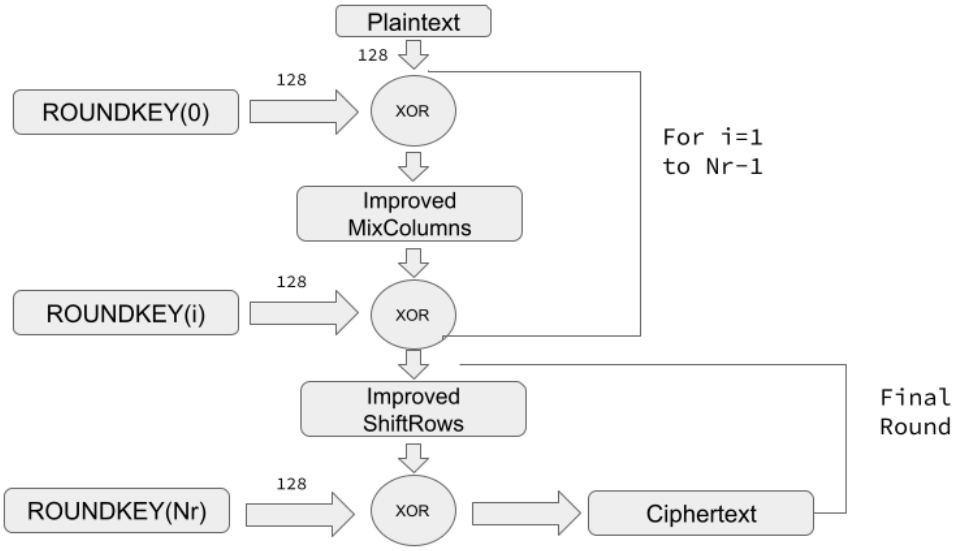
where the list of the coefficient a_j is listed. The algebraic complexity of the new S-box is 255, which is optimal and makes it more resistant to possible algebraic attacks than the AES S-box.

Algebraic expression of the new S-box:

	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0
f	00	b6	6c	30	3e	32	e5	06	68	b2	9c	8e	54	b9	0d	c8
e	01	c0	6d	aa	3a	0c	1a	7e	eb	52	48	4e	b5	cf	8a	5c
d	56	5b	1d	0b	42	43	4d	06	5c	15	37	49	02	ea	e9	d6
c	c4	35	b7	f2	ca	d0	0c	9a	28	ba	1c	8a	d7	ef	31	be
b	2e	ac	b5	6e	b1	6c	18	61	a3	06	8f	c4	10	0e	3b	c1
a	ff	55	f8	60	99	0c	b8	3a	88	90	ad	c6	61	83	a7	16
9	a4	48	5a	1b	a4	1f	b8	c4	3c	af	d5	33	4d	90	7d	60
8	cf	65	7e	5d	bb	43	b4	41	95	6c	0c	86	e0	02	b2	93
7	a2	6f	c6	e1	1d	71	6a	93	9d	12	c6	9f	d4	5e	c7	84
6	c3	84	1f	38	6e	a9	52	ea	98	97	ec	1f	bd	12	c4	32
5	49	ae	1a	63	b4	fe	7b	b4	e7	f4	04	2b	f8	e4	f2	47
4	fa	e3	04	c6	72	f8	fb	2c	bf	c8	e6	e1	0c	2a	2d	4a
3	e5	c3	73	0c	99	8a	8d	a9	25	39	16	c1	1b	3f	c0	19
2	5d	fd	9b	5d	fb	1d	f9	c7	a8	c4	03	48	63	63	15	83
1	f6	50	18	50	3c	57	96	0b	dc	dd	41	a0	fd	05	e7	50
0	13	66	d8	f8	fa	ea	93	72	a7	1d	5b	5e	0b	75	45	36



Improved AES algorithm flowchart



FUTURE SCOPE

- Use Hardware acceleration: The AES algorithm can be implemented using specialized hardware such as Field Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs).

- Implement the AES algorithm utilising parallel processing methods, which make advantage of several processor cores to encrypt and decrypt data simultaneously. As a result of using additional processing power, computing speed can be greatly increased.

- Use SIMD Instructions:The AES algorithm can be optimized by using SIMD(Single Instruction Multiple Data) instructions. These instructions allow multiple data elements to be processed simultaneously,which can improve the speed of computing.

REFERENCES

- Selent, D.. Advanced encryption standard. Rivier Academic Journal 2010.

- Sarker, M.Z.H., Parvez, M.S.. A cost effective symmetric key cryptographic algorithm for small amount of data. In: 9th International Multitopic Conference, IEEE INMIC 2005. IEEE; 2005.

- Stallings, W., Tahiliani, M.P.. Cryptography and network security: principles and practice; vol. 6. Pearson London; 2014

- Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems, Journal of Cryptology, vol.4, no.1, pp. 3–72.

- Carlet, C.: Vectorial Boolean Functions for Cryptography. In: Y. Crama & P.Hammer (Eds.), Boolean Models and Methods in Mathematics, Computer Science, and Engineering (Encyclopedia of Mathematics and its Applications, pp. 398–470. Cambridge: Cambridge University Press (2010)