**Cryptography**

### -Lab report



Submitted By: Deshant Devkota

Submitted To: Suresh Thapal (Lecturer, Cryptography)

Vedas College, Lalitpur, Nepal

**Tribhuvan University**

**Department of Bsc Csit**

Submitted Date: Thursday, July 23, 2021

Submitted Lab Reports:

1. To implement AES-128 encryption and use it to encrypt a single 128 block size plaintext.

## LAB 12

Advanced Encryption Standard

**1. Objectives:**

In this lab we were to implement AES-128 encryption and use it to encrypt a single 128 block size plaintext.

**2.Introduction:**

AES stands for Advanced Encryption Standard and it is a symmetric block encryption algorithm established by National Institute of Standards and Technology(NIST) in 2001. AES was developed and proposed for selection to NIST by two belgian cryptographers Vincent Rijmen and Joan Daemen.

AES works on block size of 128-bit and allows for different key sizes: 128, 192 and 256 bits which work in rounds of 10, 12 and 16 respectively. AES is based on Shannon’s Substitution-Permutation network and is efficient for both hardware and software implementation.

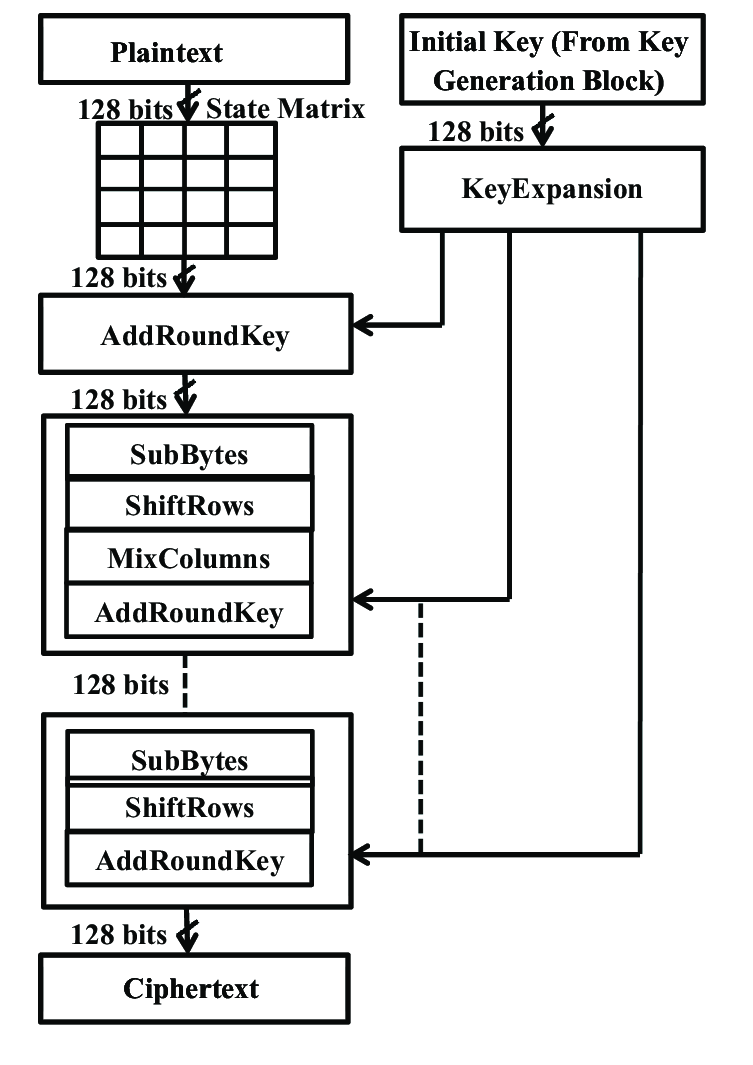
AES-128:

AES-128 works with a 128 bit key in 128 bit plaintext. i.e16 bytes. This 16 bytes are stored in 4x4 column matrix as such:

| b0 | b4 | b8 | b12 |
| --- | --- | --- | --- |
| b1 | b5 | b9 | b13 |
| b2 | b6 | b10 | b14 |
| b3 | b7 | b11 | b15 |

This matrix is also called a state matrix which is used for further processing and stores the plaintext.

It works for 10 rounds and the flow diagram for AES is given below:



src: [Block Diagram of AES Algorithm. | Download Scientific Diagram (researchgate.net)](https://www.researchgate.net/figure/Block-Diagram-of-AES-Algorithm_fig1_326596710)

Here is a brief description about different steps followed in AES:

1. Key Expansion:

The 128 bit key is expanded using key scheduling for 10 more rounds. The algorithm is based on 4 words taken from the key and is produced simultaneously using immediately preceding word and word[current-4].

1. Add Round Key

Each byte of a state is operated using bitwise xor with the byte of the round key.

1. Sub Bytes

A non linear substitution performed using a 16x16 substitution table. This table is designed to improve the strength of AES and an inverse table is also provided for inverse substitution. The table is void of identity and inverses.

1. Shift Rows

Each row of the state matrix is shifted cyclically according to the [‘row’th-1] position. That is, the first row isn’t shifted and the last row is shifted 3 times.

1. Mix Columns

This allows diffusion of the bytes and is designed such that it is simple and is defined as a Galois Field operation where multiplication is done by shifting bytes and addition is achieved using simple xor.

The transformation matrix is:

| 2 | 3 | 1 | 1 |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 1 |
| 1 | 1 | 2 | 3 |
| 3 | 1 | 1 | 2 |

This matrix is multiplied to each column of the state to produce new columns. This step isn’t followed in the last step as it’s redundant.

**3.Code:**

# Implementing AES-128 to encrypt a 128 bit plain text

# code available in https://github.com/DexA1948/cryptography

import copy

class mat44:

def \_\_init\_\_(self):

self.mat = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]

round\_key = list()

Rcon = (0x01, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36)

Sbox = (

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

)

Sbox\_inv = (

0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,

0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB,

0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,

0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,

0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,

0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,

0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,

0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,

0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,

0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,

0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,

0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,

0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,

0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,

0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,

0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D

)

def formroundkeys(key):

wordlist = list()

for row\_ in range(0, 4):

word = list()

for col\_ in range(0, 4):

word.append(key[col\_][row\_])

wordlist.append(word)

for wn in range(4, 44):

tword = copy.copy(wordlist[wn - 1])

if wn % 4 == 0:

tword[0], tword[1], tword[2], tword[3] = tword[1], tword[2], tword[3], tword[0]

tword[0], tword[1], tword[2], tword[3] = Sbox[tword[0]], Sbox[tword[1]], Sbox[tword[2]], Sbox[tword[3]]

tword[0] = tword[0] ^ Rcon[int(wn / 4)]

for i in range(0, 4):

tword[i] = tword[i] ^ wordlist[wn - 4][i]

wordlist.append(tword)

for rn in range(0, 11):

temp = mat44()

for row in range(0, 4):

temp.mat[row][0], temp.mat[row][1], temp.mat[row][2], temp.mat[row][3] = wordlist[rn \* 4][row], \

wordlist[rn \* 4 + 1][row], \

wordlist[rn \* 4 + 2][row], \

wordlist[rn \* 4 + 3][row]

round\_key.append(temp.mat)

def subbytes(given):

for row in range(4):

for col in range(4):

given[row][col] = int(Sbox[given[row][col]])

return given

def shiftrows(given):

for row in range(1, 4):

given[row][0], given[row][1], given[row][2], given[row][3] = given[row][row], given[row][(row + 1) % 4], \

given[row][(row + 2) % 4], given[row][

(row + 3) % 4]

return given

def xor(given, given2):

for row in range(4):

for col in range(4):

given[row][col] = given[row][col] ^ given2[row][col]

return given

def addroundkey(given, rn):

return xor(given, round\_key[rn])

def formmatrix(given):

matrix\_ = [[], [], [], []]

for row in range(4):

for col in range(4):

matrix\_[col].append(ord(given[row \* 4 + col]))

return matrix\_

def mixcol(given):

mixmat = [

[2, 3, 1, 1],

[1, 2, 3, 1],

[1, 1, 2, 3],

[3, 1, 1, 2]

]

temp = [

[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0]

]

for i in range(0, 4):

for j in range(0, 4):

for k in range(0, 4):

if mixmat[i][k] == 2:

if given[k][j] >= 128:

temp[i][j] = temp[i][j] ^ (((given[k][j] << 1) ^ 27) & 255)

else:

temp[i][j] = temp[i][j] ^ ((given[k][j] << 1) & 255)

elif mixmat[i][k] == 3:

if given[k][j] >= 128:

temp[i][j] = temp[i][j] ^ (given[k][j] ^ (((given[k][j] << 1) ^ 27) & 255))

else:

temp[i][j] = temp[i][j] ^ (given[k][j] ^ ((given[k][j] << 1) & 255))

else:

temp[i][j] = temp[i][j] ^ given[k][j]

return temp

def aes(plaintext, key):

plaintext = formmatrix(plaintext)

key = formmatrix(key)

formroundkeys(key)

# initial round

temp = xor(plaintext, key)

# 9 rounds of AES128

for rn in range(1, 10):

temp = subbytes(temp)

temp = shiftrows(temp)

temp = mixcol(temp)

temp = addroundkey(temp, rn)

# final round

temp = subbytes(temp)

temp = shiftrows(temp)

temp = addroundkey(temp, 10)

return temp

text = "Two One Nine Two"

key\_ = "Thats my Kung Fu"

cipher = aes(text, key\_)

print("\nPlaintext is: ", text)

print("\nCipher in hex is: ")

for i in range(0, 4):

for j in range(0, 4):

print(hex(cipher[j][i]), end=" ")

print("\n\nCipher in text is: ")

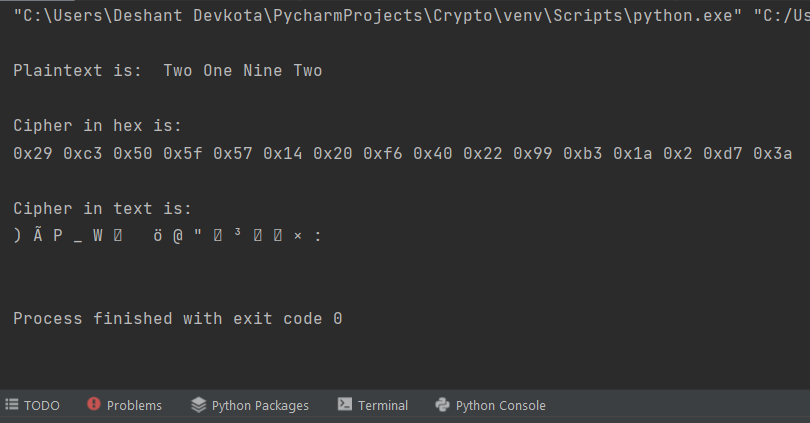
for i in range(0, 4):

for j in range(0, 4):

print(chr(cipher[j][i]), end=" ")

print("\n")

**4.Output:**



Thank You