Ex.No.4 03/07/2024	**IMPLEMENTATION OF KEY GENERATION IN ADVANCED ENCRYPTION STANDARD**

AIM:

To implement key generation in Advanced Encryption Standard using Java/Python

ALGORITHM:

S-box and Rcon:

a. The sbox is a lookup table used for substitution in the sub-word operation.
b. The rcon array contains constants used in the key expansion process.

Sub-word Function:

a. Takes a word (4 bytes) as input.
b. Applies the S-box to each byte of the word.
c. Returns the substituted word.

Rotate_word Function:

a. Takes a word as input.
b. Rotates the word one byte to the left.
c. Returns the rotated word.

Key Expansion Function:

a. Takes the original key and the number of rounds as input.
b. Initializes an empty key schedule.
c. Copies the original key into the first four words of the key schedule.
d. Iteratively generates round keys, For each word:
e. Takes the previous word as a temporary variable.
f. If the word index is a multiple of 4:
    i. Apply rotate_word and sub_word to the temporary word.
    ii. XOR the first byte of the temporary word with the corresponding Rcon value.
    iii. XOR the temporary word with the word four positions earlier in the key schedule.
g. Append the new word to the key schedule.
h. Returns the complete key schedule.

Word-to-Hex Function:

a. Takes a word as input.
b. Converts each byte to a two-digit hexadecimal string.
c. Joins the hexadecimal strings with spaces.
d. Returns the formatted hexadecimal string.

CODING:

```
from AES import KeyGeneration
if __name__ == "__main__":
    key_generation = KeyGeneration()
    hex_key_matrix = [[int(input("Enter the value: "),16) for _ in range(4)] for _ in range(4)]
    expanded_key=key_generation.key_expansion(hex_key_matrix)
    print("The Input Matrix is: ")
    print(expanded_key)
    for i in range(11):
        print(f"Round {i}:")
        for j in range(4):
            print(key_generation.word_to_hex(expanded_key[4*i+j]))
class KeyGeneration:
    def __init__(self) -> None:
        self.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]]

    self.rcon = [0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36]

  def sub_word(self, word):
    return [self.sbox[b >> 4][b & 0x0F] for b in word]

  def rotate_word(self, word):
    return word[1:] + word[:1]

  def key_expansion(self, key, round=10):
    key_schedule = []
    for i in range(4):
      key_schedule.append(key[i])
    for i in range(4, (round + 1) * 4):
      temp = key_schedule[i - 1]
      if i % 4 == 0:
        temp = self.sub_word(self.rotate_word(temp))
        temp[0] ^= self.rcon[i // 4]
      key_schedule.append([key_schedule[i - 4][j] ^ temp[j] for j in range(4)])
    return key_schedule
```

```
def word_to_hex(self, word):
    return ' '.join(f'{b:02x}' for b in word)
```

SCREEN SHOTS:

Unit Testing:

Getting inputs:

```
Enter the value: 1
Enter the value: 2
Enter the value: 7
Enter the value: 4
Enter the value: 8
Enter the value: 9
Enter the value: 6
Enter the value: 5
Enter the value: 2
Enter the value: 3
Enter the value: 1
Enter the value: 4
Enter the value: 5
Enter the value: 9
Enter the value: 3
Enter the value: 2
The Input Matrix is:
[[1, 2, 7, 4], [8, 9, 6, 5], [2, 3, 1, 4], [5, 9, 3, 2], [1, 121, 112, 111], [9, 112, 118, 106], [11, 115, 119, 110], [14, 122, 116, 108], [217, 235, 32, 196],
```

Decimal to Hexa-Decimal

```
c5 f9 b4 e2
```

Generating Round Keys

```
Round 0:
07 08 09 04
05 06 02 01
03 04 08 09
05 02 01 07
Round 1:
71 74 cc 6f
74 72 ce 6e
77 76 c6 67
72 74 c7 60
Round 2:
e1 b2 1c 2f
95 c0 d2 41
e2 b6 14 26
90 c2 d3 46
Round 3:
c0 d4 46 4f
55 14 94 0e
b7 a2 80 28
27 60 53 6e
```

RESULT:

Thus, we have successfully implement key generation in Advanced Encryption Standard using Java/Python

Evaluation

Parameter	Max Marks	Marks Obtained
Uniqueness of the Code	50	
Completion of experiment on time	10	
Documentation	15	
Total	75	
Signature of the faculty with Date		