|  |  |
| --- | --- |
| 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:**

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

**Sub-word Function:**

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

**Rotate\_word Function:**

1. Takes a word as input.
2. Rotates the word one byte to the left.
3. Returns the rotated word.

**Key Expansion Function**:

1. Takes the original key and the number of rounds as input.
2. Initializes an empty key schedule.
3. Copies the original key into the first four words of the key schedule.
4. Iteratively generates round keys, For each word:
5. Takes the previous word as a temporary variable.
6. If the word index is a multiple of 4:
   * 1. Apply rotate\_word and sub\_word to the temporary word.
     2. XOR the first byte of the temporary word with the corresponding Rcon value.
     3. XOR the temporary word with the word four positions earlier in the key schedule.
7. Append the new word to the key schedule.
8. Returns the complete key schedule.

**Word-to-Hex Function:**

1. Takes a word as input.
2. Converts each byte to a two-digit hexadecimal string.
3. Joins the hexadecimal strings with spaces.
4. Returns the formatted hexadecimal string.

|  |
| --- |
| **CODING:** |

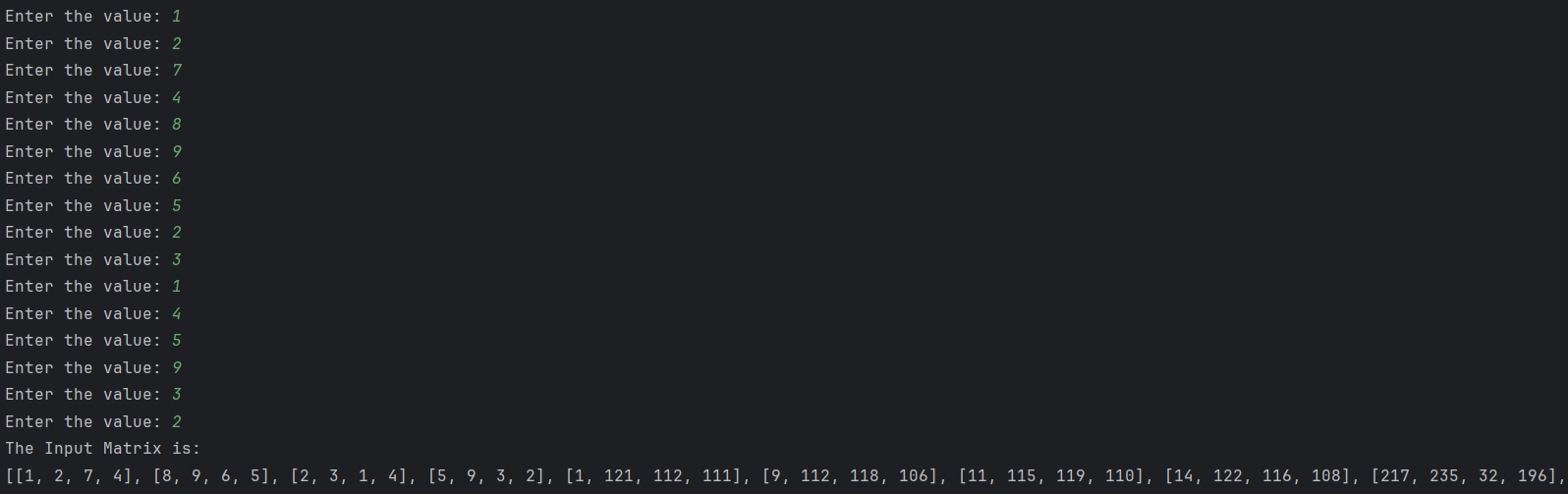
from AES import KeyGeneration  
if \_\_name\_\_ == "\_\_main\_\_":  
 key\_generation = KeyGeneration()  
 hex\_key\_matrix = [[int(input("Enter the value: "),16) for \_ in range(4)] for \_ in range(4)]  
 expanded\_key=key\_generation.key\_expansion(hex\_key\_matrix)  
 print("The Input Matrix is: ")  
 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:**

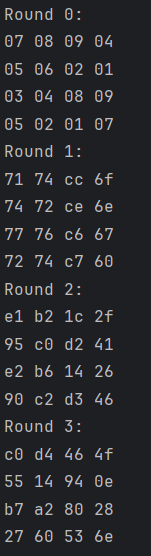
**Getting inputs:**

****

**Decimal to Hexa-Decimal**

****

**Generating Round Keys**

****

|  |
| --- |
| **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 |  |  |