AES Key Generation

Prashanth.S 19MID0020

Importing the Necessary Libraries

In [1]:
```
import numpy as np
import pandas as pd
import ast
import itertools
```

Importing the key S-Box

In [2]:
```
col_names  =['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f']
row_names = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f']

enc_key_sbox = pd.read_excel('AES tables.xlsx', index_col=0)

enc_key_sbox.columns = col_names ## replacing the column names
enc_key_sbox.index = row_names    ## replacing the row names

enc_key_sbox.head()
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	1	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	4	c7	23	c3	18	96	5	9a	7	12	80	e2	eb	27	b2	75
4	9	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84

Getting inputs

In [3]:
```
def text_hexadecimal(text): ## all the blocks --> 16bytes
    hex_text   = []
    for i in text: ## character -> ascii (decimal) -> hexa-decimal
        hex_text.append(hex(ord(i))[2:])
    return hex_text ## 16-byte representation of the text
```

In [4]:
```
key_str = 'Thats my Kung Fu'
hex_key = text_hexadecimal(key_str)
print("Hexadecimal Representation : ",hex_key)
```

```
Hexadecimal Representation :  ['54', '68', '61', '74', '73', '20', '6d', '79', '20', '4b', '75', '6e', '67', '20', '46', '75']
```

In [5]:
```
## splitting into 4*4 matrix
hex_key = [hex_key[i:i+4] for i in range(0, len(hex_key), 4)]
hex_key
```

Out[5]:
```
[['54', '68', '61', '74'],
 ['73', '20', '6d', '79'],
 ['20', '4b', '75', '6e'],
 ['67', '20', '46', '75']]
```

Key Generation

In [6]:
```
def key_sbox(element):
    row_index = element[0]
    col_index  = element[1]
    ans = enc_key_sbox.loc[row_index,[col_index]][col_index][0]
    return str(ans)
```

In [7]:
```
def binaryToDecimal(binary):
    binary1 = binary
    decimal, i, n = 0, 0, 0

    while(binary != 0):
        dec = binary % 10
        decimal = decimal + dec * pow(2, i)
        binary = binary//10
        i += 1

    hexadecimal =  hex(decimal)[-1]
    return hexadecimal
```

In [8]:
```
def bcd_hexadecimal(bcd_list):
    w4 = []

    for i in range(0,len(bcd_list),2):
        temp_str = ''
        temp_str = binaryToDecimal(int(bcd_list[i]))
        temp_str = temp_str + binaryToDecimal(int(bcd_list[i+1]))
        w4.append(temp_str)

    return w4
```

In [9]:
```
def HexaDecimaltoBCD(str):
    list1 = []
    for i in range(len(str)):
        decimal = int(str[i], 16)                                  ## hexadecimal -> decimal
        binary_num = bin(decimal).replace("0b", "")  # decimal -> binary
        list1.append(binary_num)

    ## binary in-terms of 4 bits
    for i in range(len(list1)):
        element = list1[i]
        if len(element)<4:
            diff = 4 - len(element)
            for j in range(diff):
                element = "0" + element
            list1[i] = element

    return list1
```

In [10]:
```
def Hexaword_BCD(list1):
    bcd = [ HexaDecimaltoBCD(i) for i in list1 ]
    bcd = list(np.concatenate(bcd).flat)    ## 2d list to 1d list
    return bcd
```

In [11]:
```
def XOR(list1, list2):

    def compare(element1, element2):
        ans = []
        for i,j in zip(element1,element2):
            if (i!=j):ans.append(1)
            else:ans.append(0)
        return ans

    main_ans = []
    for i in range(len(list1)):
        main_ans.append(compare(list1[i], list2[i]))

    main_ans = [ "".join(list(map(str, i))) for i in main_ans ]
    return main_ans
```

In [12]:
```
def rotword(word):
    rot=word[1:]
    rot.append(word[0])
    return rot
```

In [13]:
```
## round-constant table
## key-round and value-hexadecimal
round_constant = {1:'1', 2:'2', 3:'4', 4:'8', 5:'10', 6:'20', 7:'40', 8:'80', 9:'1B', 10:'36'}
```

In [14]:
```
def coll_generation(iteration_var, hex_key):

    ## taking the last column words
    last_col = hex_key[-1]

    ## left shifting the last column words
    #left_shift = last_col[1:] + last_col[:1]
    left_shift = rotword(last_col)

    ## sub-word generation from S-Box
    subword = []
    for i in left_shift:
        val = key_sbox(i)
        if len(val)!=2:
            val = '0' + val
        subword.append(val)
    else:
        subword.append(val)

    ## subword --> hexadecimal(subword)
    y1 = Hexaword_BCD(subword)

    ## subword (XOR) Round Constant
    element = round_constant[iteration_var]
    initial = HexaDecimaltoBCD(element)
    initial_length = len(initial)

    if (initial_length == 1):
        between = initial
        temp_1 = [ [0 for j in range(4) for i in range(1)] ]
        last = [ [0 for j in range(4)] for i in range(6) ]
        temp_1.extend(between)
        temp_1.extend(last)
        final = temp_1
        if (element==1): ## if there is a single element(0,1,2, ... ,9) from the s-box --> length=1
            beforel = [ [0 for j in range(4) for i in range(1)] ]
            beforel.extend(final)
            final = beforel

    elif (initial_length == 2):  ## if there is a two element(11, 1a, b1, ...) from the s-box --> length=2
        last = [ [0 for j in range(4)] for i in range(6) ]
        first = initial
        first.extend(last)
        final = first

    final = [ "".join(list(map(str, i))) for i in final ]
    round_list = final

    round_ans = XOR(y1, round_list) ## --> g(col4_before)

    # taking coll_before
    coll_before = [ HexaDecimaltoBCD(i) for i in hex_key[0] ]
    coll_before = list(np.concatenate(coll_before).flat)

    ## coll --> coll_before (exor) g(col4_before)
    coll = XOR(coll_before, round_ans)
    coll = bcd_hexadecimal(coll)

    return coll
```

In [15]:
```
complete_keys = []
for i in range(1,11): ## 10 times running
    temp = []

    ## 1st col word = col_before[0] (exor) g(col_before[-1])
    coll_hex = coll_generation(i, hex_key)
    coll_bin  = Hexaword_BCD(coll_hex)

    for j in range(2,5): ## each loop, 3 times running ( remaining 3 words)
        if (j==2):  ## 2nd col words
            col1_before_bin = Hexaword_BCD(hex_key[j-1])
            col2_bin = XOR(coll_before_bin, col1_bin)
            col2_hex = bcd_hexadecimal(col2_bin)

        else:  ## 3rd and 4th column words
            col2_before_bin = Hexaword_BCD(hex_key[j-1])
            col3_bin = XOR(col2_before_bin, col2_bin)
            col3_hex = bcd_hexadecimal(col3_bin)
            col2_bin = Hexaword_BCD(col3_hex)
            temp.append(col3_hex)

    ## once again generating the hex_key
    hex_key = []

    hex_key.append(coll_hex)
    hex_key.append(col2_hex)
    hex_key.extend(temp)

    complete_keys.append(hex_key)
```

Keys for all rounds

In [16]:
```
for i in range(len(complete_keys)):
    print("Round - {} keys --> ".format(i+1),end=' ')
    print(*list(itertools.chain(*complete_keys[i])))
```

```
Round - 1 keys -->  e2 32 fc f1 91 12 91 88 b1 59 e4 e6 d6 79 a2 93
Round - 2 keys -->  56 08 20 07 c7 1a b1 8f 76 43 55 69 a0 3a f7 fa
Round - 3 keys -->  d2 60 0d e7 15 7a bc 68 63 39 e9 01 c3 03 1e fb
Round - 4 keys -->  a1 12 02 c9 b4 68 be a1 d7 51 57 a0 14 52 49 5b
Round - 5 keys -->  b1 29 3b 33 05 41 85 92 d2 10 d2 32 c6 42 9b 69
Round - 6 keys -->  bd 3d c2 87 b8 7c 47 15 6a 6c 95 27 ac 2e 0e 4e
Round - 7 keys -->  cc 96 ed 16 74 ea aa 03 1e 86 3f 24 b2 a8 31 6a
Round - 8 keys -->  8e 51 ef 21 fa bb 45 22 e4 3d 7a 06 56 95 4b 6c
Round - 9 keys -->  bf 42 bf 90 45 59 fa b2 a1 64 80 b4 f7 f1 cb d8
Round - 10 keys -->  28 fd de f8 6d a4 24 4a cc c0 a4 fe 3b 31 6f 26
```

Mam, I have included my code here

https://github.com/PrashanthSingaravelan/winter_semester-2022-assignments/tree/main/CSI3002%20Applied%20Cryptography%20and%20Network%20Security/lab%20assignment/assignment_3