National University of Computer and Emerging Sciences

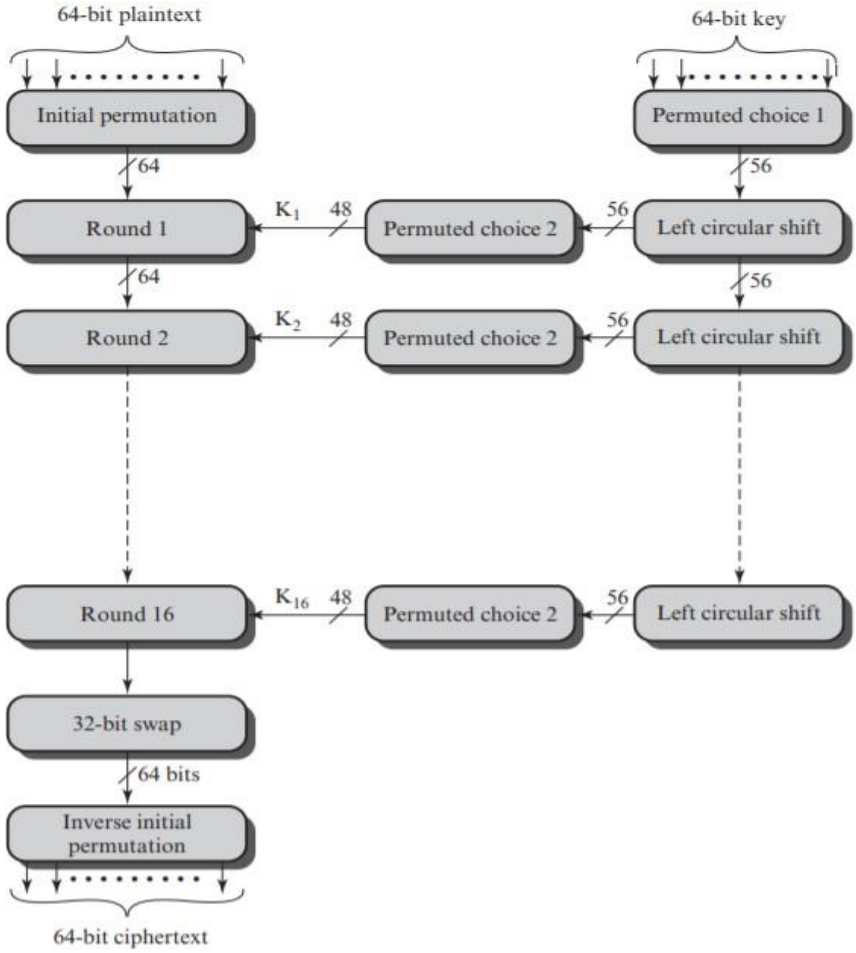
FAST School of Computing | Fall 2020 | Islamabad Campus

Assignment-01

{Implementation of DES algorithm in C++ to be tested by given test cases of Gtest}

During the US-Vietnam War NSA failed due to misinterpreting the signal. And due to this failure, they decide to make an algorithm/scheme for encryption and decryption. The Data Encryption Standard published as an official Federal Information Processing Standard (FIPS) for the United States.

Data Encryption Standard (DES) is used to encrypt until AES. You are a member of NSA and the job is to implement the DES. there are 2 inputs, the plain text and key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length. There are 16 rounds on which you have to encrypt the message/plaintext. First-round input is simply plain text and it combines with key second round input is encrypted text of first-round so on. It will be continued till completion of round 16.



Running Example

Following is the example of 1st round

DES sub-key Generation

We have to convert 64 bit to 56 bit by removing the parity bit. Let suppose,
key = **12345678ABCEDFF9**
The equivalent binary is:
0001001000110100010101100111100010101011110011101101111111111001

PC1 =
[57,49,41,33,25,17,9,1,58,50,42,34,26,18,10,2,59,51,43,35,27,19,11,3,60,52,44,36,63,55,47,39,31,23,15,7,62,54,46,38,30,22,14,6,61,53,45,37,29,21,13,5,28,20,12,4]
The permutation table has 56 number between 1–64 in a predefined order and do has 8,16,24,32,40,48,56 and 64 (these are parities bits)

Index of the binary key, will be number of PC1 (where the length of PC1 is 56)

So resultant 56-bit key will be.
11110000111011001001101011000111010101100110111110001111

Now split it into two halves of 28 bits each

Left half = 1111000011101100100110101100
Right Half =01110101011001101111110001111

In every round of key generation, both halves are circularly left-shifted and the number of bits for left shifting is round dependent as mentioned below
round_shifts = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

After shifting new left and right halves will look like
New Left half= 1110000111011001001101011001
New Right Half= 11101010110011011111100011110

Now take another permutation of left and right combine to produce the new key with final permutations.

This will take Compression table and combined both halves as input and return a 48-bits string as output

Resultant 48 bit key is produced by Final Permutation with

PC2 = [14,17,11,24,1,5,3,28,15,6,21,10,23,19,12,4,26,8,16,7,27,20,13,2,
41,52,31,37,47,55,30,40,51,45,33,48,44,49,39,56,34,53,46,42,50,36,29,32]

PC2 is of length 48 and same its values are used for the 56-bit key index.

New 48 bit key will be= **000110110000101100110011111111110011100001110010**

Except for initial permutation of the key all other steps will be used in 16 rounds.

After shifting new left and right halves these also will be used in the next round of generating keys.

Plain Text

Now coming to the plain text we will use what we actually work

For this Input will be= 0123456789ABCDEF

We will convert it to binary

Binary output is **0000000100100011010001010110011110001001101010111001101111**

Initial Permutation

So, the very first step of DES for plain text encryption is the process called Initial permutation which is just rearrangement of plaintext bits according to a given permutation table

```
INITIAL_PERMUTATION_TABLE = ['58 ', '50 ', '42 ', '34 ', '26 ', '18 ', '10 ', '2',
                             '60 ', '52 ', '44 ', '36 ', '28 ', '20 ', '12 ', '4',
                             '62 ', '54 ', '46 ', '38 ', '30 ', '22 ', '14 ', '6',
                            '64 ', '56 ', '48 ', '40 ', '32 ', '24 ', '16 ', '8',
                            '57 ', '49 ', '41 ', '33 ', '25 ', '17 ', '9 ', '1',
                             '59 ', '51 ', '43 ', '35 ', '27 ', '19 ', '11 ', '3',
                             '61 ', '53 ', '45 ', '37 ', '29 ', '21 ', '13 ', '5',
                             '63 ', '55 ', '47 ', '39 ', '31 ', '23 ', '15 ', '7']
```

Index of binary input will be the value of INITIAL_PERMUTATION_TABLE.
Output after initial permutation

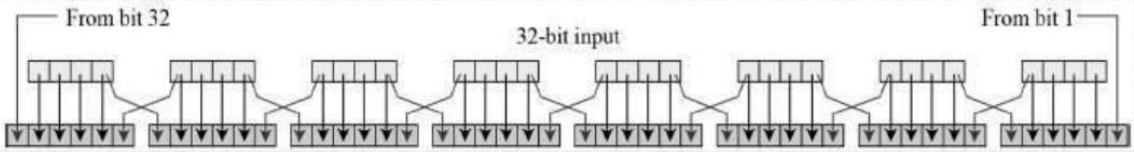
1100110000000000110011001111111111110000101010101111000010101010
Then split it into two halves each of 32 bit
Left half= 11001100000000000110011001111111
Right Half=11110000101010101111000010101010

Round Function

Round Function take input of Right half (32 bits) and expand it to 48-bit using Expansion so that it can be XOR'ed with a 48-bit key

Expansion Table
In expansion index of 32 bit is the value of the expansion table.
EXPANSION_TABLE = [32,1,2,3,4,5,4,5,6,7,8,9,8,9,10,11,12,13,12,13,14,15,16,17, 16,17,18,19,20,21,20,21,22,23,24,25,24,25,26,27,28,29,28,29,30,31,32,1]
So mapping 32 bits over expansion table to get 48-bit output

Expansion Permutation Box



32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

Output will be 48 bit: **011110100001010101010101011110100001010101010101**
Next step is XOR this input with 48-bit key which is
000110110000101100110011111111110011110000111001 0

XOR

After expansion of 32-bit to 48-bit binary string, a XOR operation is performed on expanded 48-bit and input 48-bit key. The XOR is simple operation which checks the bit in both string for each index and result into '0' if both are same i.e. ('11' and '00') else result into '1' for different value i.e. ('01' or '10').

Output of 48 bit key and 48 bit text is

011110100001010101010101011110100001010101010101
⊕ **000110110000101100110011111111110011110000111001 0**

011000010000001100110010100001000110110100100111

S-BOX Calculation

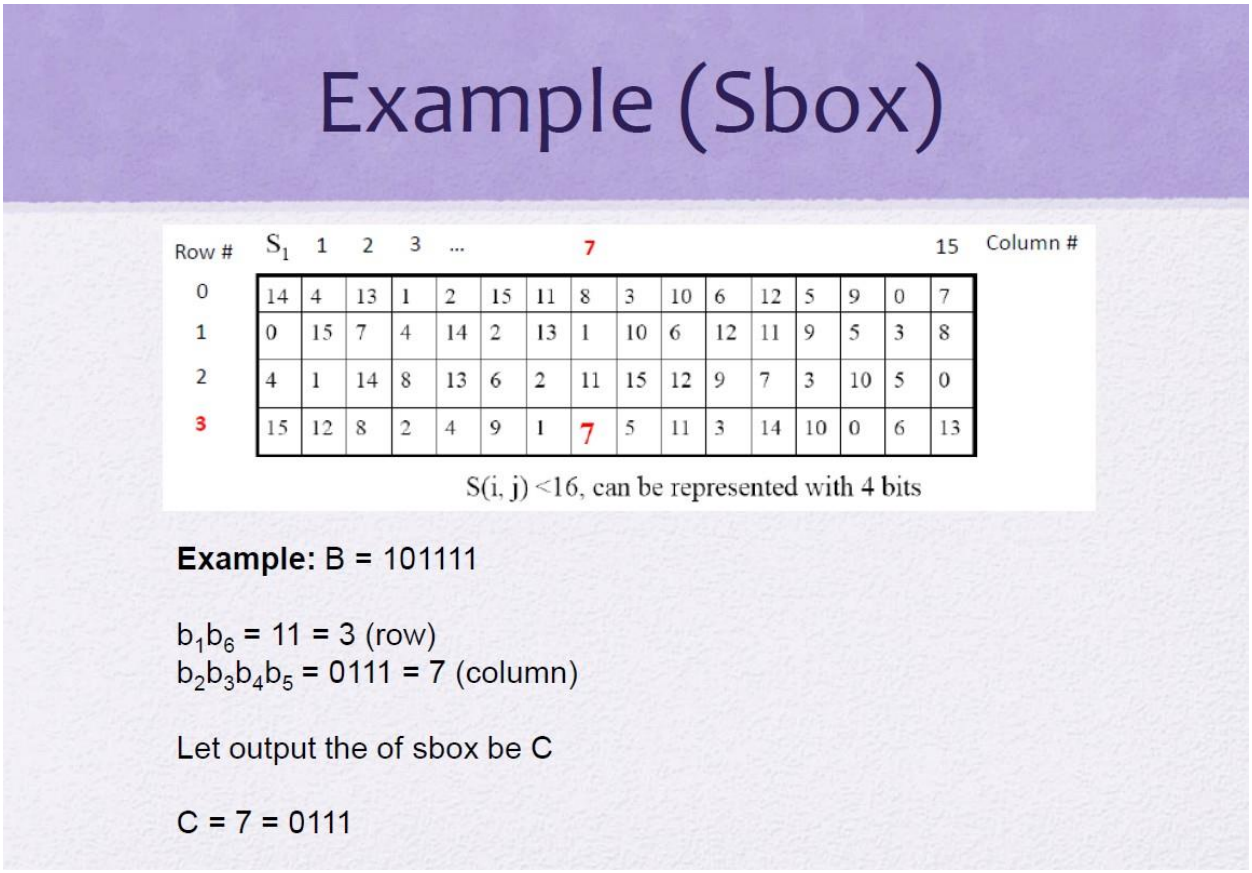
The main part of this round function is the S-box calculation. In abstract view, this takes a 48-bit binary string as input and produce a 32-bit binary string as output with the help of 8 S-boxes. Each S-box has four rows and 16 columns numbered from (0–3 and 0–15) and each row has predefined value between 0 to 15

SBOX

```
SBOX = [
# Box-1
[
[14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7],
[0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8],
[4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0],
[15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13]
],
# Box-2

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

# Box-3

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

],
```

```
# Box-4
[
[7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15],
[13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9],
[10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4],
[3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14]
],

# Box-5
[
[2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9],
[14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6],
[4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14],
[11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3]
],
# Box-6

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

],
# Box-7
[
[4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1],
[13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6],
[1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2],
[6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12]
],
# Box-8

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

To perform S-box operation the 48-bit XORed output is split into 8 part each having 6 bits. Then these 8 binary strings are passed to S-boxes one to each individual S-box. Each S-box works in a similar manner and process 6 bits as follows, the first and last bit of string is treated as row number (00, 01,10,11 or 0,1,2,3) for the S-box and middle four value is treated as column number (0000 to 1111 or 0–15). Thus having row and column number help to access the value from the S-box which will be a decimal number between 0–15 and treating these as HEX convert it to a 4-bit binary string (0000 to 1111). So, each 8 boxes will return 4-bit as output and which combined together will form 32-bit output string. In the following code snippet the split part is carried out by the use of textwrap module which makes this kind of split very easy. Then a S-box lookup method is defined which takes three parameters (boxnumber, rownumber, column_number) and return a single value as output. Below is an example of S-BOX calculation.



Final stage of Round Function

Now taking XOR'ed output 01100001000000110011001010000100011011010010011 1

and SBOX calculation results in 32 bit 01011001111100011011111101100111

After Round function Left half will be XOr'ed with Round Function Value
I.e.

```
          10110101001110111110110110101110

    ⊕     11001100000000001100110011111111
-------------------------------------------
          01111001001110110010000101010001
```

XOR'ed output will be treated as new Right half and Right half will be treated as new Left half.

These outputs will be the input of the next round.

After 16 round completion, we again have to take Final permutation of final output new Left + new Right with Final permutation Table

Final Thing

```
INVERSE_PERMUTATION_TABLE = ['40 ', '8 ', '48 ', '16 ', '56 ', '24 ', '64 ', '32',
                             '39 ', '7 ', '47 ', '15 ', '55 ', '23 ', '63 ', '31',
                             '38 ', '6 ', '46 ', '14 ',  '54 ', '22 ', '62 ', '30',
                             '37 ', '5 ', '45 ', '13 ', '53 ', '21 ', '61 ', '29',
                             '36 ', '4 ', '44 ', '12 ', '52 ', '20 ', '60 ', '28',
                             '35 ', '3 ', '43 ', '11 ', '51 ', '19 ', '59 ', '27',
                             '34 ', '2 ', '42 ', '10 ', '50 ', '18 ', '58 ', '26',
                             '33 ', '1 ', '41 ', '9 ', '49 ', '17 ', '57 ', '25']
```

This will be our final encrypted output.

National University of Computer and Emerging Sciences

FAST School of Computing | Fall 2020 | Islamabad Campus

Submission guidelines:

1- Your code must include templates.
2- All the submissions will be tested against provided test cases to assign marks.
3- In case of any discrepancy (test case failed, incomplete assignment, cheating case), you will be called for demo.
4- There is zero tolerance for the cheating cases so don't try for that.