SIL765
Assignment 1

Yash Gupta (2013CS10302), Ujjwal Sinha (2012CH70185)

Project Selection:

A1 = last_4_digits_of_entry_no_of_first_student = 0302
A2 = last_4_digits_of_entry_no_of_second_student = 0185

k = A1+A2 mod 3 = 487 mod 3 = 1
Where k is the project number. Thus, project 1 was selected.

Project 1: Implementation of DES

Problem statement:

You are required to develop a program to encrypt (and similarly decrypt) a 64-bit plaintext using DES. Instead of using an available library, I insist that you program any/every element of each of the 16 rounds of DES (and that means F-box, 32-bit exchanges, generation of sub-key required in each round).
Having done that, with one or more 64-bit plaintext(s), verify that indeed the output of the J^{th} encryption round is identical to the output of the $(16\text{-}J)^{th}$ decryption round.

DES is a ***block cipher,*** meaning that it operates on plaintext blocks of a given size (64-bits) and returns ciphertext blocks of the same size. Our version of DES uses a 56-bit private key (with additional 8 bits used to store parity of 7-bit blocks at 8^{th}, 16^{th}, … , 64^{th} bits).
We use C++ to implement and validate DES for this project.

ENCRYPTION

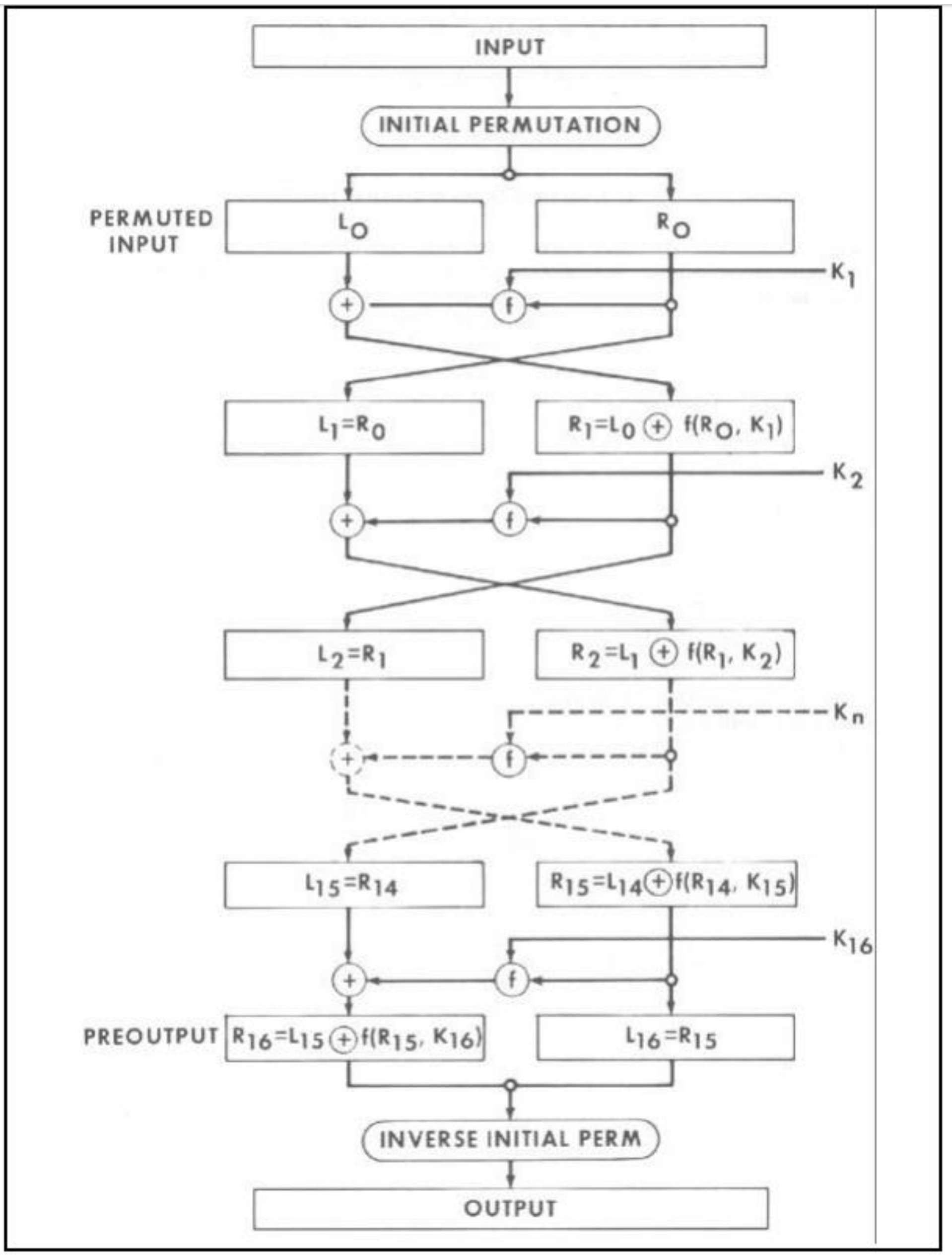

Figure 1. *Enciphering computation.*

- We have implemented a function *encrypt(plain, key)* which returns the cipher text using DES.
- Here we take plain-text to be FEED1337BEAD8787 and key to be 0E329232EA6D0D73 in hexadecimal format.
- plain: FEED1337BEAD8787

key: 0E329232EA6D0D73

- Both plain text and the key is converted in binary format and we get a 64 bit block of plain text and key. Out of 64 bits of key, 8 bits of key are redundant parity bits.
- Encrypt() proceeds in the following manner:
    - Calculate initial permutation of plaintext and split into L and R.
    - Perform 16 rounds based on fiestel cipher, with the specified DES f-box and key schedule.
    - Swap L and R, and join them.
    - Apply inverse of the initial permutation to obtain cipher text.

Step 1:

Function *initial_permutation_box(in)* permutes the input 64 bits using the permutation table defined in the DES specification:

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,
33,1,41,09,49,17,57,25

From the above table, we get the new arrangement of the bits from their initial order as such the 40^{th} bit of the plain text will become the first bit of result and similarly, 8^{th} bit of the plaintext becomes the second bit of the result.

Step 2:

After initial permutation, we divide initial permuted block into halves each of 32 bit.

Round 0 - L: 031DFBEE R: F33B33DD

Step 3:

Next we iterate through the 16 rounds, in each generating the round key using function "key_round(i, CD)" which expects two arguments (round number and internal key schedule state) and it generates the 48-bit round key "K", while updating the key schedule state (CD). Initially CD is equal to "key". Using the right 32-bit block "R" and round key "K" we execute the function "fiestel_box(R, K)" which calculates the f box as defined in DES specs to produce a 32 bit output block (H). This 32-bit output is XOR'd with "L".

Y = L XOR fiestel_box(R, K)

After this, we do a swap between Y and R for the next step. R (F33B33DD) will become the left sub half (L) after the first round and Y will become the right sub half (R). This will continue for 16 rounds.

Output for each round:

00 - L: 031DFBEE R: F33B33DD
01 - L: F33B33DD R: 51D8987D K: 36146478E1E1 CD: 0029617503EC2E3D
02 - L: 51D8987D R: A9C71419 K: 40BD1176E8FD CD: 0052C2EA07D85C7A
03 - L: A9C71419 R: 38255661 K: 45A473239DDB CD: 004B0BA81F6171E9
04 - L: 38255661 R: F7F0DE37 K: E7C4828FB533 CD: 002C2EA05D85C7A7
05 - L: F7F0DE37 R: EC67ABB5 K: 7A83826F4F64 CD: 00B0BA8146171E9F
06 - L: EC67ABB5 R: EA49F160 K: 38901B58C9DE CD: 00C2EA05285C7A7D
07 - L: EA49F160 R: D46E0D99 K: 25005EC5D49D CD: 000BA814B171E9F6

08 - L: D46E0D99 R: 8FFCB3E7 K: 264894CB36E9 CD: 002EA052C5C7A7D8
09 - L: 8FFCB3E7 R: 72EBB23C K: 54554179F633 CD: 005D40A58B8F4FB0
10 - L: 72EBB23C R: C0AF3554 K: 43C9453F4C2E CD: 007502961E3D3EC2
11 - L: C0AF3554 R: E849DBA6 K: 09E1878C79D6 CD: 00D40A5858F4FB0B
12 - L: E849DBA6 R: 3C314F65 K: 3105ABA5E2F5 CD: 0050296173D3EC2E
13 - L: 3C314F65 R: 85577A6D K: F100A1F38EC3 CD: 0040A585DF4FB0B8
14 - L: 85577A6D R: 5465A8F8 K: 918A949E871F CD: 000296175D3EC2E3
15 - L: 5465A8F8 R: EC0C0B33 K: 1432961F77C4 CD: 000A585D44FB0B8F
16 - L: EC0C0B33 R: 2EE990D4 K: 606F044C3AE7 CD: 0014B0BA89F6171E

L: Left Sub Half, R: Right Sub Half, K: Round Key, CD: key schedule state

Step 4:
In the final step, we swap the left sub half(L) and right sub half(R) of round 16 (LR: 2EE990D4EC0C0B33) and then performed an inverse permutation of LR using "inv_initial_permutation_box(in)" to get the cipher text as **1A4AE1F807D29195**.

DECRYPTION

Here we take cipher-text to be 1A4AE1F807D29195 and key to be 0E329232EA6D0D73 in hexadecimal format, and decrypt it using "decrypt(ciphertext, key)", which results in 64-bit plaintext output.

cipher-text: 1A4AE1F807D29195
key: 0E329232EA6D0D73

Both Cipher text and the key is converted in binary format and we get a 64 bit block of Cipher text and key. The cipher text is same as output of encrypt() and key is identical to the one used for encryption.

Step 1:
Initial permutation is computed using "initial_permutation_box(in)", as in "encrypt()".

Step 2:
Next, we divide initial permuted block into halves each of 32 bit and swap the left sub half (32 bits of

the cipher-text) and the right sub half (next 32 bits of cipher-text).

Step 3:
Our next step is to go through 16 iterations of decryption using “inv_key_round()” for the decryption key schedule. It expects two arguments (round number and internal key_schedule state) and it generates the key “K” for that round.

For i^{th} round of decryption, we use the same key which was used in i^{th} round of encryption.

Using the round key “K” we perform the inverse of the operation performed while encryption, which operates updates “L” and “R” using “K”.
Y = R XOR fiestel_box(L,K)

After this step, we do a swap between Y and R for the next step. L (EC0C0B33) becomes the right sub block “R” after the first round and Y becomes the left sub-block “L” for the next round.
This continues for 16 rounds.

Output for each round:

00 - L: EC0C0B33 R: 2EE990D4
01 - L: 5465A8F8 R: EC0C0B33 K: 606F044C3AE7 CD: 000A585D44FB0B8F
02 - L: 85577A6D R: 5465A8F8 K: 1432961F77C4 CD: 000296175D3EC2E3
03 - L: 3C314F65 R: 85577A6D K: 918A949E871F CD: 0040A585DF4FB0B8
04 - L: E849DBA6 R: 3C314F65 K: F100A1F38EC3 CD: 0050296173D3EC2E
05 - L: C0AF3554 R: E849DBA6 K: 3105ABA5E2F5 CD: 00D40A5858F4FB0B
06 - L: 72EBB23C R: C0AF3554 K: 09E1878C79D6 CD: 007502961E3D3EC2
07 - L: 8FFCB3E7 R: 72EBB23C K: 43C9453F4C2E CD: 005D40A58B8F4FB0
08 - L: D46E0D99 R: 8FFCB3E7 K: 54554179F633 CD: 002EA052C5C7A7D8
09 - L: EA49F160 R: D46E0D99 K: 264894CB36E9 CD: 000BA814B171E9F6
10 - L: EC67ABB5 R: EA49F160 K: 25005EC5D49D CD: 00C2EA05285C7A7D
11 - L: F7F0DE37 R: EC67ABB5 K: 38901B58C9DE CD: 00B0BA8146171E9F
12 - L: 38255661 R: F7F0DE37 K: 7A83826F4F64 CD: 002C2EA05D85C7A7
13 - L: A9C71419 R: 38255661 K: E7C4828FB533 CD: 004B0BA81F6171E9
14 - L: 51D8987D R: A9C71419 K: 45A473239DDB CD: 0052C2EA07D85C7A
15 - L: F33B33DD R: 51D8987D K: 40BD1176E8FD CD: 0029617503EC2E3D
16 - L: 031DFBEE R: F33B33DD K: 36146478E1E1 CD: 0014B0BA89F6171E
LR: 031DFBEEF33B33DD
plain: FEED1337BEAD8787

L: Left sub-block, R: Right sub-block, K: round key, CD: Internal key_schedule state

Step 4:
In the final step, we combine L and R into LR (031DFBEEF33B33DD) and then perform the inverse initial permutation on it to get the plain text as **FEED1337BEAD8787**.

FIESTEL BOX

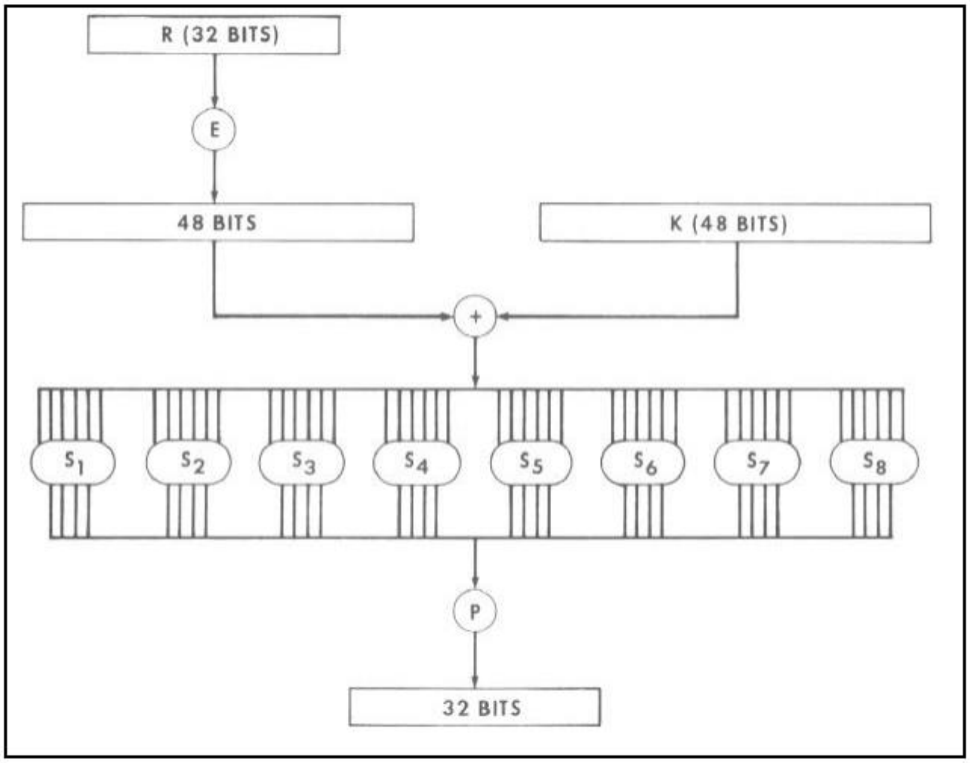

Figure 2. *Calculation of f*(R, K)

- Fiestel box is implemented by the function “fiestel_box(R,K)” which returns 32-bit output block from 32 bit input right sub-block “R” and 48 bit round key “K”.
- We first expand R to 48-bits using “expansion_box(in)” and obtain E.
- Then the expanded block “E” and round key “K”are XOR’d
- Then each 6-bit sub-block of the result is passed through a S_i-box with i = 1…8
- S_i box is implemented by the function “selection_box(i, in)”.
- Finally the function “permutation_box(in)” calculates the final 32-bit output of fiestel_box().

KEY SCHEDULE

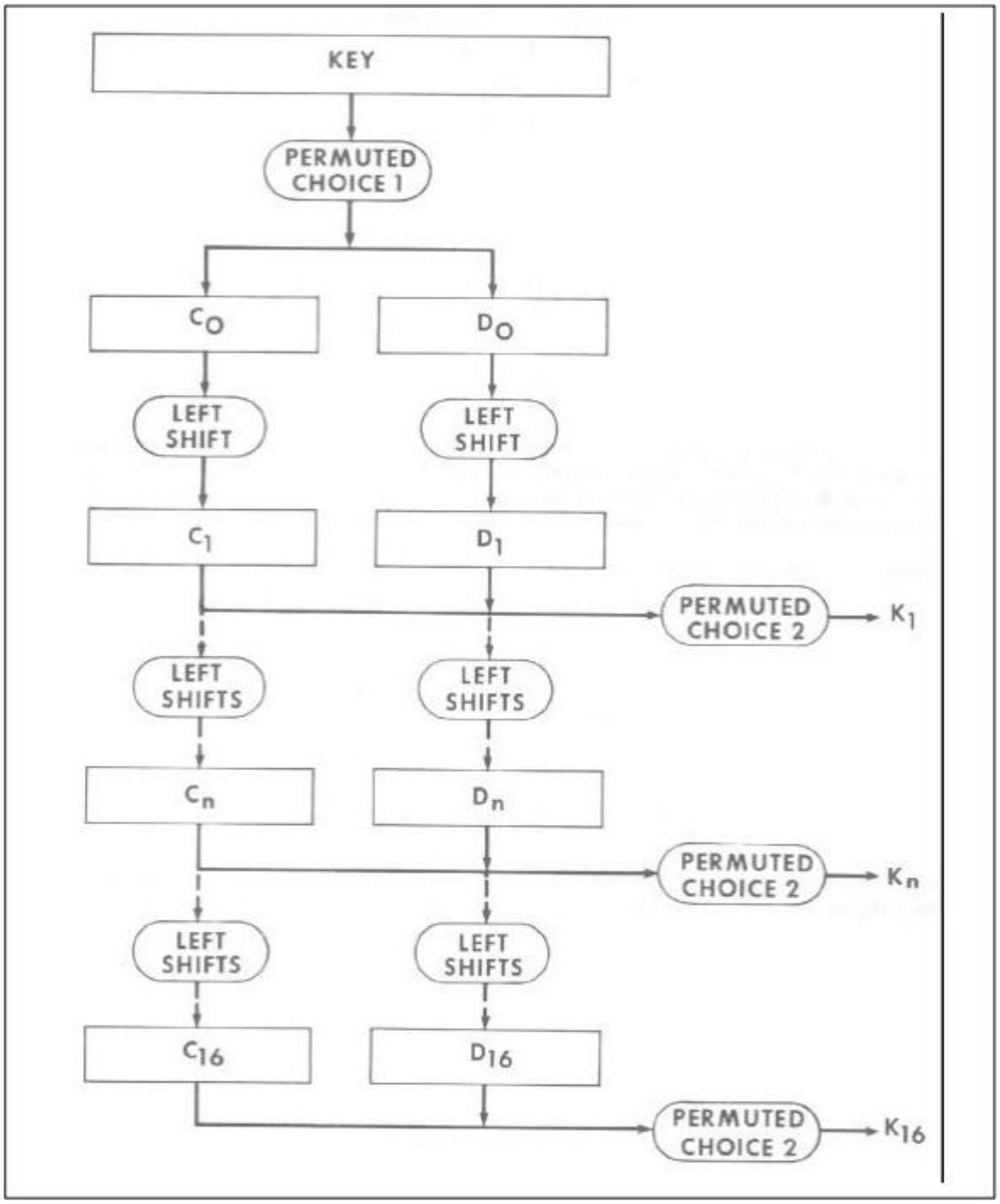

Figure 3. *Key schedule calculation*

- Key schedule is generated by functions “key_round(i, CD)” and “inv_key_round(i,CD)” for encryption and decryption respectively. Here I is the round number and CD is the internal state.
- Initially, CD is equal to the input key.
- In the first round, “permutation_choice1_box(in)” generates the actual CD as per specs.
- For each round, K is generated using “permutation_choice2_box(in)” and internal state CD is updated by left-circular shifts following a shifting schedule as defined in the specs.
- Functions “shift_boxes(count, CD)” and “inv_shift_boxes(count, CD)” perform this action during encryption and decryption respectively.

VALIDATION

To validate the DES implementation, output of the J^{th} encryption round should be identical to the output of the $(16-J)^{th}$ decryption round.

In function “main()”, we verify this by storing intermediate outputs from encryption and decryption and asserting *encrypt_outputs[i] == decypt_outputs[16-i]*, for each i.

For instance, let J = 15: It means output of 15^{th} encryption round should be identical to the output of the 1^{st} round of decryption.

Output of 15^{th} encryption round:
15 - L: 5465A8F8 R: EC0C0B33

Output of 1^{st} decryption round:
01 - L: 5465A8F8 R: EC0C0B33

RESULTS

All assertions in the program are passed and thus the implementation of DES is validated.