Lab 2 Help

Nate Lannan

Oklahoma State University

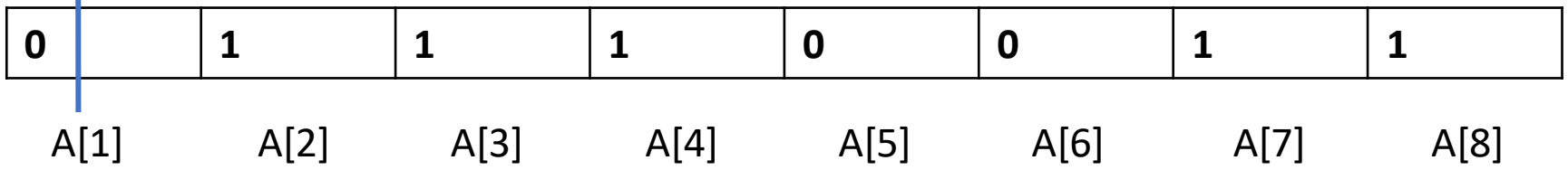
Electrical and Computer Engineering Department

Stillwater, OK 74078 USA

Bit Numbering Convention

Our convention for an arbitrary 8 bit field:
Logic [7:0] A = 8'b01110011;

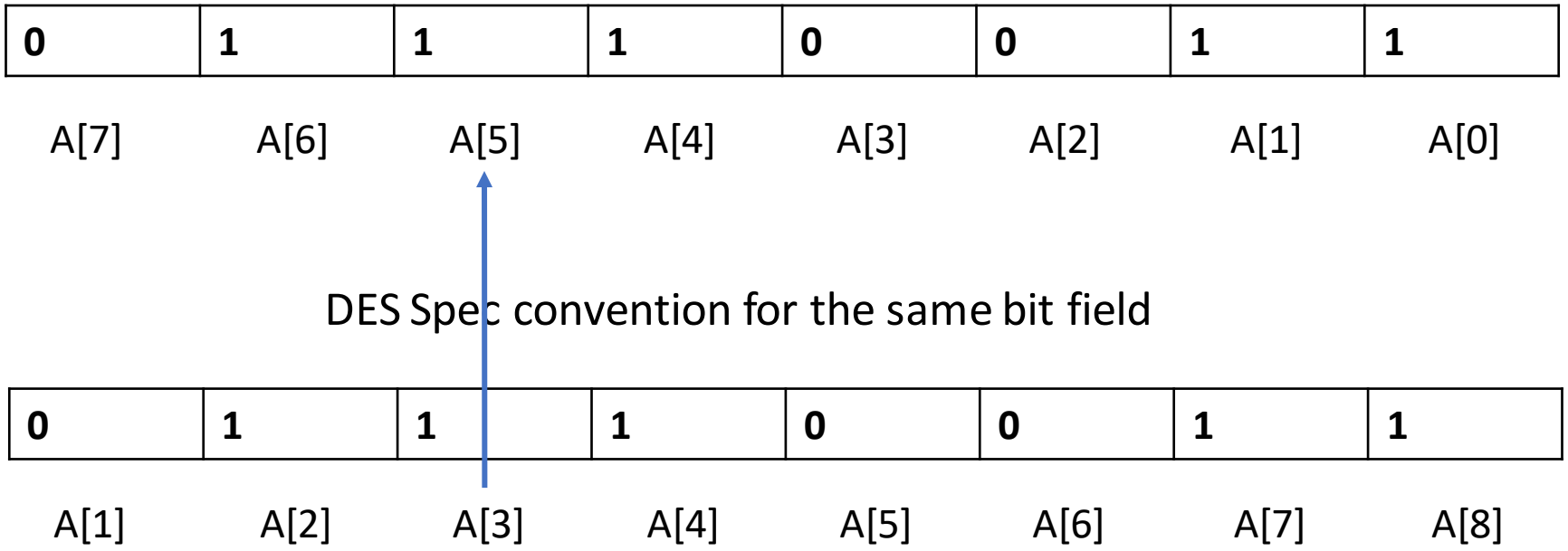
DES Spec convention for the same bit field



Bit Numbering Convention

Reconciling the different number conventions:

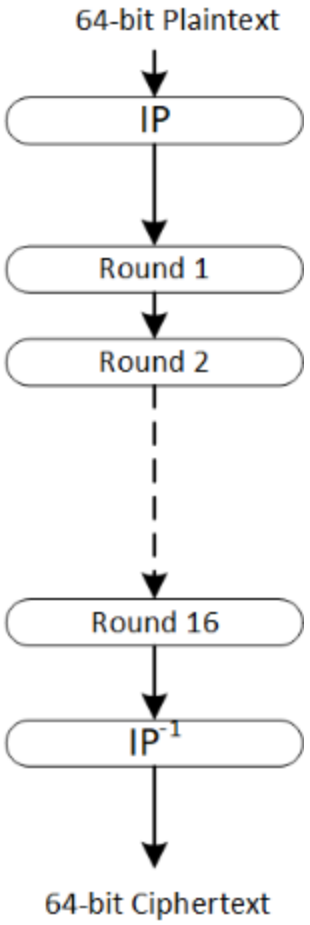
- Bit # in our numbering system = (Max number of the DES numbering) – (DES value we want to convert)



What is A[3] from DES numbering to our numbering scheme?

- Max number = 8, DES Value = 3
- Bit number in our numbering scheme = 8 – 3 = 5

Bit Numbering Convention IP Block



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

Table 5: Initial Permutation (IP) Function

```
assign out_block[63] = inp_block[64-58];
assign out_block[62] = inp_block[64-50];
assign out_block[61] = inp_block[64-42];
assign out_block[60] = inp_block[64-34];
assign out_block[59] = inp_block[64-26];
assign out_block[58] = inp_block[64-18];
assign out_block[57] = inp_block[64-10];
assign out_block[56] = inp_block[64-2];
assign out_block[55] = inp_block[64-60];
assign out_block[54] = inp_block[64-52];
assign out_block[53] = inp_block[64-44];
assign out_block[52] = inp_block[64-36];
assign out_block[51] = inp_block[64-28];
assign out_block[50] = inp_block[64-20];
assign out_block[49] = inp_block[64-12];
assign out_block[48] = inp_block[64-4];
assign out_block[47] = inp_block[64-62];
assign out_block[46] = inp_block[64-54];
assign out_block[45] = inp_block[64-46];
assign out_block[44] = inp_block[64-38];
assign out_block[43] = inp_block[64-30];
assign out_block[42] = inp_block[64-22];
assign out_block[41] = inp_block[64-14];
assign out_block[40] = inp_block[64-6];
assign out_block[39] = inp_block[64-64];
assign out_block[38] = inp_block[64-56];
assign out_block[37] = inp_block[64-48];
assign out_block[36] = inp_block[64-40];
assign out_block[35] = inp_block[64-32];
assign out_block[34] = inp_block[64-24];
assign out_block[33] = inp_block[64-16];
assign out_block[32] = inp_block[64-8];
assign out_block[31] = inp_block[64-57];
assign out_block[30] = inp_block[64-49];
assign out_block[29] = inp_block[64-41];
assign out_block[28] = inp_block[64-33];
assign out_block[27] = inp_block[64-25];
assign out_block[26] = inp_block[64-17];
```

Where Should I Start?

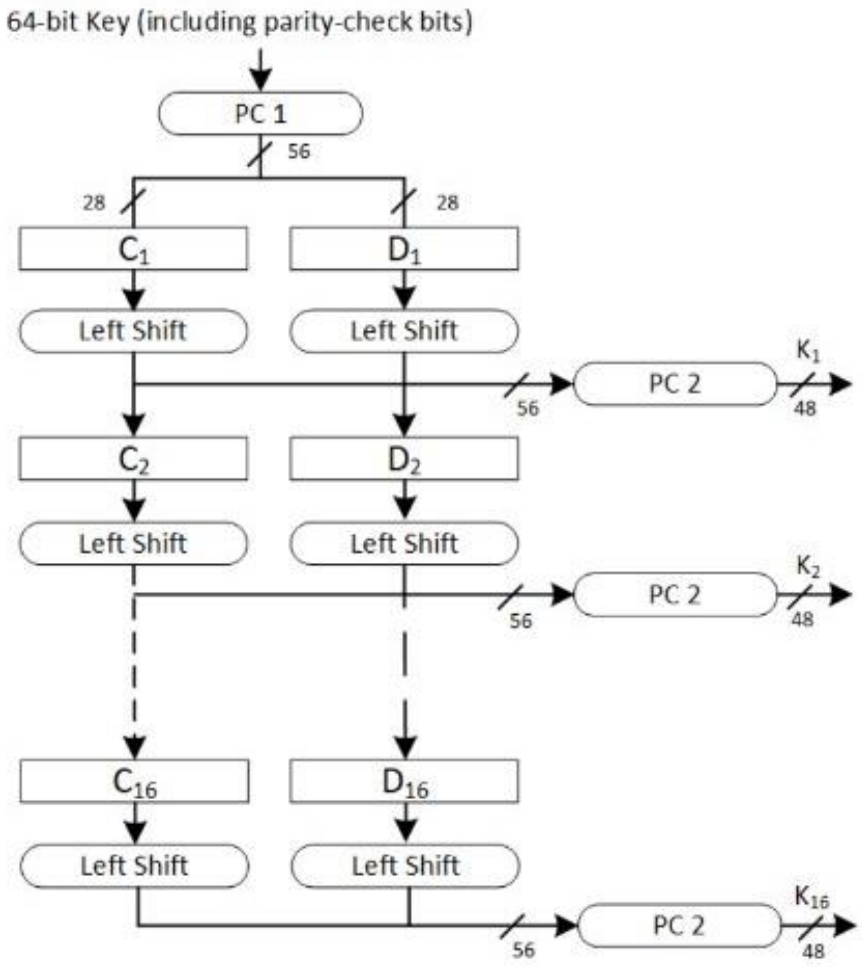
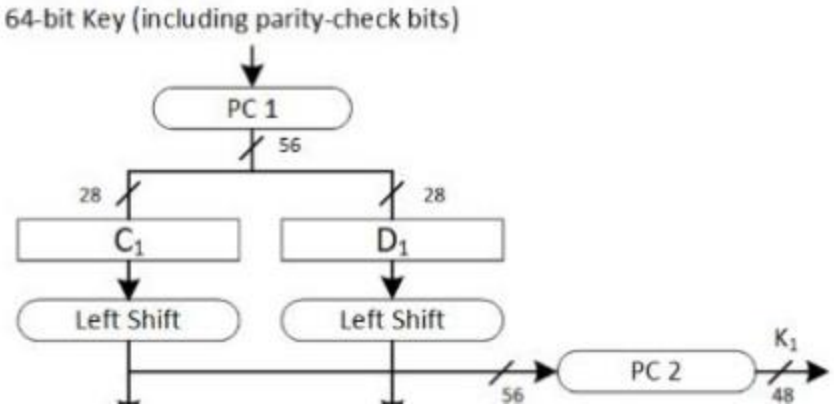


Figure 3: DES SubKey Permutation Diagram



Break anything you do in engineering into bite sized pieces. This way we can just repeat what we have done.

First step: Implement PC 1

PC 1 shuffles bits almost exactly like the IP block, except it throws away parity bits and splits the result in half

Where Should I Start?

Left						
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
Right						
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

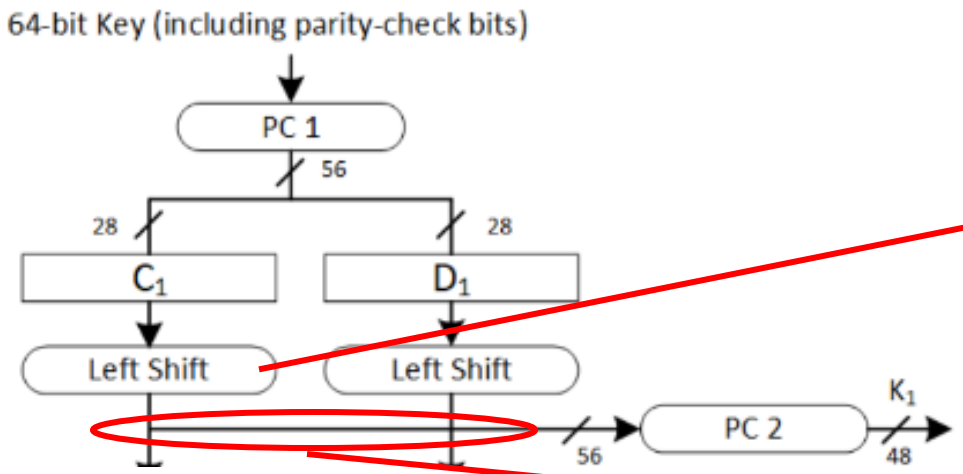
Table 3: Permutation Choice 1 (PC-1) Function[2]

```
module PC1 (key, left_block, right_block);

  input logic [63:0]  key;
  output logic [27:0] left_block;
  output logic [27:0] right_block;

  assign left_block[27] = key[64-57];
  assign left_block[26] = key[64-49];

  //fill in the rest

  assign right_block[27] = key[64-63];
  assign right_block[26] = key[64-55];
```

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

Table 4: Permutation Choice 2 (PC-2) Bit Function

Left Shifting and Concatenation



ShiftedC1 = {C1[26:0],C1[27]} //1 bit left circular shift

logic [55:0] Concatenated;
Concatenated = {C1,D1}

Extend to All Rounds

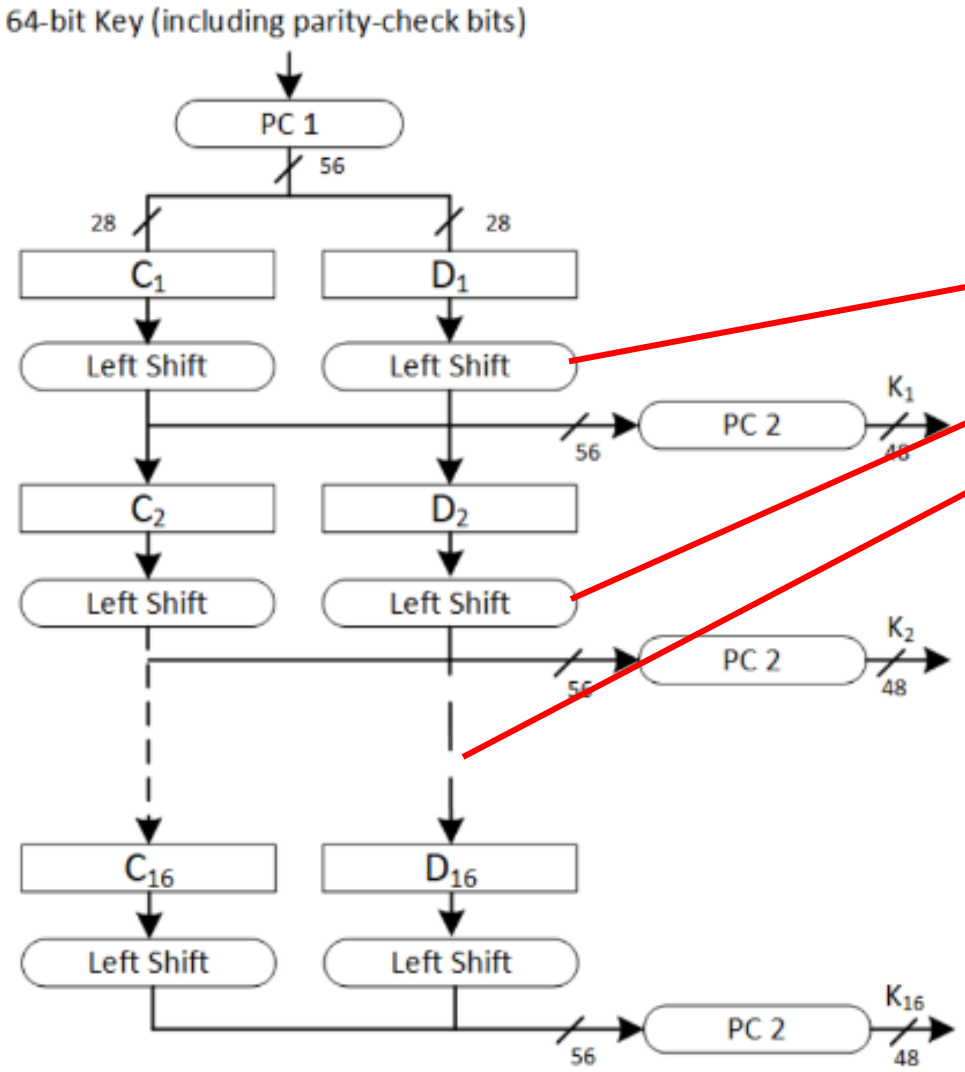


Figure 3: DES SubKey Permutation Diagram

Iteration #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Left Shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 2: Left Rotation within SubKey Generation

Shifted= {C3[25:0],C3[27:26]} //2 bit left circular shift

Note: Remember that the input to the next round is the shifted values from the previous round NOT the original output from PC1

How do we check our results?

```
PS C:\Users\amerigo\Documents\GitHub\dldLab2_Spring23\Lab2\java> javac -d . -classpath . DES.java
PS C:\Users\amerigo\Documents\GitHub\dldLab2_Spring23\Lab2\java> java DES
Original plain Text:    2579DB866C0F528C
Key:                    433E4529462A4A62

Encryption:

After initial permutation: 5646B9278C13B66C
After splitting: L0=5646B927 R0=8C13B66C

Round 1         8C13B66C F3EFC169 208066A253BA
Round 2         F3EFC169 25DAF255 C0B6508F6DC2
Round 3         25DAF255 1890CFBF 44D6422CC355
Round 4         1890CFBF AFB98FA0 62D142D3C4C6
Round 5         AFB98FA0 8F76DBD7 28C143CC8789
Round 6         8F76DBD7 C176D0E5 21411B9A764D
Round 7         C176D0E5 C7401A8C 2501917AD3A0
Round 8         C7401A8C B748825A 170891906D2B
Round 9         B748825A 61239171 084949255DD5
Round 10        61239171 FE28B577 01690D8B80F3
Round 11        FE28B577 CDB650DE 012D81C7CF05
Round 12        CDB650DE 8B8270E5 512CA11A07DC
Round 13        8B8270E5 DDDBEE19 D1A480D9D185
Round 14        DDDBEE19 5F82D63F 5086864266A9
Round 15        5F82D63F B35B4964 709006FA390D
Round 16        B35B4964 850AC7BE C03E202F8437

Cipher Text: FCB54739A1832EC5
```

Original text (unencrypted)

64 bit Key (in Hex) to send to GenerateKeys()

Output of IP Block

C1, D1, and SubKey1

Encrypted Output

Encryption Using the SubKeys

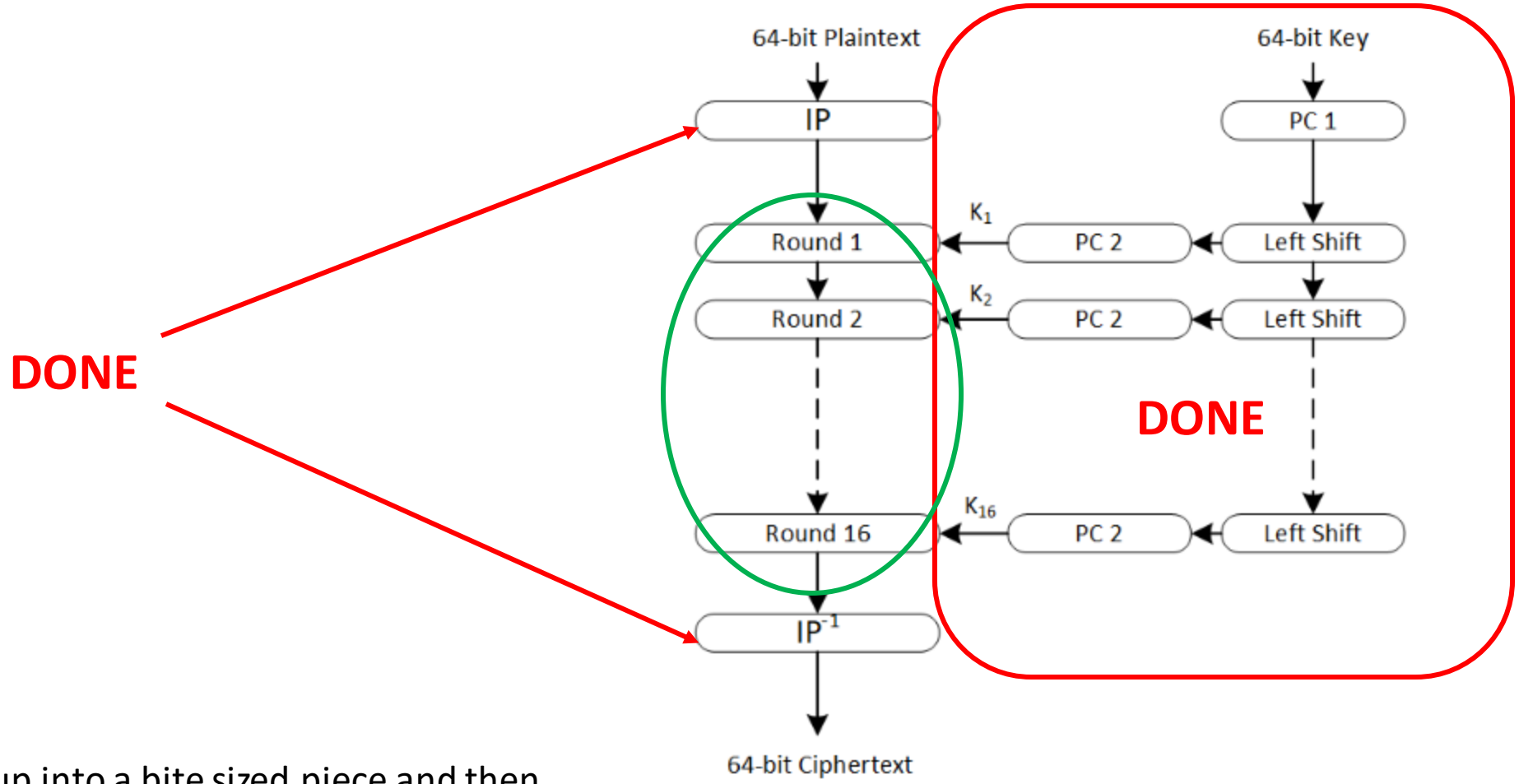


Figure 2: DES Block Diagram

Rounds: Break up into a bite sized piece and then instantiate it 16 times

Round

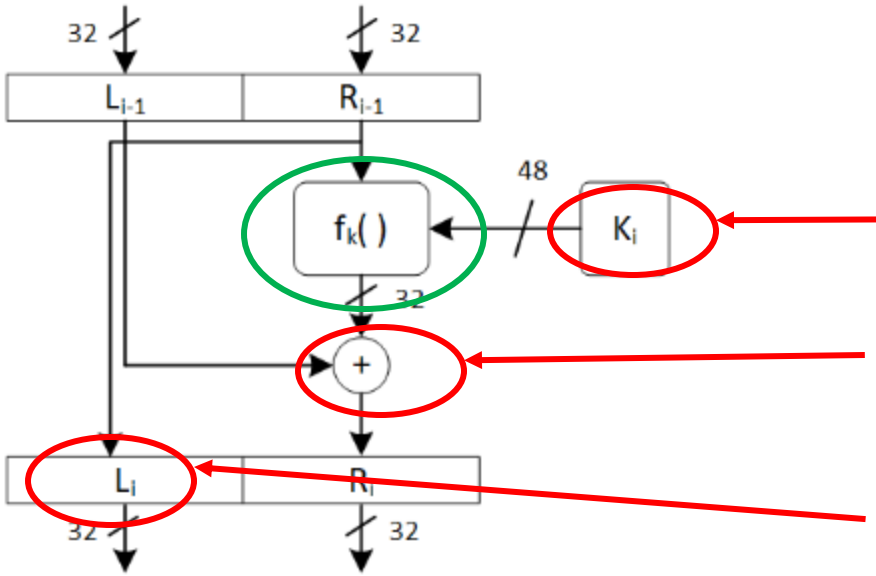


Figure 5: DES Round Block Diagram

SubKey from GenerateKeys()

Bitwise Exclusive OR
Ri = feistelOut ^ Li-1;

Just Swapping the lower 32 bits with upper 32 bits
Li = Ri-1;

Feistel Block

Bitwise Exclusive OR

Already implemented
Just need to break up 48-bit input into 8 6-bit inputs to feed each Sbox

Concatenate all the 4-bit outputs into 1 32-bit number:
StraightIn = {S8_out, S7_out, …, S1_out};

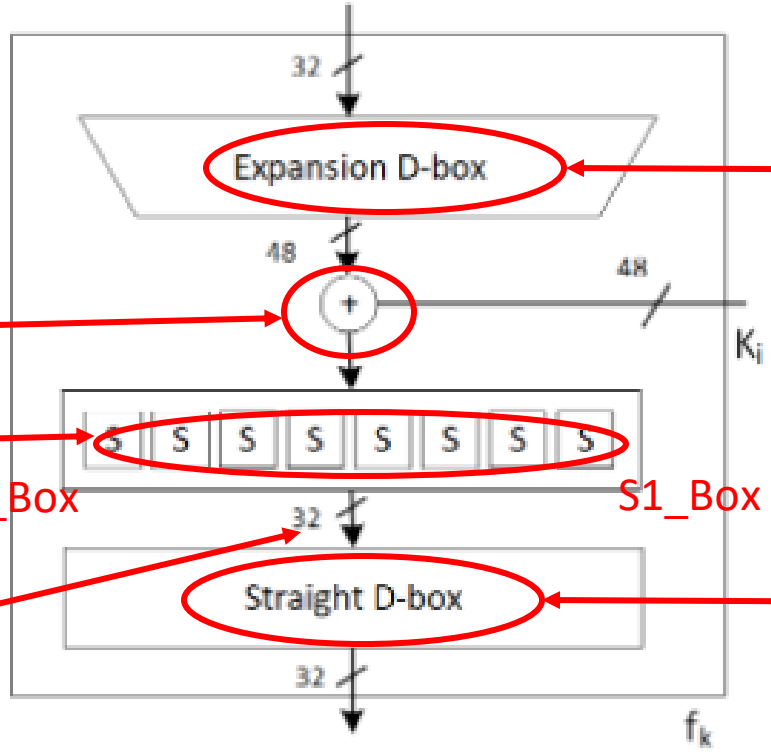


Figure 6: Feistel (f_K) Block Diagram

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

Table 7: Expansion Function or Permutation (EP) Function

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Table 8: Straight Diffusion-box (D-box) Function

Nearing the end

- Repeat for all 16 rounds
- Output of 16th round should be the input of FP()
- Output of FP will be the encrypted data
- Create an input to your sytem which dictates if you are encrypting or decrypting. Modify your system so that you can encrypt or decrypt based on this signal.
  - Hint1: the difference between encrypting and decrypting is just reversing the order of the subkeys.
  - Hint2: the ternary operator can operate on multiple bitfields at once if you use concatenation:
    - assign {resultA, resultB} = decide ? {option1A, option1B} : {option2A, option2B};

Extra Credit – DES in CBC Mode

There several modes in DES encryption, implement a switch input that selects between ECB (Electronic Code Book - Default) and CBC (Cipher Block Chaining) modes.

- CBC encryption process also requires a 64-bit Initialization Vector (IV), otherwise the process is the same

The definitive step in CBC is to XOR the plaintext block with an IV – a unique, fixed-length conversion function – to create a random, or pseudorandom, output. That is, it is simply that the plaintext is XOR'ed with the Iniatialization Vector (IV) right in the beginning of the process.

$$Plaintext_{new} = IV \oplus Plaintext_{input}$$

- The key to this is that the IV must be processed in the beginning during encryption and towards the end in decryption.