The DES Algorithm Illustrated

J. Orlin Grabbe

DES Algorithm Overview

DES is a symmetric block cipher that encrypts 64-bit blocks using a 56-bit key through 16 Feistel rounds. Each round uses a different 48-bit subkey derived from the main key.

- **Block size:** 64 bits
- **Key size:** 56 bits (64 bits with parity)
- **Rounds:** 16
- **Structure:** Feistel network

Example Parameters

We will use the following example throughout the explanation:

```
Plaintext M = 0123456789ABCDEF
Key K = 133457799BBCDFF1
```

In binary format:

$$M = 0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000\ 1001\ 1010\ 1011\ 1100\ 1101\ 1110\ 1111$$

$$K = 00010011\ 00110100\ 01010111\ 01111001\ 10011011\ 10111100\ 11011111\ 11110001$$

1 Key Generation Process

Key Generation Steps:

1. **PC-1 Permutation: Convert 64-bit key to 56-bit by dropping every 8th bit**
2. **Split: Divide into two 28-bit halves (C and D)**
3. **Left Shifts: Rotate each half according to shift schedule (16 rounds)**
4. **PC-2 Compression: Combine and compress 56 bits to 48-bit subkeys**

1.1 PC-1: Initial Key Permutation

The 64-bit key is permuted using PC-1 to 56 bits:

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

```
K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001
K+ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111
```

1.2 Initial Splitting

Split K+ into two 28-bit halves:

$$C_0 = 1111000\ 0110011\ 0010101\ 0101111$$
$$D_0 = 0101010\ 1011001\ 1001111\ 0001111$$

1.3 Left Shift Schedule

With C_0 and D_0 defined, we now create sixteen blocks C_n and D_n, $1 \leq n \leq 16$. Each pair of blocks C_n and D_n is formed from the previous pair C_{n-1} and D_{n-1}, respectively, for $n = 1, 2, \ldots, 16$, using the following schedule of "left shifts" of the previous block.

Left Shift Operation: Move each bit one place to the left, except for the first bit, which is cycled to the end of the block.

Rounds	Shifts	Pattern
1, 2, 9, 16	1	Single shift
3–8, 10–15	2	Double shift

1.3.1 Key Rotation Examples

$$C_0 = 11110000110011001010101011111$$
$$D_0 = 01010101011001100111100011111$$
$$C_1 = 11100001100110010101010111111$$
$$D_1 = 10101010110011001111100011110$$
$$C_2 = 11000011001100101010101111111$$
$$D_2 = 01010101100110011111000111101$$
$$C_3 = 00001100110010101010111111111$$
$$D_3 = 01010110011001111100011110101$$

1.4 PC-2: Subkey Permutation

Generate 16 subkeys using PC-2 permutation. For each round, the previous C_n and D_n are concatenated to form a 56-bit block, which is then compressed to 48 bits using the PC-2 table:

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

1.4.1 Generated Subkeys

$$K_1 = 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010$$
$$K_2 = 011110\ 011010\ 111011\ 011001\ 110110\ 111100\ 100111\ 100101$$
$$K_3 = 010101\ 011111\ 110010\ 001010\ 010000\ 101100\ 111110\ 011001$$
$$\vdots$$
$$K_{16} = 110010\ 110011\ 110110\ 001011\ 000011\ 100001\ 011111\ 110101$$

2 Data Encryption Process

Data Encryption Steps:

1. **Initial Permutation (IP): Rearrange 64-bit plaintext block**
2. **Split: Divide into 32-bit left and right halves (L and R)**
3. **16 Rounds: For each round:**
   - **Expand right half from 32 to 48 bits (E-table)**
   - **XOR with round subkey**
   - **S-box substitution (48 bits → 32 bits)**
   - **Permutation (P-table)**
   - **XOR with left half**
4. **Final Swap: Swap left and right halves**
5. **Final Permutation (IP^{-1}): Produce ciphertext**

2.1 Initial Permutation (IP)

Rearranges bits of 64-bit plaintext 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

```
M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010
```

Split into halves:

$$L_0 = 1100\ 1100\ 0000\ 0000\ 1100\ 1100\ 1111\ 1111$$

$$R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$$

3 The 16-Round Iterative Process

We now proceed through 16 iterations, for $1 \leq n \leq 16$, using a function f which operates on two blocks–a data block of 32 bits and a key K_n of 48 bits–to produce a block of 32 bits. Let $+$ denote XOR addition (bit-by-bit addition modulo 2). Then for n going from 1 to 16 we calculate:

$$L_n = R_{n-1}$$
$$R_n = L_{n-1} + f(R_{n-1}, K_n)$$

Round Function Steps:

1. **Expansion (E): Expand 32-bit R_{n-1} to 48 bits using expansion table E**
2. **XOR with Subkey: Compute $E(R_{n-1}) \oplus K_n$ (48-bit result)**
3. **S-box Substitution: Split into 8 groups of 6 bits, each processed by S-boxes S_1 to S_8 to produce 8 groups of 4 bits (32-bit total)**
4. **Permutation (P): Apply permutation P to the 32-bit S-box output**
5. **XOR with Left Half: Compute $L_{n-1} \oplus f(R_{n-1}, K_n)$**
6. **Swap Halves: Set $L_n = R_{n-1}$ and $R_n =$ XOR result for next round**

3.1 Expansion (E) Table

Expands the 32-bit R_{n-1} block to 48-bit output by duplicating and rearranging bits according to the following 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

Expansion Process:

- Each row shows which bit from R_{n-1} goes to each output position
- Some bits are used multiple times (duplicated)
- Output is organized as 8 blocks of 6 bits each

3.1.1 Expansion Example

From our Round 1 calculation with R_0:

$$R_0 = \texttt{1111 0000 1010 1010 1111 0000 1010 1010}$$
$$E(R_0) = \texttt{011110 100001 010101 010101 011110 100001 010101 010101}$$

How the expansion works:

- The first output block 011110 comes from bits: 32,1,2,3,4,5 of R_0
- Bit 32 of R_0 is 1 $\rightarrow$ becomes bit 1 of output
- Bit 1 of R_0 is 1 $\rightarrow$ becomes bit 2 of output
- Bit 2 of R_0 is 1 $\rightarrow$ becomes bit 3 of output
- Bit 3 of R_0 is 1 $\rightarrow$ becomes bit 4 of output
- Bit 4 of R_0 is 0 $\rightarrow$ becomes bit 5 of output
- Bit 5 of R_0 is 0 $\rightarrow$ becomes bit 6 of output

Note that bits 32 and 1 appear at both the beginning and end of the expansion table, creating the duplication effect.

Each block of 4 original bits from R_{n-1} has been expanded to a block of 6 output bits in $E(R_{n-1})$.

3.2 XOR with Subkey

Next in the f calculation, we XOR the output $E(R_{n-1})$ with the key K_n:

$$K_n + E(R_{n-1})$$

3.2.1 XOR Example

From our Round 1 calculation:

$$K_1 = \texttt{000110 110000 001011 101111 111111 000111 000001 110010}$$
$$E(R_0) = \texttt{011110 100001 010101 010101 011110 100001 010101 010101}$$
$$K_1 + E(R_0) = \texttt{011000 010001 011110 111010 100001 100110 010100 100111}$$

The result of $K_n + E(R_{n-1})$ is written as 8 blocks of 6 bits each:

$$K_n + E(R_{n-1}) = B_1B_2B_3B_4B_5B_6B_7B_8$$

where each B_i is a 6-bit block that will be processed by the corresponding S-box S_i.

3.3 S-boxes

Each S-box maps 6 bits to 4 bits. We now calculate:

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$$

where $S_i(B_i)$ refers to the output of the i-th S-box.

The tables defining the functions $S_1, \ldots, S_8$ are shown below:

S-Box 1

Row	Column															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-Box 2

Row	Column															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S-Box 3

Row	Column															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S-Box 4

Row	Column															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S-Box 5

Row	Column															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S-Box 6

Row	Column															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S-Box 7

Row	Column															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S-Box 8

Row	Column															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

S-box Lookup Method:

- The **first and last bits** of the 6-bit input determine the **row** (0-3)
- The **middle 4 bits** determine the **column** (0-15)
- Look up the number at the intersection of row and column
- Convert the number (0-15) to a 4-bit binary output

3.3.1 S-box Example

For input block $B = 011011$:

- First bit = 0, last bit = 1 → Row = 01 = 1
- Middle 4 bits = 1101 = 13 → Column = 13
- In S1, row 1, column 13 → Value = 5
- 5 in binary = 0101
- Therefore, $S_1(011011) = 0101$

3.3.2 Round 1 S-box Application

From our previous calculation: $K_1 + E(R_0) =$ 011000 010001 011110 111010 100001 100110 010100 10

Applying each S-box:

$$S_1(011000) = 0101 \qquad S_2(010001) = 1100$$
$$S_3(011110) = 1000 \qquad S_4(111010) = 0010$$
$$S_5(100001) = 1011 \qquad S_6(100110) = 0101$$
$$S_7(010100) = 1001 \qquad S_8(100111) = 0111$$

Final S-box output: 0101 1100 1000 0010 1011 0101 1001 0111

3.4 Permutation P

The final stage in the calculation of f is to do a permutation P of the S-box output to obtain the final value of f:

$$f = P(S_1(B_1)S_2(B_2)\ldots S_8(B_8))$$

The permutation P yields a 32-bit output from a 32-bit input by permuting the bits of the input block. The permutation table is defined as follows:

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

Example: If the S-box output before permutation P is:

$$f = 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$$

3.5 Completing the Round

After computing the round function $f(R_{n-1}, K_n)$ through all the previous steps, the final operations complete round n and prepare the data for the next round:

Final Round Steps:

XOR with Left Half: Combine the round function output with the left half:

$$R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$$

4 Final Permutation and Result

Final Steps:

1. **Swap Halves: After 16 rounds, swap L_{16} and R_{16} to undo the last swap**
2. **Final Permutation (IP^{-1}): Apply inverse of initial permutation**
3. **Output: 64-bit ciphertext block**

4.1 Final Swap

After completing all 16 rounds, we perform a final swap of the halves. This is necessary because in the Feistel network structure, each round ends with swapping the halves. After 16 rounds, we would have an odd number of swaps, so we need one final swap to return to the original left-right order:

$$\text{Final Block} = R_{16}L_{16}$$

This ensures that the decryption process can correctly reverse the encryption using the same algorithm.

4.2 Final Permutation (IP^{-1})

The final permutation IP^{-1} is the inverse of the initial permutation IP applied at the beginning. It rearranges the 64-bit block according to the following 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

Example:

$$R_{16}L_{16} = \texttt{00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100}$$
$$IP^{-1}(R_{16}L_{16}) = \texttt{10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101}$$
$$C = \texttt{85E813540F0AB405}$$

The output is the final 64-bit ciphertext block, completing the DES encryption process.