CSc 466/566

Computer Security

6 : Cryptography — Symmetric Key

Version: 2012/02/22 16:14:16

Department of Computer Science
University of Arizona

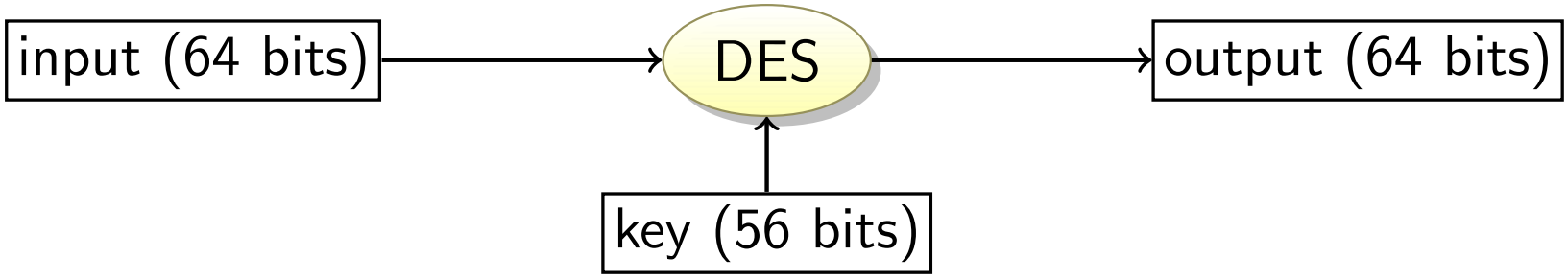
collberg@gmail.com

Christian Collberg

Outline

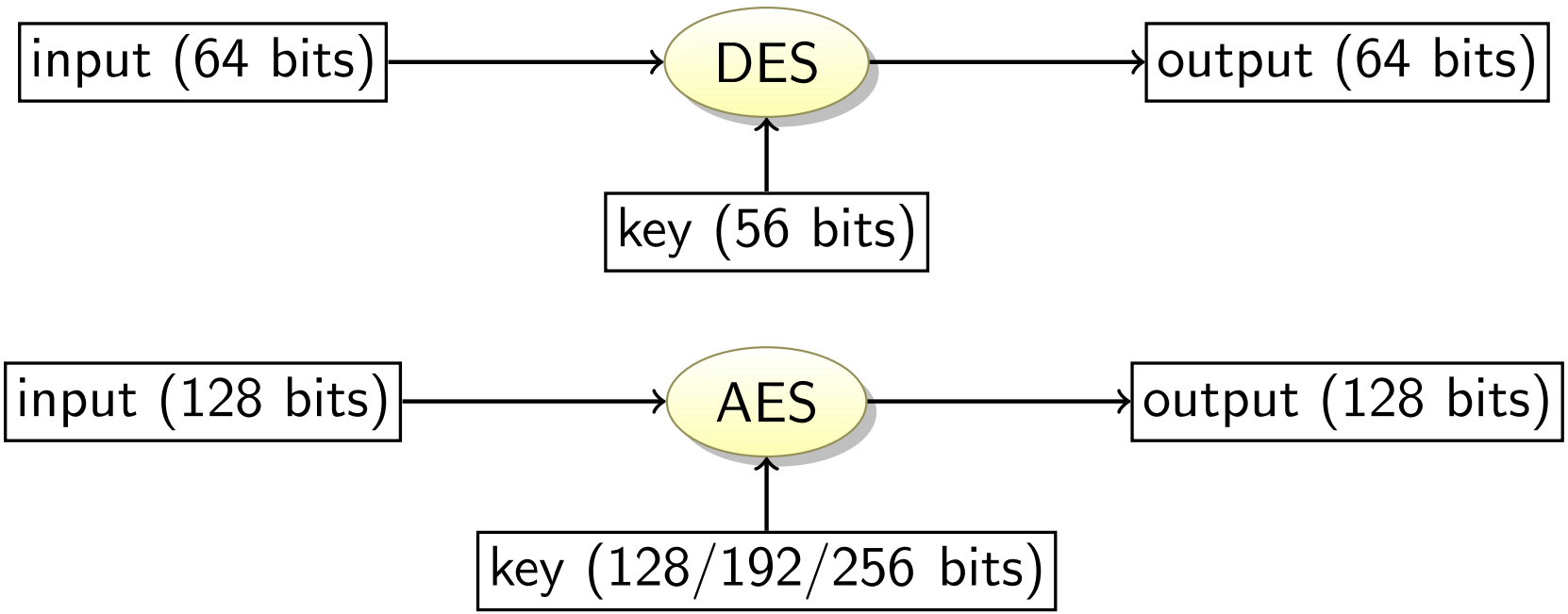
Block Ciphers

- Block ciphers work on one block of data at a time. Different ciphers use different block size and key length:



Block Ciphers

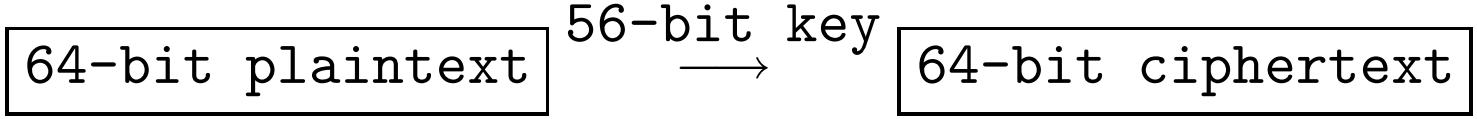
- Block ciphers work on one block of data at a time. Different ciphers use different block size and key length:



Outline

DES

- *Data Encryption Standard*. Designed by IBM and "certified" by NSA. Widely used in industry. The NSA is rumored to be able to break it in 3–15 minutes.
- A block cipher. Blocks are 64 bits, keys 56 bits:



- DES is a symmetric algorithm; the same key and algorithm is used for encryption and decryption.
- The running example has been taken from J. Orlin Grabbe, The DES Algorithm Illustrated, http://orlingrabbe.com/des.htm.

DES

- RSA Conference 2011 Keynote - The Cryptographers' Panel:

  http://www.youtube.com/watch?v=0NlZpyk3PKI

Confusion and Diffusion

- DES is a combination of two basic principles:

  confusion: Confusion scrambles up the letters of the plaintext so that it has no direct relationship to the ciphertext. Substitution ciphers do this.

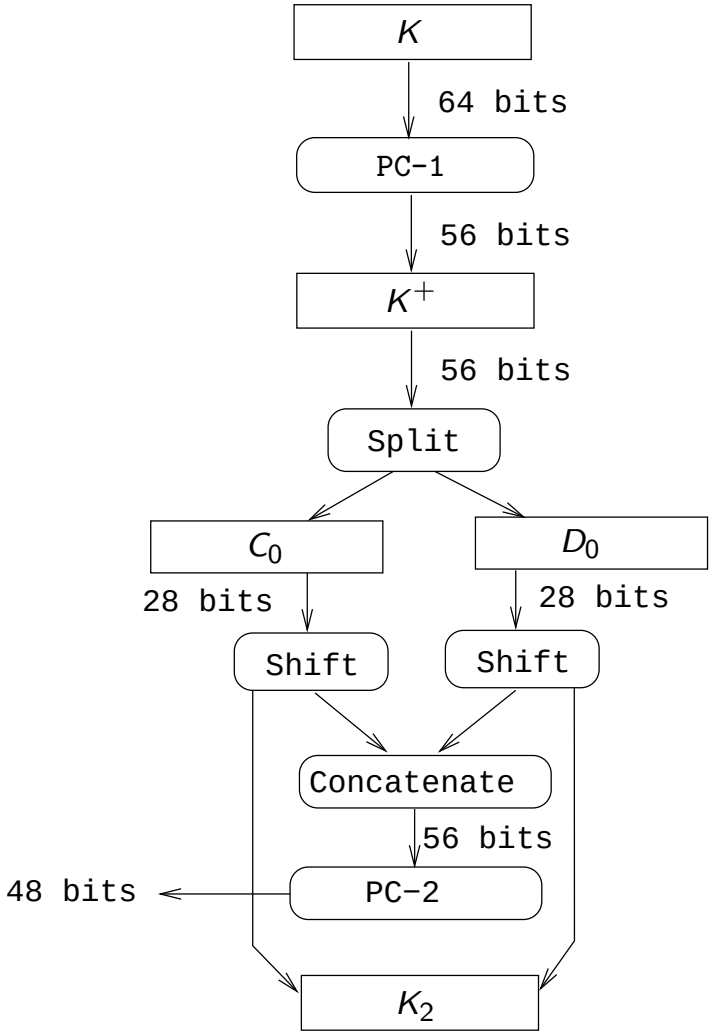
  diffusion: Diffusion spreads the plaintext out over the ciphertext. Transposition (AKA permutation) ciphers do this.

- DES has 16 rounds. Each round consists of a substitution followed by a permutation.

Key Schedule

- DES runs in 16 rounds.
- At each round we need a 48 bit key.
- We take the original 64 bit key, and extract 56 bits.
- In each round a new version of the key is generated to be used in the next round:
  1. Split the key into two halves, each 28 bits.
  2. Shift each half by 1 or 2 steps.
  3. Compress the two halves into 48 bits using the *Compression Permutation*,
  4. Merge the two shifted halves into the key for the next round.

Key Schedule. . .



Key Schedule — Initial Permutation

- We permute the 64 bits of the key according to table PC-1:

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

- Bit 57 of original key $\longrightarrow$ bit 1 of permuted key.
- Bit 49 of original key $\longrightarrow$ bit 2 of permuted key.
- Note: there are 56 entries in the table $\Rightarrow$ we discard 8 original bits. I.e. bits numbered 8, 16, 24, 32, 40, 48, 56, and 64 are missing in the table.

Key Schedule — Initial Permutation — Example

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

Original 64-bit key, K

00010011	00110100	01010111	01111001	10011011	10111100	11011111	111100

56-bit permutation, K^+

1111000	0110011	0010101	0101111	0101010	1011001	1001111	0001111

Key Schedule — Split the Key

- Next, the 56-bit permuted key is split key into a left and a right half, C_0 and D_0, each 28 bits.

56-bit permutation, K^+							
1111000	0110011	0010101	0101111	0101010	1011001	1001111	0001111

Split in left and right half								
i	C_i				D_i			
0	1111000	0110011	0010101	0101111	0101010	1011001	1001111	0001111

Key Schedule — Rotate the Key

- Next, generate 16 key-halves C_i, D_i, $1 \leq i \leq 16$ by rotating 1 or 2 bits.
- The table below expresses by how much (1 or 2 steps) the key should be circularly shifted each round.
- Because of this shift, a different part of the key will be used in each round.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Shift	1	1	2	2	2	2	2	2	2	1	2	2	2	2	2	1

i	C_i	D_i	$<<$
0	11110000110011001010101111	01010101011001100111110001111	
1	11100001100110010101011111	10101010110011001111100011110	1
2	11000011001100101010111111	01010101100110011111000111101	1
3	00001100110010101010111111111	01010110011001111100011110101	2
4	00110011001010101011111111100	01011001100111110001111010101	2
5	11001100101010101111111110000	01100110011111000111101010101	2
6	00110010101010111111111000011	10011001111100011110101010101	2
7	11001010101011111111100001100	01100111110001111010101010110	2
8	00101010101111111110000110011	10011110001111010101011001	2
9	01010101011111111100001100110	00111100011110101010110011	2
10	01010101111111110000110011001	11110001111010101011001100	2
11	01010111111111000011001100101	11000111101010101100110011	1
12	01011111111000011001100101010	00011110101010101100110011111	2
13	01111111110000110011001010101	01111010101010110011001111100	2
14	11111110000110011001010101010	11101010101011001100111110001	2
15	11111000011001100101010101111	10101010101100110011111000111	2
16	11110000110011001010101011111	01010101011001100111110001111	1

Key Schedule — Concatenate the Keys

- Next, concatenate the key, forming a 56 bit key.

i	$C_i D_i$
0	1111000011001100101010101111010101010110011001111100011111
1	1110000110011001010101011111101010101100110011111000111110
2	1100001100110010101010111111010101011001100111110001111101
3	0000110011001010101010111111110101010011001100111110001111010101
4	0011001100101010101111111100010110011001111000111101010101
5	1100110010101010111111110000011001100111100011110101010101
6	0011001010101011111111000011100110011110001111010101010101
7	1100101010101111111100001100011001111000111101010101010110
8	0010101010111111110000110011100111100011110101010101011001
9	0101010101111111100001100110001111000111101010101011001100
10	0101010111111110000110011001111100011110101010101100110011
11	0101011111111000011001100101110001111010101010110011001111
12	0101111111100001100110010101000111101010101100110011001111
13	0111111110000110011001010101011110101010101100110011110000
14	1111111000011001100101010101111010101010110011001111000111
15	1111100001100110010101010111101010101011001100111100011111
16	1111000011001100101010101111010101010110011001111000111111

Key Schedule — Compress the Keys

- Next, permute and compress the key into 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

- Bit 1 of K_n is bit 14 of C_nD_n.
- Bit 2 of K_n is bit 17 of C_nD_n.

Key Schedule — Compress the Keys...

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

- Example: $C_1 D_1 =$

  1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110

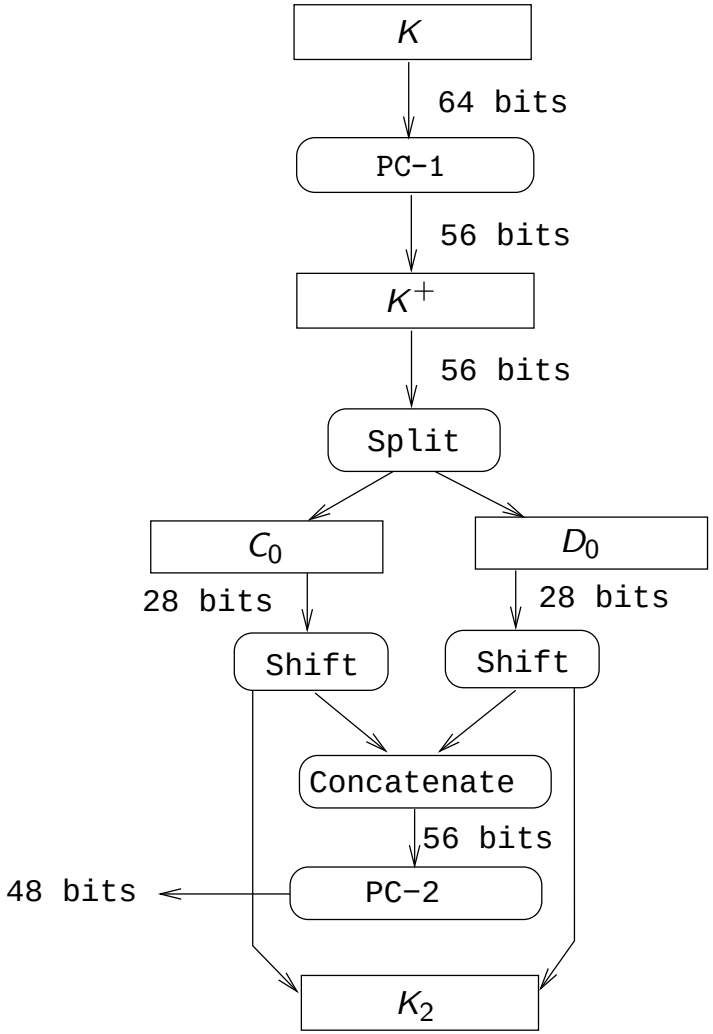
- Efter permutation becomes $K_1 =$

  000110 110000 001011 101111 111111 000111 000001 110010

Key Schedule — Final Keys

i	K_i							
1	000110	110000	001011	101111	111111	000111	000001	110010
2	011110	011010	111011	011001	110110	111100	100111	100101
3	010101	011111	110010	001010	010000	101100	111110	011001
4	011100	101010	110111	010110	110110	110011	010100	011101
5	011111	001110	110000	000111	111010	110101	001110	101000
6	011000	111010	010100	111110	010100	000111	101100	101111
7	111011	001000	010010	110111	111101	100001	100010	111100
8	111101	111000	101000	111010	110000	010011	101111	111011
9	111000	001101	101111	101011	111011	011110	011110	000001
10	101100	011111	001101	000111	101110	100100	011001	001111
11	001000	010101	111111	010011	110111	101101	001110	000110
12	011101	010111	000111	110101	100101	000110	011111	101001
13	100101	111100	010111	010001	111110	101011	101001	000001
14	010111	110100	001110	110111	111100	101110	011100	111010
15	101111	111001	000110	001101	001111	010011	111100	001010
16	110010	110011	110110	001011	000011	100001	011111	110101

Key Schedule — Summary



Initial Data Permutation

- We first permute the 64-bit data block with permutation table `IP`:

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

- Bit 58 of M becomes the bit 1.
- Bit 50 of M becomes the bit 2.

Initial Data Permutation...

- Plaintext $M = 0123456789ABCDEF$

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

$\mathrm{IP}(M)$

1100	1100	0000	0000	1100	1100	1111	1111	1111	0000	1010	1010	1111	0000	1010	1010

Splitting the Input Block

- Next, we split the permuted 64-bit data block into a left and a right half, L_0 and R_0:

$\mathrm{IP}(M)$

1100	1100	0000	0000	1100	1100	1111	1111	1111	0000	1010	1010	1111	0000	1010	1010

L_0								R_0							
1100	1100	0000	0000	1100	1100	1111	1111	1111	0000	1010	1010	1111	0000	1010	1010

The Rounds

- We iterate 16 times, for $1 \leq i \leq 16$, computing:

$$\begin{aligned} L_n &= R_{n-1} \\ R_n &= L_{n-1} \oplus f(R_{n-1}, K_n) \end{aligned}$$

- In each iteration, the right 32 bits of the previous result become the left 32 bits of the current step.
- We XOR the left 32 bits of the previous step with the calculation f on the key and the right 32 bits.
- At the end, we have a final block $L_{16}R_{16}$.

The Rounds...

1. Split 64-bit input into L_0 and R_0, each 32 bits.
2. Round 1:

$$\begin{aligned} L_1 &= R_0 \\ R_1 &= L_0 \oplus f(R_0, K_1) \end{aligned}$$

3. Round i:

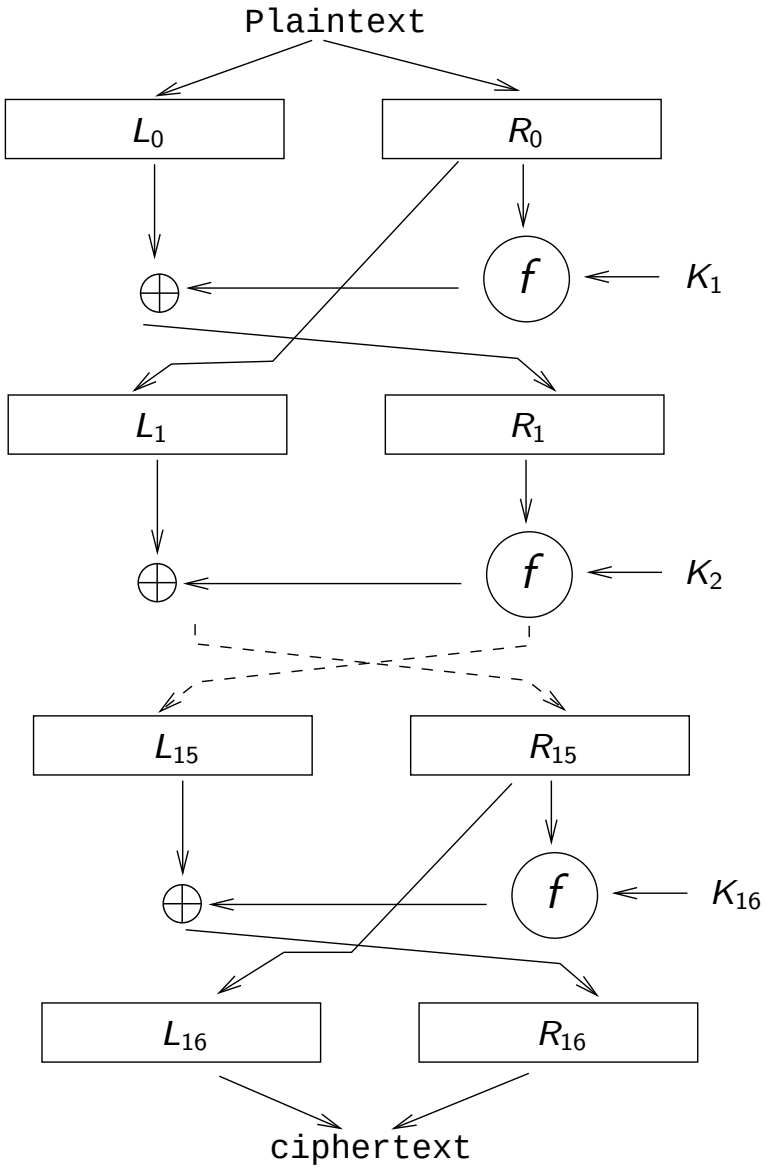
$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i) \end{aligned}$$

4. Round 16:

$$\begin{aligned} L_{16} &= R_{15} \\ R_6 &= L_{15} \oplus f(R_{15}, K_{16}) \end{aligned}$$

5. Return the 64-bit ciphertext $L_{16}R_{16}$.

The Rounds...



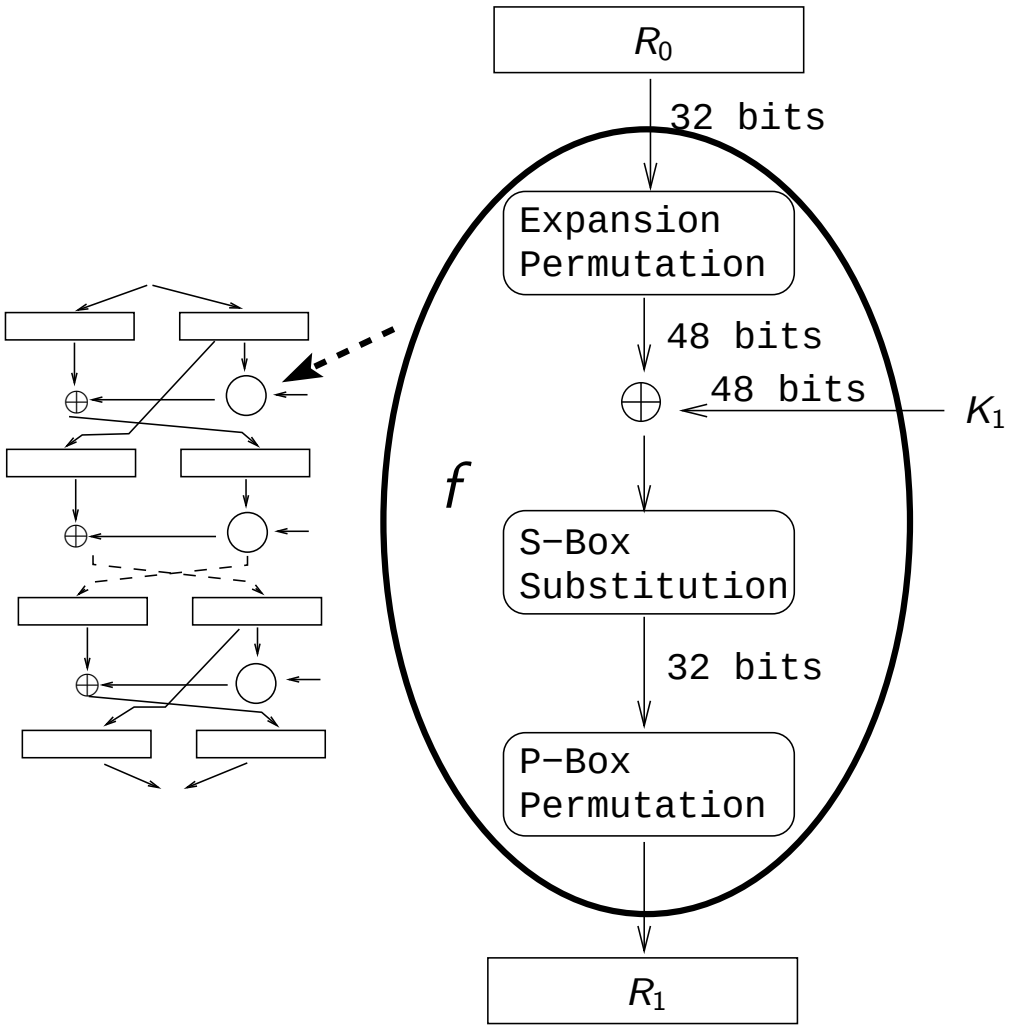
Example

- Let $n = 1$.
- $K_1 =$
  000110 110000 001011 101111 111111 000111 000001 110010
- $L_1 = R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$
- $R_1 = L_0 + f(R_0, K_1)$

The f Function

- f is the function that combines the data (the right 32-bit half, R) with the key during each round:
  1. Get 48 bits of the key,
  2. Expand R to 48 bits using the *expansion permutation*,
  3. XOR the expanded R and the compressed key,
  4. Send the result through 8 S-boxes using the *S-Box Substitution* to get 32 new bits,
  5. Permute the result using the *P-Box Permutation*,
- The result of the f function is XOR:ed with the left half (L) to get the new right half.
- f stands for Feistel, named after German-born physicist and cryptographer Horst Feistel.

The f Function...



Expansion Permutation

- The expansion permutation takes the 32 bits of R, permutes these bits, and, by copying some of them, expands R into 48 bits.
- The table E below says that bit 32 of R moves into position 1, bit 1 moves to 2, 2 to 3, 4 to 5, 5 to 6, 4 to 7, 5 to 8, etc.

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 Permutation...

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

- Calculate $E(R_0)$ from R_0
- $R_0 = 1111\ 0000\ 1010\ 1010\ 1111\ 0000\ 1010\ 1010$
- $E(R_0) =$
  $011110\ 100001\ 010101\ 010101\ 011110\ 100001\ 010101\ 010101$
- Notice how every 4-bit block has now become 8 bits.

XOR:ing in the Key

- Next the output $E(R_n - 1)$ is XOR:ed the key K_n:

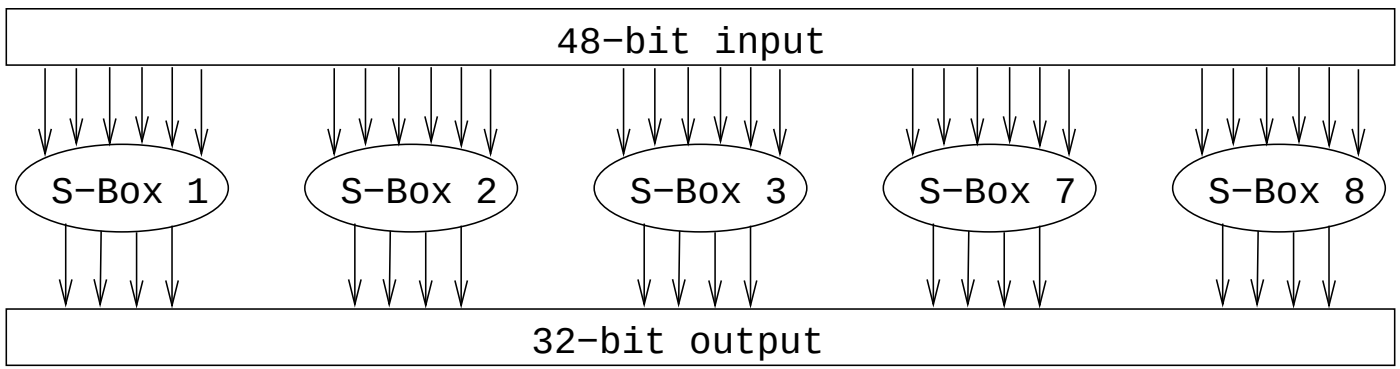
$$K_n \oplus E(R_n - 1).$$

- For K_1 , $E(R_0)$:

$$\begin{aligned} K_1 &= 000110\ 110000\ 001011\ 101111\ 111111\ 000111\ 000001\ 110010 \\ E(R_0) &= 011110\ 100001\ 010101\ 010101\ 011110\ 100001\ 010101\ 010101 \\ K_1 \oplus E(R_0) &= 011000\ 010001\ 011110\ 111010\ 100001\ 100110\ 010100\ 100111. \end{aligned}$$

S-Box Substitution

- The expanded right (R_{i-1}) and the compressed key (K_i) are XORed together. The 48-bit result is then passed through 8 S-Boxes (Substitution Boxes) to form 32 bits.
- Each one of the 8 S-Boxes is different. Each takes 6 bits of input and produces 4 bits of output:



S-Box Substitution Tables

- Each S-Box is given by a table consisting of 4 rows of 16 numbers. Each number is a 4-bit quantity.
- The input to each S-Box is 6 bits:

b_1	b_2	b_3	b_4	b_5	b_6

- The S-Box table is indexed by taking the outer 2 bits (b_1 and b_6) and forming a number between 0 and 3. This is used to get the row number, The middle 4 bits ($b_2 \cdots b_5$) get the column number.
- The S-Boxes are what gives DES its security.

S-Box Substitution: S-Box 1

	column number															
row	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

- Input block $B = b_1 b_2 b_3 b_4 b_5 b_6 = 011011$.
- Row index $= b_1 b_6 = 01_2 = 1_{10}$
- Column index $= b_2 b_3 b_4 b_5 = 1101_2 = 13_{10}$.
- $\Rightarrow S_1(011011) = 5_{10} = 0101_2$.

S-Box Substitution

S-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

S-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

S-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

S-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

S-Box Substitution

S-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

S-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

S-Box 7

4	11	2	14	15	0	8	13	3	12	9	7	6	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

S-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

P-Box Permutation Tables

- The final operation in the f function is the P-Box.
- It's a stright permutation of the 32 bits that come out of the S-Box.
- Bit number 16 moves into position 1, bit 7 into 2, 20 into 3, etc:

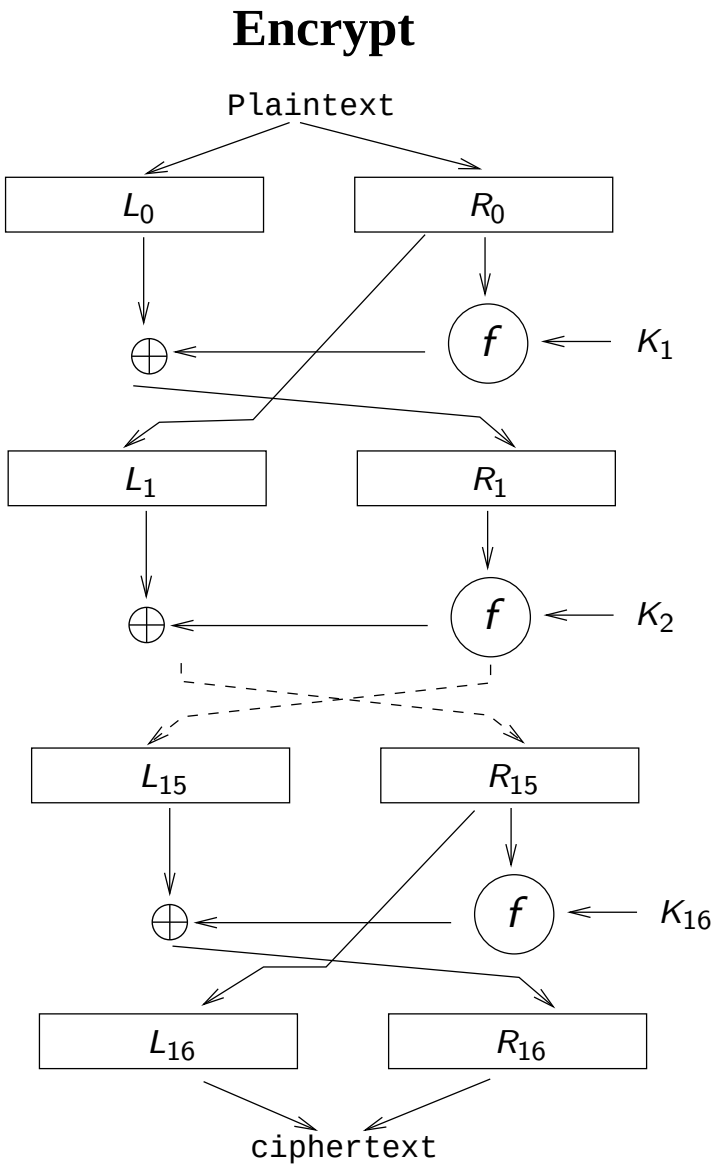
16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

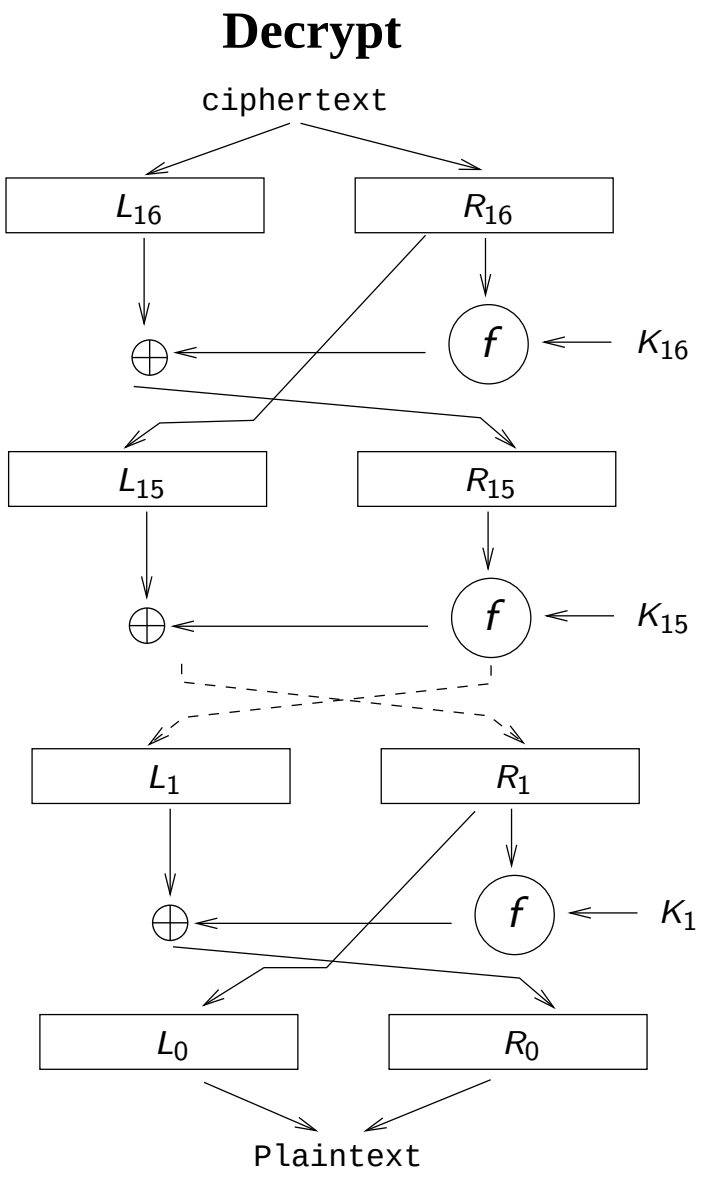
Decryption

- The same algorithm we just described for encryption can also be used for decryption.
- However, we have to use the keys in the reverse order.
- That is, during decryption we use the keys

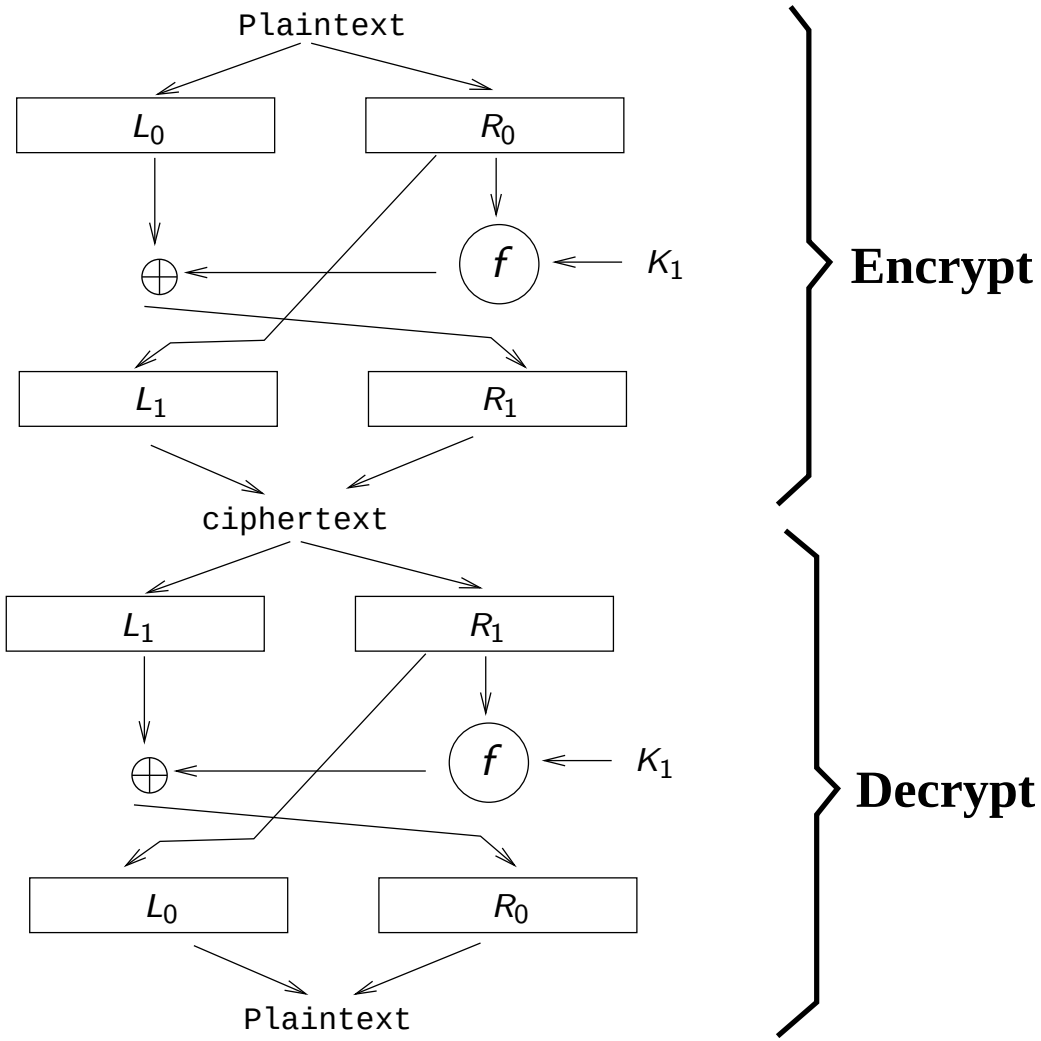
$$K_{16}, K_{15}, \cdots, K_2, K_1.$$

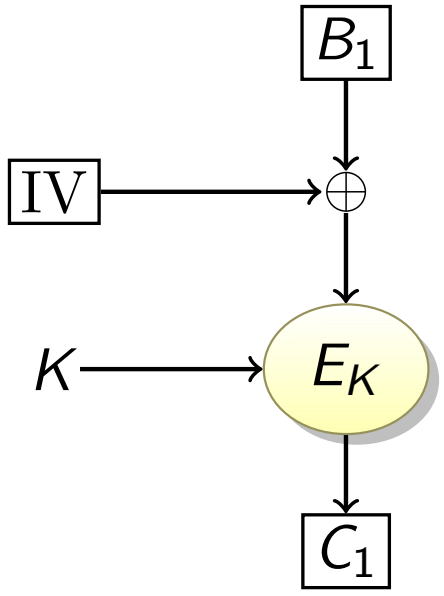
Decryption

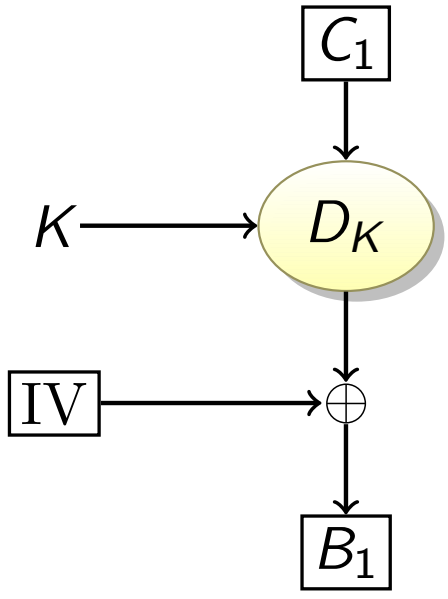

Encrypt
Plaintext
L_0
R_0
f
K_1
L_1
R_1
K_2
L_{15}
R_{15}
K_{16}
L_{16}
R_{16}
ciphertext

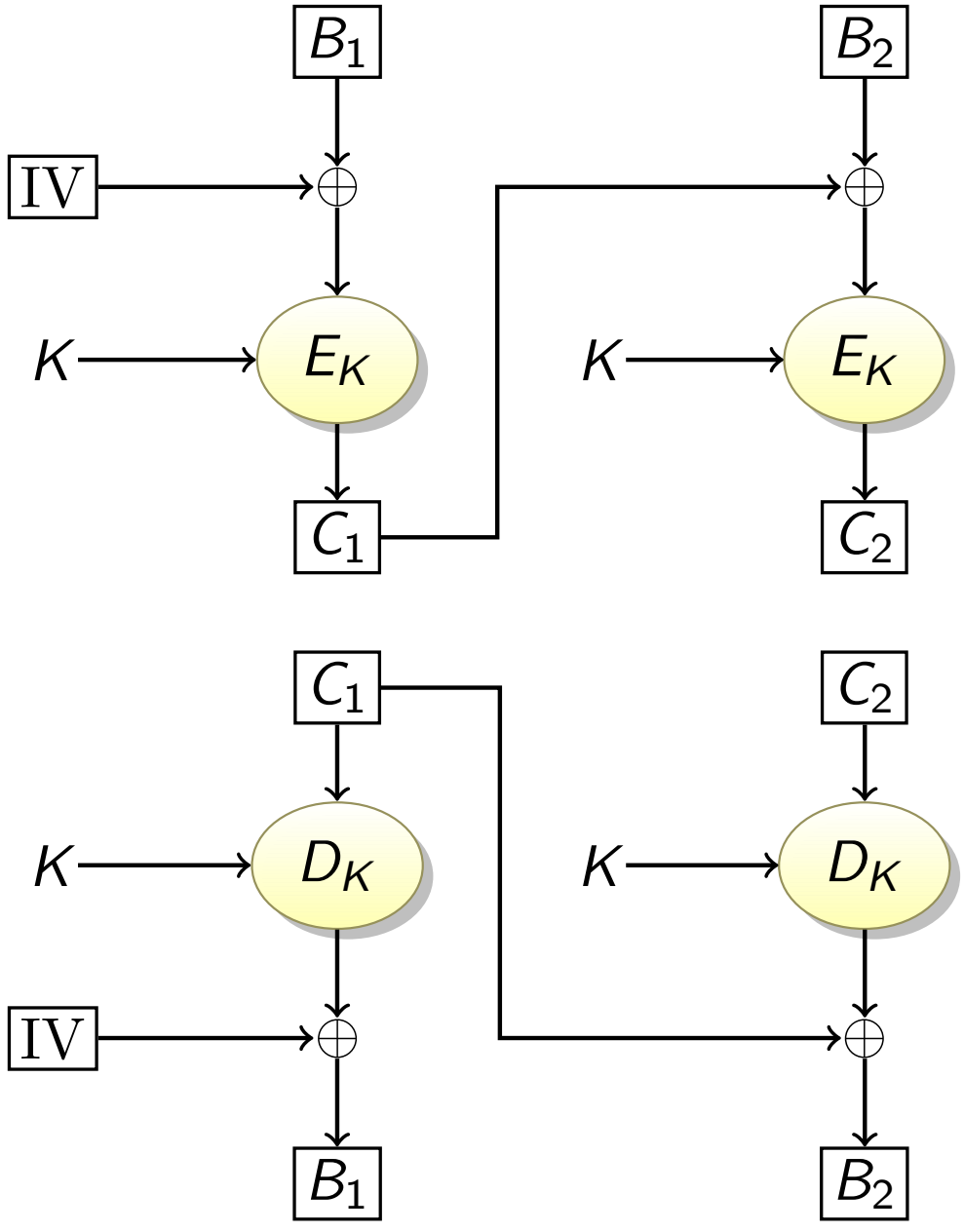

Decrypt
ciphertext
L_{16}
R_{16}
f
K_{16}
L_{15}
R_{15}
K_{15}
L_1
R_1
K_1
L_0
R_0
Plaintext

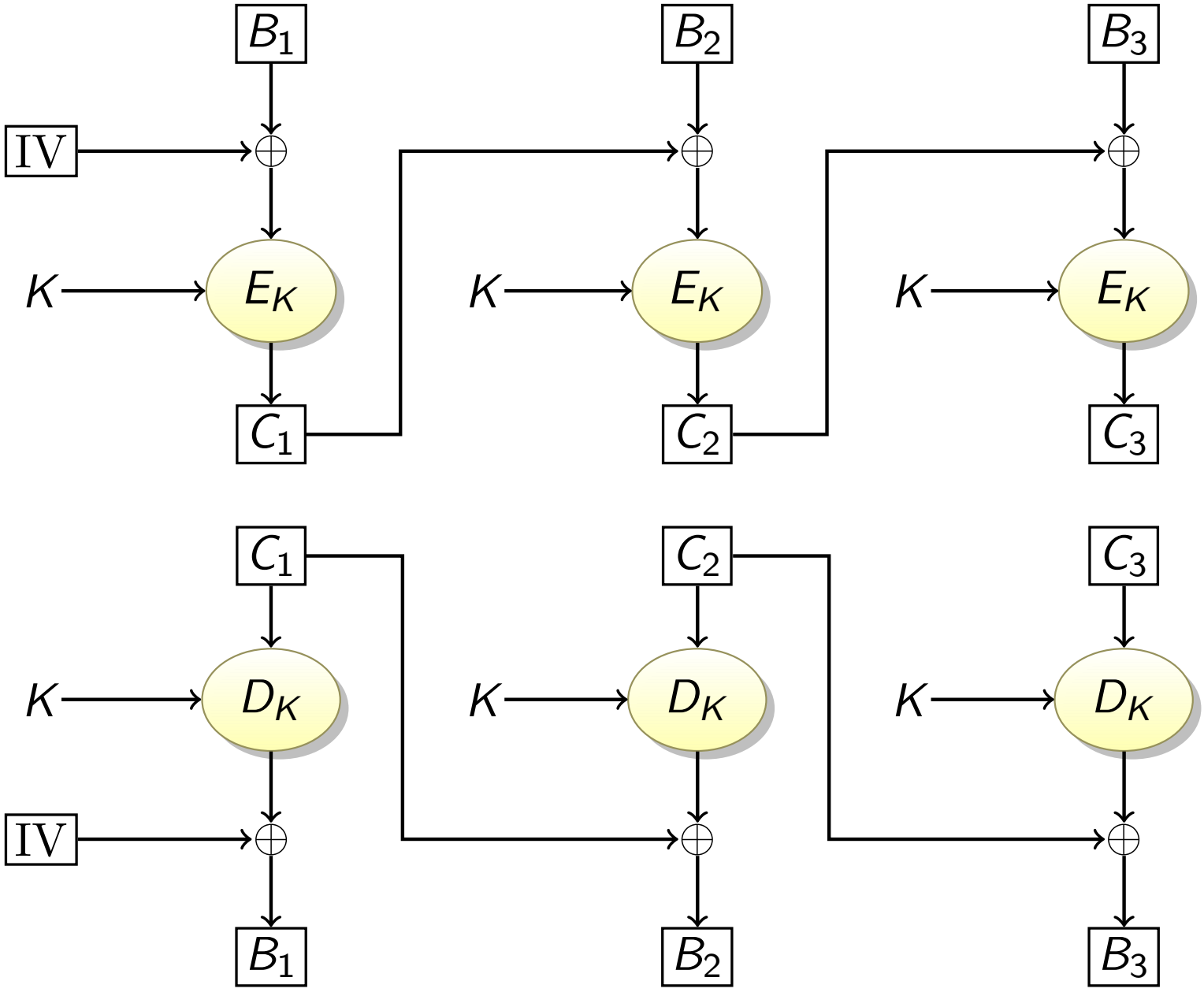
Decryption — 1 Round




B_1
IV
K
E_K
C_1


C_1
K
D_K
IV
B_1


B_1
B_2
IV
K
E_K
K
E_K
C_1
C_2
C_1
C_2
K
D_K
K
D_K
IV
B_1
B_2


B_1
B_2
B_3
IV
K
E_K
K
E_K
K
E_K
C_1
C_2
C_3
C_1
C_2
C_3
K
D_K
K
D_K
K
D_K
IV
B_1
B_2
B_3

Triple DES

- Three DES keys, K_1, K_2 and K_3, each 64 bits (56 bits of actual key + 8 parity bits).
- Blocks are still 64 bits.
- Encryption:

$$C = E_{K_3}(D_{K_2}(E_{K_1}(M)))$$

  DES encrypt with K_1, DES decrypt with K_2, then DES encrypt with K_3.
- Decryption:

$$M = D_{K_1}(E_{K_2}(D_{K_3}(C)))$$

  Decrypt with K_3, encrypt with K_2, then decrypt with K_1.

Triple DES — Picking keys

- There are three options for picking keys:
  1. All three keys are independent (168 key bits)
  2. K_1 and K_2 are independent, $K_3 = K_1$ (112 key bits).
  3. $K_1 = K_2 = K_3$ (56 key bits).
- Option 3 is equivalent to DES, good for backwards compatibility.

Weak keys

These keys are weak or semi-weak and should be avoided (p is **0** or **1**, P is **e** or **f**):

`0x0p0p0p0p0p0p0p0p`	`0x0p1P0p1P0p0P0p0P`
`0x0pep0pep0pfp0pfp`	`0x0pfP0pfP0pfP0pfP`
`0x1P0p1P0p0P0p0P0p`	`0x1P1P1P1P0P0P0P0P`
`0x1Pep1Pep0Pfp0Pfp`	`0x1PfP1PfP0PfP0PfP`
`0xep0pep0pfp0pfp0p`	`0xep1Pep1pfp0Pfp0P`
`0xepepepepepepepep`	`0xepfPepfPfpfPfpfP`
`0xfP0pfP0pfP0pfP0p`	`0xfP1PfP1PfP0PfP0P`
`0xfPepfPepfPepfPep`	`0xfPfPfPfPfPfPfPfP`

Software – openssl

```
> cat message
Attack at dawn, actually, no, let's make it 3 am
since, you know, they expect us at dawn, but,
maybe they will expect us at 3 am since, we
said we'd attack at dawn, so let's attack at
down anyway! Unless...
> openssl enc -e -des -in message -a \
          -out message.asc -pass pass:Alice4Evah
> cat message.asc

U2FsdGVkX1/hHZThWEArEyxcOZ6dT4s1zfFDUovG+yCBNEpk1M+m7jevbg+bJpE9
IgxkZLVzncgIWyKQQePLjbmBJfKVRBHCFdfbS+oi4dxj3PfMygm/HN5QjeC7TIJL
muXX+cOSExJ4GKifxBO8VBT8baj8aO6k2Qfex9JuXhOdHSqCfO2GHgr4b7s/5us0
OHgrVZP9RIaCMBEB+9uu6oDVlApK/GoWipf8/S5jz7qbdjjaT6CwOO+HImNPEE9w
k1xYs5CkNU6fVHH3b1SIyJHxxf6REUkxA3immtFVidsn2sOHJ0g36A==
```

Software – openssl

```
> openssl enc -d -des -in message.asc \
          -a -pass pass:Alice4Evah
Attack at dawn, actually, no, let's make it 3 am
since, you know, they expect us at dawn, but,
maybe they will expect us at 3 am since, we
said we'd attack at dawn, so let's attack at
down anyway! Unless...
```

Outline

AES - Rijndael

- Joan Daemen and Vincent Rijmen created Rijndael
- AES is based on Rijndael: iterated block cypher with variable block and key lengths
- AES and Rijndael differ only in the range of supported values for the block length and cipher key length
- Rijndael block length and key length can be specified to any multiple of 32 bits with a minimum of 128 bits
- AES fixes the block length to 128 bits and supports key lengths of 128, 192, or 256
- A Stick Figure Guide to the Advanced Encryption Standard (AES): http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

Rijndael Design

- Rijndael design criteria
  - Resistance against all known attacks
  - Speed and code compactness on many platforms
  - Design simplicity

Design Rationale

- Wide Trail Strategy provides resistance against linear and differential cryptanalysis
- Wide Trail Strategy layers:
  - Linear mixing layer: guarantees high diffusion over multiple rounds
  - Non-linear layer: parallel application of S-boxes
  - Key addition layer: XOR of the Round Key to the intermediate state

Rijndael Specification - Transformations

- The different transformations operate on intermediate results called States
- Round(State,RoundKey){
  ByteSub(State);
  ShiftRow(State);
  MixColumn(State);
  AddRoundKey(State,RoundKey);}
- Final Round does not include MixColumn(State)

Rijndael Specification - Key Schedule

- Round keys are derived from Cipher Key through a key schedule
- This consists of Key Expansion and Round Key Selection
- Total number of Round Key bits is the block length multiplied by the number of rounds plus 1 (for bl=128 and r=9, RK=1280)
- Cipher Key is expanded to an Expanded Key via S-boxes
- Round keys are taken from the Expanded Key

Outline

Readings and References

- Chapter 8.1.6 in *Introduction to Computer Security*, by Goodrich and Tamassia.
- The Enigma Secret: http://www.youtube.com/watch?v=IJToxIZMbZQ&feature=related
- J. Orlin Grabbe, The DES Algorithm Illustrated, http://orlingrabbe.com/des.htm.

Acknowledgments

Additional material and exercises have also been collected from these sources:

1. Igor Crk and Scott Baker, *620—Fall 2003—Basic Cryptography*.
2. J. Orlin Grabbe, The DES Algorithm Illustrated, http://orlingrabbe.com/des.htm.