Secret Key Systems

Encrypting a small block of text (say 64 bits)

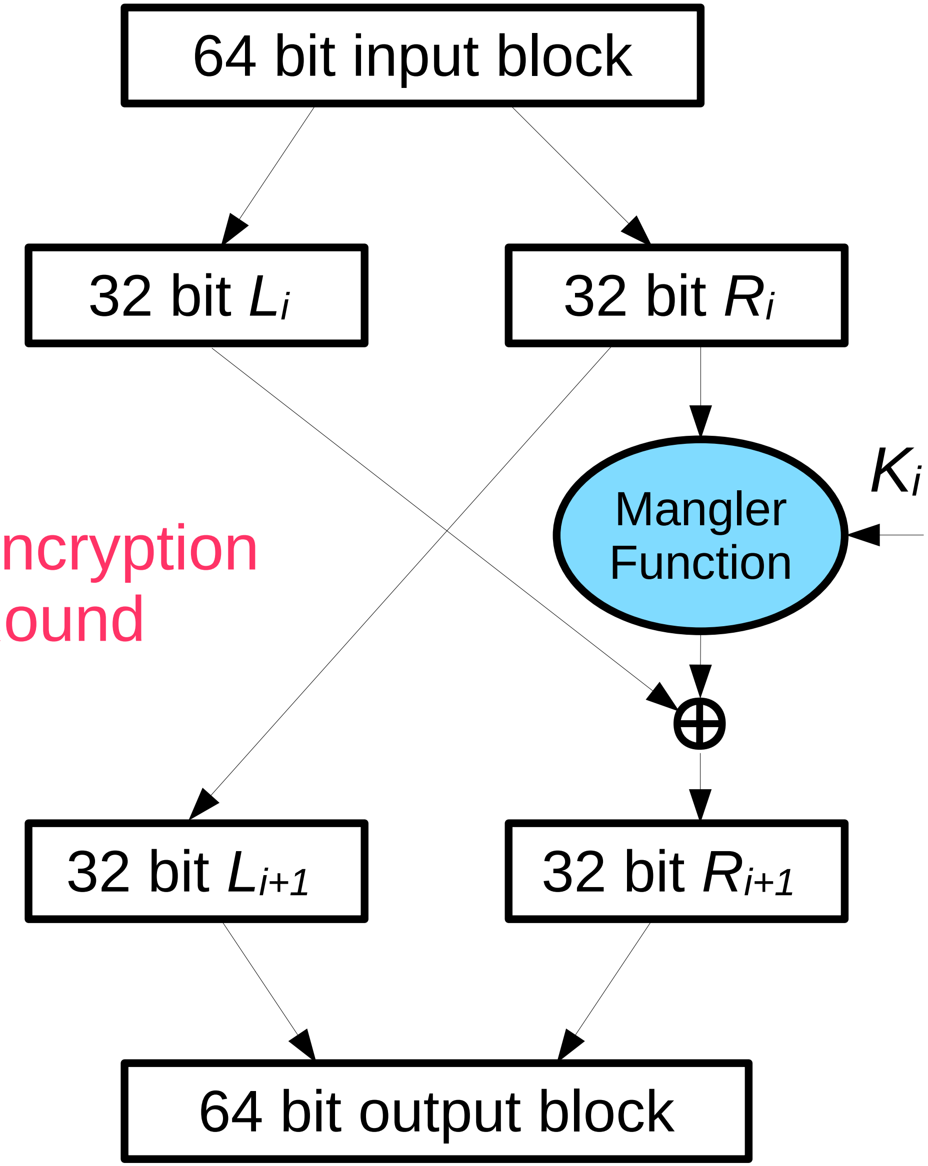
General Considerations:

- Encrypted data should look random.
  As though someone flipped a fair coin 64 times and heads means 1 and tails 0.
  Any change in one bit of output corresponds to a huge change in the input (bits are uncorrelated).
  Should be about as many 1's as 0's usually.
- Try to spread the influence of each input bit to all output bits
  and change in one input bit should have 50% chance of changing any of the output bits (hence many rounds).
- Operations should be invertible – hence *xor* and table lookup.
  Use of one key for both encryption and decryption.
- Attacks may be mitigated if they rely on operations that are not efficiently implemented in hardware yet allow normal operation to complete efficiently, even in software (permute).

Secret Key Systems - DES

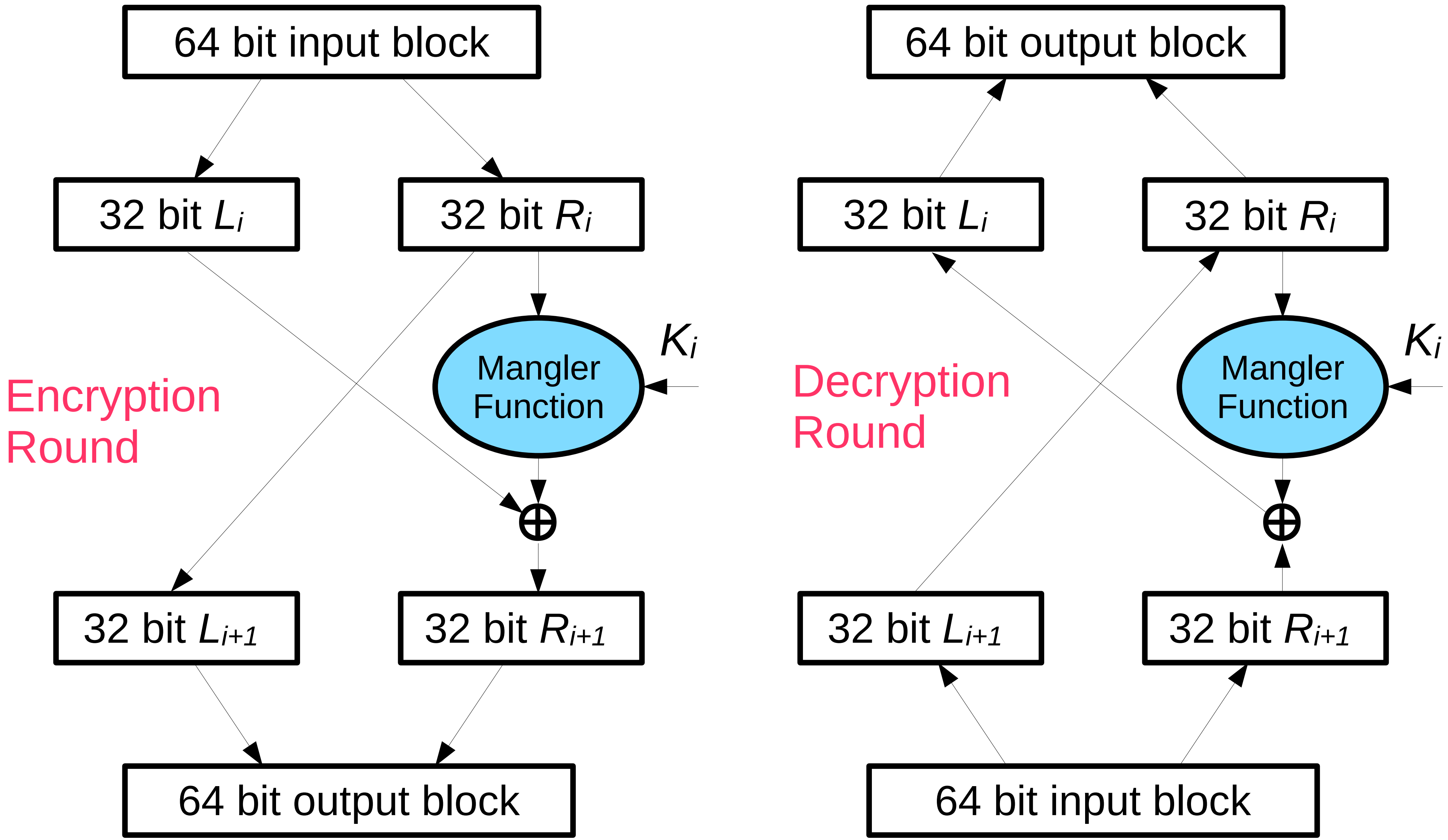
IBM/NSA 1977 - 64 bit blocks, 56 bit key, 8 bits parity, 16 rounds

Encryption Round



Secret Key Systems - DES

IBM/NSA 1977 - 64 bit blocks, 56 bit key, 8 bits parity, 16 rounds



Secret Key Systems - DES

Generating per round keys K_1 K_2 ... K_{16} from the 56 bit Key + 8 parity bits

Key bits:

1...8	9...16	17...24	25...32	33...40	41...48	49...56	57...64

Secret Key Systems - DES

Generating per round keys K_1 K_2 ... K_{16} from the 56 bit Key + 8 parity bits

Key bits:

1...8	9...16	17...24	25...32	33...40	41...48	49...56	57...64

C_0:

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

D_0:

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

Create two sequences of permutations of key bits.

Secret Key Systems - DES

Generating per round keys K_1 K_2 ... K_{16} from the 56 bit Key + 8 parity bits

Key bits:

1...8	9...16	17...24	25...32	33...40	41...48	49...56	57...64

C_0:

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

D_0:

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

Create two sequences of permutations of key bits. **In Cryptol:**

```
KP : [56][6]
KP = [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 first 2 lines are C_0, the next two lines are D_0

Secret Key Systems - DES

Generating per round keys K_1 K_2 ... K_{16} from the 56 bit Key + 8 parity bits

Key bits:

1...8	9...16	17...24	25...32	33...40	41...48	49...56	57...64

C_0:

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

D_0:

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

Create two sequences of permutations of key bits. **In Cryptol:**

```
KP : [56][6]
KP = [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]
```

But, since sequence indices start with 0, not 1, the following is needed:

```
zeroBasify XP = [ i - 1 | i <- XP ]
KPz = zeroBasify KP
```

Secret Key Systems - DES

Generating per round keys K_1 K_2 ... K_{16} from the 56 bit Key + 8 parity bits

Key bits:	1...8	9...16	17...24	25...32	33...40	41...48	49...56	57...64

Each round: K_i has 48 bits assembled in 2 halves permuted from 24 bits each of C_i and D_i, K_{i+1} is obtained by rotating C_i and D_i left to form C_{i+1} and D_{i+1} (rotation is 1 bit for rounds 1,2,9,16 and 2 bits for other rounds)

Permutations:

Left half C_i: (9,18,35,43 are missing)

14 17 11 24 1 5 3 28 15 6 21 10 23 19 12 4 26 8 16 7 27 20 13 2

Right half D_i: (22,38,43,54 are missing)

41 52 31 37 47 55 30 40 51 45 33 48 44 49 39 56 34 53 46 42 50 36 29 32

```
CP : [48][6]
CP  = [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]

CPz = zeroBasify CP
```

Secret Key Systems - DES

Generating per round keys K_1 K_2 ... K_{16} from the 56 bit Key + 8 parity bits

Key bits:								
	1...8	9...16	17...24	25...32	33...40	41...48	49...56	57...64

Each round: K_i has 48 bits assembled in 2 halves permuted from 24 bits each of C_i and D_i, K_{i+1} is obtained by rotating C_i and D_i left to form C_{i+1} and D_{i+1} (rotation is 1 bit for rounds 1,2,9,16 and 2 bits for other rounds)

In Cryptol:

```
sums : [16][8] -> [16][8]
sums xs = ys
  where ys = [ x + y | x <- xs | y <- [0] # ys ]

expand: [2][28] -> [16][48]
expand [kl, kr] =
    [ ((kl <<< i) # (kr <<< i)) @@ CPz
    | i <- sums [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]]
```

`expand` needs to be supplied with C_0 and D_0 from key information
This is done on the next slide.

Secret Key Systems - DES

Generating per round keys K_1 K_2 ... K_{16} from the 56 bit Key + 8 parity bits

Key bits:								
	1...8	9...16	17...24	25...32	33...40	41...48	49...56	57...64

Each round: K_i has 48 bits assembled in 2 halves permuted from 24 bits each of C_i and D_i, K_{i+1} is obtained by rotating C_i and D_i left to form C_{i+1} and D_{i+1} (rotation is 1 bit for rounds 1,2,9,16 and 2 bits for other rounds)

In Cryptol:

```
expandKey : [64] -> [16][48]
expandKey key = expand (split (key @@ KPz))
```

Example:

```
let key = 0x0001020304050607
let res1 = (key @@ KPz)
res1 is 0x0000000ccf0000 (56 bits)
let res2 = (split (key @@ KPz)):[2][28]
res2 is [0x0000000, 0xccf0000]
let keys = expand (split (key @@ KPz))
```

Secret Key Systems - DES

Generating per round keys K_1 K_2 ... K_{16} from the 56 bit Key + 8 parity bits

Key bits:	1...8	9...16	17...24	25...32	33...40	41...48	49...56	57...64

Each round: K_i has 48 bits assembled in 2 halves permuted from 24 bits each of C_i and D_i, K_{i+1} is obtained by rotating C_i and D_i left to form C_{i+1} and D_{i+1} (rotation is 1 bit for rounds 1,2,9,16 and 2 bits for other rounds)

Example:

```
   keys is
[0x000000102307, 0x000000340105, 0x0000000220c6, 0x00000064a181,
 0x00000022044b, 0x0000004e9102, 0x000000044568, 0x000000489840,
 0x000000488078, 0x00000081dc08, 0x000000081630, 0x000000994824,
 0x000000004a90, 0x000000912015, 0x000000a30280, 0x000000132282]
```

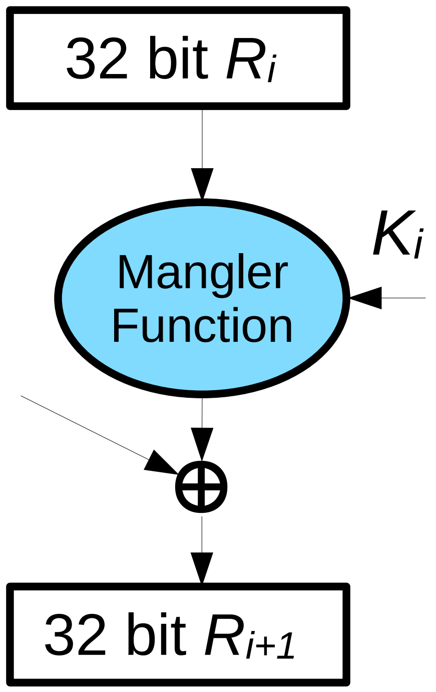
These would be the 16 per round keys generated from the 56 bit key

```
key @@ KPz = 0x0000000ccf0000
```

Of course, this is a bad key but was used to simplify your observations

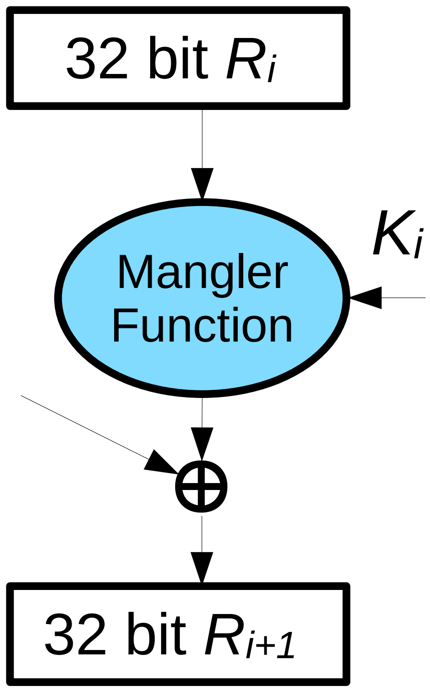
Secret Key Systems - DES

The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits



Secret Key Systems - DES

The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits

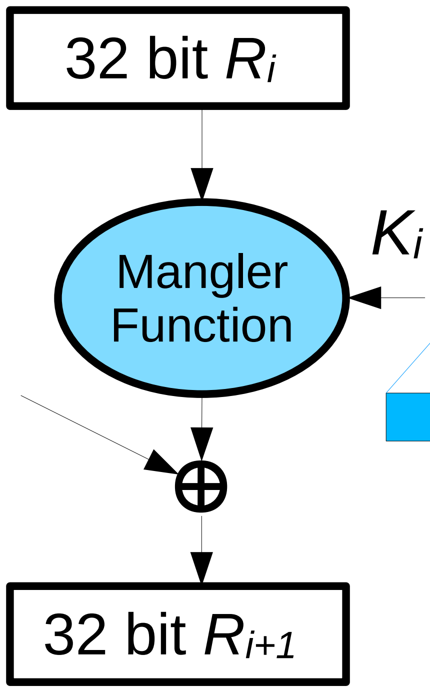


1. Expansion of input bits:

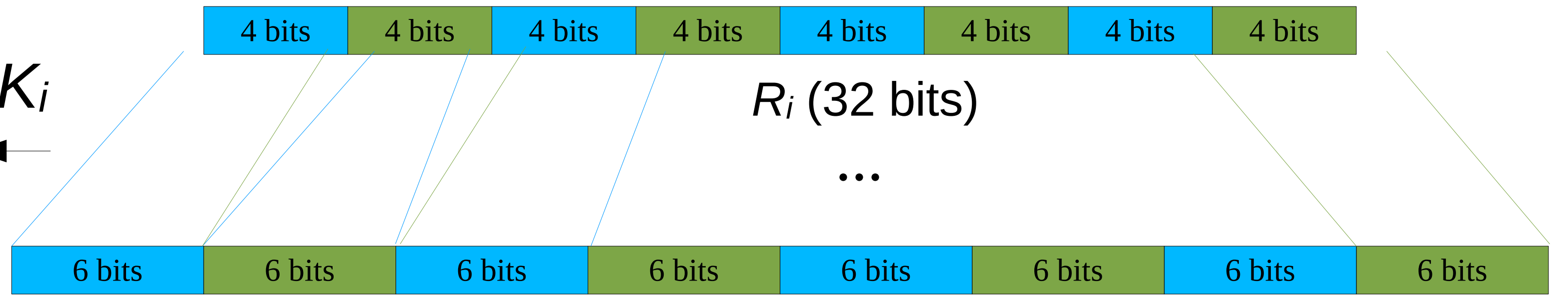
4 bits	4 bits	4 bits	4 bits	4 bits	4 bits	4 bits	4 bits

Secret Key Systems - DES

The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits

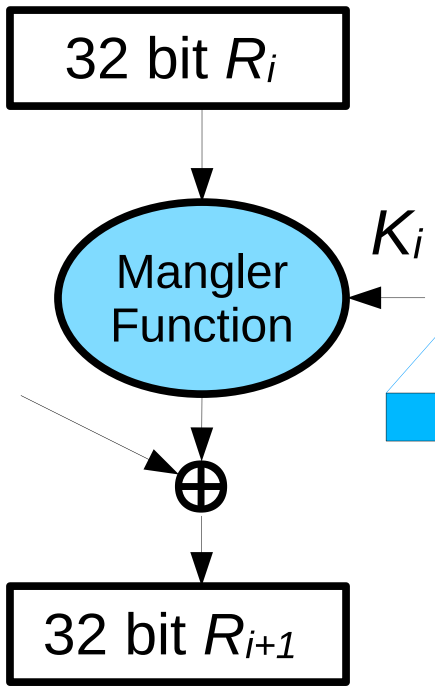


1. Expansion of input bits:

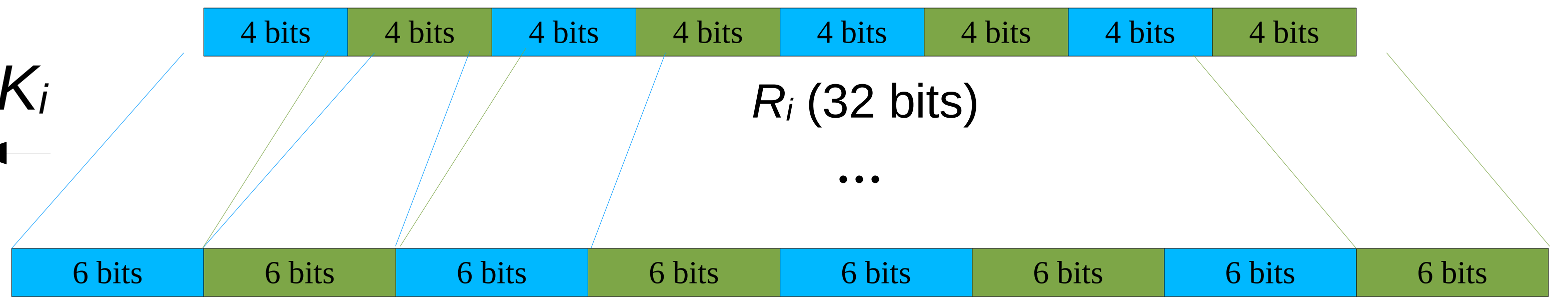


Secret Key Systems - DES

The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits



1. Expansion of input bits:

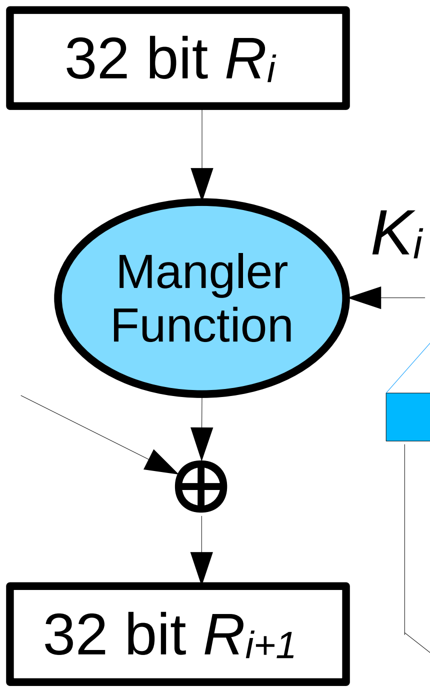


In Cryptol, expand 32 input bits to 48 bits:

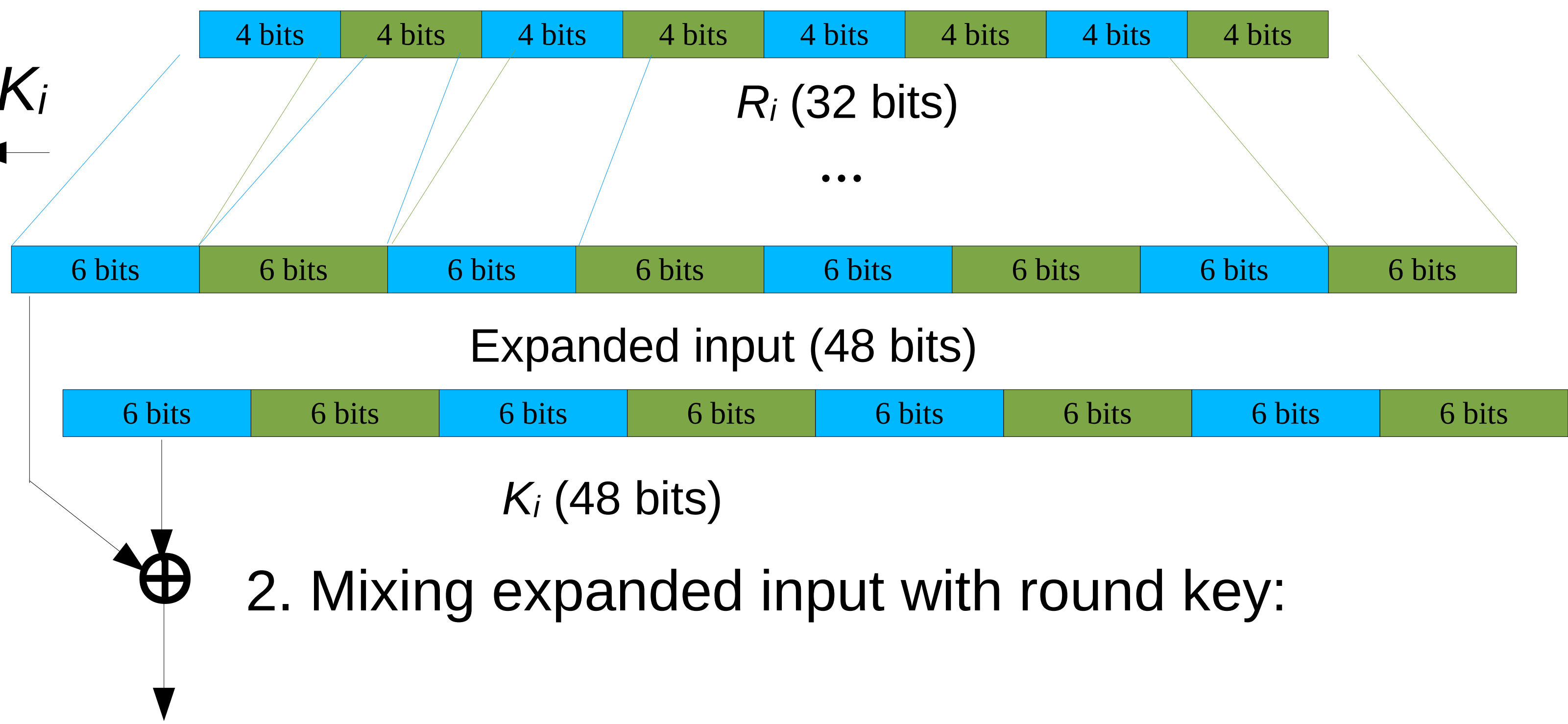
```
EP : [48][6]
EP  = [32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9,
        8, 9,10,11,12,13,12,13,14,15,16,17,
       16,17,18,19,20,21,20,21,22,23,24,25,
       24,25,26,27,28,29,28,29,30,31,32, 1]
EPz = zeroBasify EP
let R3 = 0x10111213 (32 bits, at round 3 as example)
let res3 = R3 @@ EPz (res3 is 0x8a00a28a40a6 – 48 bits)
```

Secret Key Systems - DES

The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits



1. Expansion of input bits:



2. Mixing expanded input with round key:

`let` K_3 `= keys @ 3` (just an example – round 3 key)

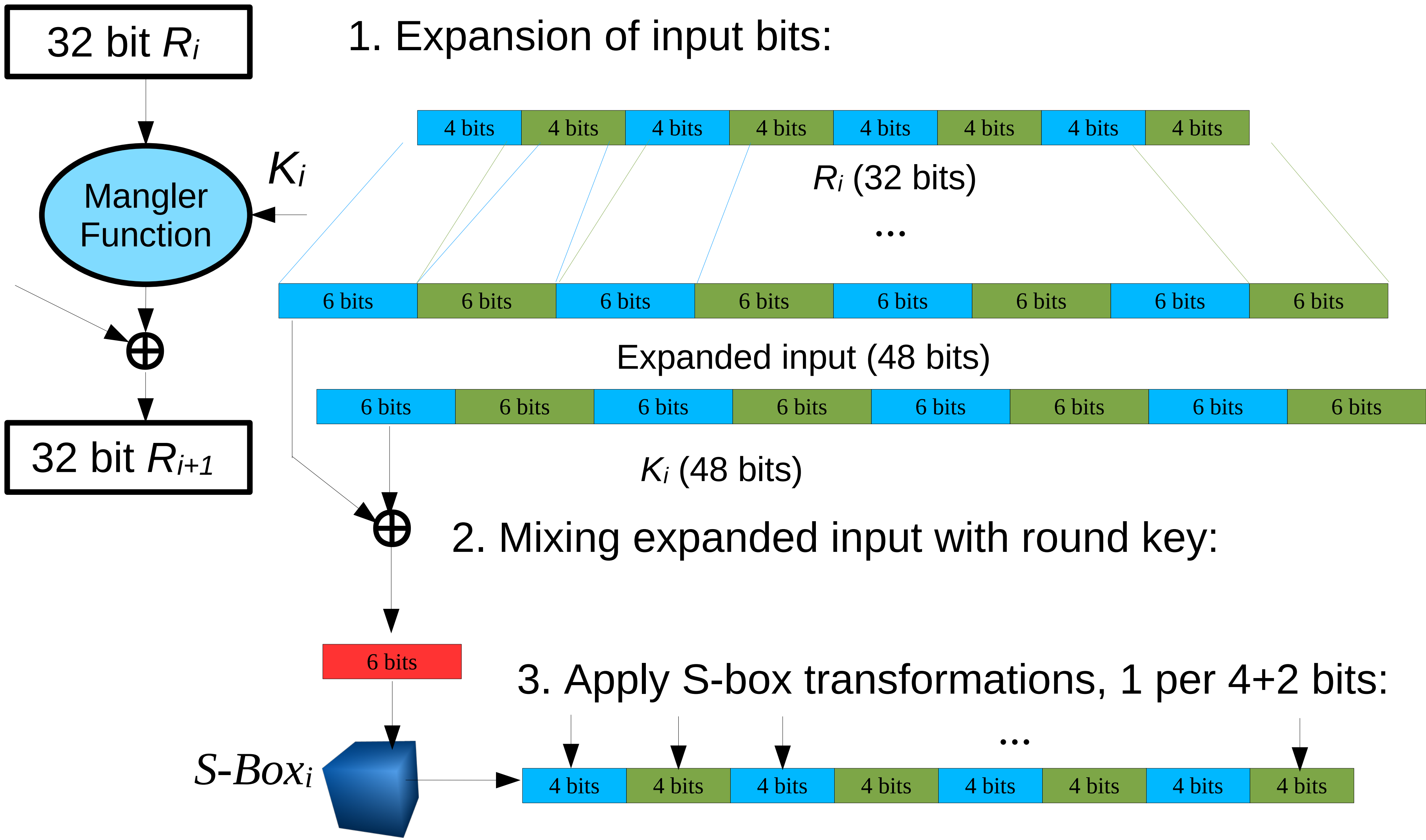
K_3 is `0x00000064a181` (48 bits)

`let res4 = (`R_3 `@@ EPz) ^` K_3 (xor of expanded R_3 and K_3)

`res4 is 0x8a00a2eee127` (48 bits)

Secret Key Systems - DES

The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits



Secret Key Systems - DES

The *S-Box*: maps 6 bit blocks to 4 bit sections
S-Box₁ (first 6 bits):

Input bits 2,3,4,5

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111
01	0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011	1000
10	0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101	0000
11	1111	1100	1000	0010	0100	1001	0001	0111	0101	1011	0011	1110	1010	0000	0110	1101

Input bits 1 and 6

```
sbox1 : [4][16][4]
sbox1 = [[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]]
```

This is just one of 8 S-Boxes, they are all different. Input to an S-box is 6 bits, output is 4 bits, determined by the coordinates of a row and column.

Secret Key Systems - DES

The *S-Box*: maps 6 bit blocks to 4 bit sections

S-Box$_1$ (first 6 bits):

Input bits 2,3,4,5

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111
01	0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011	1000
10	0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101	0000
11	1111	1100	1000	0010	0100	1001	0001	0111	0101	1011	0011	1110	1010	0000	0110	1101

Input bits 1 and 6

```
let sboxes = [sbox1, sbox2, … , sbox8]
// n is the sbox, b is the 6 bit input to the sbox
// returns 4 bit translation
sbox (n,b) = (s @ [b1, b6] @ [b2, b3, b4, b5])
  Where
    s = sboxes @ (n-1)
    b1 = b @ 0
    . . .
    b6 = b @ 5
```

Secret Key Systems - DES

The *S-Box*: maps 6 bit blocks to 4 bit sections

S-Box$_1$ (first 6 bits):

Input bits 2,3,4,5

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111
01	0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011	1000
10	0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101	0000
11	1111	1100	1000	0010	0100	1001	0001	0111	0101	1011	0011	1110	1010	0000	0110	1101

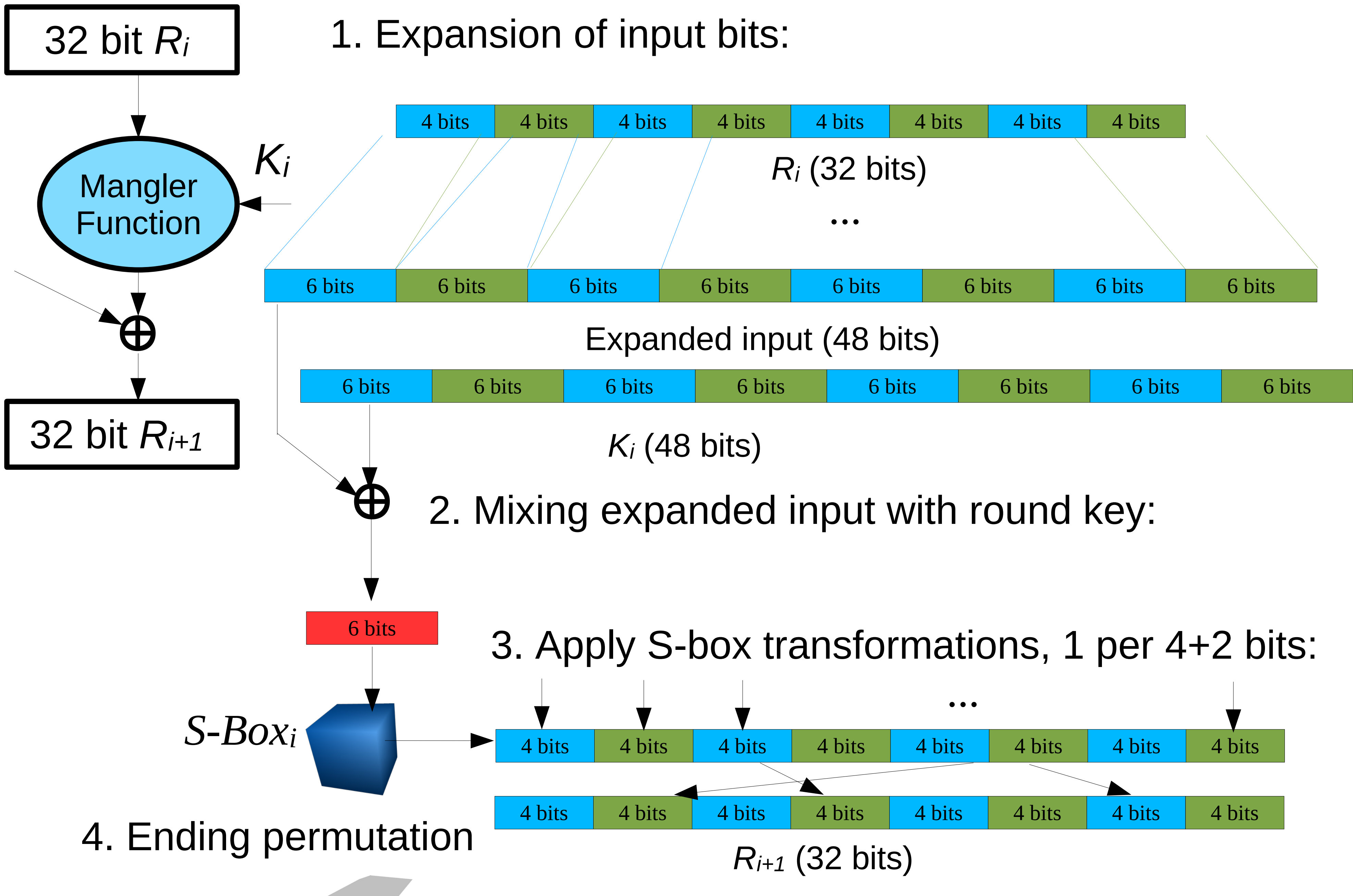
Input bits 1 and 6

```
// Combine results from 8 sboxes - x is the 48 bit
// input. returns the 32 bit output
SBox : [48] -> [32]
SBox x = join [ sbox (n, b) | n <- [1 .. 8] | b <- split x ]
```

Since the type signature for sbox is `([4],[6]) -> [4]` `split` knows to split 48-bit x into a sequence of 8 6-bit sections.

Secret Key Systems - DES

The Mangler function: mixes 32 bit input with 48 bit key to produce 32 bits



Secret Key Systems - DES

Ending permutation:

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

In Cryptol:

```
PP : [32][6] // Ending permutation applied after Sbox op
PP = [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]
PPz = zeroBasify PP
```

The 48 bit result of xoring the per round key with expanded input:

(R_i @@ EPz) ^ K_i

The 32 bit result of SBox operations after the above operation:

SBox((R_i @@ EPz) ^ K_i)

The 32 bit result after application of the Ending Permutation:

(SBox((R_i @@ EPz) ^ K_i)) @@ PPz

As a function, to be applied on any of 16 rounds:

```
f(r,k) = (SBox((r @@ EPz) ^ k) @@ PPz
```

Secret Key Systems - DES

The operation of a round – `r` is right half of 64 bit input and `l` is the left half of the 64 bit input, `k` is the per-round 48 bit key:

```
round (k, [l, r]) = r # (l ^ f (r, k))
```

All 16 rounds, encryption and decryption. Plaintext is `pt`, `IPz` is the Initial Permutation on the 64 bit input (see below) `keys` is obtained from the `expandKeys` function, last round result is `lst` – the last operation is to swap the last block right half with the left half and apply Final Permutation `FPz` (see below):

```
des pt keys = (swap (split lst)) @@ FPz
  where
      pt' = pt @@ IPz
      iv  = [ round (k, split lr)
            | k  <- keys
            | lr <- [pt'] # iv ]
      lst = iv @ (length keys - 1)
```

Finally the encryption and decryption functions cleanly stated:

```
encrypt key pt = des pt (expandKey key)
decrypt key ct = des ct (reverse (expandKey key))
```

Secret Key Systems - DES

The two permutations not previously defined:

```
IP : [64][7] // Redo of C0 & D0 but applied to all 64 bits
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]
IPz = zeroBasify IP

FP : [64][7]
FP = [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]
FPz = zeroBasify FP
```

Secret Key Systems - DES

Weak and semi-weak keys:

If key is such that C_0 *or* D_0 are:

1) all 0s;
2) all 1s;
3) alternating 1s and 0s,

then attack is easy. There are 16 such keys. Keys for which C_0 *and* D_0 are both 0 or both 1 are called *weak* (encrypting with key gives same result as decrypting).

Secret Key Systems - DES

Weak and semi-weak keys:

If key is such that C_0 *or* D_0 are:

1) all 0s;
2) all 1s;
3) alternating 1s and 0s,

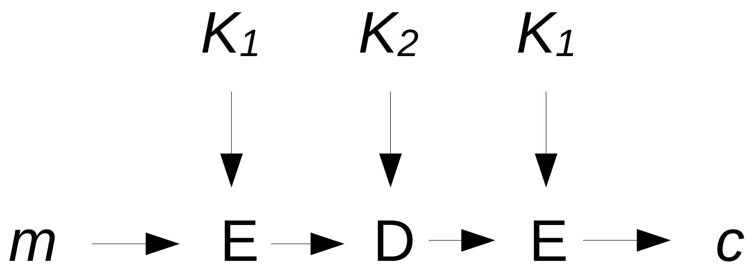
then attack is easy. There are 16 such keys. Keys for which C_0 *and* D_0 are both 0 or both 1 are called *weak* (encrypting with key gives same result as decrypting).

Discussion:

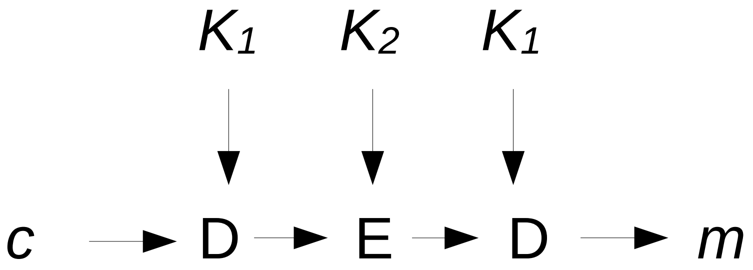
1. Not much known about the design – not made public
   Probably attempt to prevent known attacks
2. Changing S-Boxes has resulted in provably weaker system

Secret Key Systems - DES

Two keys K_1 and K_2:



encryption



decryption

Secret Key Systems - DES

Two keys K_1 and K_2:

$$\begin{tikzcd} & K_1 \arrow[d] & K_2 \arrow[d] & K_1 \arrow[d] & \\ m \arrow[r] & \mathrm{E} \arrow[r] & \mathrm{D} \arrow[r] & \mathrm{E} \arrow[r] & c \end{tikzcd}$$

encryption

$$\begin{tikzcd} & K_1 \arrow[d] & K_2 \arrow[d] & K_1 \arrow[d] & \\ c \arrow[r] & \mathrm{D} \arrow[r] & \mathrm{E} \arrow[r] & \mathrm{D} \arrow[r] & m \end{tikzcd}$$

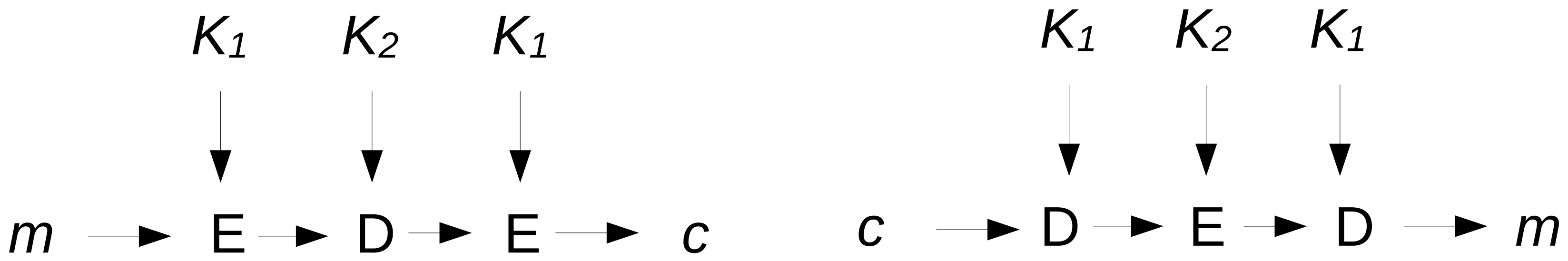
decryption

Why not 2DES

1. Double encryption with the same key still requires searching 2^{56} keys

Secret Key Systems - DES

Two keys K_1 and K_2:



encryption decryption

Why not 2DES

1. Double encryption with the same key still requires searching 2^{56} keys
2. Double encryption with two different keys is just as vulnerable as DES due to the following, assuming some <*m,c*> pairs are known:

Secret Key Systems - DES

Two keys K_1 and K_2:

$$\begin{tikzcd} & K_1 \arrow[d] & K_2 \arrow[d] & K_1 \arrow[d] & \\ m \arrow[r] & E \arrow[r] & D \arrow[r] & E \arrow[r] & c \end{tikzcd}$$

encryption

$$\begin{tikzcd} & K_1 \arrow[d] & K_2 \arrow[d] & K_1 \arrow[d] & \\ c \arrow[r] & D \arrow[r] & E \arrow[r] & D \arrow[r] & m \end{tikzcd}$$

decryption

Why not 2DES

1. Double encryption with the same key still requires searching 2^{56} keys
2. Double encryption with two different keys is just as vulnerable as DES due to the following, assuming some <m,c> pairs are known:

	m: K_1	$E(K_{1,}m)$
	101010	1000011
2^{56}	...	...
	100010	0101111
	001011	0001101

Secret Key Systems - DES

Two keys K_1 and K_2:

$$
\begin{tikzcd}
 & K_1 \arrow[d] & K_2 \arrow[d] & K_1 \arrow[d] & \\
m \arrow[r] & \mathrm{E} \arrow[r] & \mathrm{D} \arrow[r] & \mathrm{E} \arrow[r] & c
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
 & K_1 \arrow[d] & K_2 \arrow[d] & K_1 \arrow[d] & \\
c \arrow[r] & \mathrm{D} \arrow[r] & \mathrm{E} \arrow[r] & \mathrm{D} \arrow[r] & m
\end{tikzcd}
$$

encryption — decryption

Why not 2DES

1. Double encryption with the same key still requires searching 2^{56} keys
2. Double encryption with two different keys is just as vulnerable as DES due to the following, assuming some <m,c> pairs are known:

m: (2^{56} rows)

K_1	$E(K_1, m)$
101010	1000011
...	...
100010	0101111
001011	0001101

c: (2^{56} rows)

K_2	$D(K_2, c)$
101110	0001101
...	...
001110	1000011
001011	0001101

Secret Key Systems - DES

Two keys K_1 and K_2:

$$m \longrightarrow \overset{K_1}{\overset{\downarrow}{\mathrm{E}}} \longrightarrow \overset{K_2}{\overset{\downarrow}{\mathrm{D}}} \longrightarrow \overset{K_1}{\overset{\downarrow}{\mathrm{E}}} \longrightarrow c$$

encryption

$$c \longrightarrow \overset{K_1}{\overset{\downarrow}{\mathrm{D}}} \longrightarrow \overset{K_2}{\overset{\downarrow}{\mathrm{E}}} \longrightarrow \overset{K_1}{\overset{\downarrow}{\mathrm{D}}} \longrightarrow m$$

decryption

Why not 2DES

1. Double encryption with the same key still requires searching 2^{56} keys
2. Double encryption with two different keys is just as vulnerable as DES due to the following, assuming some $<m,c>$ pairs are known:

2^{56}

m: K_1	$\mathrm{E}(K_{1,} m)$
101010	1000011
...	...
100010	0101111
001011	0001101

2^{56}

c: K_2	$\mathrm{D}(K_{2,} c)$
101110	0001101
...	...
001110	1000011
001011	0001101

Secret Key Systems - DES

Two keys K_1 and K_2:

$$
\begin{tikzcd}
 & K_1 \arrow[d] & K_2 \arrow[d] & K_1 \arrow[d] & \\
m \arrow[r] & \mathrm{E} \arrow[r] & \mathrm{D} \arrow[r] & \mathrm{E} \arrow[r] & c
\end{tikzcd}
$$

encryption

$$
\begin{tikzcd}
 & K_1 \arrow[d] & K_2 \arrow[d] & K_1 \arrow[d] & \\
c \arrow[r] & \mathrm{D} \arrow[r] & \mathrm{E} \arrow[r] & \mathrm{D} \arrow[r] & m
\end{tikzcd}
$$

decryption

Why not 2DES

1. Double encryption with the same key still requires searching 2^{56} keys
2. Double encryption with two different keys is just as vulnerable as DES due to the following, assuming some <*m*,*c*> pairs are known:

m: (2^{56} rows)

K_1	$E(K_{1,}m)$
101010	1000011
...	...
100010	0101111
001011	0001101

c: (2^{56} rows)

K_2	$D(K_{2,}c)$
101110	0001101
...	...
001110	1000011
001011	0001101

Test matches on other <*m*,*c*> pairs

$m \rightarrow E(K_1,m)$

$D(K_2,c) \leftarrow c$

Secret Key Systems - DES

Why not 2DES

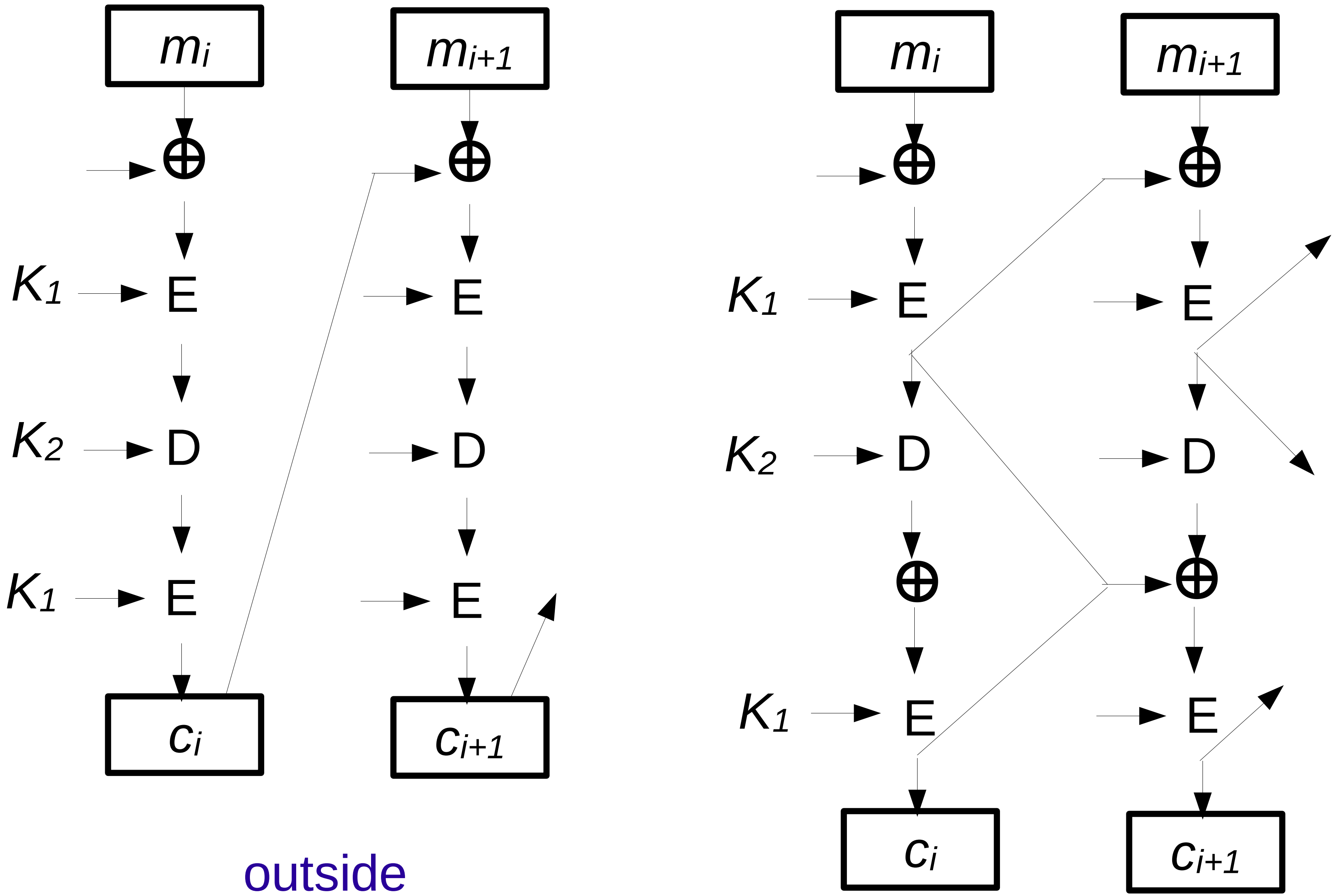
2. Double encryption with two different keys is just as vulnerable as DES due to the following, assuming some <*m,c*> pairs are known: How many <*m*,*c*> pairs do you need?

   2^{64} possible blocks
   2^{56} table entries
   each block has probability 2^{-8} = 1/256 of showing in a table
   probability a block is in both tables is 2^{-16}
   average number of matches is 2^{48}
   average number of matches for two <*m*,*c*> pairs is about 2^{32}
   for three <*m*,*c*> pairs is about 2^{16}
   for four <*m*,*c*> pairs is about 2^{0}

3. Triple encryption with two different keys
   - 112 bits of key is considered enough
   - straightforward to find a triple of keys that maps a given plaintext to a given ciphertext (no known attack with 2 keys)
   - EDE: use same keys to get DES, EEE: effect of permutations lost

Secret Key Systems - DES

CBC with 3DES:



outside

inside

Secret Key Systems - DES

CBC with 3DES:

1. On the outside – same attack as with CBC – change a block with side effect of garbling another
2. On the inside – attempt at changing a block results in all block garbled to the end of the message.
3. On the inside – use three times as much hardware to pipeline encryptions resulting in DES speeds.
4. On the outside – EDE simply is a drop-in replacement for what might have been there before.