Practical Constructions of Block Ciphers

Reading: K&L Section 6.2 (skipping 6.2.6)

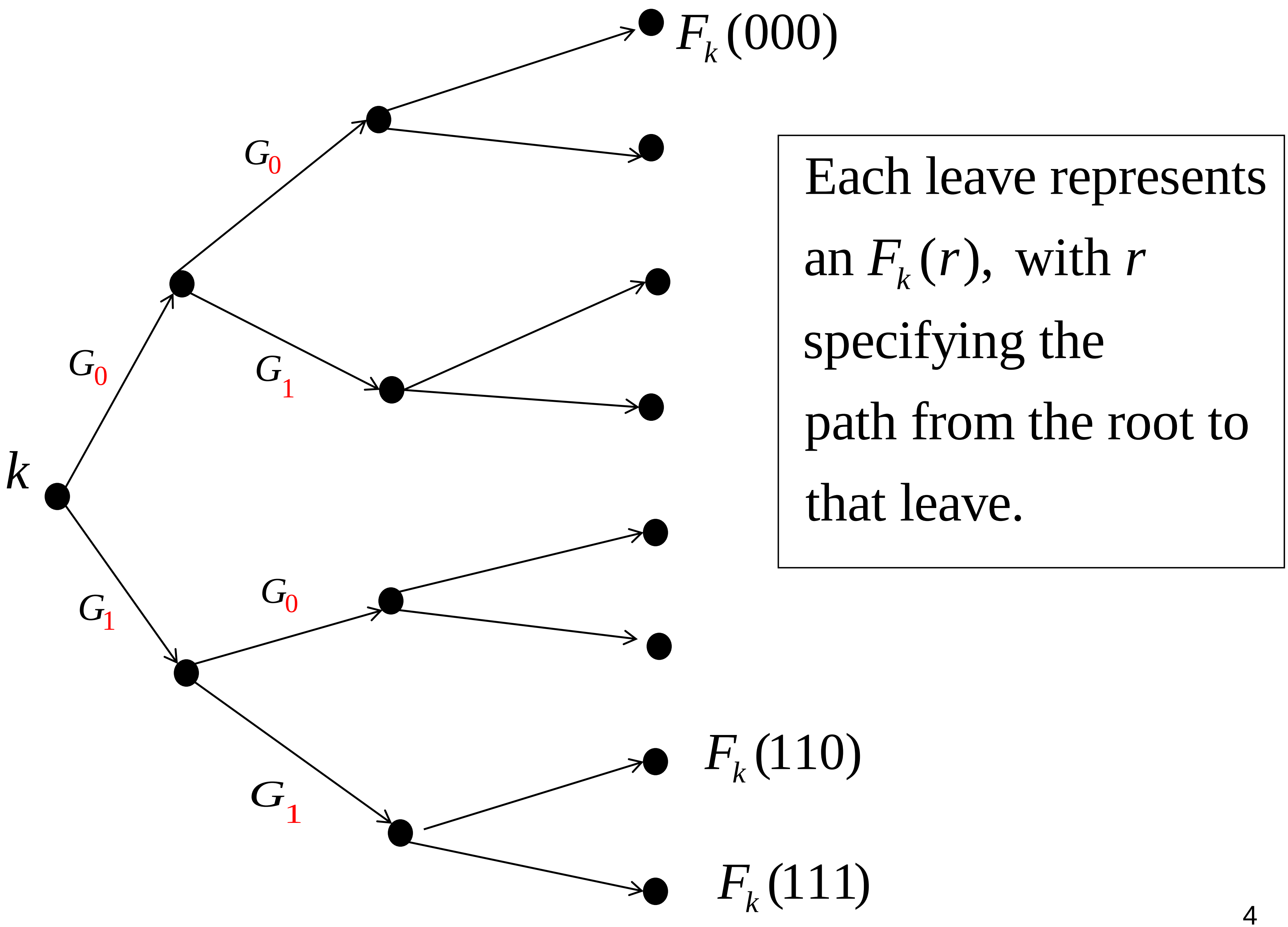
Practical constructions of block ciphers

- There are methods to construct pseudorandom functions/permutations from one-way functions.
  - One-way functions $\Rightarrow$ pseudorandom generators
    $\Rightarrow$ pseudorandom functions
    $\Rightarrow$ pseudorandom permutations
  - Extremely slow
- In practice, block ciphers are constructed using
  - Feistel networks (e.g., DES)
  - Substitution-permutation networks (e.g., AES)
- Block ciphers: "approximate" pseudorandom permutations with some fixed key length and block length.

Constructing pseudorandom functions

- A pseudorandom function F can be constructed from a pseudorandom generator.
- Let $G: \{0,1\}^n \rightarrow \{0,1\}^{2n}$ be a pseudorandom generator.
- Write $G(s) = G_0(s) \,\|\, G_1(s)$.
- For all $k \in \{0,1\}^n$ and $r = b_1 b_2 b_3 \ldots b_n \in \{0,1\}^n$, define

$$F_k(r) = G_{b_n}\left(G_{b_{n-1}}\left(\cdots G_{b_3}\left(G_{b_2}\left(G_{b_1}(k)\right)\right)\right)\right).$$


k
G_0
G_1
G_0
G_1
G_0
G_1
$F_k(000)$
$F_k(110)$
$F_k(111)$
Each leave represents an $F_k(r)$, with r specifying the path from the root to that leave.

The confusion-diffusion paradigm

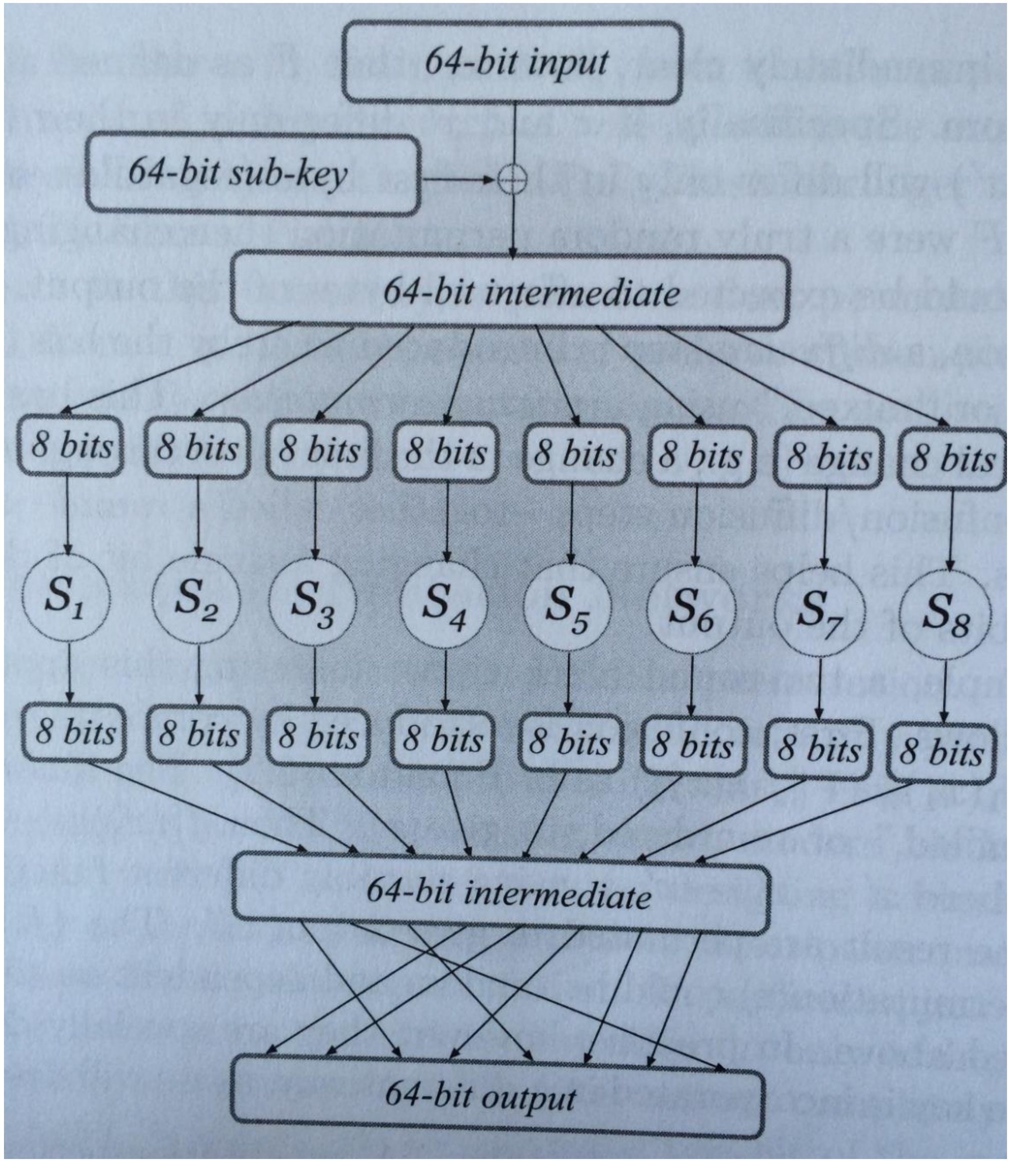
- Introduced by Shannon. Suppose we want to design a 128-bit (keyed) random-looking permutation F.
- First, design an 8-bit (keyed) random-looking permutation f.
- To compute $F_k(x)$:
  - Divide the input block x into sixteen 8-bit blocks $x_1, \ldots, x_{16}$.
  - Use the key k to specify 16 permutations $f_{k_1}, \ldots, f_{k_{16}}$.

    (I.e., derive a round key $\langle k_1, \ldots, k_{16} \rangle$ from the master key k.)
  - Let $x' = f_{k_1}(x_1) \| \cdots \| f_{k_{16}}(x_{16})$ (confusion/substitution).
  - Permutate the 128 bits of x' (diffussion/permutation).
  - Repeat the process several times (rounds).

Substitution-permutation networks

- An implementation of the confusion-diffusion paradigm.
- Harder to design a (keyed) random-looking permutation f.
- So, instead, design 16 (unkeyed) 8-bit permutations $f_1, \ldots, f_{16}$, called S-boxes and denoted by $S_1, \ldots, S_{16}$.
- To compute $F_k(x)$:
  - Divide the input block x into 8-bit blocks $x_1, \ldots, x_{16}$.
  - Derive a round key $\langle k_1, \ldots, k_{16} \rangle$ from the master key k.
  - Let $x' = S_1(x_1 \oplus k_1) \,||\, \cdots \,||\, S_{16}(x_{16} \oplus k_{16})$ (key-mixing & substitution).
  - Permutate the 128 bits of x' (permutation).
  - Repeat the process several times (rounds), followed by a final key-mixing.

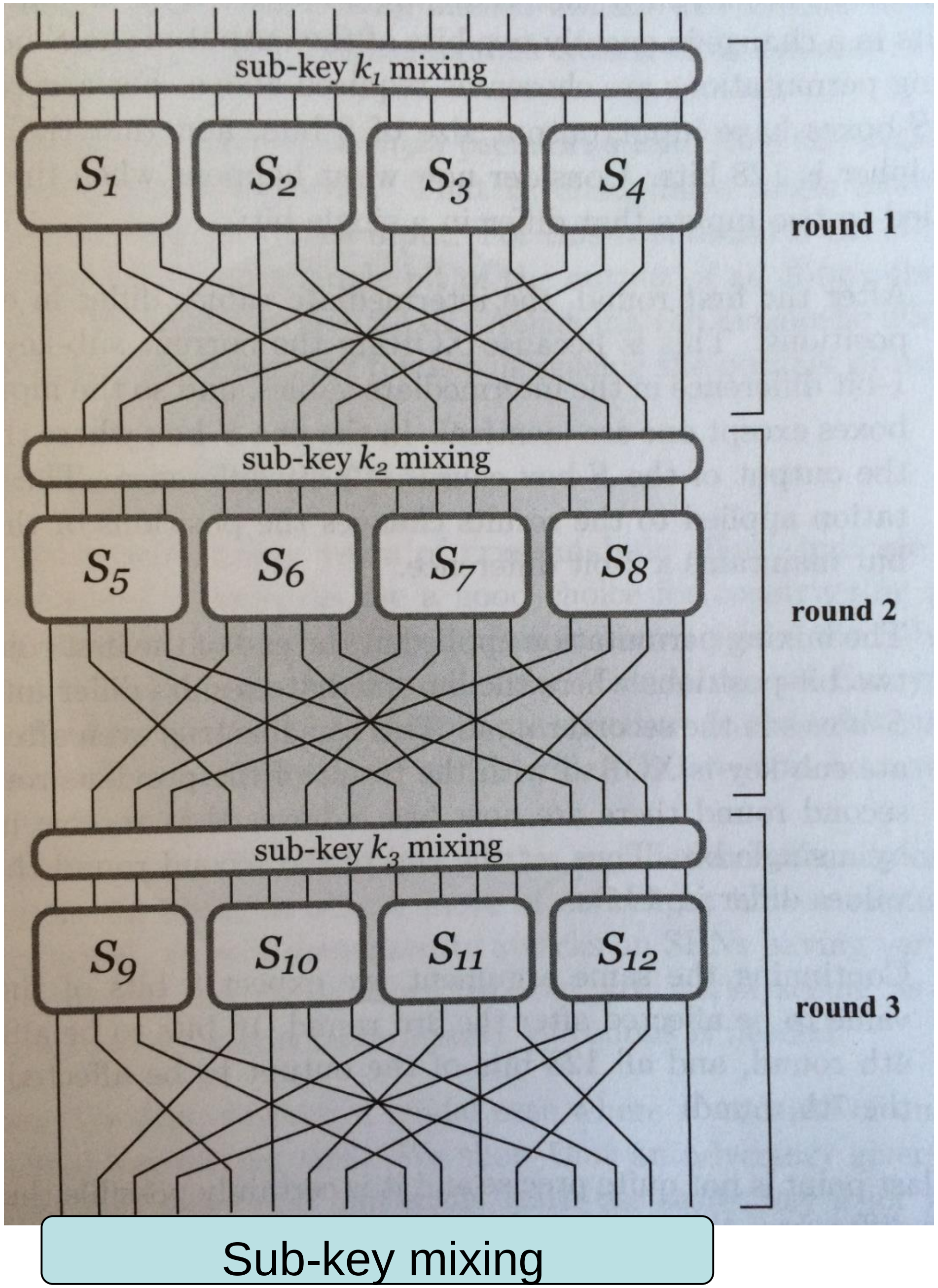
Substitution-permutation network



Key-mixing

Substitution

Permutation



In practice, all rounds use the same set of boxes, say $\{S_1, S_2, S_3, S_4\}$.

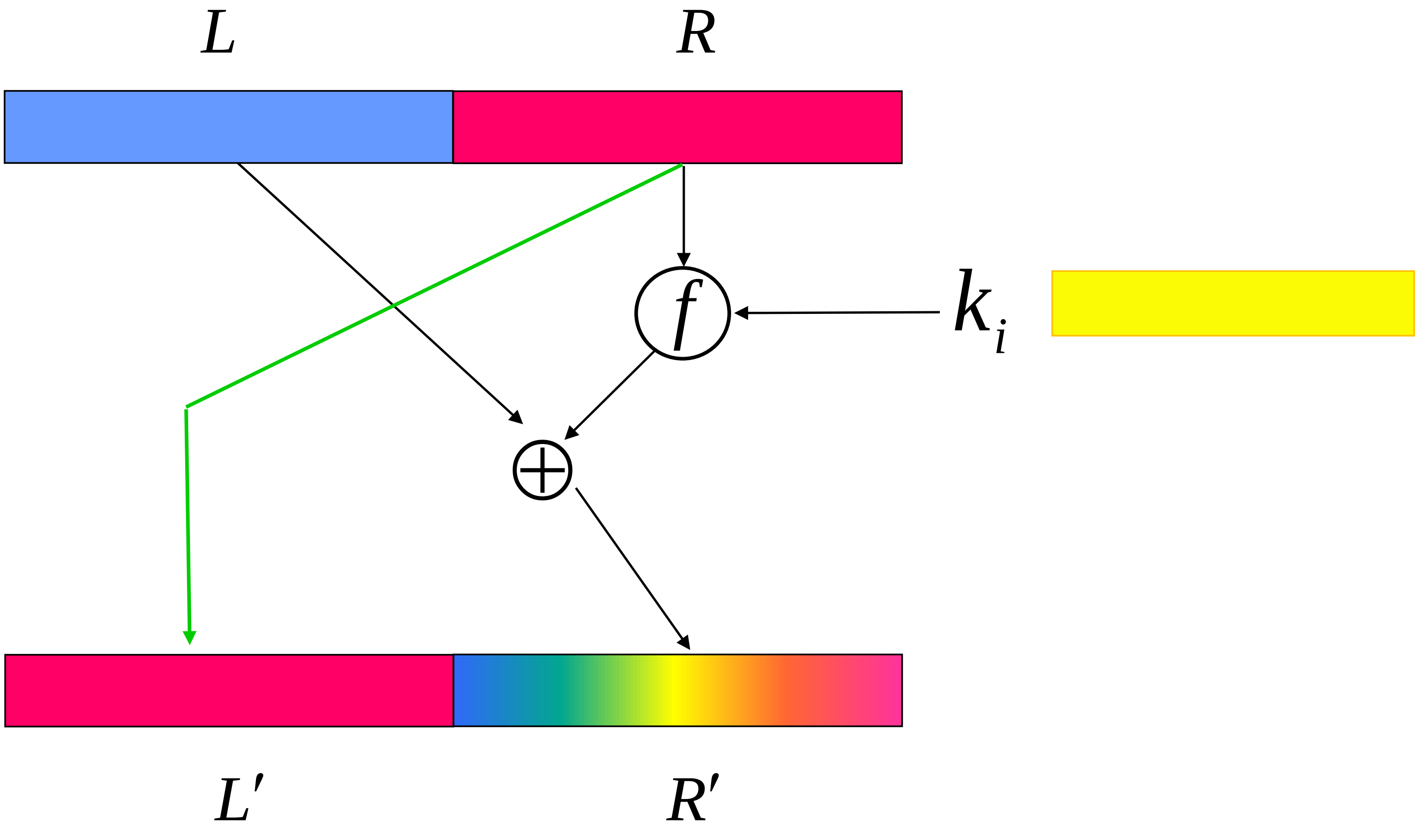
Feistel Networks and
Data Encryption Standard (DES)

Feistel Network/Cipher

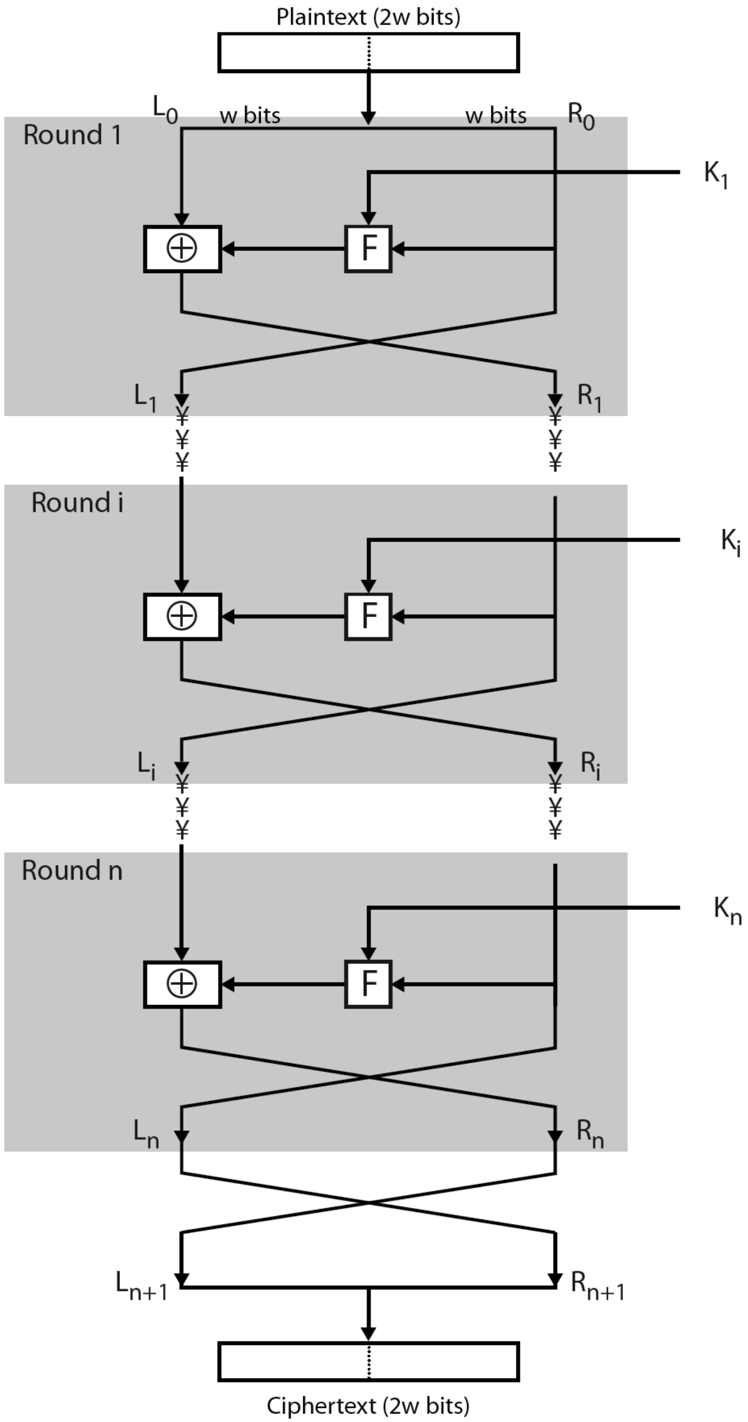
- Proposed by Feistel (in 1970s). Suppose we want to design an ℓ-bit (keyed) random-looking permutation F.
- First, design an $\ell/2$-bit (keyed) random-looking function f, which is not necessarily invertible.
- To compute $F_k(x)$:
  - Divide the input block x into two halves L and R.
  - Derive a round key k_i (for round i) from master key k.
  - Let $x' = R \,||\, L \oplus f_{k_i}(R)$.
  - Repeat the process several times (rounds).
  - (Typically there is a final swap of L and R.)

Round i

f is not invertible



The Feistel Network Structure



Note: Read F as *f*.

Feistel Network/Cipher (Mathematical Description)

- Let L_i and R_i denote the output half-blocks of the ith round.
- So L_{i-1} and R_{i-1} are the input of the ith round.
- We have

$$L_i := R_{i-1}$$
$$R_i := L_{i-1} \oplus f_{k_i}(R_{i-1})$$

- The ith round can be viewed as a composite function $\mu \circ \phi_i$.

$$\phi_i : (L, R) \rightarrow (L \oplus f_{k_i}(R), R).$$
$$\mu : (L, R) \rightarrow (R, L).$$

- Note that $\phi_i^{-1} = \phi_i$ and $\mu^{-1} = \mu$.

- Assume 16 rounds.
- A Feistel cipher with key k and input block x will output:

$$y = F_k(x) = \mu \circ \mu \circ \phi_{16} \circ \cdots \circ \mu \circ \phi_2 \circ \mu \circ \phi_1(x)$$

- The inverse $F_k^{-1}(y)$ will be:

$$F_k^{-1}(y) = \phi_1^{-1} \circ \mu^{-1} \circ \phi_2^{-1} \circ \cdots \circ \mu^{-1} \circ \phi_{16}^{-1} \circ \mu^{-1} \circ \mu^{-1}(y)$$
$$= \mu \circ \mu \circ \phi_1 \circ \mu \circ \phi_2 \circ \cdots \circ \mu \circ \phi_{16}(y)$$

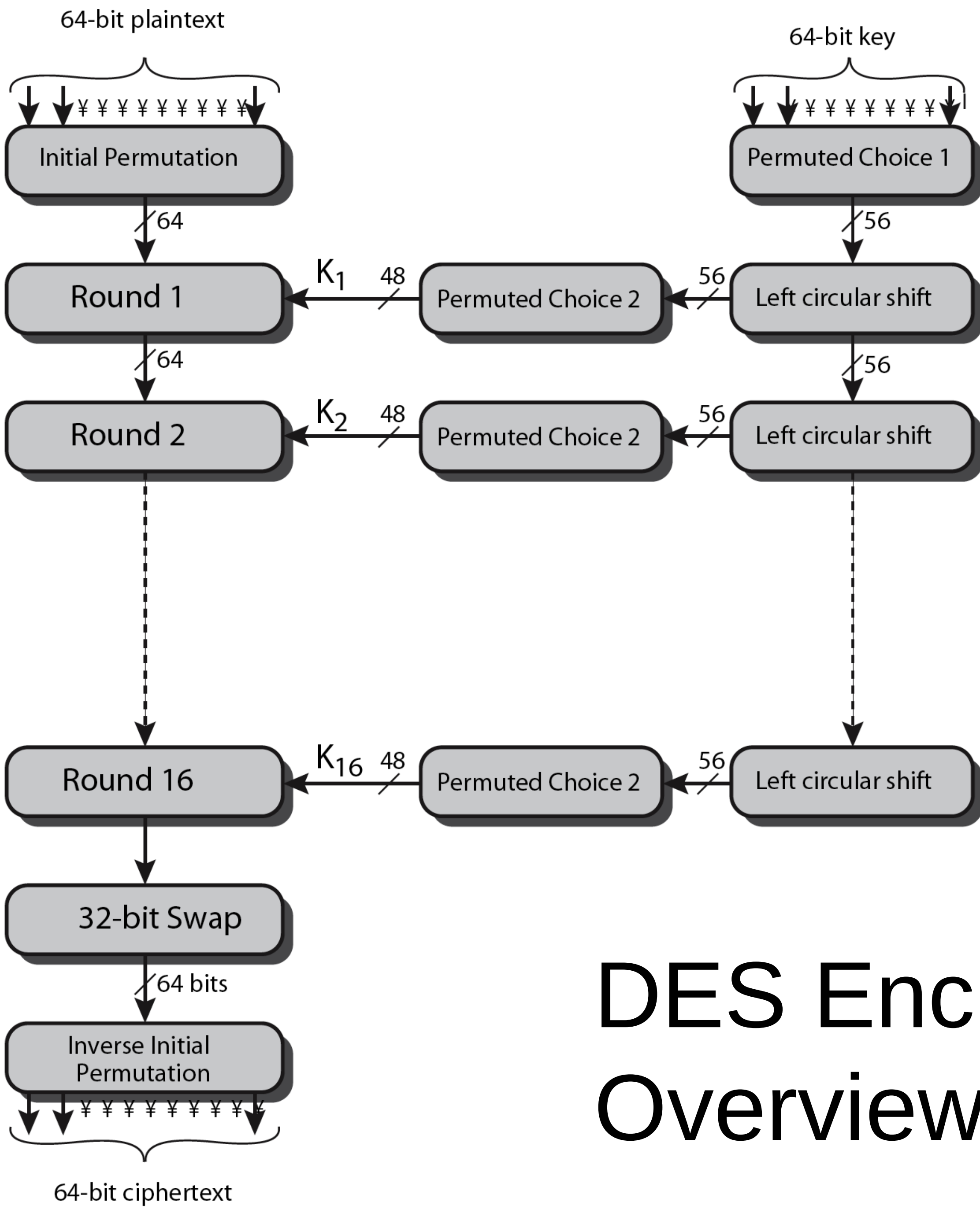
- F_k^{-1} is the same as F_k, but uses the round keys in the reverse order.

DES: The Data Encryption Standard

- Once most widely used block cipher in the world.
- Adopted by NIST in 1977.
- Based on the Feistel cipher structure with 16 rounds of processing.
- Block = 64 bits
- Key = 56 bits
- What is specific to DES is the design of the f function and how the round keys are derived from the main key.

Design Principles of DES

- To achieve high degree of **confusion** and **diffusion**.
- Confusion: making the relationship between the encryption key and the ciphertext, as well as that between the plaintext and the ciphertext, as complex as possible.
- Diffusion: making each plaintext bit affect as many ciphertext bits as possible.



DES Encryption Overview

Round Key Generation

- Main key: 64 bits, but only 56 bits are used.
- 16 round keys (48 bits each) are generated from the main key by a sequence of permutations.
- Select and permute 56-bits using Permuted Choice One (PC1). Then divide them into two **28-bit halves**.
- At each round:
  - Rotate **each half** separately by either 1 or 2 bits according to a rotation schedule.
  - Select 24-bits from each half & permute them (48 bits) by PC2.
  - This forms a round key.

Permuted Choice One (PC1)

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

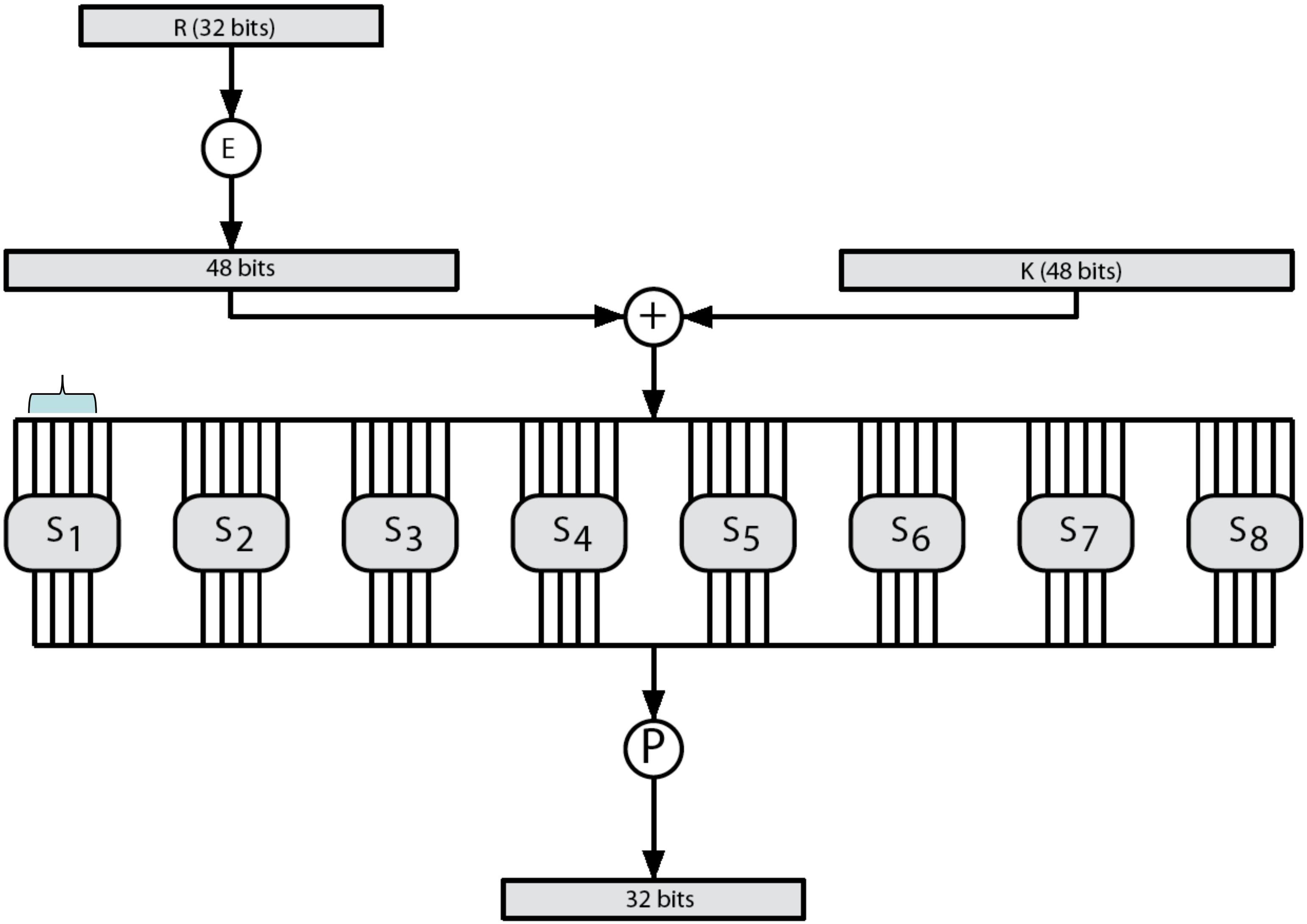
DES Round Structure

- L & R each has 32 bits.
- As in any Feistel cipher:

$$L_i := R_{i-1}$$
$$R_i := L_{i-1} \oplus f_{k_i}(R_{i-1})$$

- f takes 32-bit R and 48-bit round key k_i:
  - expands R to 48-bits using expansion perm E
  - adds to the round key using XOR
  - shrinks to 32-bits using 8 S-boxes
  - finally permutes using 32-bit perm P

The DES *f* function



The E 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

The S-Boxes

- Eight S-boxes each map 6 to 4 bits
- Each S-box is a 4 x 16 table
  - each row is a permutation of 0-15
  - outer bits 1 & 6 of input are used to select one of the four rows/permutations
  - inner 4 bits of input are used to select a column
- All the eight boxes are different.

Box S_1

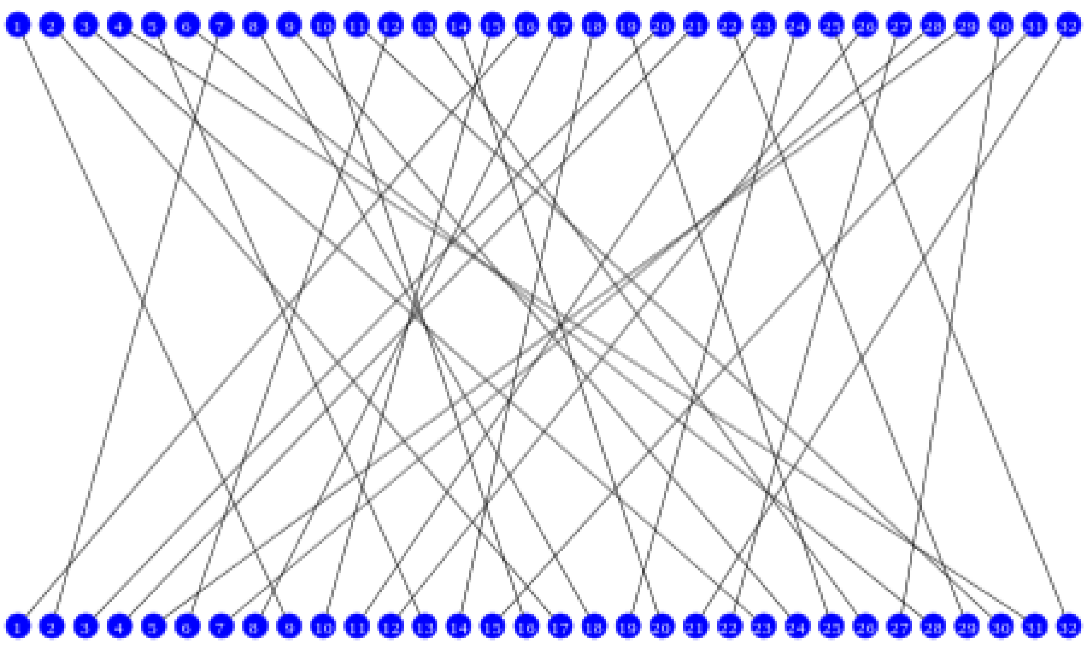
S_1

	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	6	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

- For example, $S_1(101010) = 6 = 0110$.

P-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



Avalanche Effect

- Avalanche effect: a key desirable property of any encryption algorithm:
  - A small change in the plaintext or in the key results in a significant change in the ciphertext.
  - (an evidence of high degree of diffusion and confusion)
- DES exhibits a strong avalanche effect
  - Changing 1 bit in the plaintext affects 34 bits in the ciphertext on average.
  - 1-bit change in the key affects 35 bits in the ciphertext on average.

Attacks on DES

- Brute-force key search
  - Needs only two plaintext-ciphertext samples
  - Trying 1 key per microsecond would take 1000+ years on average, due to the large key space size, $2^{56} \approx 7.2 \times 10^{16}$.
- Differential cryptanalysis
  - Possible to find a key with 2^{47} plaintext-ciphertext samples
  - Known-plaintext attack
- Linear cryptanalysis:
  - Possible to find a key with 2^{43} plaintext-ciphertext samples
  - Known-plaintext attack

Attacks on DES

- **DES Cracker**:
  - A DES key search machine
  - containing 1536 chips
  - could search 88 billion keys per second
  - In 1998, won RSA Laboratory's **DES Challenge II-2** by successfully finding a DES key in 56 hours.
  - Cost: $250,000
- The vulnerability of DES is due to its short key length.
- Remedy: 3DES

Multiple Encryption with DES

- In 2001, NIST published the Advanced Encryption Standard (AES) to replace DES.
- But users in commerce and finance are not ready to give up on DES.
- As a temporary solution to DES's security problem, one may encrypt a message (with DES) multiple times using multiple keys:
  - 2DES is not much securer than the regular DES
  - So, 3DES with either 2 or 3 keys is used

2DES

- Use two DES keys, say $k_1,\ k_2$.
- Encryption: $c := Enc_{k_2}\left(Enc_{k_1}(m)\right)$
- Key length: $56 \times 2 = 112$ bits
- Would this thwart brute-force attacks?

Meet-in-the-Middle Attack on 2DES

$$m \rightarrow \boxed{\mathrm{Enc}_{k_1}} \rightarrow \boxed{\mathrm{Enc}_{k_2}} \rightarrow c$$

- Given a known pair (m, c), attack as follows:
  - Encrypt m with all 2^{56} possible keys for k_1.
  - Decrypt c with all 2^{56} possible keys for k_2.
  - Find two keys $\tilde{k}_1, \tilde{k}_2$ such that $\mathrm{Enc}_{\tilde{k}_1}(m) = \mathrm{Dec}_{\tilde{k}_2}(c)$.
  - Try $\tilde{k}_1, \tilde{k}_2$ on another pair (m', c'): Is $\mathrm{Enc}_{\tilde{k}_1}(m') = \mathrm{Dec}_{\tilde{k}_2}(c')$?
  - If works, $(\tilde{k}_1, \tilde{k}_2) = (k_1, k_2)$ with high probability.
  - Takes $\Theta(2^{56})$ steps, not much more than attacking 1-DES.
- It is a known-plaintext attack.

3DES with 2 keys

- A straightforward implementation would be:

$$c := Enc_{k_1}\left(Enc_{k_2}\left(Enc_{k_1}(m)\right)\right)$$

- In practice: $c := Enc_{k_1}\left(Dec_{k_2}\left(Enc_{k_1}(m)\right)\right)$
  - Also referred to as EDE encryption
- Reason: if $k_1 = k_2$, then 3DES $=$ 1DES.
  Thus, a 3DES software can be used as a single-DES.
- Standardized in ANSI X9.17 & ISO 8732.
- No practical attacks are known.
- Not recommended: key size 112 bits is shorter than the current minimum recommendation of 128 bits.

3DES with 3 keys

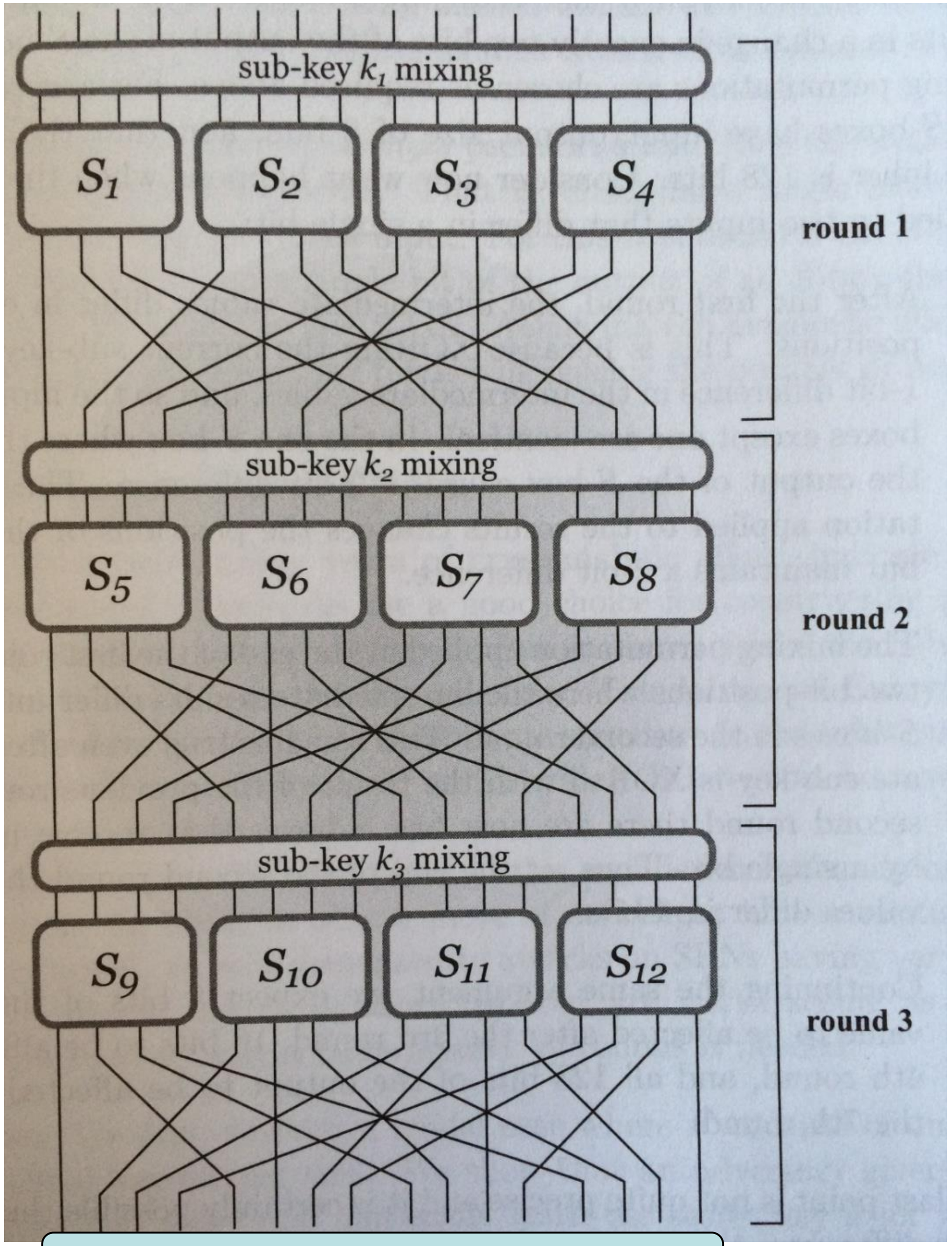
- Encryption: $c := Enc_{k_3}\left(Dec_{k_2}\left(Enc_{k_1}(m)\right)\right).$
- If $k_1 = k_3,$ it becomes 3DES with 2 keys.
- If $k_1 = k_2 = k_3,$ it becomes the regular DES.
- So, it is backward compatible with both 3DES with 2 keys and the regular DES.
- Some internet applications adopt 3DES with three keys, e.g. PGP and S / MIME.

AES: Advanced Encryption Standard

Finite field: The mathematics used in AES.

AES: Advanced Encryption Standard

- In1997, NIST began the process of choosing a replacement for DES and called it the **Advanced Encryption Standard**.
- Requirements: block length of 128 bits, key lengths of 128, 192, and 256 bits.
- In 2000, **Rijndael** cipher (by Rijmen and Daemen) was selected.
- An iterated cipher, with 10, 12, or 14 rounds.
- Rijndael allows various block lengths.
- AES allows only one block size: 128 bits.



Sub-key mixing

In practice, all rounds use the same set of boxes, say $\{S_1,S_2,S_3,S_4\}$.

Structure of Rijndael

- N_b: block size (number of words). For AES, $N_b = 4$.
- N_k: key length (number of words).
- N_r: number of rounds, depending on N_b, N_k.
- Assume: $N_b = 4,\ \ N_k = 4,\ \ N_r = 10.$
- *state*: a variable of 4 words, holding the data block, viewed as a 4×4 matrix of bytes; each column is a word.
- Key schedule: $N_r + 1$ round keys $key_0, key_1, \ldots, key_{10}$ are computed from the main key k.

Rijndael algorithm $\left(\text{input: plaintext } m, \text{ key } k\right)$

$state \leftarrow m$

$\text{AddKey}(state, key_0)$

$\text{for } i \leftarrow 1 \text{ to } N_r - 1 \text{ do}$

$\quad \text{SubBytes}(state)$

$\quad \text{ShiftRows}(state)$

$\quad \text{Mixcolumns}(state)$

$\quad \text{AddKey}(state, key_i)$

$\text{SubBytes}(state)$

$\text{ShiftRows}(state)$

$\text{AddKey}(state, key_{N_r})$

$\text{return}(state)$

AddKey(*state*, key_i)

$$state \leftarrow state \oplus key_i$$

SubBytes(*state*)

- Each byte z in *state* is substituted with another byte according to a table.

ShiftRows(*state*)

- Left-shift row i circularly by i bytes, $0 \le i \le 3$.

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \rightarrow \begin{pmatrix} a & b & c & d \\ f & g & h & e \\ k & l & i & j \\ p & m & n & o \end{pmatrix}$$

MixColumns(*state*)

- Operates on each column of the *state* matrix.
- View each column $a = (a_0,\ a_1,\ a_2,\ a_3)$ as a polynomial with coefficients in $\mathrm{GF}(2^8)$:

$$a(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_0$$

- A fixed polynomial: $c(x) = 03x^3 + 01x^2 + 01x + 02.$
- The MixColumns operation maps each column

$$a(x) \mapsto a(x) \cdot c(x) \ \mathrm{mod}\ (x^4 + 1)$$

Rijndael Decryption

- Each step of Rijndael encryption is invertible.

Rijndael key schedule

- Round keys are derived from the main key

[A Rijndael Animation by Enrique Zabala]