Lecture 3
Symmetric Encryption II

Stefan Dziembowski
www.crypto.edu.pl/Dziembowski

University of Warsaw

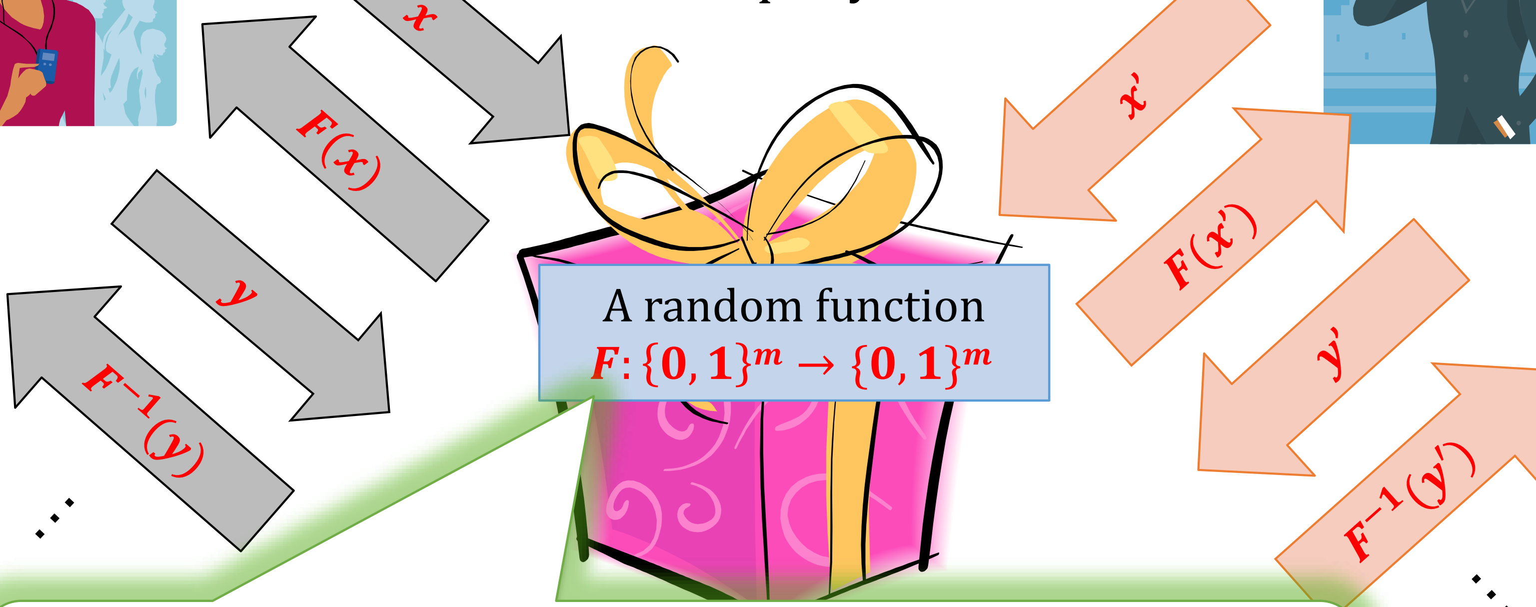
19.10.16 **version 1.0**

Plan

1. Pseudorandom functions
2. Block cipher modes of operation
3. Feistel ciphers
4. Substitution-permutation networks
5. Cascade ciphers
6. Practical considerations

Random permutations

Suppose we have a box
with a “random function”
that Alice and Bob can
query



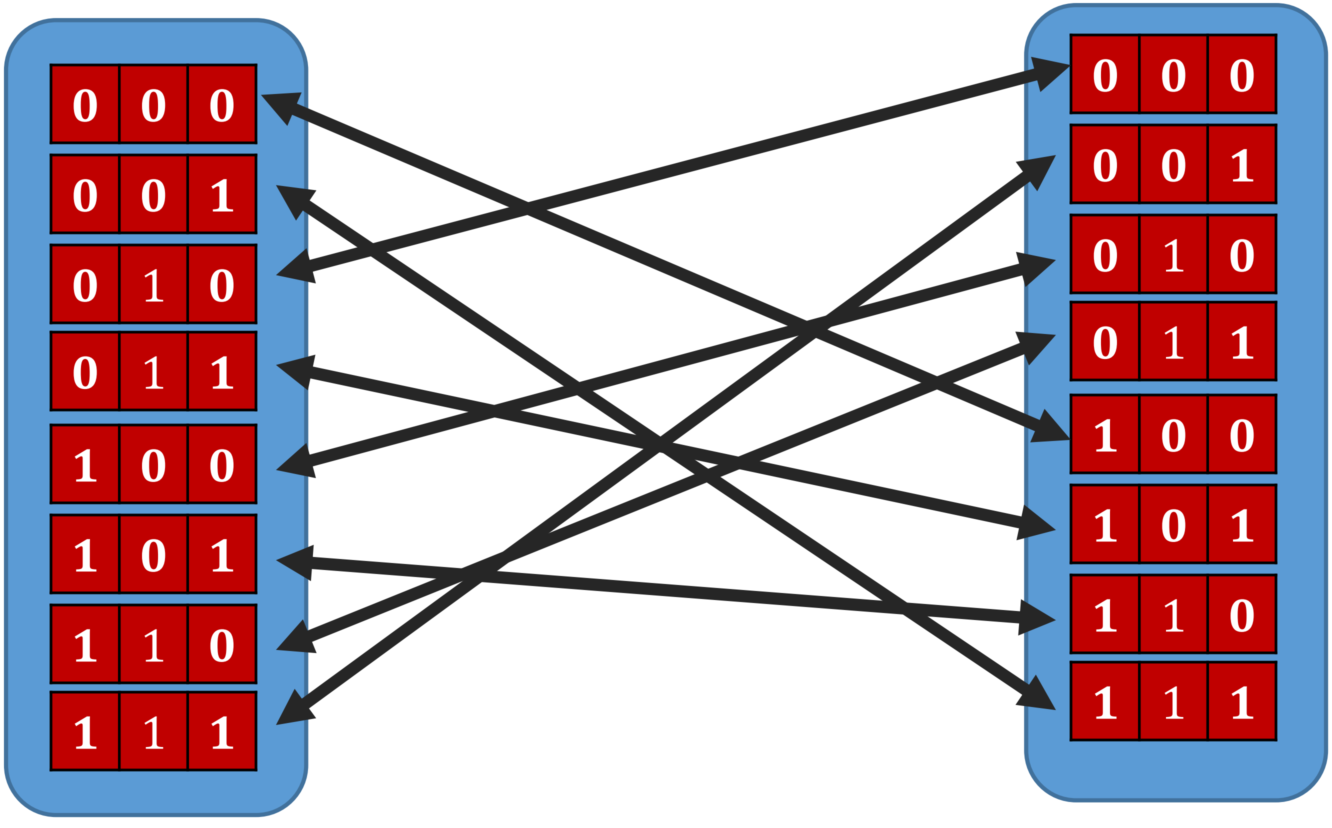
A random function
$F: \{0,1\}^m \to \{0,1\}^m$

suppose F is a bijection
In other words: it is a **permutation on** $\{0,1\}^m$

Note

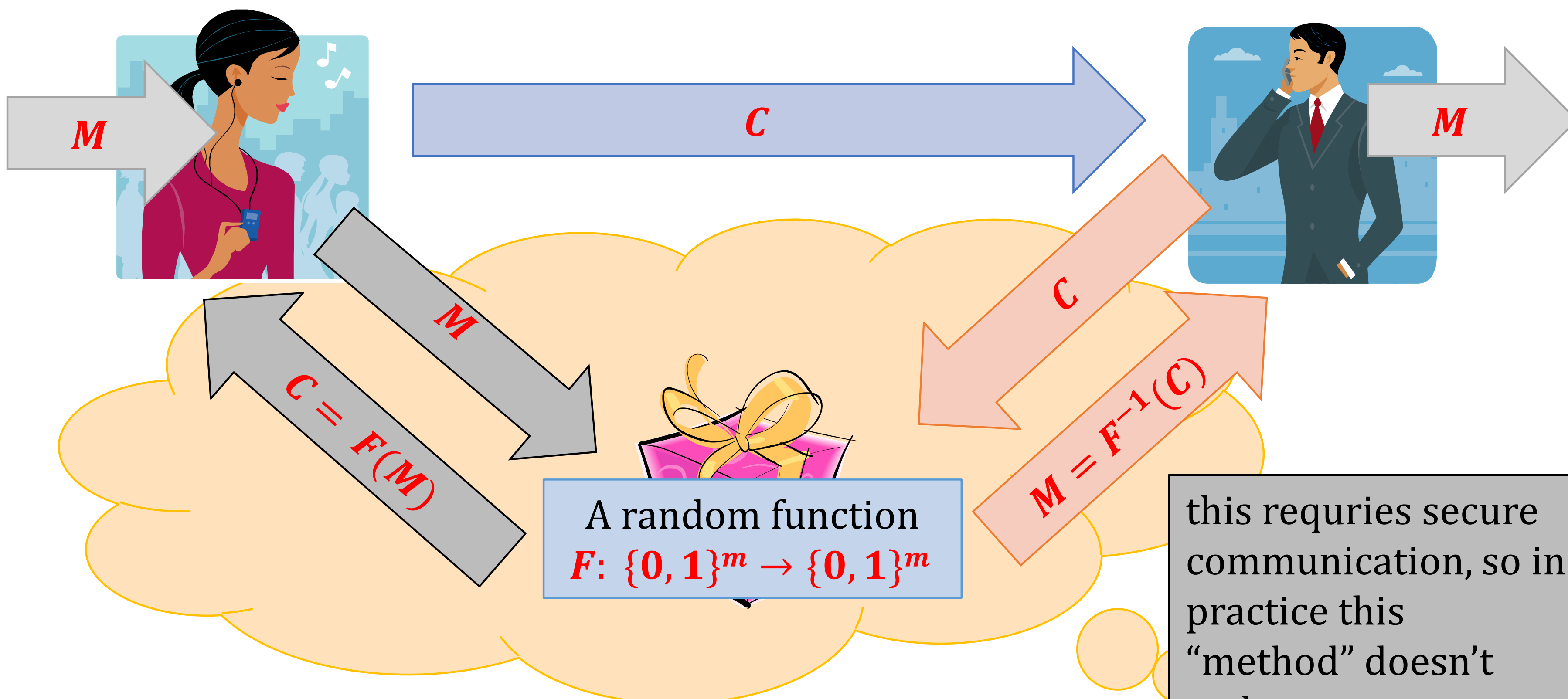
We consider permutations on $\{\mathbf{0}, \mathbf{1}\}^m$, **not** on $\{\mathbf{1}, \dots, \boldsymbol{m}\}$

Example:



Example of an application: “encryption”

Suppose that $\mathcal{M} = \{0,1\}^m$. If only one message is sent then Alice and Bob can do the following:

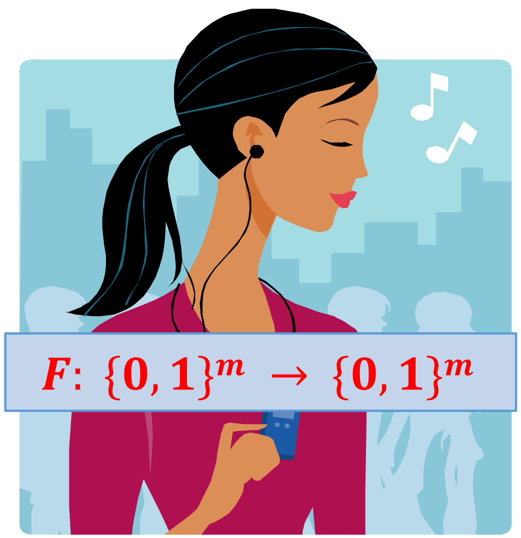


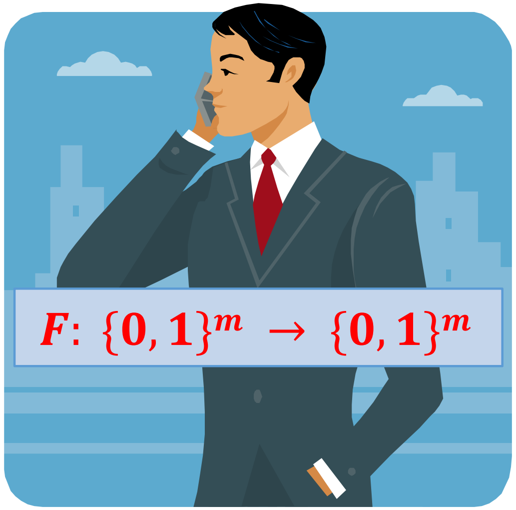
this requries secure communication, so in practice this “method” doesn’t make sense...

Can this box be simulated in real life?

<u>Naive solution</u>:

Select a random permutation $\boldsymbol{F}: \{\mathbf{0}, \mathbf{1}\}^m \rightarrow \{\mathbf{0}, \mathbf{1}\}^m$ and give it to both parties.





<u>Problem</u>:

The number of possible permutations is $(\mathbf{2}^m)!$

An idea

One **cannot** describe a random permutation

$$F: \{0,1\}^m \rightarrow \{0,1\}^m$$

in a short space.

But maybe one can do it for a function that "behaves almost like random"?

Answer:

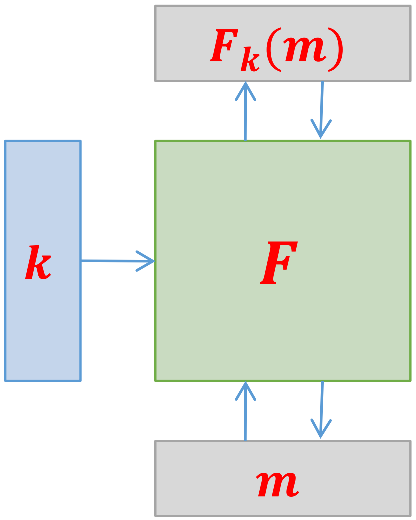
<u>YES, it is possible!</u> (under certain assumptions)

objects like these are called

- **pseudorandom permutations (by the theoreticians)**
- **block ciphers (by the practitioners)**

Keyed permutations

For a partial function
$$F: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$$
let $F_k(m)$ denote $F(k,m)$.



A **keyed-permutation** is a function
$F: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ such that

1. for every k function F_k is a permutation on some $\{0,1\}^n$

2. for every k functions F_k and F_k^{-1} are poly-time computable.

n is a function of $|k|$

for simplicity assume: $n = |k|$

Pseudorandom permutations

Intuition:

A keyed permutation ***F*** is **pseudorandom** if it cannot be distinguished from a completely random permutation.

This has to be formalized

Scenario **0**

security parameter
$\mathbf{1^n}$

distinguisher $\boldsymbol{D}$

oracle
chooses a random $\boldsymbol{k} \in \{\mathbf{0}, \mathbf{1}\}^n$.

$\boldsymbol{m_1} \in \{\mathbf{0}, \mathbf{1}\}^n$ →

← $\boldsymbol{F_k(m_1)}$

$\boldsymbol{m_2} \in \{\mathbf{0}, \mathbf{1}\}^n$ →

← $\boldsymbol{F_k(m_2)}$

⋮

$\boldsymbol{m_t} \in \{\mathbf{0}, \mathbf{1}\}^n$ →

← $\boldsymbol{F_k(m_t)}$

outputs $\boldsymbol{b} \in \{\mathbf{0}, \mathbf{1}\}$

Scenario 1

security parameter
$\mathbf{1^n}$

distinguisher D

$m_1 \in \{0,1\}^n$

$F(m_1)$

$m_2 \in \{0,1\}^n$

$F(m_2)$

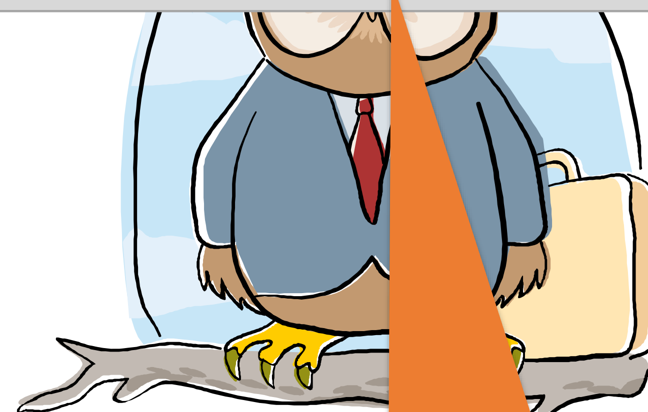
$\vdots$

$m_t \in \{0,1\}^n$

$F(m_t)$

outputs $b \in \{0,1\}$

oracle
chooses a random function
$F : \{0,1\}^n \rightarrow \{0,1\}^n$



This of course **cannot** be done efficiently, but it doesn't matter

Pseudorandom permutations – the definition

We say that a **keyed-permutation** $\boldsymbol{F}\colon \{\mathbf{0},\mathbf{1}\}^* \times \{\mathbf{0},\mathbf{1}\}^* \to \{\mathbf{0},\mathbf{1}\}^*$ is a **pseudorandom permutation (PRP)** if

any **polynomial-time randomized distinguisher** $\boldsymbol{D}$

cannot distinguish **scenario 0** from **scenario 1** with a **non-negligible advantage**.

That is:

$$|\boldsymbol{P}(\boldsymbol{D}\ \textbf{outputs "1" in scenario 0}) - \boldsymbol{P}(\boldsymbol{D}\ \textbf{outputs "1" in scenario 1})|$$

is negligible in $\boldsymbol{n}$.

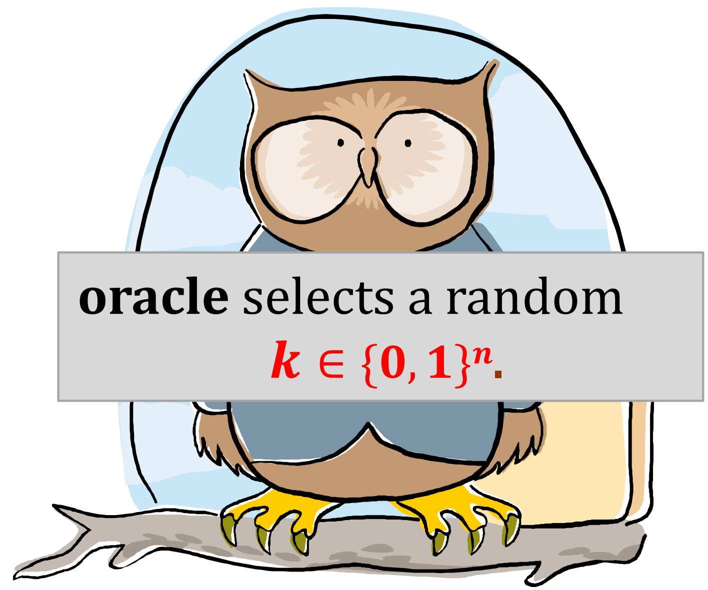
Strong pseudorandom permutations

Suppose we allow the distinguisher to **additionally** ask the oracle for inverting $\boldsymbol{F}$:

distinguisher $\boldsymbol{D}$

$\boldsymbol{c_i \in \{0,1\}^n}$ →

← $\boldsymbol{F_k^{-1}(c_i)}$



oracle selects a random $\boldsymbol{k \in \{0,1\}^n}$.

Then we get a definition of a **strong pseudorandom permutation.**

PRFs vs PRP

If we drop the assumption that

F_k **has to be a permutation**

we obtain an object called

a "pseudorandom **function (PRF)**".

The security definition doesn't change.

In fact those two objects are **indistinguishable** for a polynomial-time adversary.

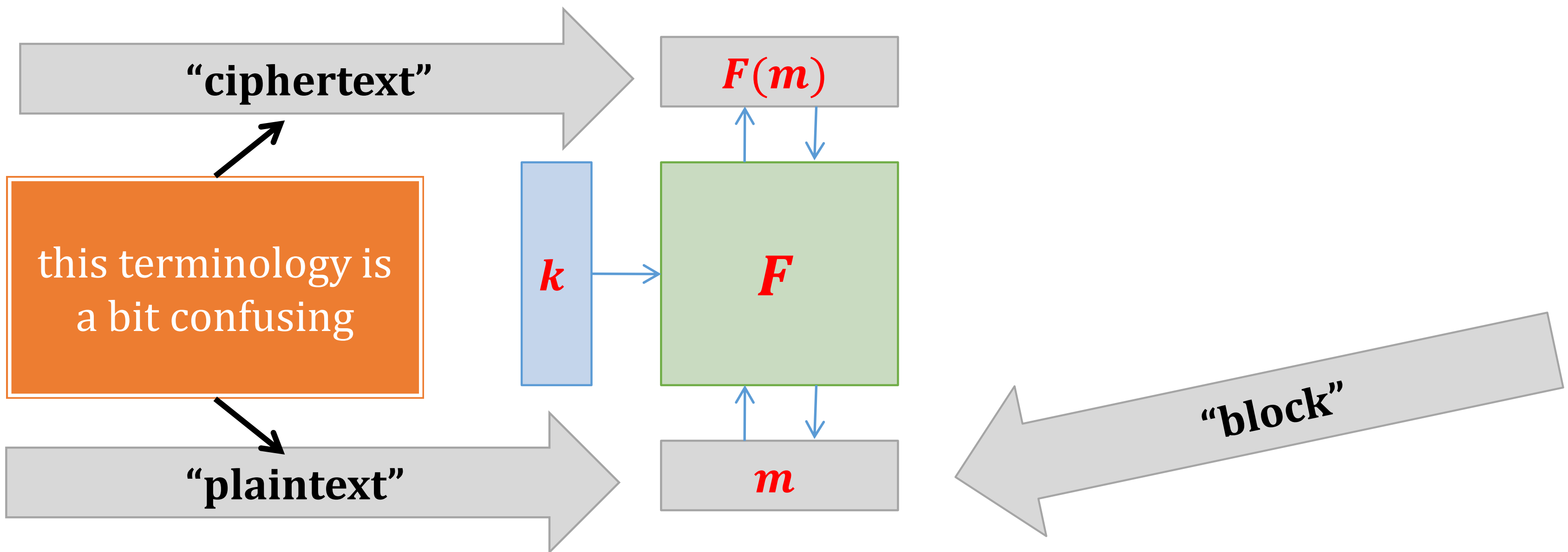
Terminology

Before we had:

stream ciphers ≈ **pseudorandom generators**

Similarly:

block ciphers ≈ **pseudorandom <u>permutations</u>**



Another way to look at the stream ciphers :

$\boldsymbol{m}$ is a parameter

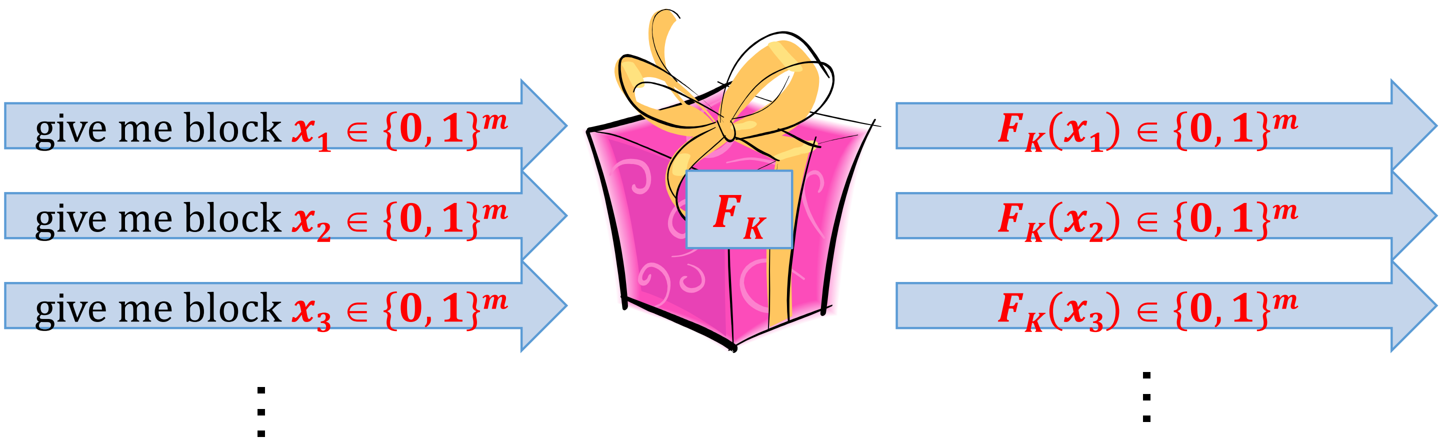
<u>Requirement</u>:

$$G_K(1), G_K(2), G_K(3), \ldots$$

has to “look random” if is $\boldsymbol{K}$ random and secret.

Block ciphers:

$\boldsymbol{m}$ is a parameter



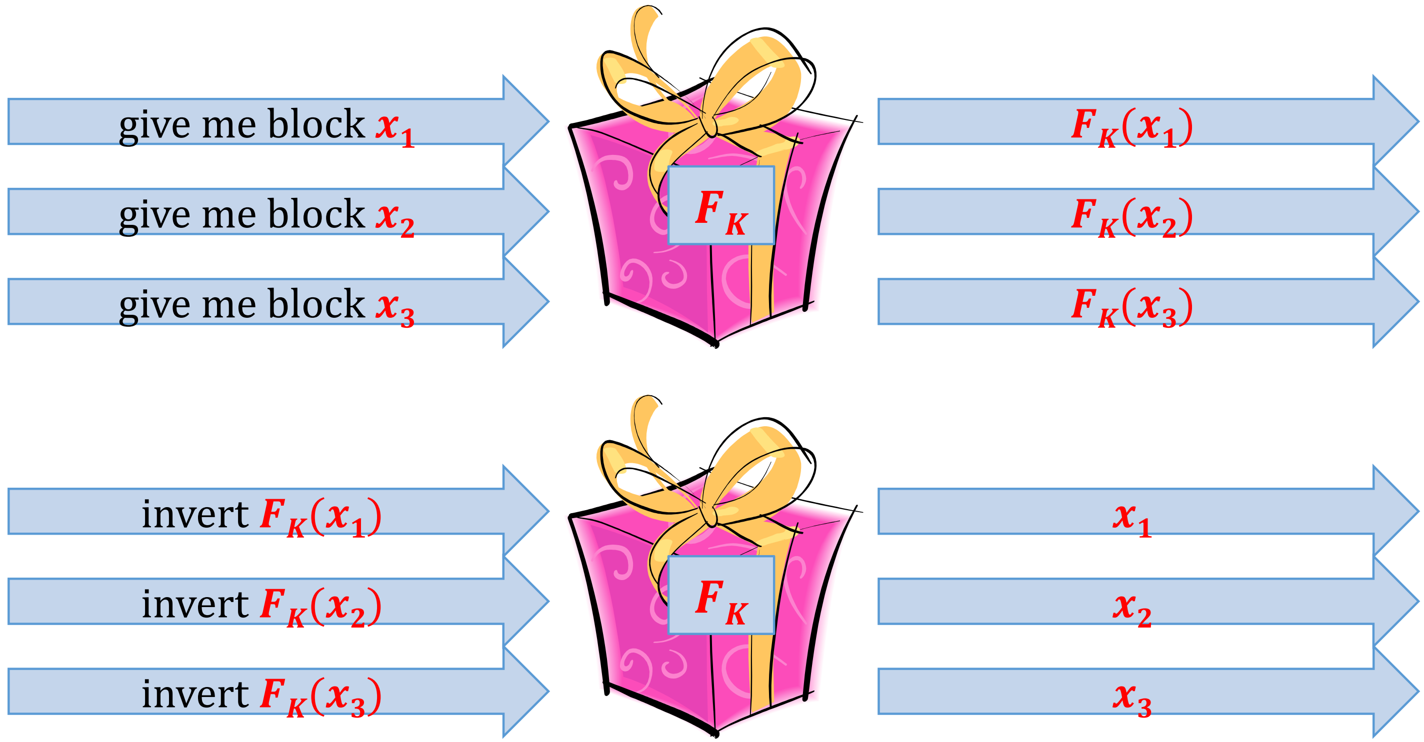
for $x_1, x_2, x_3 \ldots$ chosen adversarially

<u>Requirement</u>:

$$F_K(x_1), F_K(x_2), F_K(x_3), \ldots$$

has to “look random” if is K random and secret.

Additional property of the block ciphers



Popular block ciphers

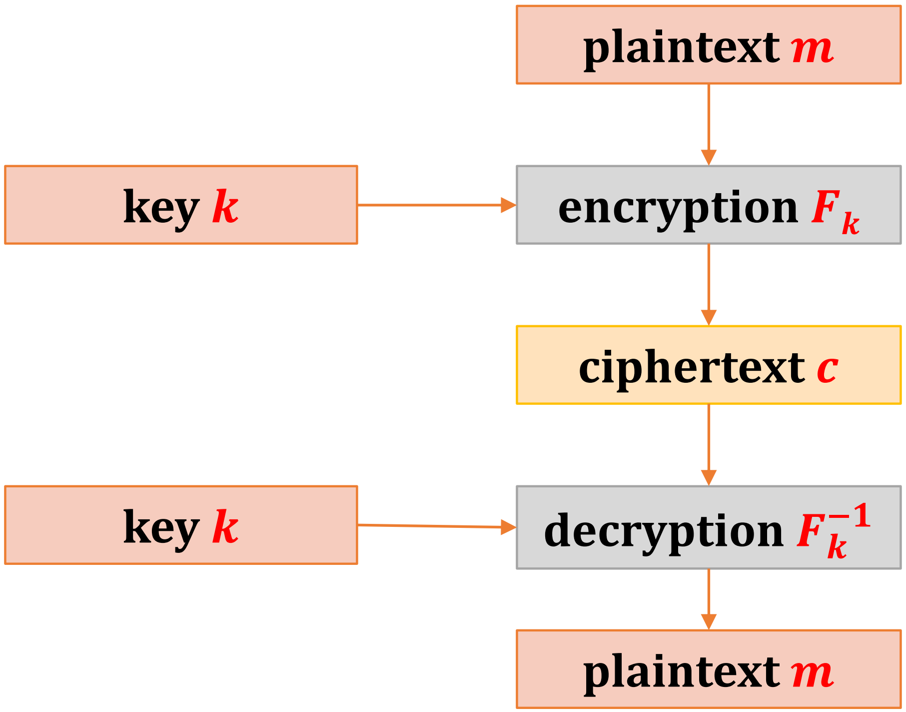
A great design.
The only practical weakness: **short key**.
Can be broken by a **brute-force attack**.

	key length	block length
DES (1976) (**Data Encryption Standard**)	**56**	**64**
IDEA (1991) (**International Data Encryption Algorithm**)	**128**	**64**
AES (1998) (**Advanced Encryption Standard**)	**128**, **192** or **256**	**128**

Other: **Blowfish, Twofish, Serpent,...**

How to encrypt using the block ciphers?

<u>A naive (wrong) idea</u>: Encrypt short blocks:



Problems:

1. the messages have to be short
2. it is **deterministic** and **has no state**, so it **cannot be CPA-secure**.

Plan

1. Pseudorandom functions
2. Block cipher modes of operation
3. Feistel ciphers
4. Substitution-permutation networks
5. Cascade ciphers
6. Practical considerations

Block cipher modes of operation

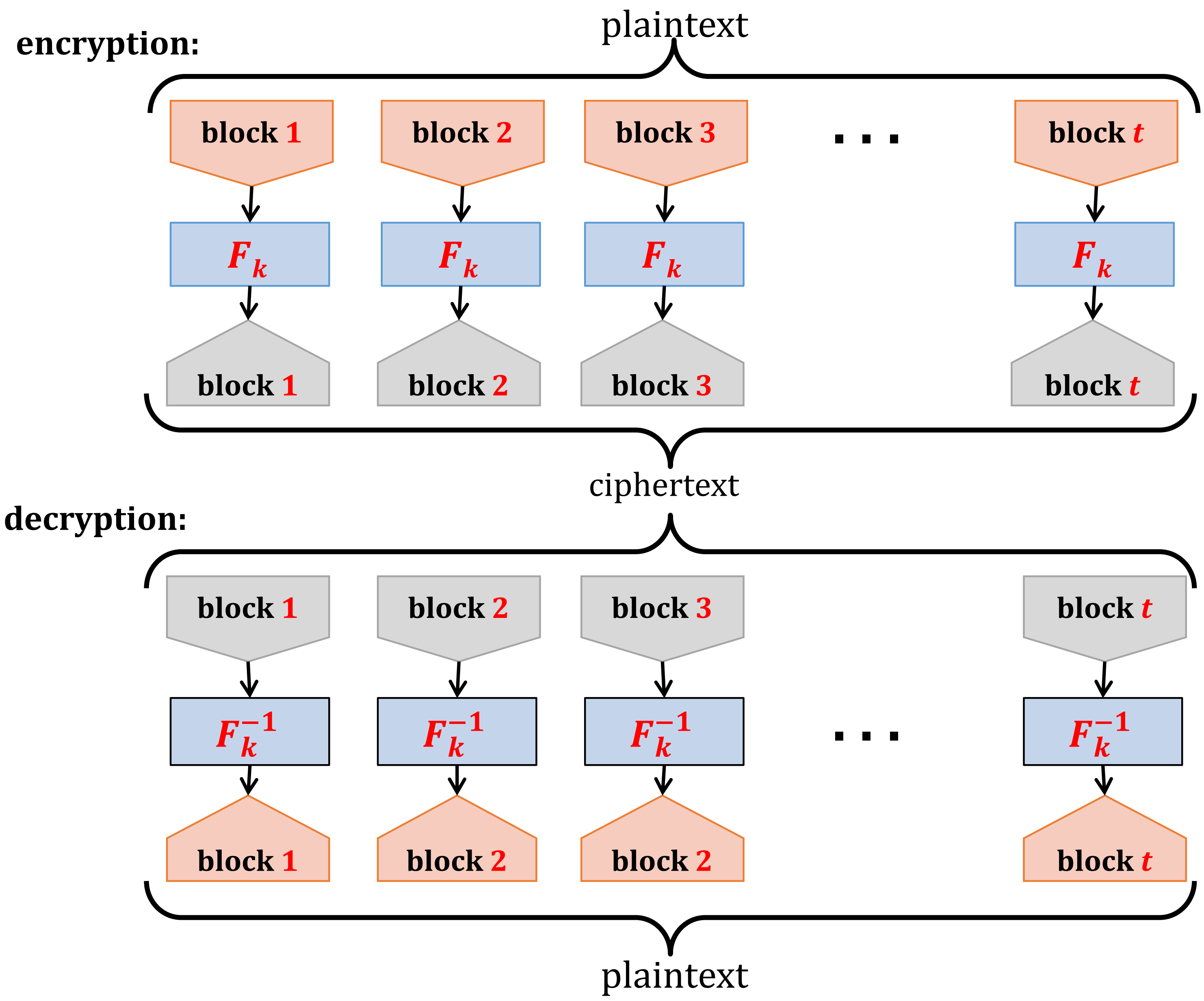
Block ciphers **cannot be used directly for encryption**.

They are always used in some "**modes of operation**"

1. **Electronic Codebook (ECB)** mode ← **not secure**,
2. **Cipher-Block Chaining (CBC)** mode,
3. **Output Feedback (OFB)** mode,
4. **Counter (CTR)** mode,

   ...

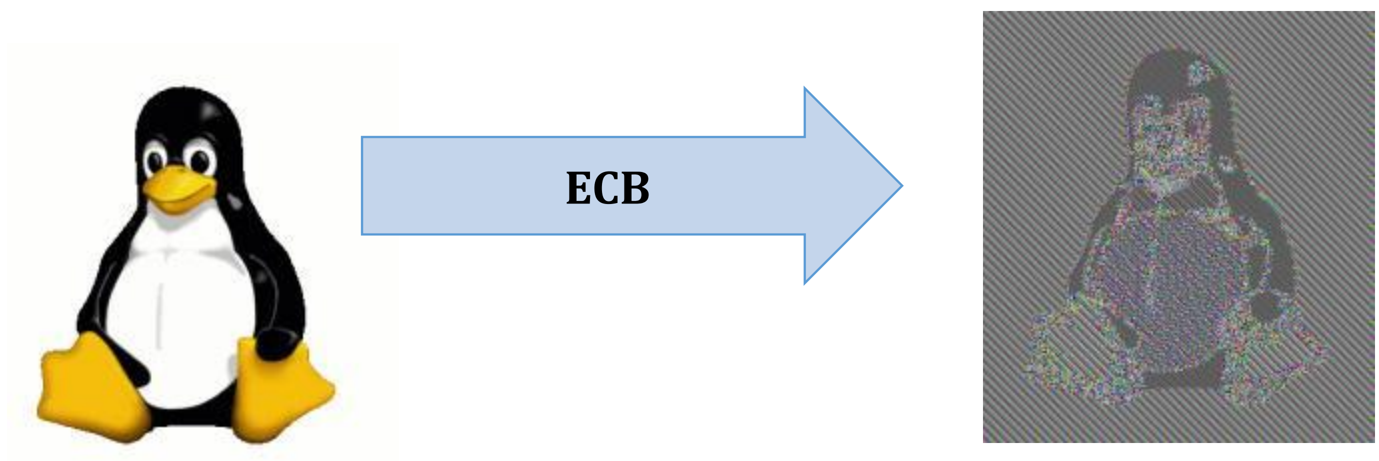
Electronic Codebook mode



This mode was used in the past.

It is not secure, and should not be used.

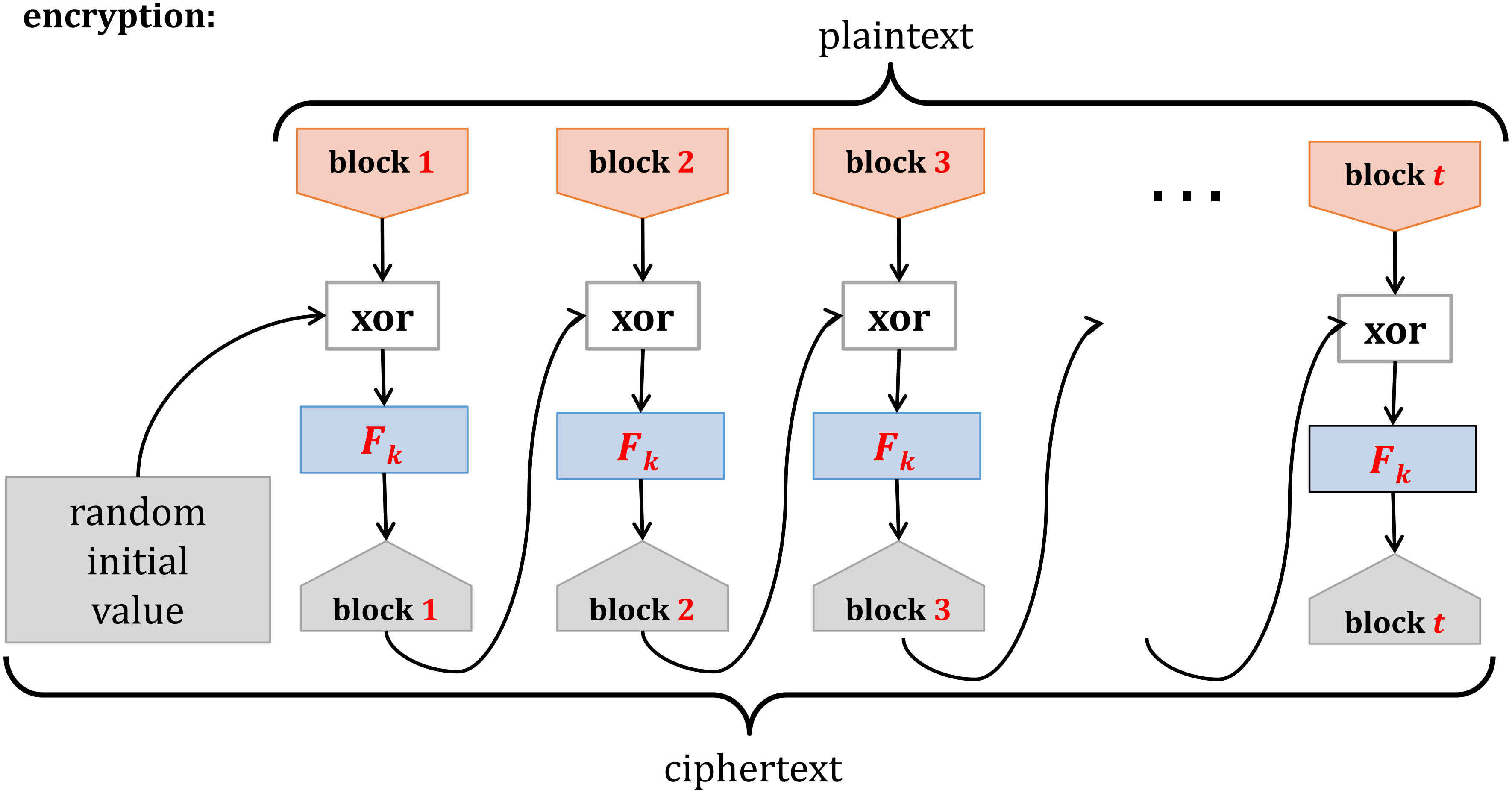
Example:



© *wikipedia*

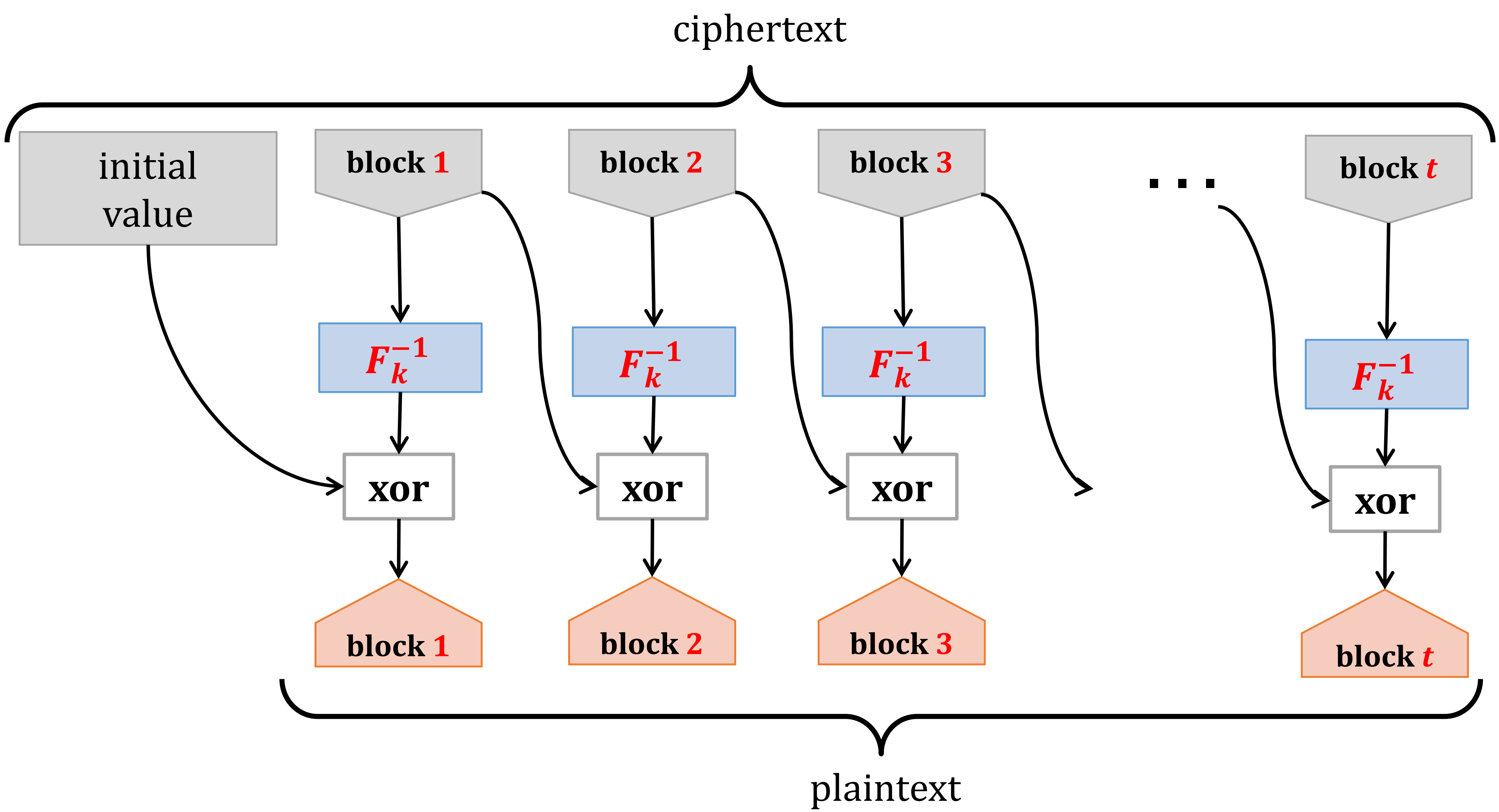
Cipher-Block Chaining (CBC)

encryption:



Cipher-Block Chaining (CBC)

decryption:



CBC mode – properties

Error propagation?

Error in block c_i affects only c_i and c_{i+1}.
So, errors don't propagate (This mode is **self-synchronizing**)

Can encryption be parallelized?

No

Can decryption be parallelized?

Yes

What if one bit of plaintext is changed?

Everything needs to be recomputed
(not so good e.g. for disc encryption)

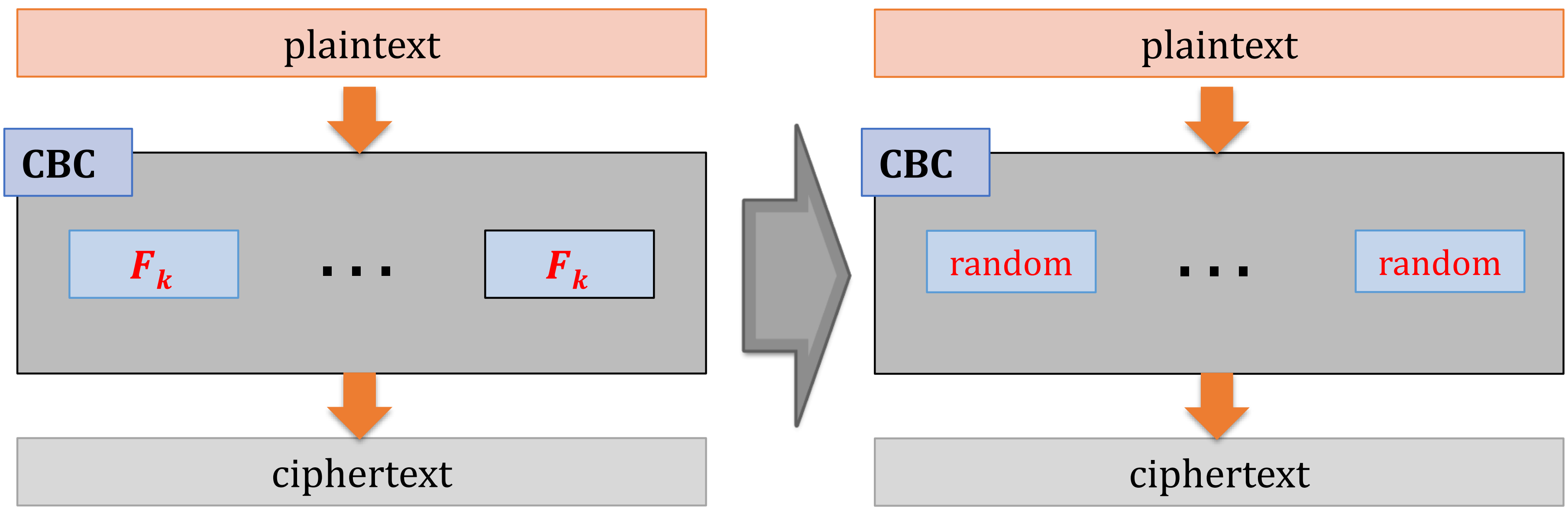
CBC mode is secure

Theorem. If F is a **PRP** then F**-CBC** is secure.

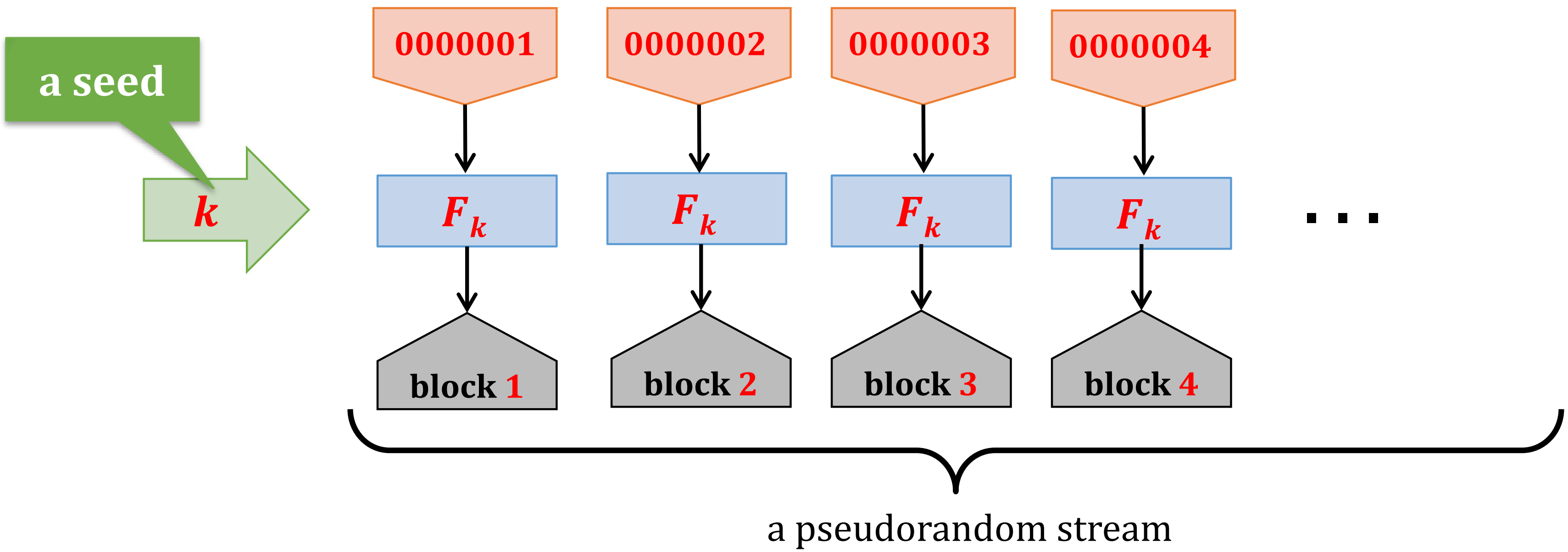
[M. Bellare, A. Desai, E. Jokipii and P. Rogaway 1997]

In the proof one can assume that F_k is a completely random function.

(If **CBC** behaves differently on a pseudorandom function, then one could construct a distiguisher.)



How to convert a pseudorandom **permutation** into a pseudorandom **generator**?

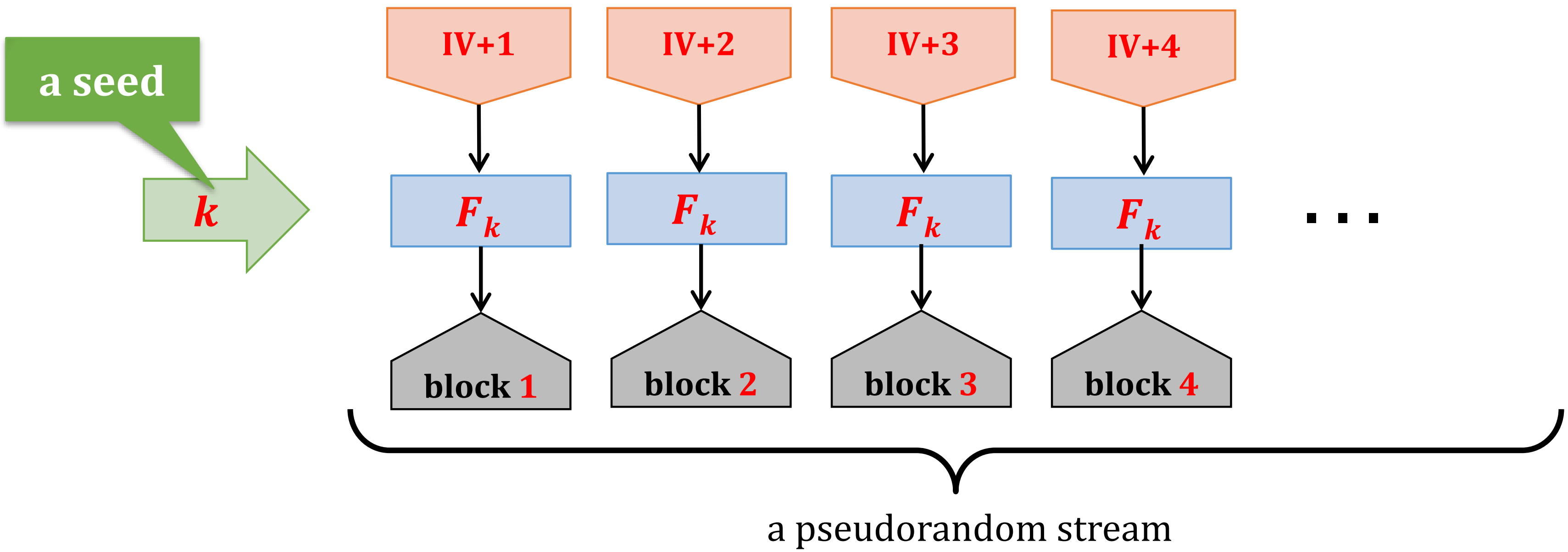


$$G(k) := F_k(1) \,||\, F_k(2) \,||\, F_k(3) \,||\, \cdots$$

Essentially, this is called a "**counter mode**" (**CTR**).

How to “randomize” this?

take some random **IV**



$$G(k, \mathrm{IV}) := F_k(\mathrm{IV}+1) \,||\, F_k(\mathrm{IV}+2) \,||\, F_k(\mathrm{IV}+3) \,||\, \cdots$$

Note:

We have to be sure that **IV + *i*** never repeats.

This is why it is bad if the block length is too small (like in **DES**).

CTR mode – properties

Error propagation?

Error in block c_i affects only c_i.

(But this mode is **<u>not</u> self-synchronizing**)

Can encryption be parallelized?

Yes

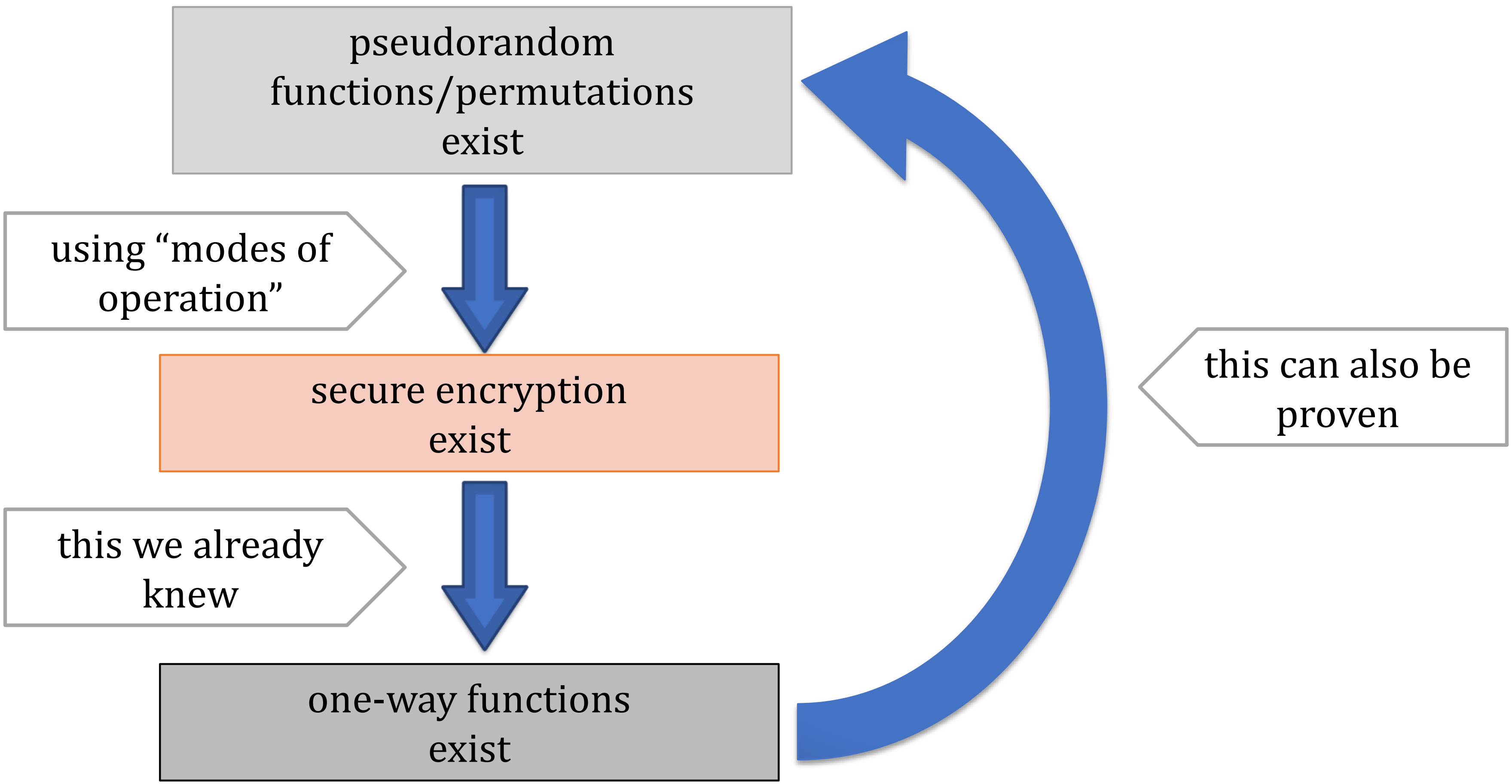
Can decryption be parallelized?

Yes

What if one bit of plaintext is changed?

Only one block needs to be recomputed

One more member of minicrypt!



There are many constructions of block ciphers that are **believed** to be secure

Why do we believe it?

- Someone important say “it is secure”.

  (But is he honest?)

- Many people tried to break it and they failed...

Plan

1. Pseudorandom functions
2. Block cipher modes of operation
3. Feistel ciphers
4. Substitution-permutation networks
5. Cascade ciphers
6. Practical considerations

DES (Digital Encryption Standard)

- **Key length**:
  - effective: **56** bits
  - formally: **64** bits (**8** bits for checking parity).
- **Block length**: **64** bits

# of DES

- First version designed by **IBM** in **1973-74**, based on a **Lucifer** cipher (by **Horst Feistel**).

- **National Security Agency (NSA)** played some role in the design of **DES**.

- Made public in **1975**.

- Approved as a **US federal standard** in November **1976**.

Criticism of DES

- The key is to short (only **56** bits).
- Unclear role of **NSA** in the design
  - hidden **backdoor**?
  - $\mathbf{2^{56}}$: feasible for **NSA**, infeasible for the others (in the **1970s**)?

Security of DES

- The main weakness is the **short key** (**brute-force** attacks are possible).
- Also the **block length is too small**.

Apart from this – **a very secure design**:

after **4 decades** still the most practical attack is **brute-force**!

The only attacks so far:
- **differential cryptanalysis**
- **linear cryptanalysis**

are rather theoretical

The role of NSA

The **United States Senate Select Committee on Intelligence** (**1978**):

"In the development of **DES**, **NSA** convinced **IBM** that a reduced key size was sufficient; indirectly assisted in the development of the **S-box** structures; and certified that the final **DES** algorithm was, to the best of their knowledge, free from any statistical or mathematical weakness."

"**NSA** did not tamper with the design of the algorithm in any way. **IBM** invented and designed the algorithm, made all pertinent decisions regarding it, and concurred that the agreed upon key size was **more than adequate** for all commercial applications for which the **DES** was intended."

Brute-force attacks on DES

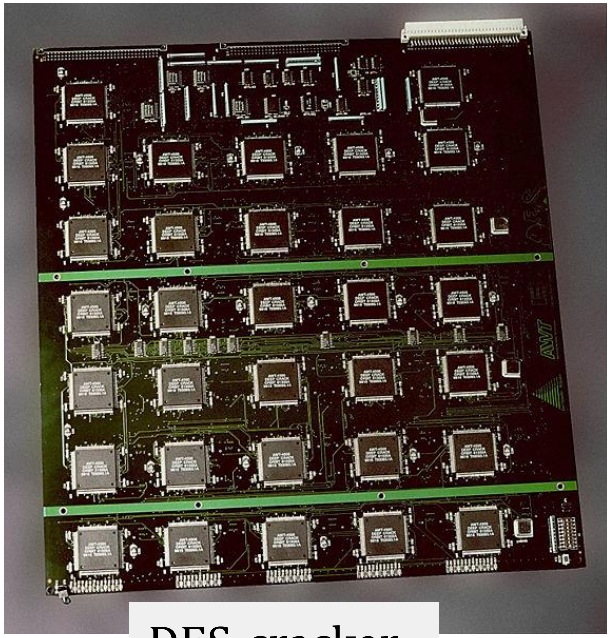
- **1977**
  Diffie and **Hellman** proposed a machine costing **20 million $** breaking DES in **1 day**.

- **1993**
  Wiener proposed a machine costing **1 million $** breaking DES in **7 hours**.

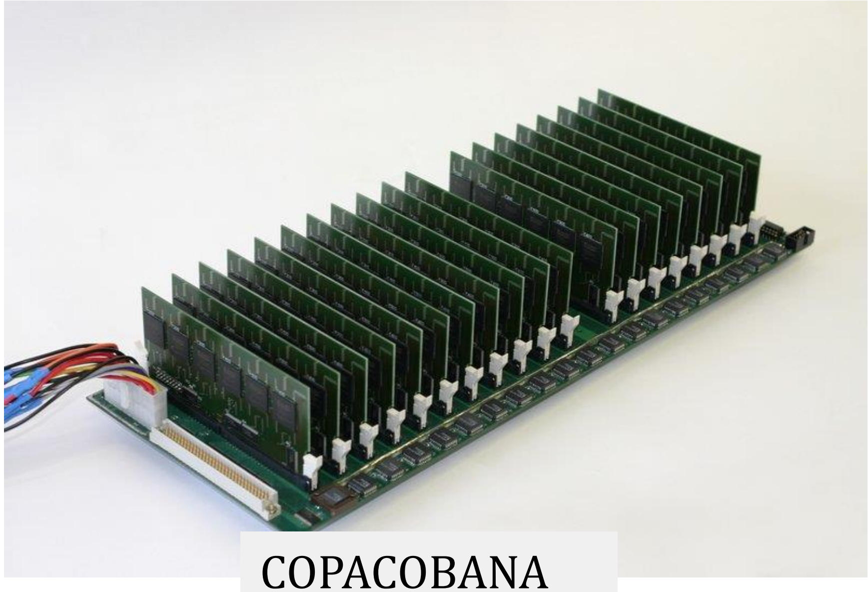
- **1997**
  DESCHALL Project broke a "**DES Challenge**" (published by **RSA**) in **96 days** using idle cycles of thousands of computers across the Internet.

- **1998**
  a **DES-cracker** was built by the **Electronic Frontier Foundation (EFF),** at the cost of approximately **250,000$**

- **COPACOBANA** (the Cost-Optimized Parallel COde Breaker) breaks **DES** in **1 week** and costs **10,000$**



DES-cracker



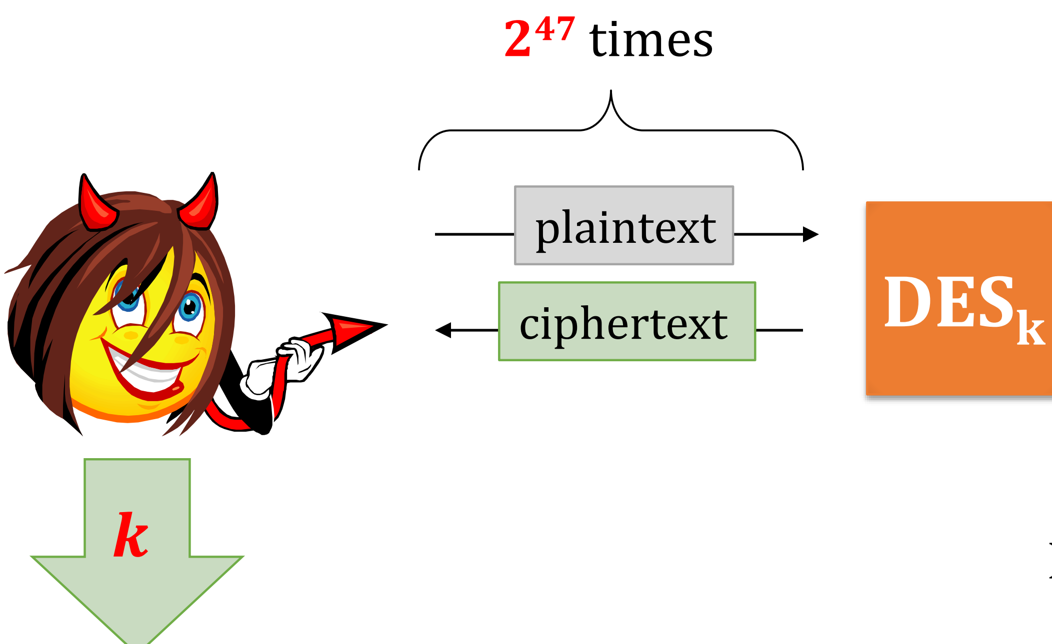
COPACOBANA

Theoretical attacks on DES – differential cryptoanalysis

Biham and Shamir (late 1980s):

differential cryptoanalysis

They show how to break **DES** using a **chosen-plaintext attack**.



Not very practical...

Differential cryptoanalysis – an interesting observation

A **small change** in the design of **DES** would make the **differential cryptanalysis** much more sucesfull.

Moral

NSA and **IBM** knew it!

"After discussions with NSA, it was decided that **disclosure of the design considerations would reveal the technique of differential cryptanalysis**, a powerful technique that could be used against many ciphers. This in turn **would weaken the competitive advantage the United States enjoyed over other countries** in the field of cryptography."

Don Coppersmith, IBM

see: Coppersmith, Don (May 1994). "[The Data Encryption Standard (DES) and its strength against attacks](http://www.research.ibm.com/journal/rd/383/coppersmith.pdf)" (PDF). *IBM Journal of Research and Development* **38** (3): 243. http://www.research.ibm.com/journal/rd/383/coppersmith.pdf.

Theoretical attacks on DES – linear cryptoanalysis

Matsui (early **1990s**):

linear cryptoanalysis

uses a **known-plaintext attack**

2^{43} (plaintext, ciphertext) pairs

this means:
the adversary
doesn't need to
choose the
plaintexts

Block ciphers – typical requirements

- **security**: ideally the best attack should be the **brute force key search**.
- **efficiency** when implemented on:
  - **8 bit microcontrollers** and **smart cards** with limited memory
  - **tablets**, **phones**, **palmtops**,
  - **PCs**, **workstations**, **servers**,
  - dedicated hardware (**ASICs**, **FPGAs**) – here we might require speeds up to **gigabits/second**
- **key agility** – **changing the key** can be done very efficiently

Block ciphers – more "informal" requirements

- **simplicity** – advantages:
  - **easier to implement**
  - more confidence that there is **no backdoor**
- **symmetry** (repeating patterns):
  - **smaller circuits** (in hardware)
  - **easier to program** (in software).

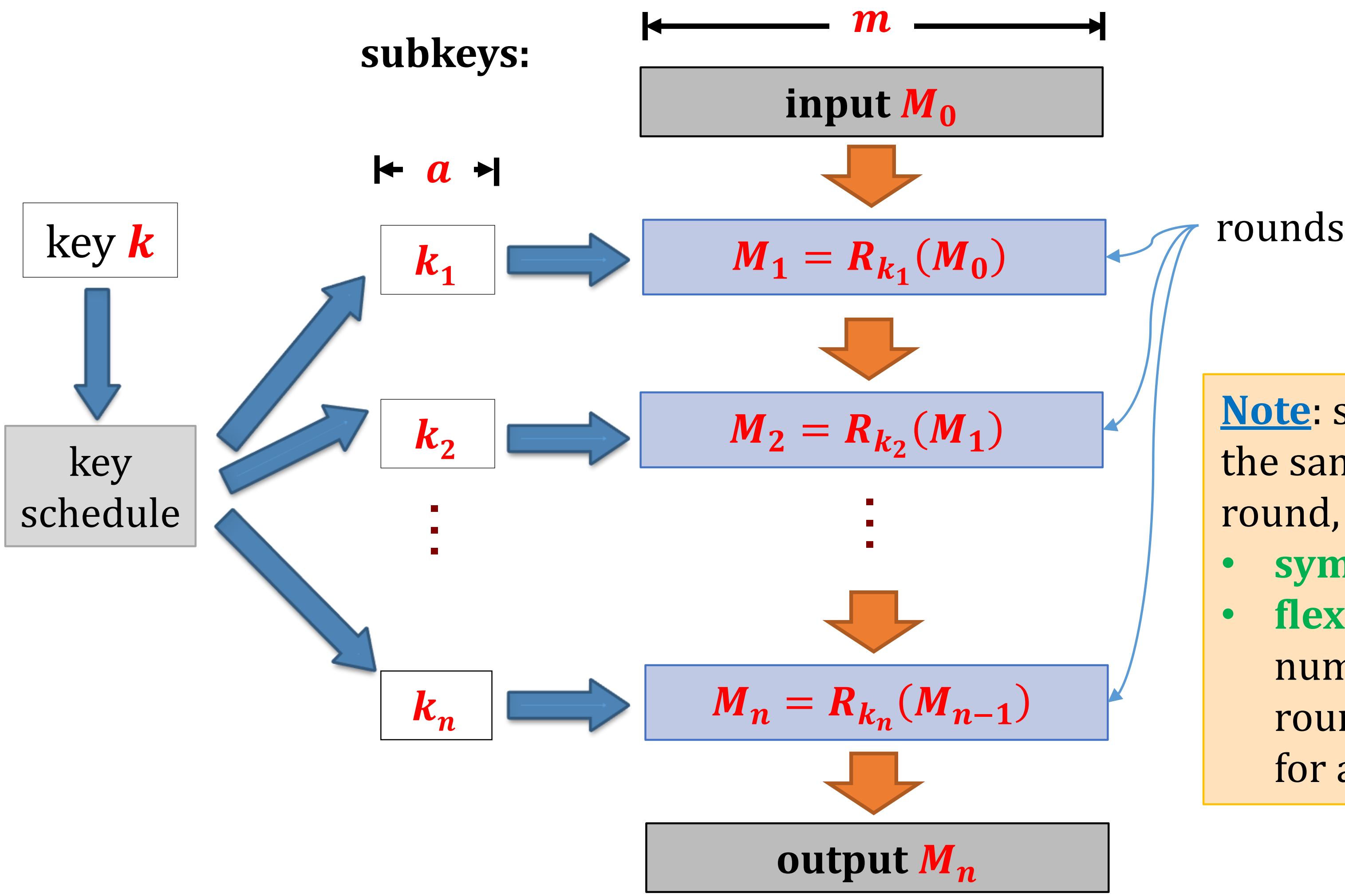
Block ciphers –advanced security requirements

- resistance to the **side-channel attacks**,
- resistance to the **key-related attacks**.

A very popular paradigm: iterated ciphers

$\boldsymbol{R}: \{0,1\}^a \times \{0,1\}^m \rightarrow \{0,1\}^m$ – a **round function**

Typically we write the first argument in a subscript.



Note: since we use the same $\boldsymbol{R}$ in each round, we get:

- **symmetry**
- **flexibility** in the number of rounds (useful for analysis)

Popular types of iterated ciphers

1. **Feistel** ciphers
2. **Substitution-permutation networks**
3. **Lai-Massey** ciphers

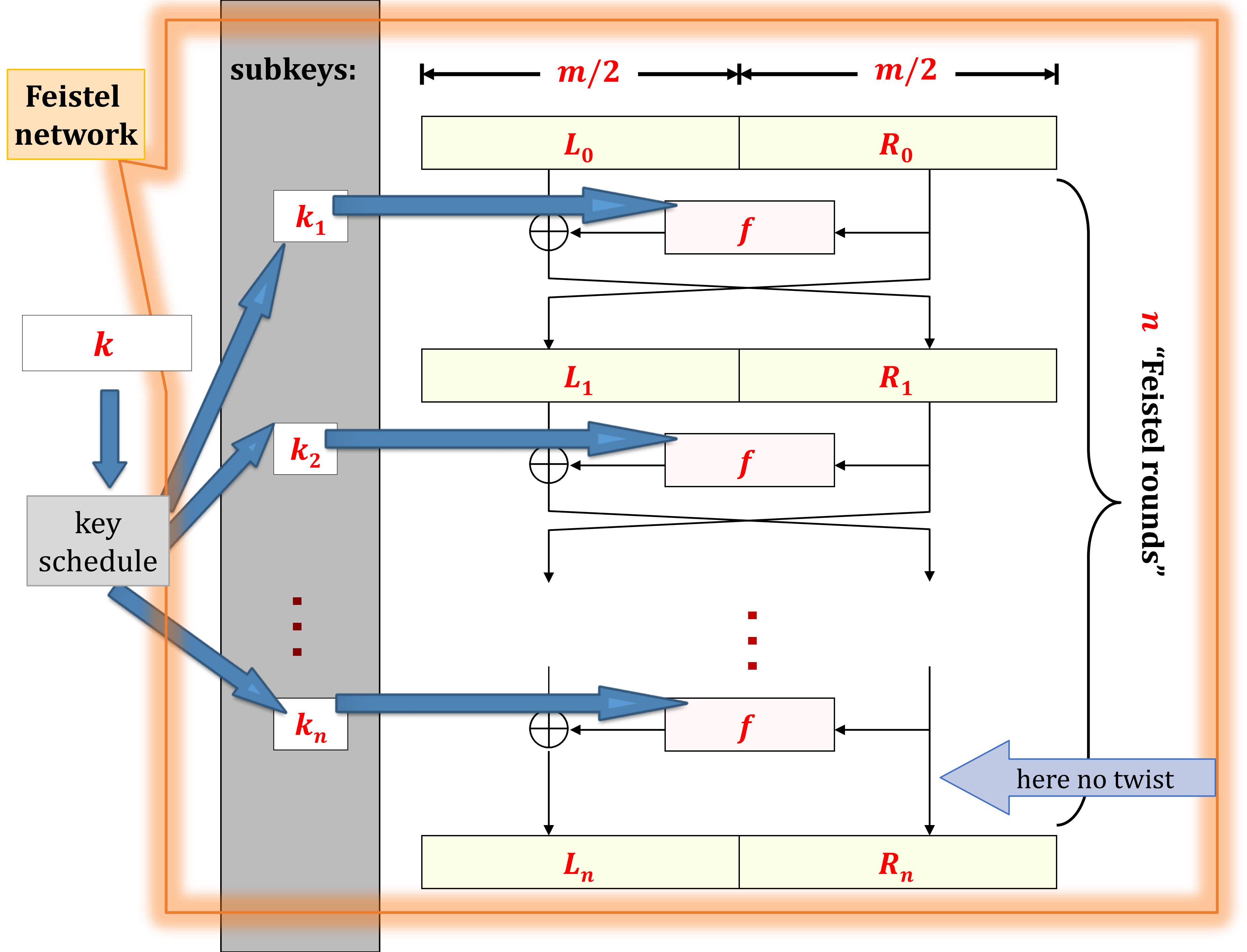
Feistel ciphers

Invented by **Horst Feistel** (1915-1990) in **1970s** while working at **IBM**.

First used in **Lucifer**. Most famous use: **Data Encryption Standard (DES)**.

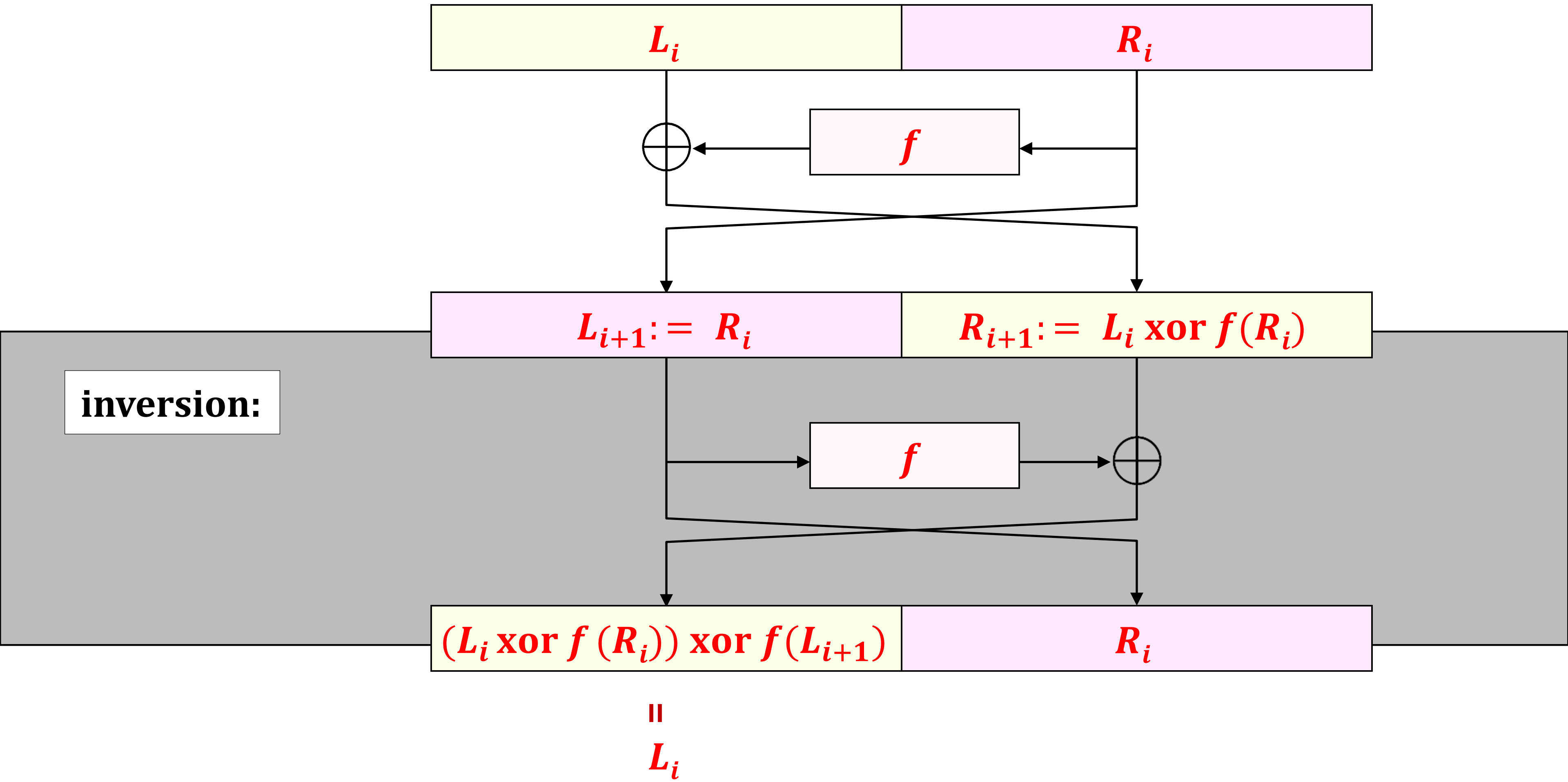
Other ciphers that use it:

Blowfish, Camellia, CAST-128, FEAL, GOST 28147-89, ICE, KASUMI, LOKI97, MARS, MAGENTA, MISTY1, RC5, Simon, TEA, Twofish, XTEA,...


Feistel network
subkeys:
m/2
m/2
k
key schedule
k1
k2
kn
L0
R0
f
L1
R1
f
f
Ln
Rn
n "Feistel rounds"
here no twist

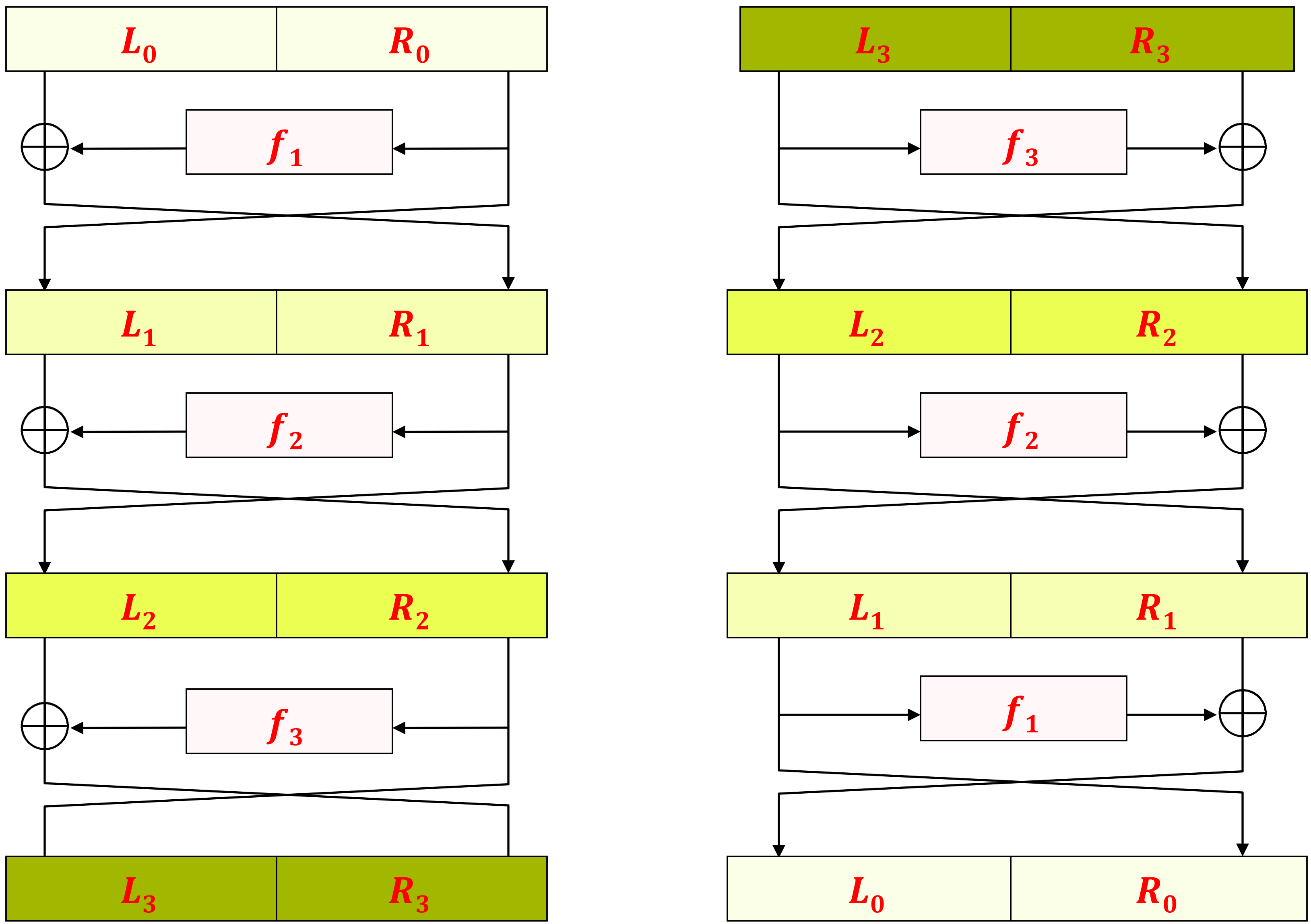
A nice propery of Feistel rounds

Even if f is **not** easily invertible, each round **can be easily inverted**!

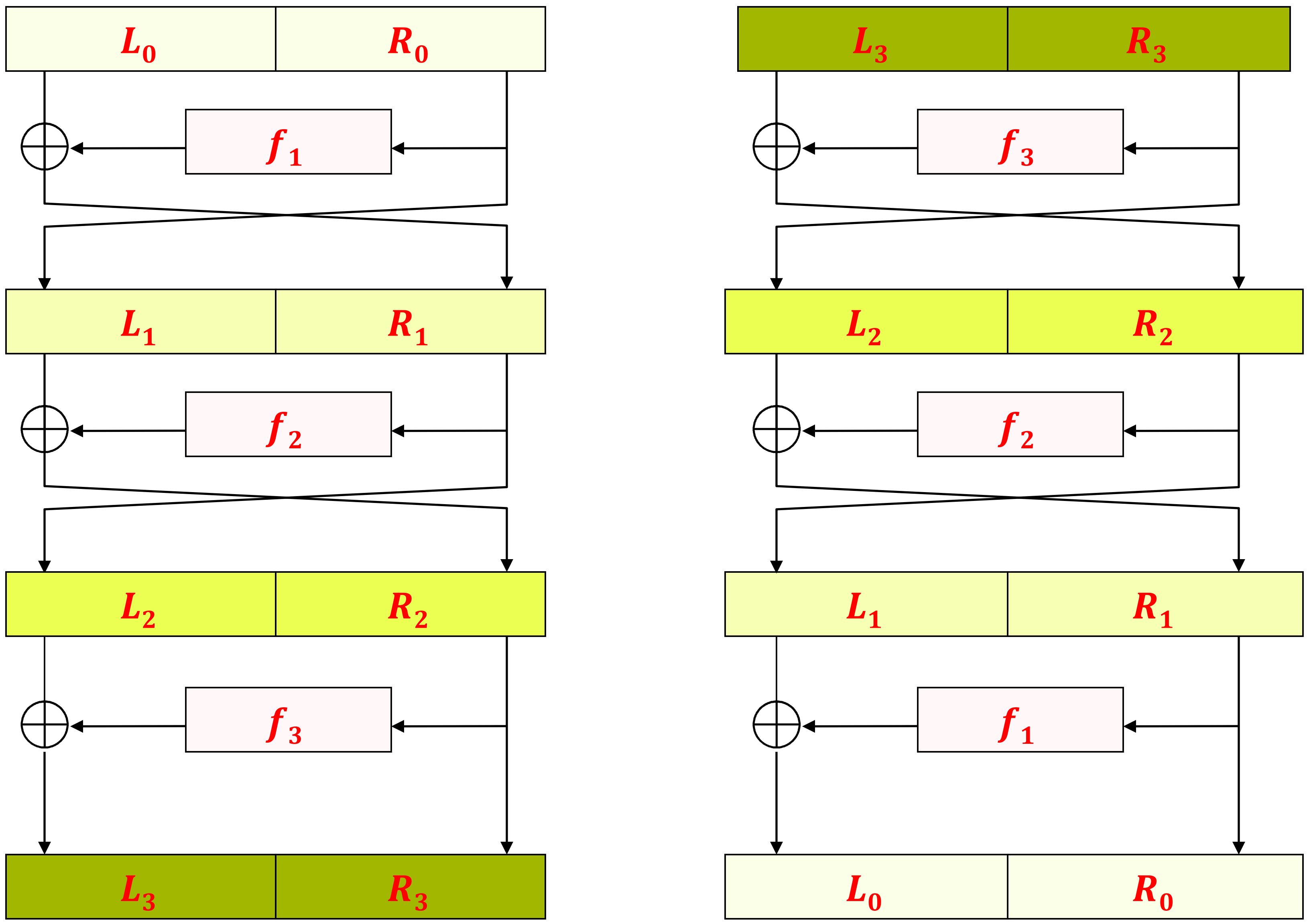


Hence: the Feistel network can be "inverted"!

Example: 3 round Feistel network

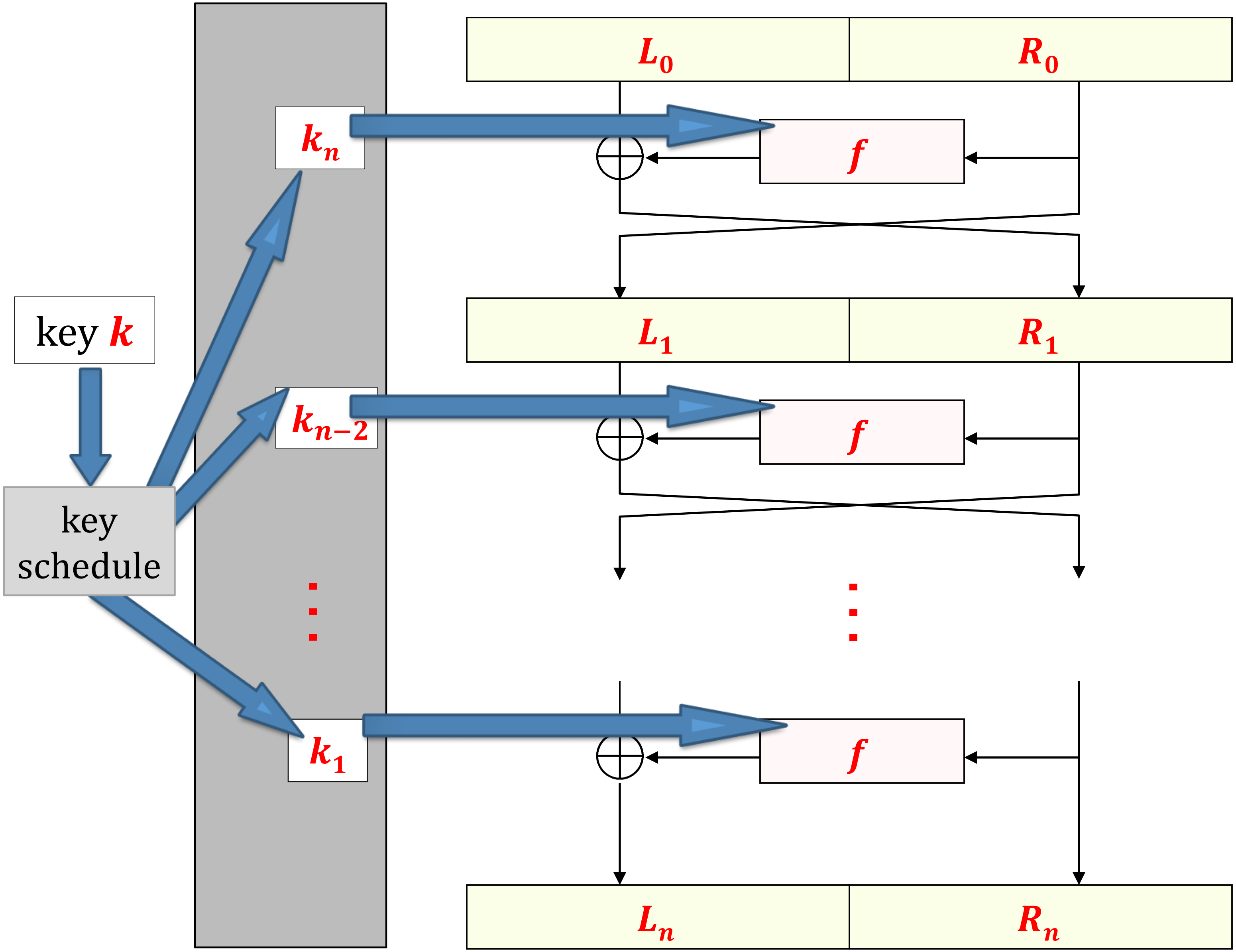


Without a “twist” in the last round:



How to decrypt?

Reverse the key schedule (**note**: **symmetry**)!



Feistel networks are also studied by the theoreticians

Suppose ***f*** is a pseudorandom **function**, and we use it to construct a Feistel network.

Then:

- the **3**-round Feistel network is a **pseudorandom permutation**,
- the **4**-round Feistel network is a **strong pseudorandom permutation**.

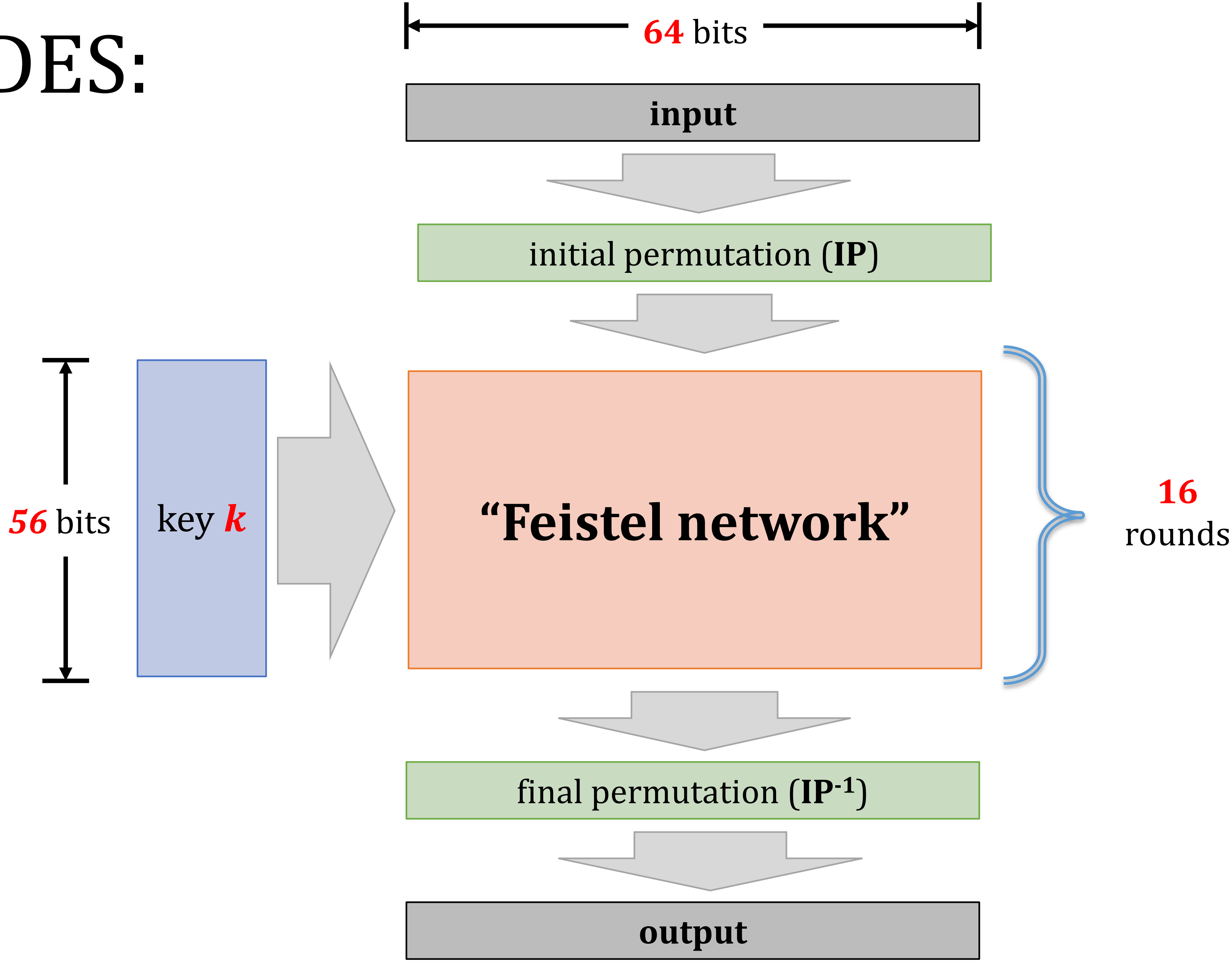
see **M. Luby and C. Rackoff. "How to Construct Pseudorandom Permutations and Pseudorandom Functions." In *SIAM J. Comput.*, vol. 17, 1988, pp. 373-386**.

How is the Feistel network used in DES?

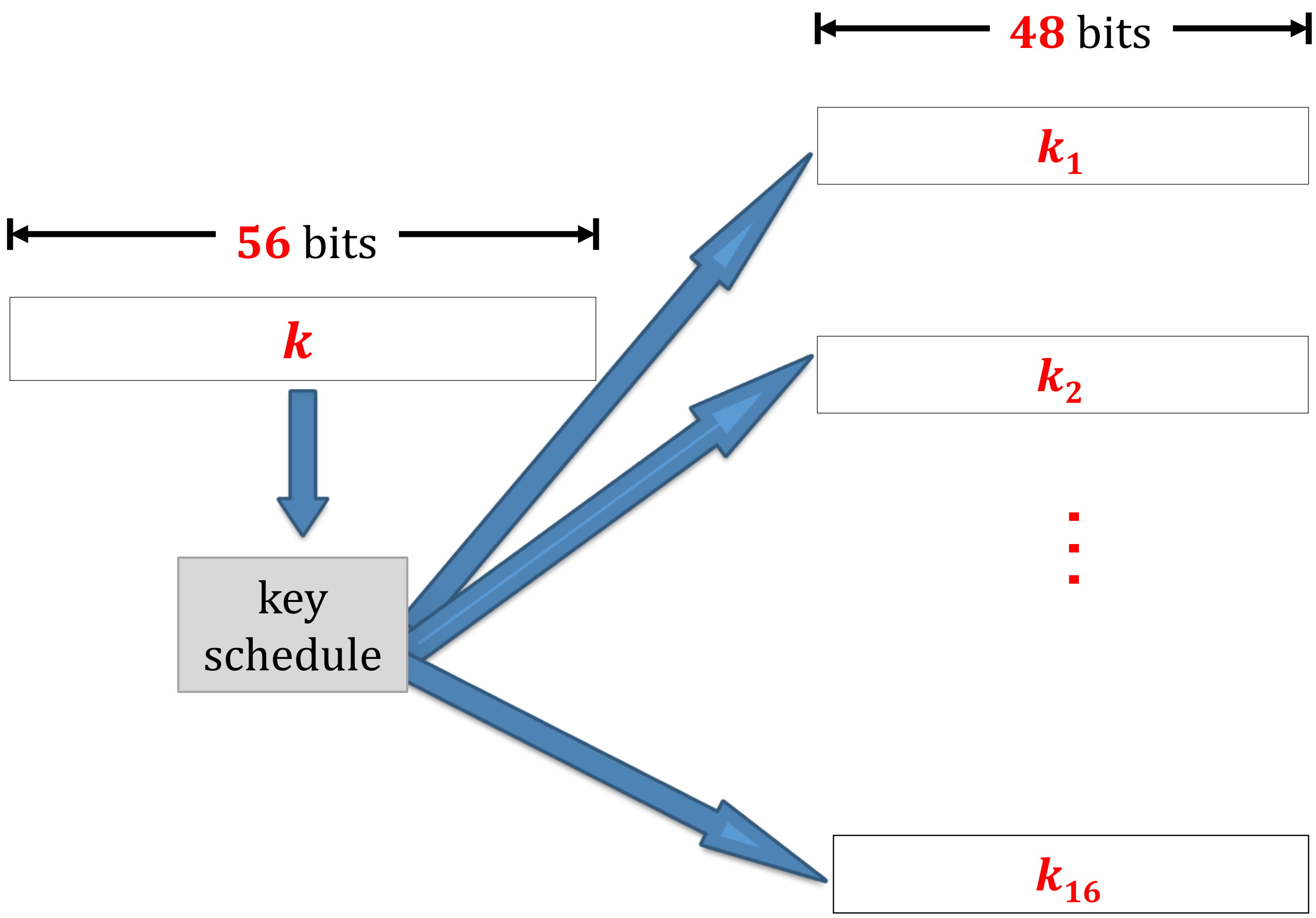
The following needs to be described:

1. The concrete parameters
2. The key schedule algorithm.
3. The functions $\boldsymbol{f}$.

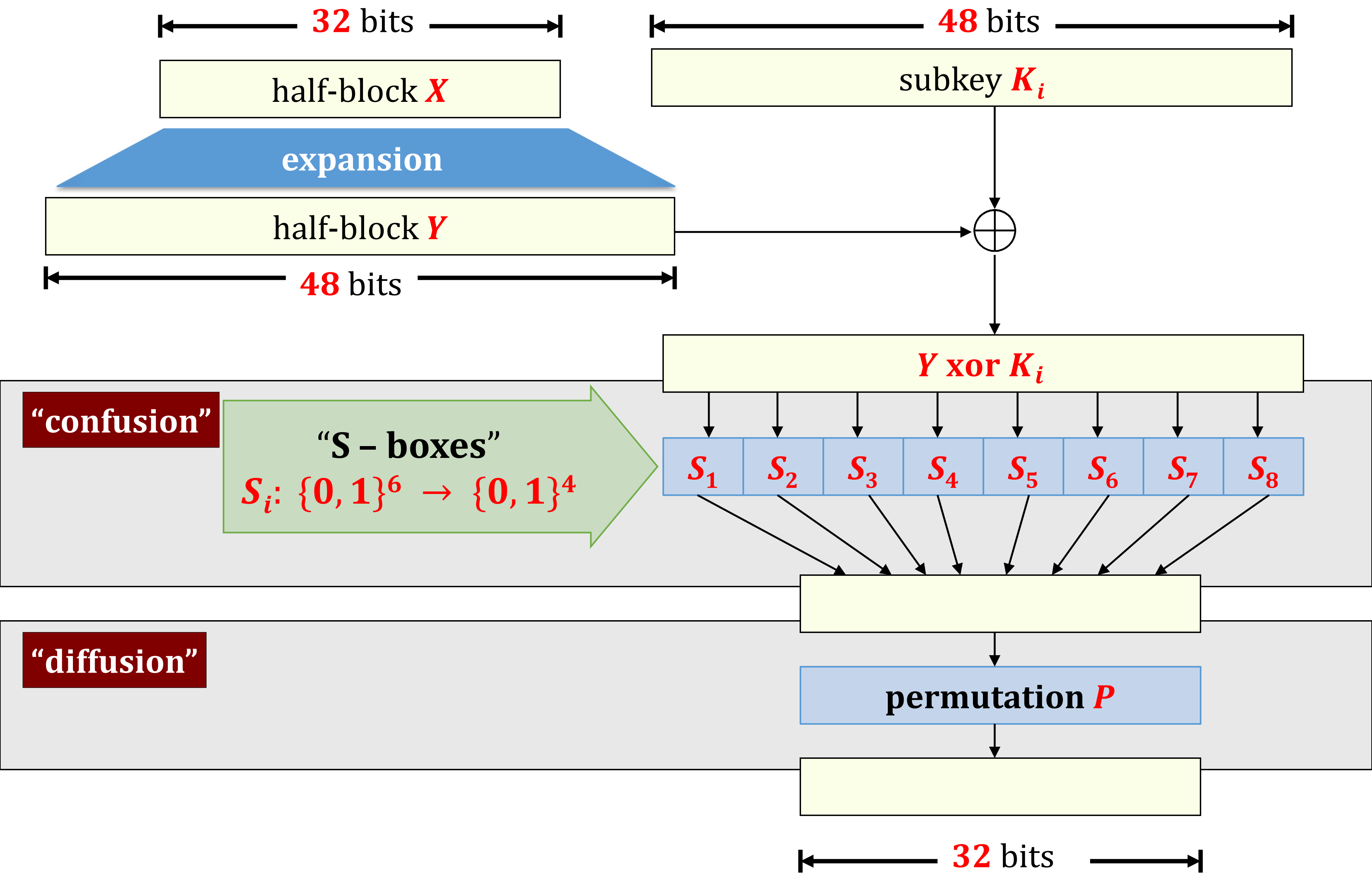
DES:


64 bits
input
initial permutation (IP)
56 bits
key k
"Feistel network"
16
rounds
final permutation (IP-1)
output

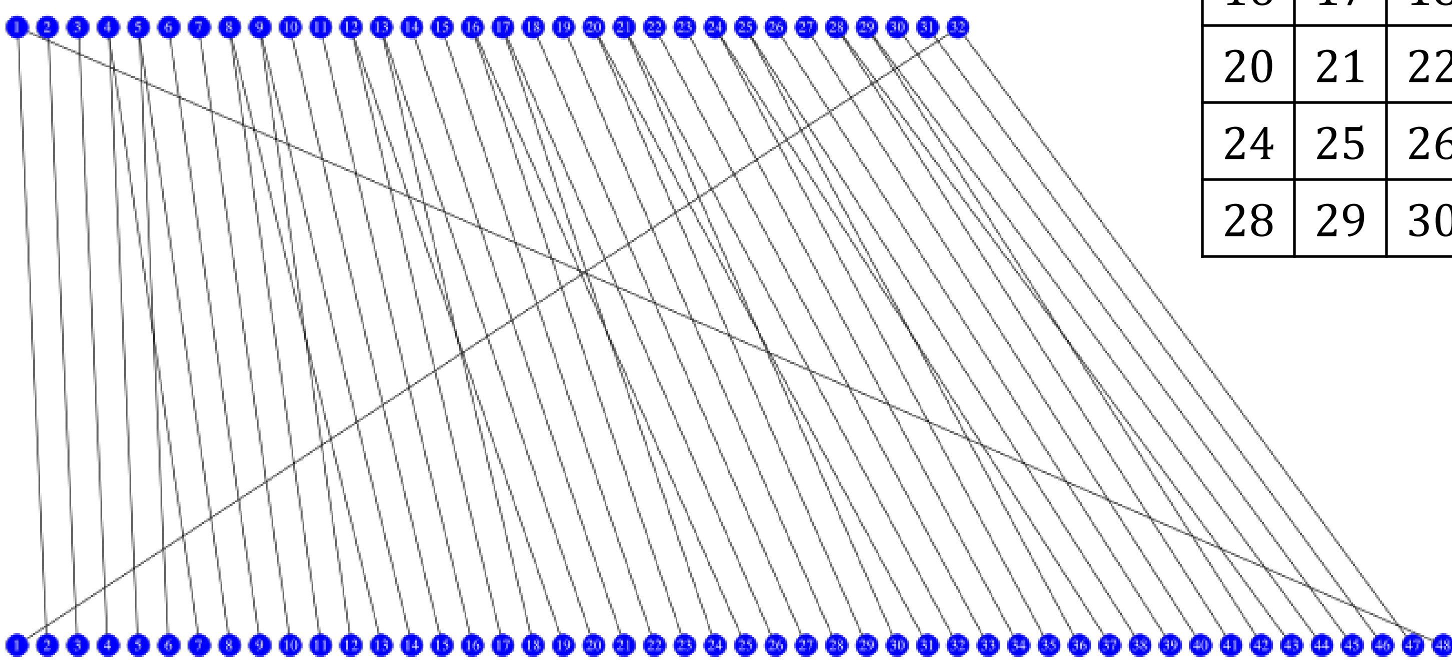
DES key schedule



each subkey k_i consists of some bits of k (we skip the details) function f:


32 bits
half-block X
48 bits
subkey K_i
expansion
half-block Y
48 bits
Y xor K_i
"confusion"
"S – boxes"
S_i: $\{0,1\}^6 \rightarrow \{0,1\}^4$
S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8
"diffusion"
permutation P
32 bits

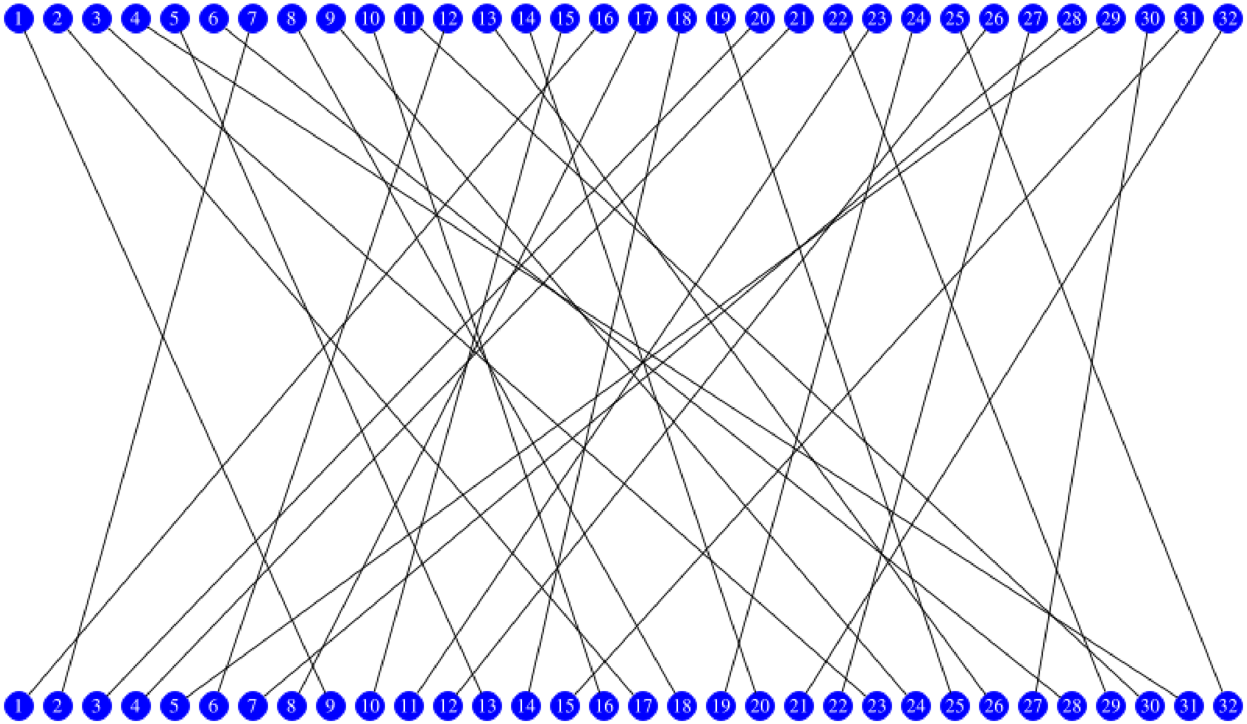
The expansion function



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

Permutation *P*

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



Properties of ***P***

The construction of ***P*** looks a bit ad-hoc.

Still, **some properties** of it are known:

- The **four bits** output from an **S-box** are distributed so that they **affect six different S-boxes** in the following round.
- If an **output bit** from **S-box** ***i*** affects one of the **two middle input bits** to **S-box** ***j*** (in the next round), then an output bit from **S-box** ***i*** cannot affect a **middle bit** of **S-box** ***i***.
- The **middle six inputs** to two neighbouring **S-boxes** (those not shared by any other **S-boxes**) are constructed from the outputs from **six different S-boxes** in the previous round.
- The **middle ten input bits to three neighbouring S-boxes**, **four bits from the two outer S-boxes** and **six from the middle S-box**, are constructed from the outputs from all **S-boxes** in the previous round.

The substitution boxes (S-boxes)

Example of an **S-box**

S_5		**Middle 4 bits of input**															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

Properties of **S-boxes** [1/2]

The design of **S-boxes** was controversial from the beginning (we discussed it before).

Responding to this the designers of **DES** published the **criteria** that they were using in this design:

- No **S-box** is a **linear** or **affine** function of the input.
- Changing **one bit** in the input to an **S-box** results in changing at **least two output bits**.
- The **S-boxes** were chosen to minimise the difference between the number of **1**'s and **0**'s when any single bit is held constant.

Properties of **S-boxes** [2/2]

- For any **S-box** $\boldsymbol{S}$, it holds that

$$\boldsymbol{S}[\boldsymbol{x}] \text{ and } \boldsymbol{S}[\boldsymbol{x} \oplus \mathbf{001100}]$$

  differ in at least two bits.

- For any **S-box** $\boldsymbol{S}$, it holds that

$$\boldsymbol{S}[\boldsymbol{x}] \neq \boldsymbol{S}[\boldsymbol{x} \oplus \mathbf{11}\boldsymbol{rs}\mathbf{00}]$$

  for any binary values $\boldsymbol{r}$ and $\boldsymbol{s}$.

- If two different **48**-bit inputs to the ensemble of eight **S-boxes** result in equal outputs, then there must be different inputs to at least three neighbouring **S-boxes**.

- For any **S-box** it holds for any nonzero **6**-bit value $\boldsymbol{\alpha}$, and for any **4**-bit value $\boldsymbol{\beta}$, that the number of solutions (for $\boldsymbol{x}$) to the equation

$$\boldsymbol{S}[\boldsymbol{x}] \oplus \boldsymbol{S}[\boldsymbol{x} \oplus \boldsymbol{\alpha}] = \boldsymbol{\beta}$$

  is at most **16**.

makes the **differential cryptanalysis** harder

DES – the conclusion

- The design of **DES** is extremally good.
- The only weaknesses: **short key** and **block**.
- Enormous impact on research in cryptography!

One practical weakness of Feistel networks

Only **half of the message** is processed at one time.

Hence: it is hard to **parallelize** the computiation.

Question: Is there any alternative construction that does not have this problem?

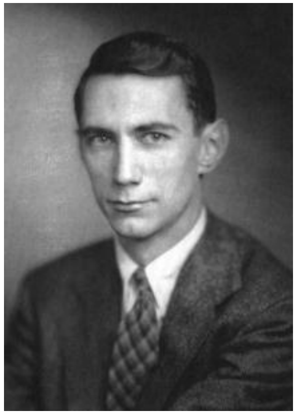
Yes! (**substitution-permutation networks**)

Plan

1. Pseudorandom functions
2. Block cipher modes of operation
3. Feistel ciphers
4. Substitution-permutation networks
5. Cascade ciphers
6. Practical considerations

Substitution-permutation networks

Based on the ideas of **Claude Shannon** (1916–2001) from **1949**.



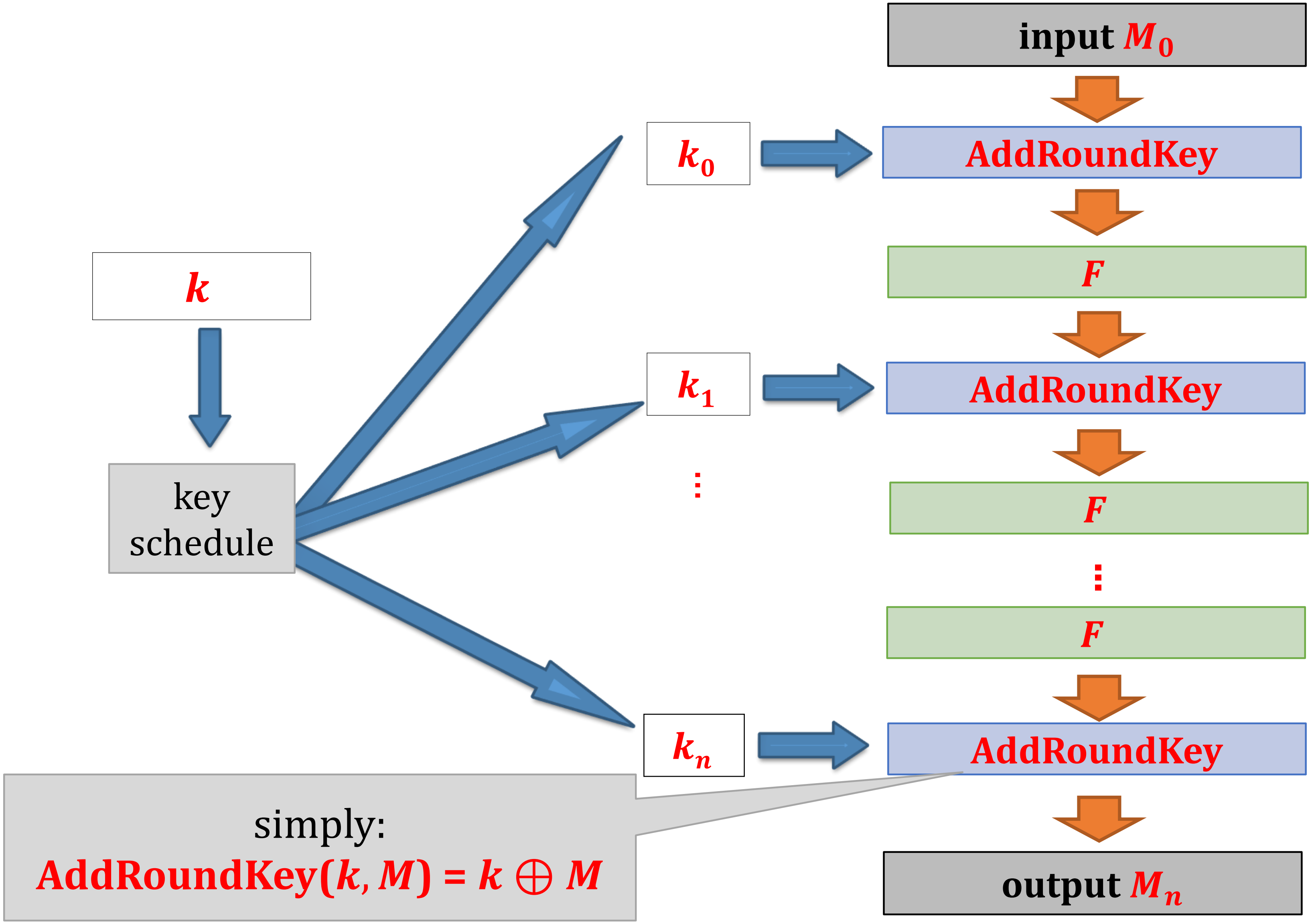
Used in **AES (Rijndael), 3-Way, SAFER, SHARK, Square**...

Advanced Encryption Standard (AES)

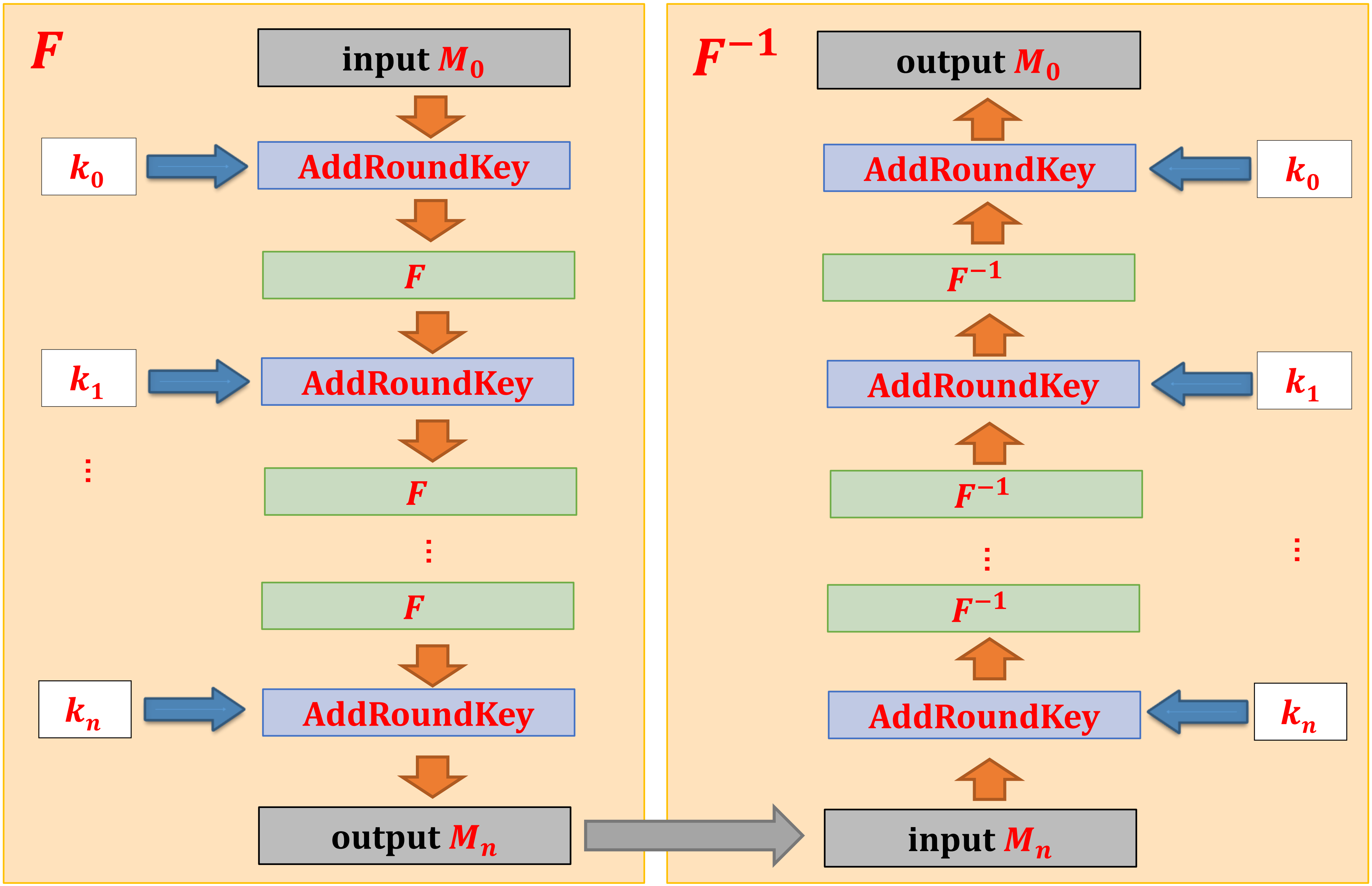
- Competition for **AES** announced in **January 1997** by the US **National Institute of Standards and Technology** (NIST)
- **15** ciphers submitted
- **5** finalists: **MARS, RC6, Rijndael, Serpent**, and **Twofish**
- **October 2, 2000**: **Rijandel** selected as the winner.
- **November 26, 2001**: **AES** becomes an official standard.

- Authors : **Vincent Rijmen**, **Joan Daemen** (from Belgium)
- **Key sizes**: **128**, **192** or **256** bit, **block size**: **128** bits

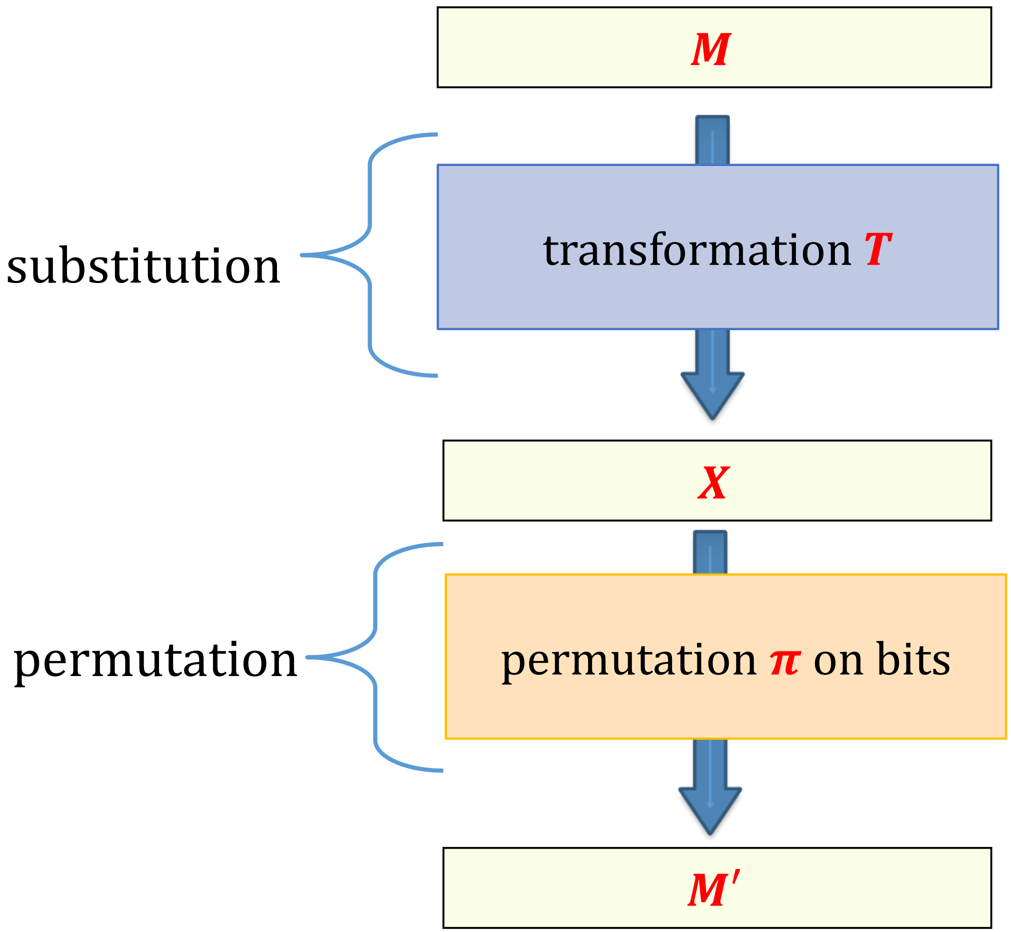
Substitution-permutation networks



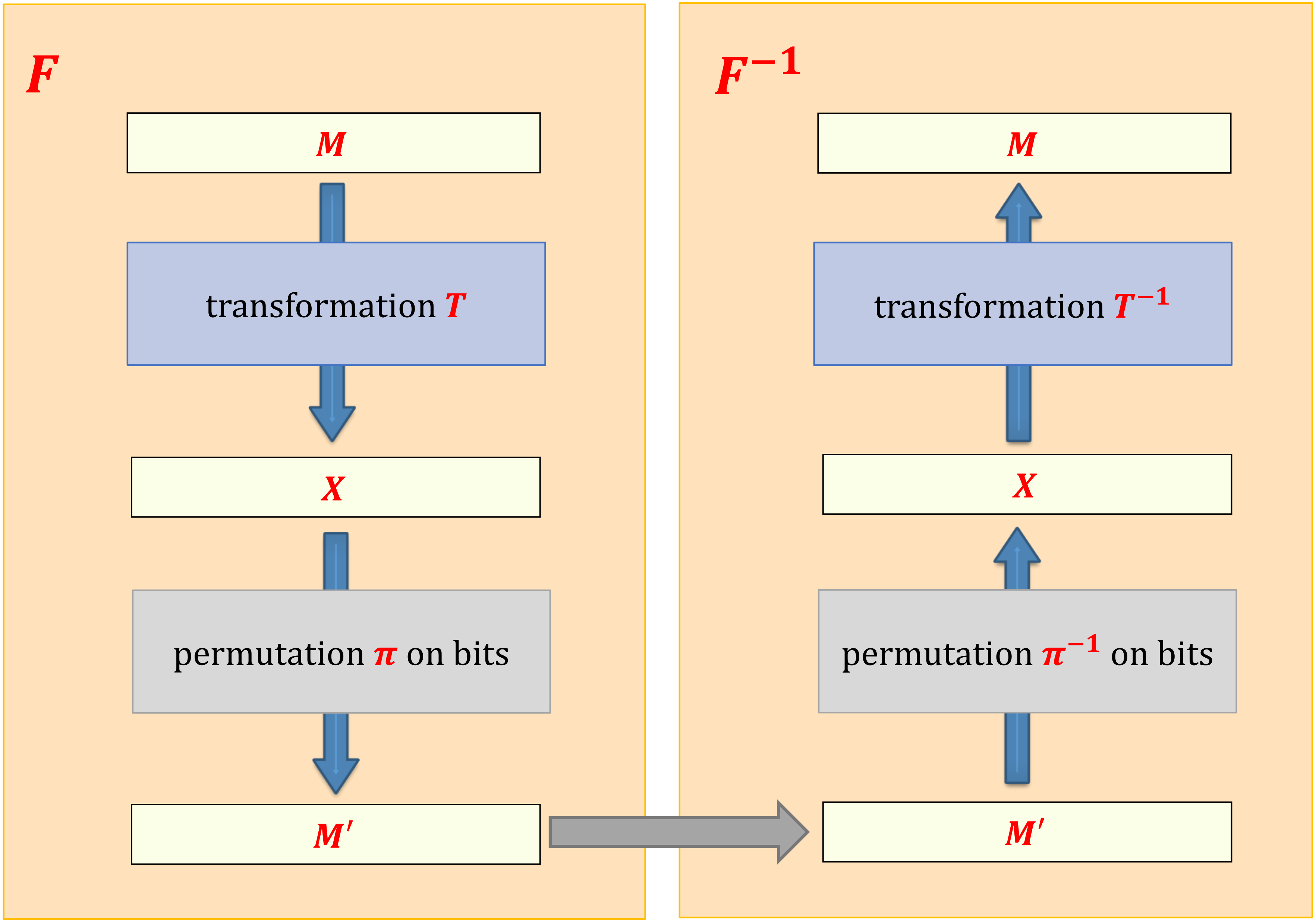
To invert: invert the order and apply F^{-1} instead of F.



A construction of F

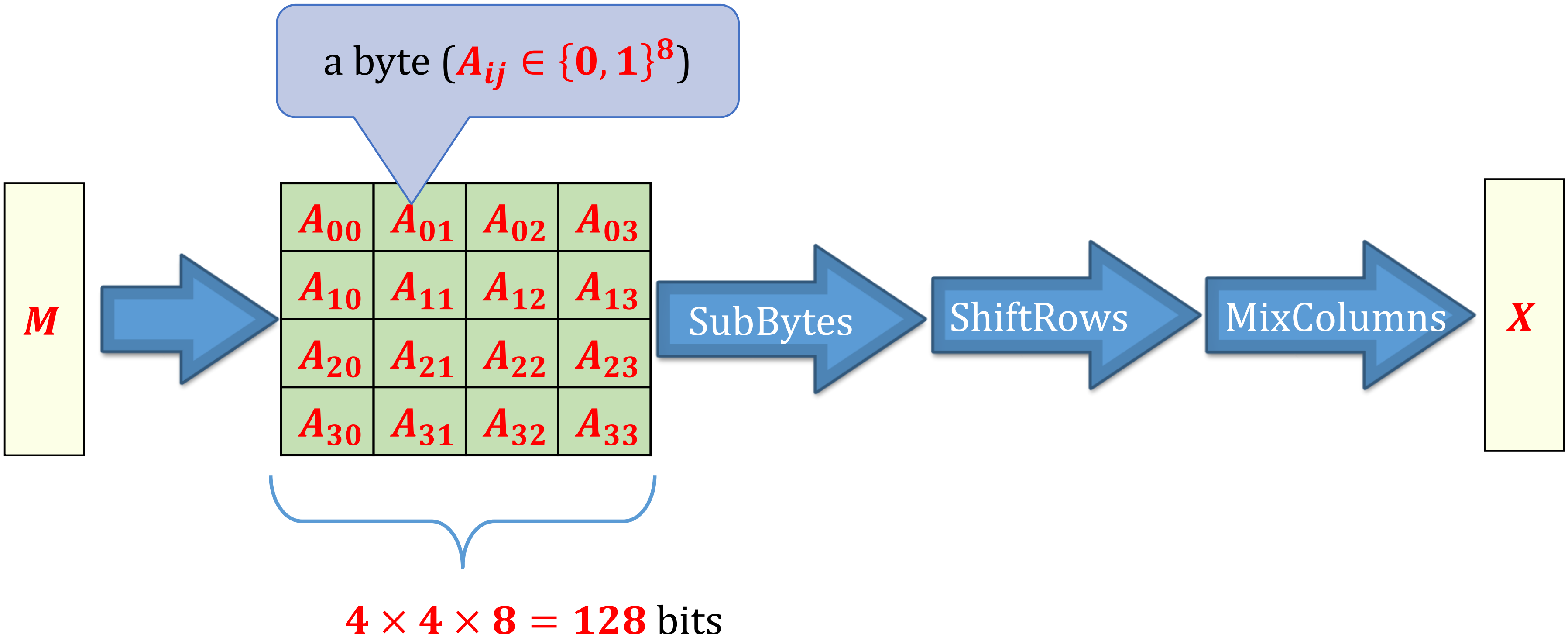


How to construct F^{-1}?



Transformation T in AES

In AES M is represented as a 4×4-matrix of bytes.



Main challenge

This transformation needs to be **invertible**...

On the other hand: it cannot be "too simple".

AES idea: use **finite field algebra**.

Groups

A **group** is a set G along with a binary operation $\circ$ such that:

- [**closure**] for all $g, h \in G$ we have $g \circ h \in G$,
- there exists an **identity** $e \in G$ such that for all $g \in G$ we have
$$e \circ g = g \circ e = g,$$
- for every $g \in G$ there exists an **inverse of**, that is an element h such that
$$g \circ h = h \circ g = e,$$
- [**associativity**] for all $g, h, k \in G$ we have
$$g \circ (h \circ k) = (g \circ h) \circ k$$
- [**commutativity**] for all $g, h \in G$ we have
$$g \circ h = h \circ g$$

if this holds, the group is called **abelian**

Additive/multiplicative notation

Convention:

[**additive notation**]
If the groups operation is denoted with $+$, then:
the inverse of g is denoted with $-g$,
the neutral element is denoted with $\mathbf{0}$,
$g + \cdots + g$ (n times) is denoted with ng.

[**multiplicative notation**]
If the groups operation is denoted "$\times$" or "$\cdot$", then:
sometimes we write gh instead of $g \cdot h$,
the inverse of g is denoted g^{-1} or $1/g$.
the neutral element is denoted with $\mathbf{1}$,
$g \cdot \cdots \cdot g$ (n times) is denoted with g^n
$\left(g^{-1}\right)^n$ is denoted with g^{-1}.

Fields

$(\boldsymbol{F}, +, \times)$ is a **field** if

- $(\boldsymbol{F}, +)$ is an **additive group** with **neutral element** $\mathbf{0}$
- $(\boldsymbol{F} \setminus \{\mathbf{0}\}, \times)$ is a **multiplicative group**
- **Distributivity of multiplication over addition**:
  for all $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \boldsymbol{F}$, we have
  $$\boldsymbol{a} \times (\boldsymbol{b} + \boldsymbol{c}) = \boldsymbol{a} \times \boldsymbol{b} + \boldsymbol{a} \times \boldsymbol{c}$$

How to define a “field over bytes”?

A very simple additive group over $\{\mathbf{0},\mathbf{1}\}^{\boldsymbol{n}}$:

$$(\{\mathbf{0},\mathbf{1}\}^{\boldsymbol{n}},+)$$

where

$$(\boldsymbol{a}_1,\ldots,\boldsymbol{a}_n)+(\boldsymbol{b}_1,\ldots,\boldsymbol{b}_n)$$
$$=(\boldsymbol{a}_1\oplus\boldsymbol{b}_1,\ldots,\boldsymbol{a}_n\oplus\boldsymbol{b}_n)$$

xor

Extremely efficient to implement.

How to extend $(\{0,1\}^n, +)$ to a field?

"**Galois fields**" $\mathbf{GF}(2^n)$:

Represent each $(a_0, \ldots, a_{n-1}) \in \{0,1\}^n$ as a polynomial A over Z_2 of degree $n-1$.

$$A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

Note: if $(b_1, \ldots, b_{n-1})$ is represented as a polynomial $B(x) = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}$, then

$$(A+B)(x) = \sum_{i=0}^{n-1} (a_i + b_i) \cdot x^i \pmod 2$$

Observe: this is the same as the **xor** operation on bits.

How to multiply?

Suppose:

$$\boldsymbol{A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}}$$
$$\boldsymbol{B(x) = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}}.$$

Then $\boldsymbol{A \times B}$ is a polynomial of max degree $\boldsymbol{2n-2}$.

How to reduce this degree?

do not exits non-constant polynomials $\boldsymbol{q_0, q_1}$ such that $\boldsymbol{p = q_0 \cdot q_1}$

Take an **irreducible polynomial** $\boldsymbol{p}$ of degree $\boldsymbol{n}$, and compute

$$\boldsymbol{C = A \cdot B \pmod{p}}$$

Then $\boldsymbol{C}$ is a polynomial of degree $\boldsymbol{n-1}$.

Write $\boldsymbol{C(x) = c_0 + c_1 x + \cdots + c_{n-1}\, x^{n-1}}$.

Define: $\boldsymbol{(a_0, \dots, a_{n-1}) \times (b_1, \dots, b_{n-1}) = (c_1, \dots, c_{n-1})}$

Fact from algebra: this defines a field.

AES field

AES uses $\mathbf{GF(2^8)}$, where the polynomial $\boldsymbol{p}$ is defined as

$$\boldsymbol{p(x) = 1 + x + x^3 + x^4 + x^8}$$

First step of SubBytes

$$\begin{array}{|c|c|c|c|}\hline A_{00} & A_{01} & A_{02} & A_{03} \\ \hline A_{10} & A_{11} & A_{12} & A_{13} \\ \hline A_{20} & A_{21} & A_{22} & A_{23} \\ \hline A_{30} & A_{31} & A_{32} & A_{33} \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|c|}\hline A_{00}^{-1} & A_{01}^{-1} & A_{02}^{-1} & A_{03}^{-1} \\ \hline A_{10}^{-1} & A_{11}^{-1} & A_{12}^{-1} & A_{13}^{-1} \\ \hline A_{20}^{-1} & A_{21}^{-1} & A_{22}^{-1} & A_{23}^{1} \\ \hline A_{30}^{-1} & A_{31}^{-1} & A_{32}^{-1} & A_{33}^{-1} \\ \hline \end{array}$$

Invert every $\mathbf{A_{ij}}$ (in the multiplicative group of $\mathbf{GF(2^n)}$).

Convention: $\mathbf{0^{-1} = 0}$.

Another observation

We can look at Z_2^n as a **linear space**.

AES defines the following affine transformation:

$$\varphi(x_1, \ldots, x_8) :=$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Advantages:

- **SubBytes** is not an operation only in $\mathbf{GF}(2^n)$.
- The constant vector is chosen in such a way that there are no
  - **fixpoints** $\varphi(X) = X$
  - **anti-fixpoints** $\varphi(X) = \overline{X}$
- φ is invertible.

Complete SubBytes

A_{00}	A_{01}	A_{02}	A_{03}
A_{10}	A_{11}	A_{12}	A_{13}
A_{20}	A_{21}	A_{22}	A_{23}
A_{30}	A_{31}	A_{32}	A_{33}

A_{00}^{-1}	A_{01}^{-1}	A_{02}^{-1}	A_{03}^{-1}
A_{10}^{-1}	A_{11}^{-1}	A_{12}^{-1}	A_{13}^{-1}
A_{20}^{-1}	A_{21}^{-1}	A_{22}^{-1}	A_{23}^{1}
A_{30}^{-1}	A_{31}^{-1}	A_{32}^{-1}	A_{33}^{-1}

$\varphi(A_{00}^{-1})$	$\varphi(A_{01}^{-1})$	$\varphi(A_{02}^{-1})$	$\varphi(A_{03}^{-1})$
$\varphi(A_{10}^{-1})$	$\varphi(A_{11}^{-1})$	$\varphi(A_{12}^{-1})$	$\varphi(A_{13}^{-1})$
$\varphi(A_{20}^{-1})$	$\varphi(A_{21}^{-1})$	$\varphi(A_{22}^{-1})$	$\varphi(A_{23}^{-1})$
$\varphi(A_{30}^{-1})$	$\varphi(A_{31}^{-1})$	$\varphi(A_{32}^{-1})$	$\varphi(A_{33}^{-1})$

Observe that

$$A_{ij} \mapsto A_{ij}^{-1} \mapsto \varphi(A_{ij}^{-1})$$

is invertible (since $A_{ij} \mapsto A_{ij}^{-1}$ and φ are invertible)

ShiftRows

Cyclic shifts of rows:

B_{00}	B_{01}	B_{02}	B_{03}
B_{10}	B_{11}	B_{12}	B_{13}
B_{20}	B_{21}	B_{22}	B_{23}
B_{30}	B_{31}	B_{32}	B_{33}

$\Rightarrow$

B_{00}	B_{01}	B_{02}	B_{03}	
B_{11}	B_{12}	B_{13}	B_{10}	shift **1** cell left
B_{22}	B_{23}	B_{20}	B_{21}	shift **2** cells left
B_{33}	B_{30}	B_{31}	B_{32}	shift **3** cells left

Clearly: **ShiftRows** is invertible.

MixColumns

SubBytes → ShiftRows → **MixColumns**

Multiply every column by a matrix $\boldsymbol{M}$ (in $\mathbf{GF(2^8)}$):

$$\begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \qquad \begin{bmatrix} c'_{00} & c'_{01} & c'_{02} & c'_{03} \\ c'_{10} & c'_{11} & c'_{12} & c'_{13} \\ c'_{20} & c'_{21} & c'_{22} & c'_{23} \\ c'_{30} & c'_{31} & c'_{32} & c'_{33} \end{bmatrix}$$

$$\boldsymbol{M}: \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} * \begin{bmatrix} c_{02} \\ c_{12} \\ c_{22} \\ c_{32} \end{bmatrix} =: \begin{bmatrix} c'_{02} \\ c'_{12} \\ c'_{22} \\ c'_{32} \end{bmatrix}$$

Clearly $\boldsymbol{M}$ is invertible, so the whole operation also is.

AES construction – more details

<u>Concrete parameters:</u>

key size: 128, **192** or **256** bit,
block size: **128** bits

We omit the description of the key schedule.

<u>Security</u>:

best known attack: **biclique attacks [Bogdanov, Khovratovich, and Rechberger, 2013]**:

- **AES-128** complexity $\mathbf{2^{126.1}}$,
- **AES-192** complexity $\mathbf{2^{189.7}}$,
- **AES-256** complexity $\mathbf{2^{254.4}}$.

Plan

1. Pseudorandom functions
2. Block cipher modes of operation
3. Feistel ciphers
4. Substitution-permutation networks
5. Cascade ciphers
6. Practical considerations

An idea

The main problem of **DES** is the short key!

Maybe we could increase the length of the key?

But how to do it?

Idea: cascade the ciphers!

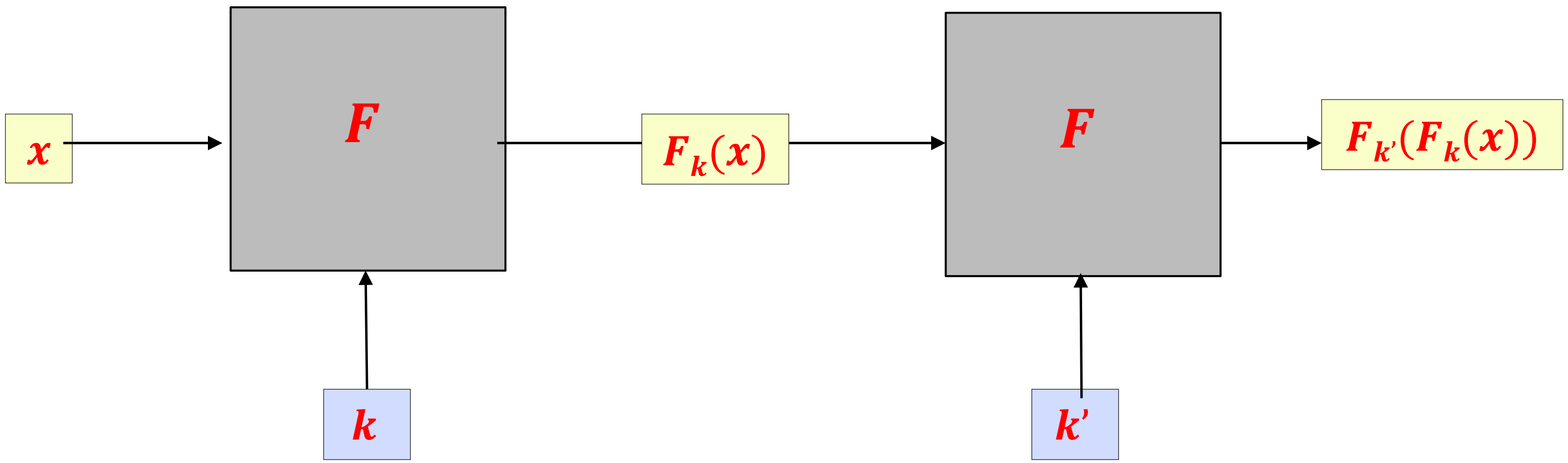
We now describe it in an abstract way (for any block cipher ***F***)

How to increase the key size?

Cascade encryption.

For example **double encryption** is defined as:

$$F'_{(k,k')}(x) := F_{k'}(F_k(x))$$



Does it work?

- Double encryption – not really...
- Triple encryption is much better!

Double encryption

$\boldsymbol{n}$ = block length = key length

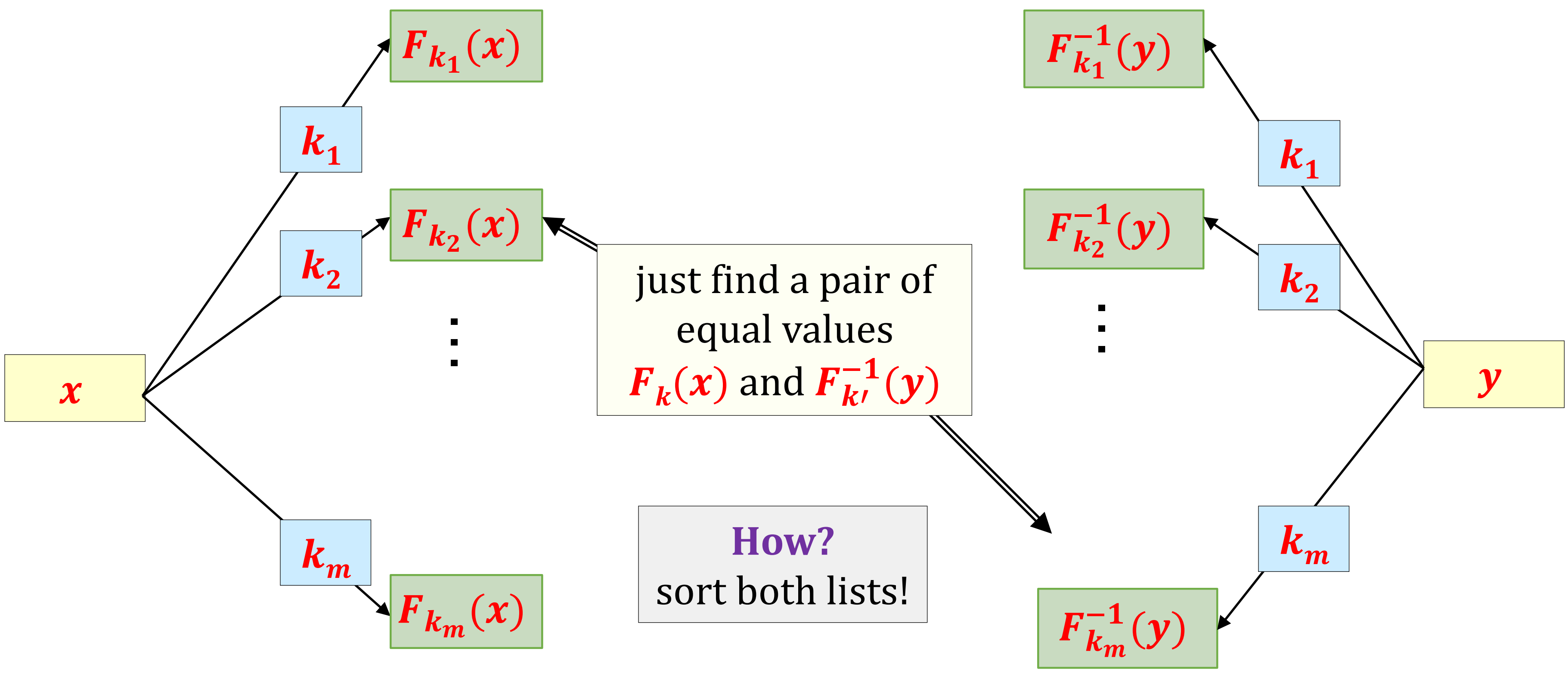
Double encryption can be broken using

- time $\boldsymbol{O(2^n)}$,
- space $\boldsymbol{O(2^n)}$,
- and **3** **(plaintext,ciphertext)** pairs.

The attack is called "**meet in the middle**".

Meet-in-the middle attack – the idea

Goal: Given (x, y) find (k, k') such that $y = F_{k'}(F_k(x))$ $m = 2^n$



Meet-in-the middle attack – the algorithm

Goal: Given (x, y) find (k, k') such that $y = F_{k'}(F_k(x))$.

Algorithm:

1. For each k compute $z = F_k(x)$ and store (z, k) in a list L.
2. For each k compute $z = F_k^{-1}(y)$ and store (z, k') in a list L'.
3. Sort L and L' by their first components.
4. Let **S** denote the list of all pairs all pairs (k, k') such that for some z we have
$$(z, k) \in L \quad \text{and} \quad (z, k') \in L'.$$
5. Output S.

Meet-in-the middle attack – analysis [1/2]

Suppose: n = block length = key length, x and y are fixed

$$P\ (\textbf{a random pair } (k, k') \textbf{ satisfies } y = F_{k'}(F_k(x))) \approx 2^{-n}$$

why?
because
$F_{k'}(F_k(x))$
can take
2^n
values

The number of all pairs (k, k') is equal to 2^{2n}. Therefore

$$E(|S|) \approx 2^{2n} \cdot 2^{-n} = 2^n.$$

So we have around 2^n "candidates" for the correct pair (k, k').

How to eliminate the "false positives"?

For each "positive" check it against another pair (x', y').

Meet-in-the middle attack – analysis [2/2]

The probability that (k, k') is a false positive for (x, y) and for (x', y') is around

$$2^{-n} \cdot 2^{-n} = 2^{-2n}.$$

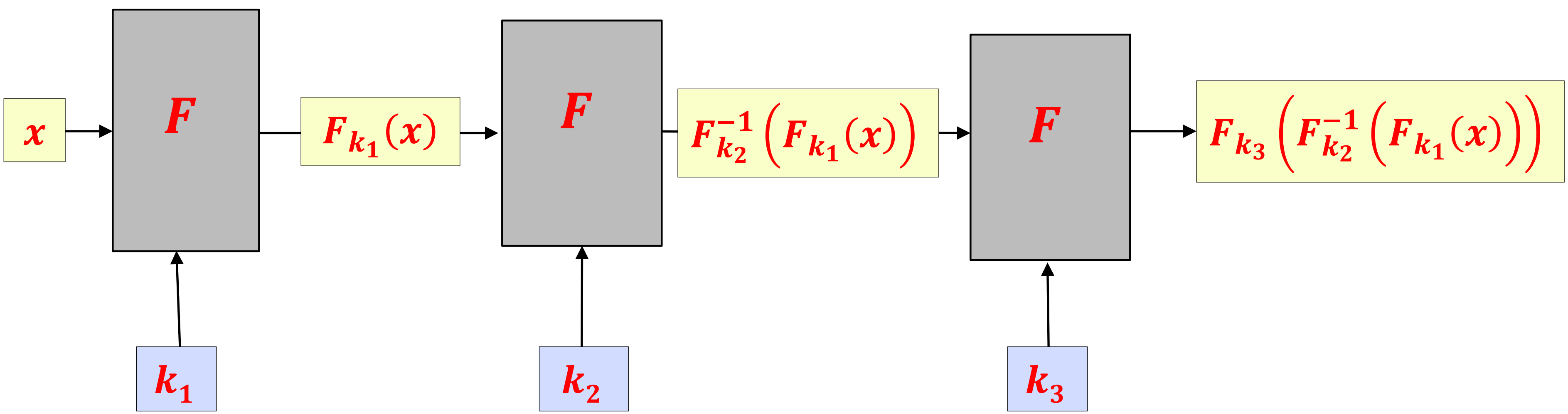
Hence, the expected number of "false positives" is around

$$2^{2n} \cdot 2^{-2n} = 1$$

An additional pair (x'', y'') allows to eliminate the false positive.

A much better idea: triple encryption

$$F_{(k_1,k_2,k_3)}(x) := F_{k_3}\left(F_{k_2}^{-1}\left(F_{k_1}(x)\right)\right)$$



Sometimes $k_1 = k_3$.

Triple DES (3DES) is a standard cipher.

Disadvantages:

- rather slow,
- small block size.

Plan

1. Pseudorandom functions
2. Block cipher modes of operation
3. Feistel ciphers
4. Substitution-permutation networks
5. Cascade ciphers
6. Practical considerations

Benchmarks

	Algorithm	MiB/Second	Cycles Per Byte
stream	Salsa20	643	2.7
	Sosemanuk	727	2.4
	AES/CTR (128-bit key)	139	12.6
	AES/CTR (192-bit key)	113	15.4
	AES/CTR (256-bit key)	96	18.2
	AES/CBC (128-bit key)	109	16
block	AES/CBC (192-bit key)	92	18.9
	AES/CBC (256-bit key)	80	21.7
	DES/CTR	32	54.7
	DES-EDE3/CTR	13	134.5

Source: www.cryptopp.com/benchmarks.html
All algorithms coded in C++, compiled with Microsoft Visual C++ 2005 SP1 (whole program optimization, optimize for speed), and ran on an Intel Core 2 1.83 GHz processor under Windows Vista in 32-bit mode.

Hardware implementations of AES

(taken from J Daemen, V Rijmen The design of Rijndael, 2001):

Example of a hardware record:

ASIC. H. Kuo and I. Verbauwhede report a throughput of 6.1 Gbit/s, using 0.18 μm standard cell technology [55] for an implementation without pipelining. Their design uses 19 000 gates. B. Weeks et al. report a throughput of 5 Gbit/s [91] for a fully pipelined version. They use a 0.5 μm standard cell library that is not available outside NSA.

Stream ciphers vs. block ciphers

- Stream ciphers are a **bit more efficient**.
- But they appear to be "**less secure**".
- It is easier to misuse them (use the same stream twice).
- If you encrypt a stream of data you can always use a block cipher in a **CTR** mode.
- Probably at the moment **block ciphers are a better choice** for most of the applications.