

Chapter 2

Symmetric Encryption

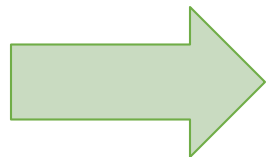
Stefan Dziembowski

www.crypto.edu.pl/Dziembowski

University of Warsaw



Plan



1. Computational definitions of security
2. If semantically-secure encryption exists, then **P \neq NP**
3. A proof that “the PRGs imply secure encryption”
4. Theoretical constructions of PRGs
5. Stream ciphers
6. Pseudorandom functions and permutations
7. Block ciphers
8. Practical considerations

How to change the security definition?

we will require that m_0, m_1 are chosen by a **poly-time adversary**

An encryption scheme is **perfectly secret** if for every $m_0, m_1 \in \mathcal{M}$
 $\text{Enc}(K, m_0)$ and $\text{Enc}(K, m_1)$ are identically distributed

we will require that no **poly-time adversary** can distinguish
 $\text{Enc}(K, m_0)$ from $\text{Enc}(K, m_1)$

A game

(Enc,Dec) – an encryption scheme



adversary
(polynomial-time probabilistic Turing machine)

security parameter
 1^n



oracle

chooses m_0, m_1 such that
 $|m_0| = |m_1|$

m_0, m_1

1. selects k randomly from $\{0,1\}^n$
2. chooses a random $b = 0,1$
3. calculates
 $c := \text{Enc}(k, m_b)$

has to guess b

c

Alternative name: **has indistinguishable encryptions in the presence of an eavesdropper**

Security definition:

We say that **(Enc,Dec)** is **semantically-secure** if any **polynomial time** adversary guesses b correctly with probability at most $\frac{1}{2} + \epsilon(n)$, where ϵ is negligible.

Testing the definition

Suppose the adversary can compute k from $\text{Enc}(k,m)$.
Can he win the game?

YES!

Suppose the adversary can compute **some bit of** m
from $\text{Enc}(k,m)$. Can he win the game?

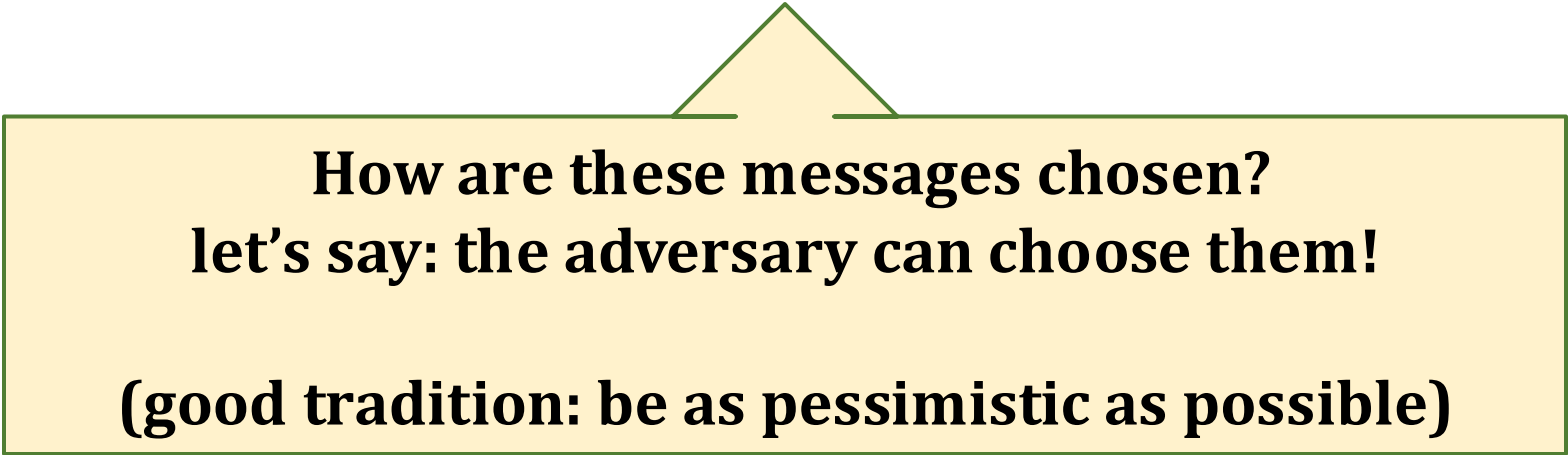
YES!

Multiple messages

In real-life applications we need to encrypt **multiple messages with one key**.

The adversary may learn something about the key by looking at

ciphertexts c_1, \dots, c_t of
some messages m_1, \dots, m_t .

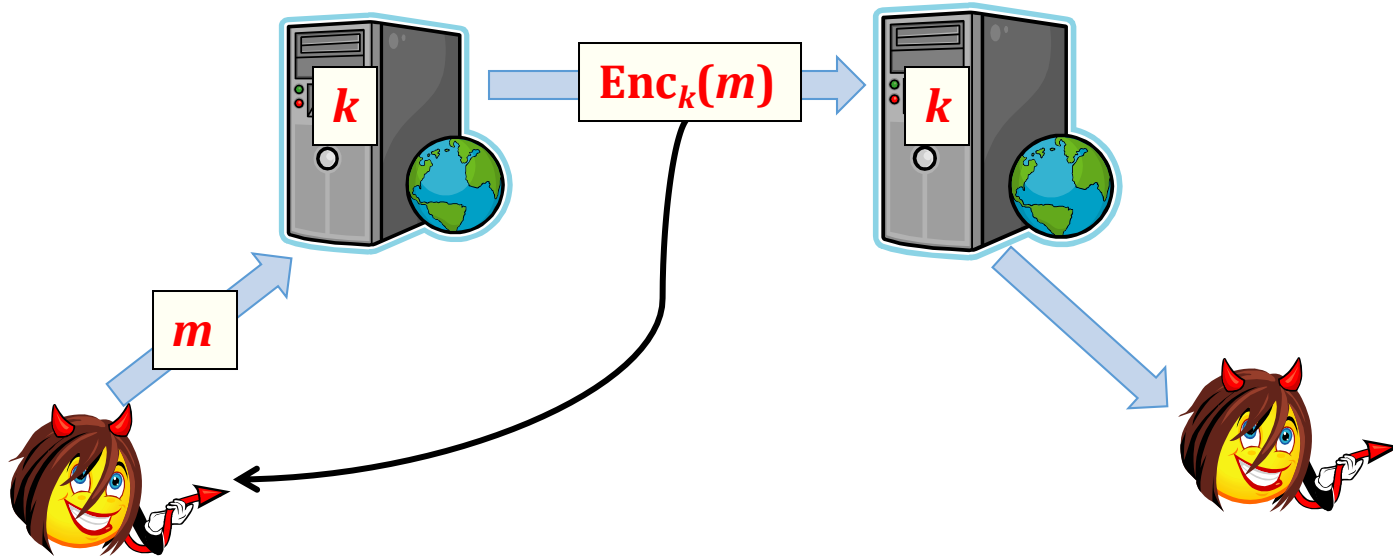


How are these messages chosen?
let's say: the adversary can choose them!

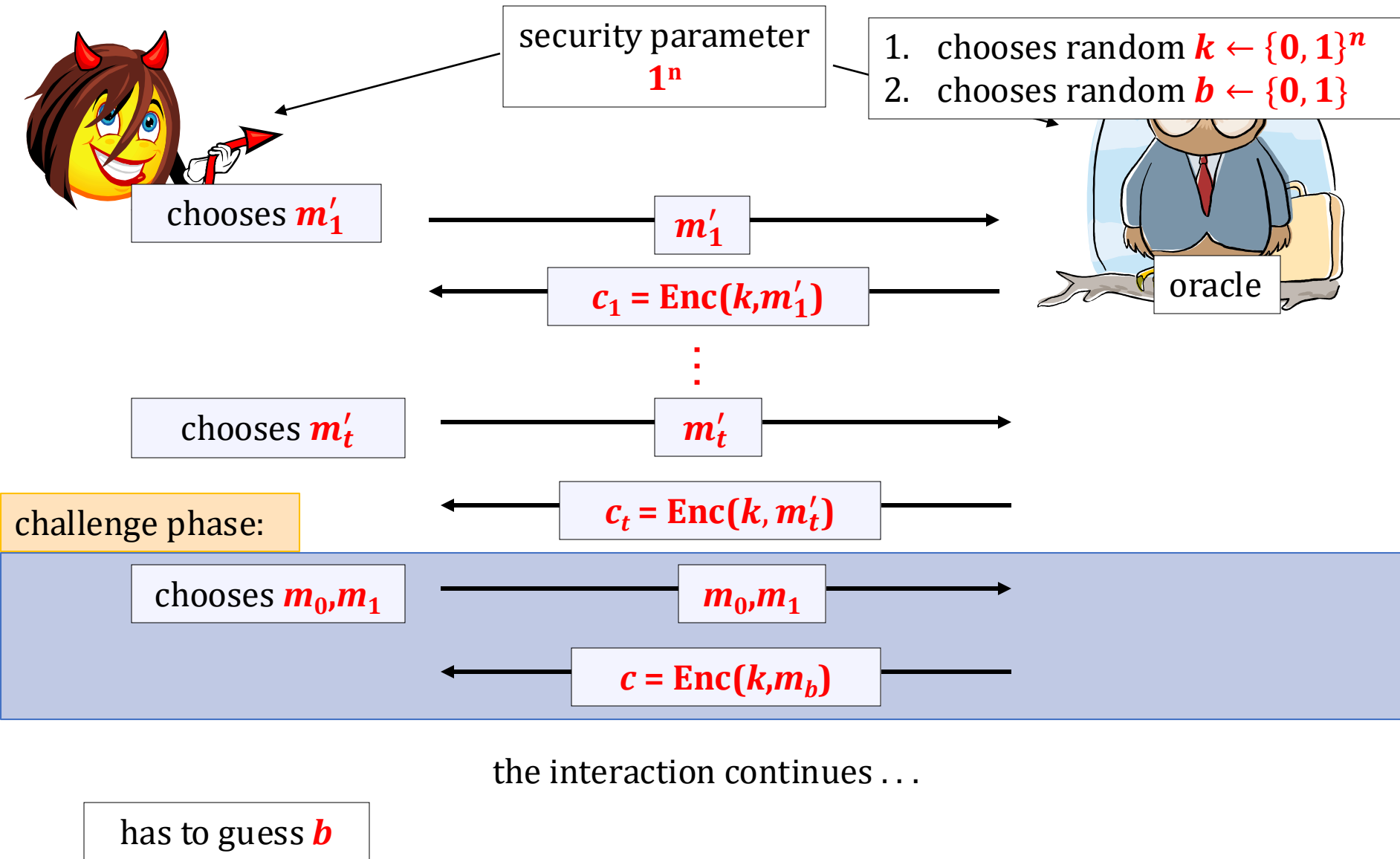
(good tradition: be as pessimistic as possible)

A real-life example

Example: routing



A chosen-plaintext attack (CPA)



CPA-security

We will sometimes say: **IND-CPA-secure**

Security definition

We say that **(Enc,Dec)** has **indistinguishable encryptions under a chosen-plaintext attack (CPA)** if

every **randomized polynomial time** adversary
guesses **b** correctly
with probability at most $\frac{1}{2} + \epsilon(n)$, where **ϵ** is negligible.

Observation

A **CPA-secure** encryption scheme cannot be deterministic.

Typical options:

- **Enc** has a “state” (e.g. a counter)
- **Enc** is randomized, i.e., it takes as additional input:
 - some perfect randomness **R**, or
 - takes as an **nonce R**

weaker
requirement

nonce = “**n**umber used **once**”

Other attacks known in the literature

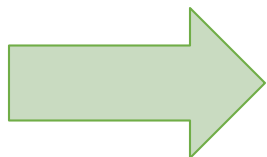


- **ciphertext-only attack** – the adversary has no information about the plaintext
- **known plaintext attack** – the plaintext are drawn from some distribution that the adversary does not control
- **batch chosen-plaintext attack** – like the **CPA** attack, but the adversary has to choose m'_1, \dots, m'_t at once.

(“our” **CPA**-attack is also called the “**adaptive CPA**-attack”)

- **chosen ciphertext attack** – we will discuss it later...

Plan



1. Computational definitions of security
2. If semantically-secure encryption exists, then **P \neq NP**
3. A proof that “the PRGs imply secure encryption”
4. Theoretical constructions of PRGs
5. Stream ciphers
6. Pseudorandom functions and permutations
7. Block ciphers
8. Practical considerations

Is it possible to prove security?

Bad news:

Theorem

If semantically-secure
encryption exists
(with $|k| < |m|$)



then

$P \neq NP$

Intuition: if $P = NP$ then the adversary can guess the key...
(formal proof – exercises)

Moral:

“If **P=NP**, then the semantically-secure encryption is broken”

Is it 100% true?

Not really...

This is because even if **P=NP** we do not know what are the constants.

Maybe **P=NP** in a very “inefficient way”...

To prove security of a cryptographic scheme we need to show a lower bound on the computational complexity of some problem.

In the “asymptotic setting” that would mean that
at least
we show that **$P \neq NP$** .

Does the implication in the other direction hold?
(that is: does **$P \neq NP$** imply anything for cryptography?)

No! (at least as far as we know)

Therefore

proving that an encryption scheme is secure is probably much
harder than proving that **$P \neq NP$** .

What can we prove?

We can prove conditional results.



That is, we can show theorems of a type:

Suppose that some
“computational
assumption **A**”
holds



then scheme **X** is
secure.

Suppose that some
scheme **Y** is secure



then scheme **X** is
secure.

Research program in cryptography

Base the security of cryptographic schemes on a small number of well-specified “computational assumptions”.

Examples of **A**:

f is one-way

“decisional Diffie-Hellman assumption”

“strong RSA assumption”

Some “computational
assumption **A**”
holds

in this we
have to
“believe”

interesting only if
this is “far from
trivial”



then scheme **X** is
secure.

the rest is
provable

Example

Suppose that G is a
“cryptographic
pseudorandom generator”



we can construct a secure
encryption scheme based on G

A natural question

What is “**the minimal assumption**” needed for cryptography?

Answer: existence **one-way functions**.

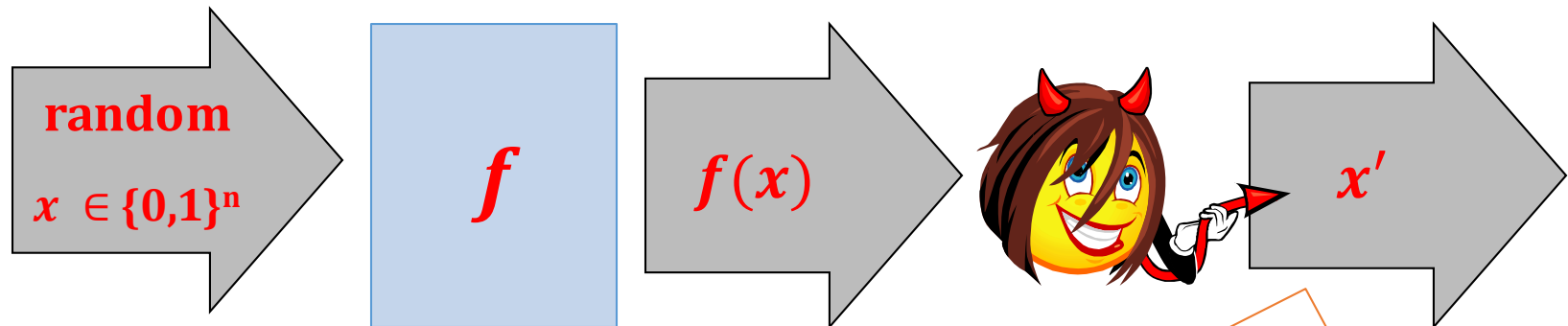
We introduce them now.

One-way functions

A function

$$f: \{0,1\}^* \rightarrow \{0,1\}^*$$

is **one-way** if it is: **(1)** poly-time computable, and **(2)** “hard to invert it”.



probability that any poly-time adversary
outputs x' such that

$$f(x) = f(x')$$

is negligible in n .

A real-life analogue: phone book



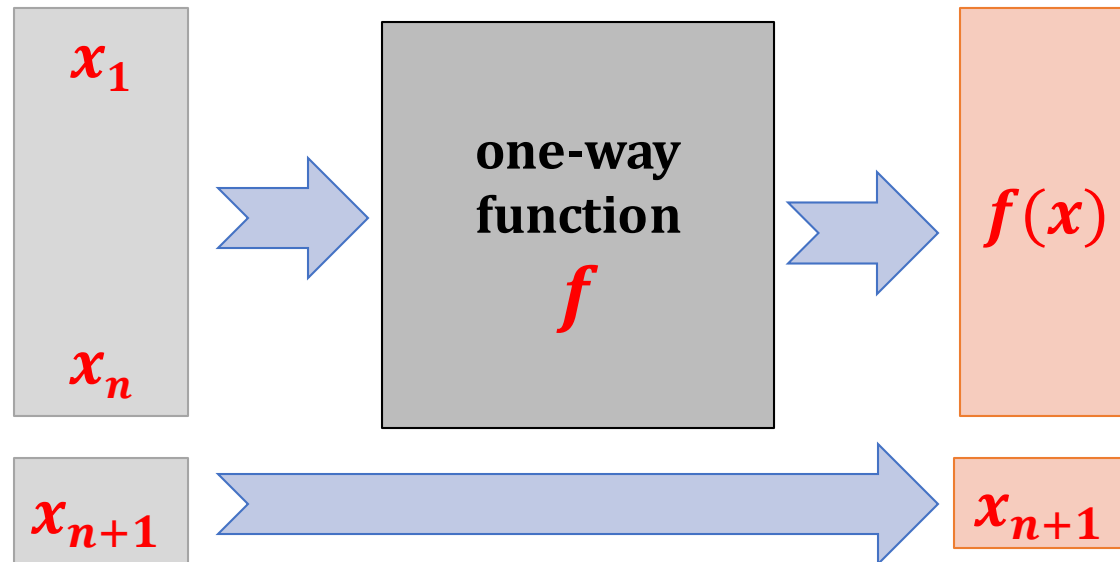
A function:

people → numbers

is “one way”.

One way functions do not “hide all the input”

Example:



$f'(x_1, \dots, x_{n+1}) := f(x_1, \dots, x_n) \parallel x_{n+1}$ is also a one-way function

One way-functions – main properties

They are “**hard to invert**” on random input x , but:

- “**inverting**” can also mean finding some other x' such that

$$f(x) = f(x')$$

- it's ok if **partial information** on x can be computed from $f(x)$.

Their existence is the “**minimal assumption**” needed for cryptography.

Why “minimal assumption?”

Recall the following idea from the previous lecture:

- (m – a message)
1. the key K is chosen uniformly at random
 2. $C := \text{Enc}_K(m)$ is given to the adversary

An idea

“The adversary should not be able to compute K .”

Observation: it’s not a sufficient condition for security.

But it is **necessary**.

if the adversary can compute K from $C := \text{Enc}_K(m)$ from some m then (Enc, Dec) is **not** secure

Moral: f defined as $f(K) := \text{Enc}_K(m)$ has to be **one-way**.

One-way functions more formally

experiment (machine M , function f)

1. pick a random element $x \leftarrow \{0, 1\}^n$
2. let $y := f(x)$,
3. let x' be the output of M on y
4. we say that M won if $f(x') = y$.

We will say that a poly-time computable $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ is **one-way** if

\forall $P(M \text{ wins})$ is negligible

polynomial-time
Turing Machine M

Example of a (candidate for) a one-way function

If **P=NP** then **one-way functions don't exist**.

Therefore currently no function can be proven to be one-way.
But there exist candidates.

Example:

$f(p, q) = pq$, where p and q are primes such that $|p| = |q|$.

this function is defined on

primes \times primes,

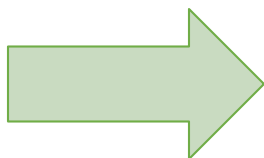
not on

$\{0,1\}^*$

but it's just a technicality

Plan

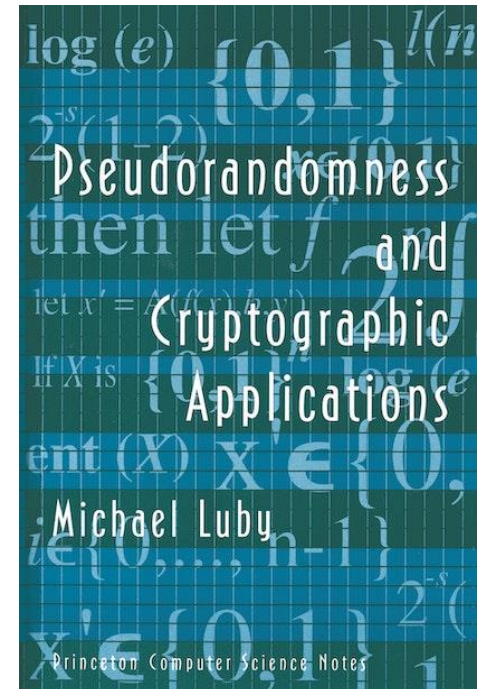
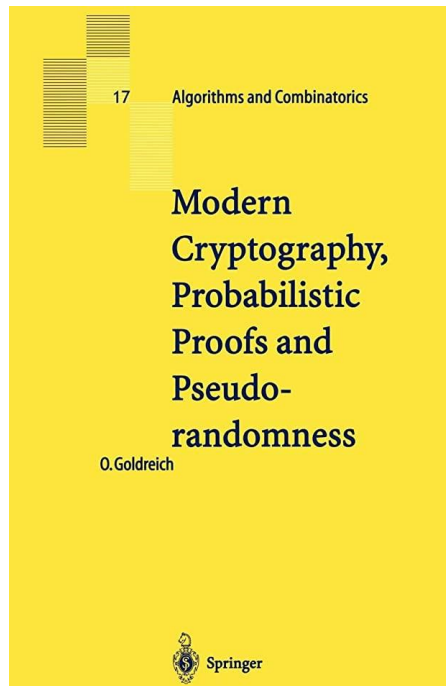
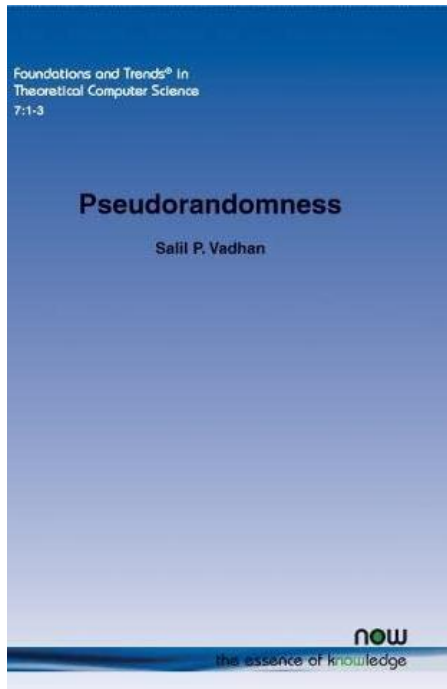
1. Computational definitions of security
2. If semantically-secure encryption exists, then **P \neq NP**
3. A proof that “the PRGs imply secure encryption”
4. Theoretical constructions of PRGs
5. Stream ciphers
6. Pseudorandom functions and permutations
7. Block ciphers
8. Practical considerations



Our tool: pseudorandomness

An important topic on its own.

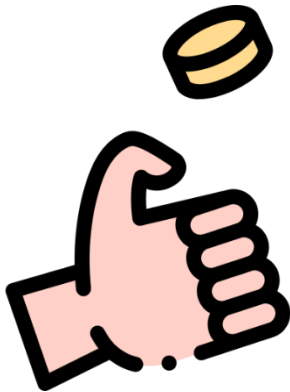
Applications outside of cryptography (e.g.:
derandomization of randomized algorithms).



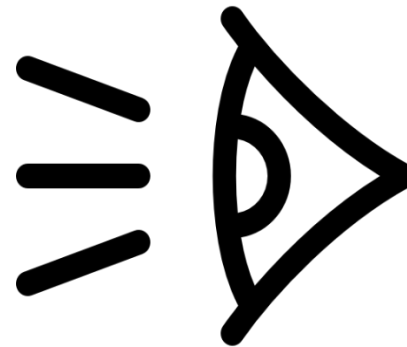
Intuition

Randomness depends on the “**power of an observer**”.

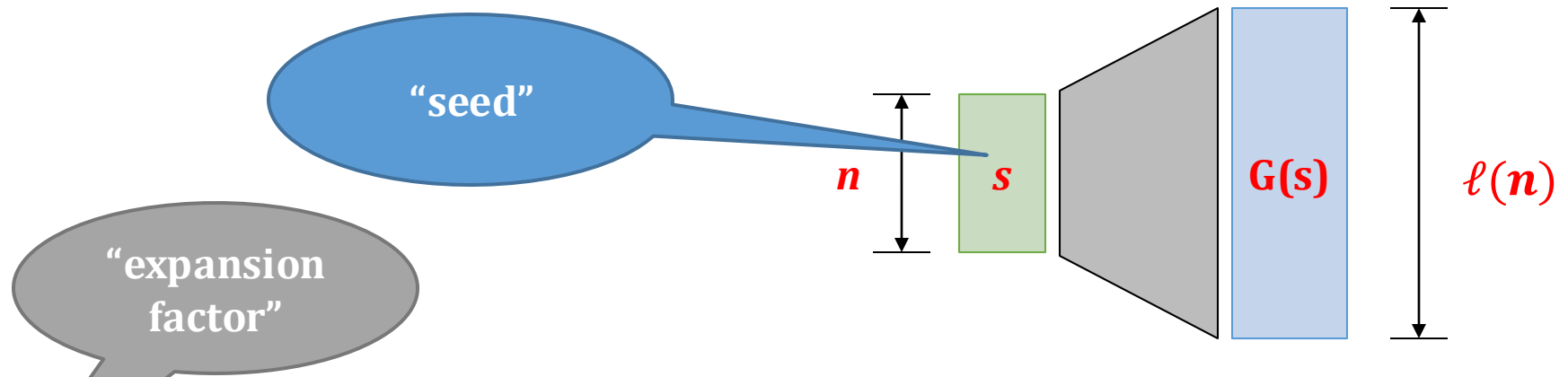
Think of a **coin-toss**:



An observer that can record
and process coin behavior
can predict the outcome



Pseudorandom generators



Definition

ℓ – polynomial such that always $\ell(n) > n$

An algorithm $G : \{0,1\}^* \rightarrow \{0,1\}^*$ is called a **pseudorandom generator (PRG)** if

for every n

and for every s such that $|s| = n$

we have

$$|G(s)| = \ell(n).$$

this has to
be
formalized

and for a random s the value $G(s)$ "looks random".

Idea

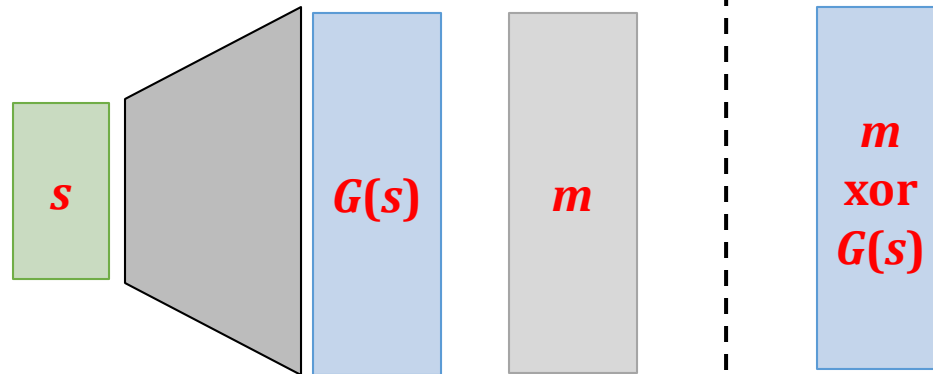
Use PRGs to “shorten” the key in the one time pad

for a moment just
consider a **single
message case**

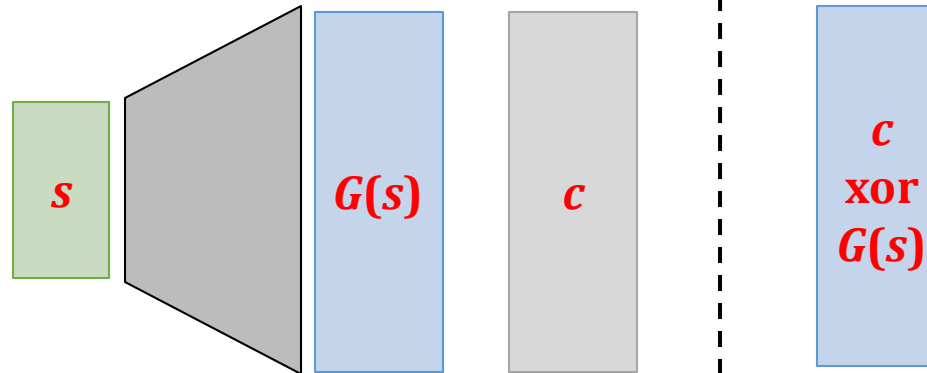
Key: random string of length n

Plaintexts: strings of length $\ell(n)$

Enc(s, m)



Dec(s, m)



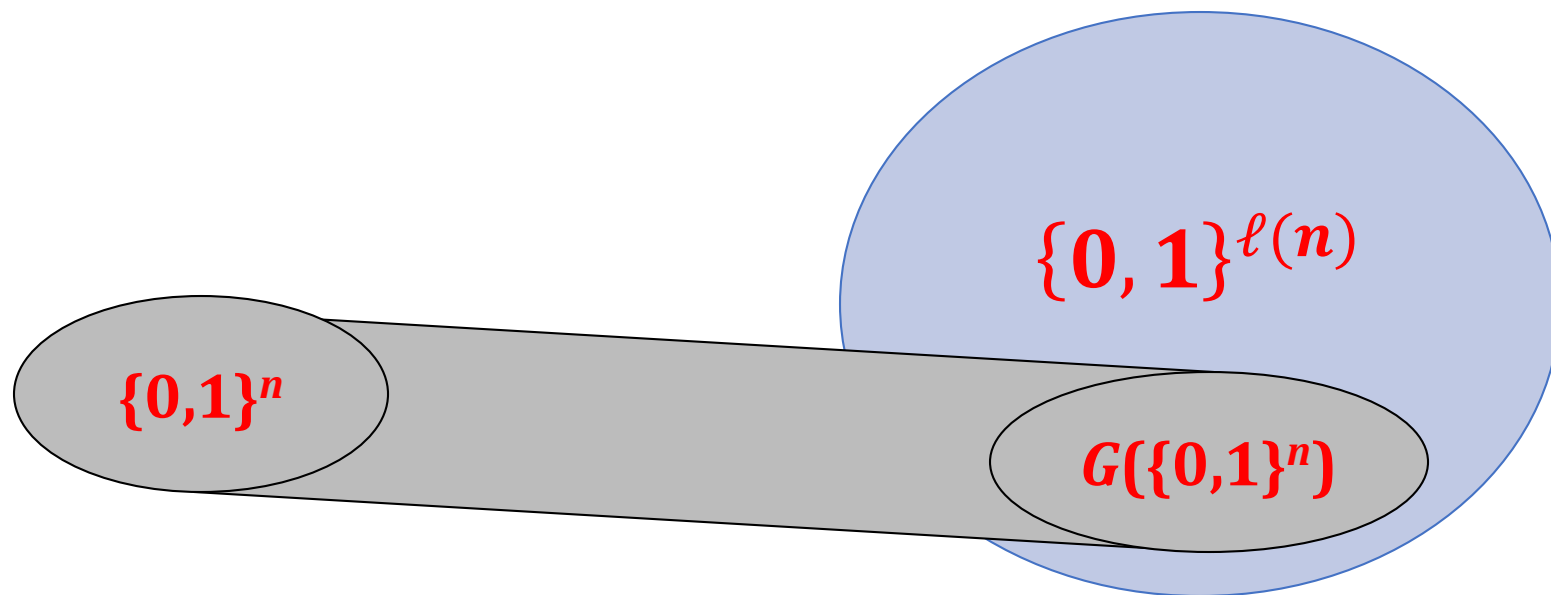
If we use a “normal PRG” – this idea doesn’t work
We have to use the **cryptographic PRGs**.

“Looks random”

Suppose $s \in \{0,1\}^n$ is chosen randomly.

Can $G(s) \in \{0,1\}^{\ell(n)}$ be uniformly random?

No!



“Looks random”

What does it mean?

Non-cryptographic applications:

should pass **some statistical tests**.

Cryptography:

should pass **all polynomial-time tests**.

Non-cryptographic PRGs

Example: **Linear Congruential Generators (LCG)**
defined recursively

- $X_0 \in \mathbb{Z}_m$ – the key
- for $n = 1, 2, \dots$ let
$$X_{n+1} := (a \cdot X_n + c) \bmod m$$

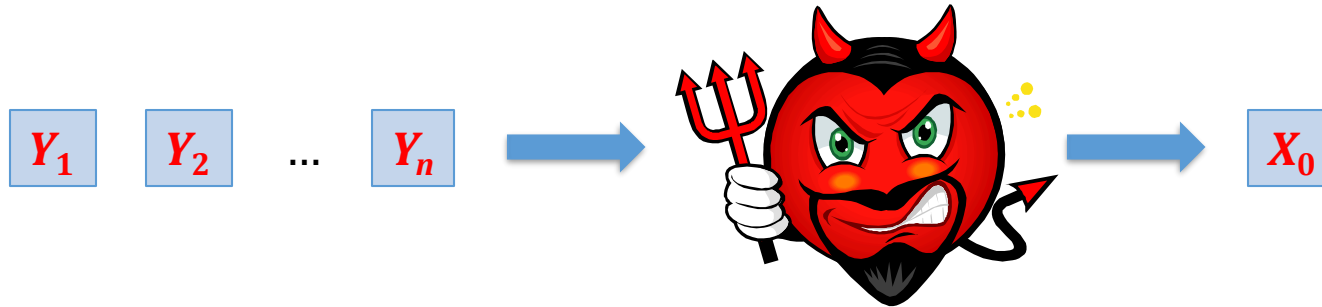
output: Y_1, Y_2, \dots where

Y_i = first t bits of each X_i

rand() function in Windows – an **LCG** with

$a = 214013, c = 2531011, m = 2^{32}, t = 15$

How to break an LRS



Solve linear equations with “partial knowledge” (because you only know only first t bits)

See:

G. Argyros and A. Kiayias: I Forgot Your Password: Randomness Attacks Against PHP Applications, *USENIX Security '12*

(successful attacks on password-recovery mechanisms in PHP)

PRG – main idea of the definition

scenario 0

a random string R

should not be able to distinguish...

scenario 1

$G(S)$



a probabilistic
polynomial-time
distinguisher D

outputs:

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

Cryptographic PRG

outputs:

a random string R

or

$G(S)$ (where S random)



0 if he thinks it's R

1 if he thinks it's $G(S)$

Should not be able to distinguish...

Definition

n – a parameter

S – a variable distributed uniformly over $\{0, 1\}^n$

R – a variable distributed uniformly over $\{0, 1\}^{\ell(n)}$

G is a **cryptographic PRG** if

for every polynomial-time Turing Machine D

we have that

$$|P(D(R) = 1) - P(D(G(S)) = 1)|$$

is negligible in n .

Constructions

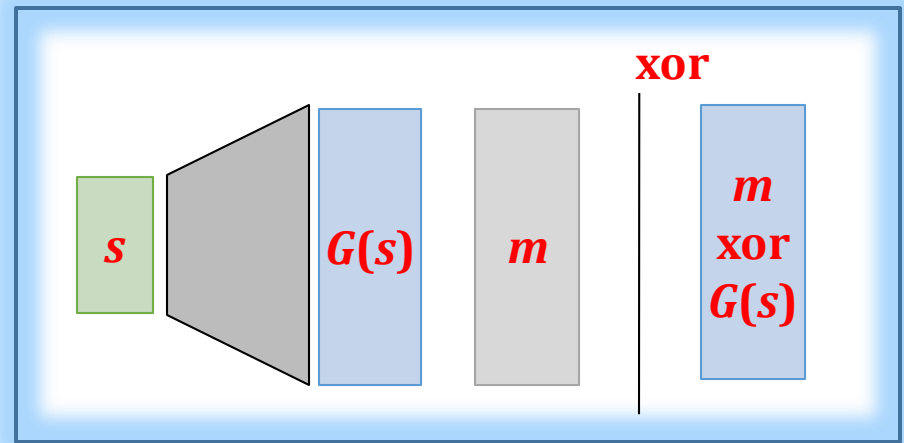
There exists constructions of cryptographic pseudorandom-generators, that are **conjectured** to be secure.

We will discuss them later...

Theorem

(for simplicity consider only the single message case)

If **G** is a **cryptographic PRG** then the encryption scheme constructed before is semantically secure.



cryptographic PRGs
exist



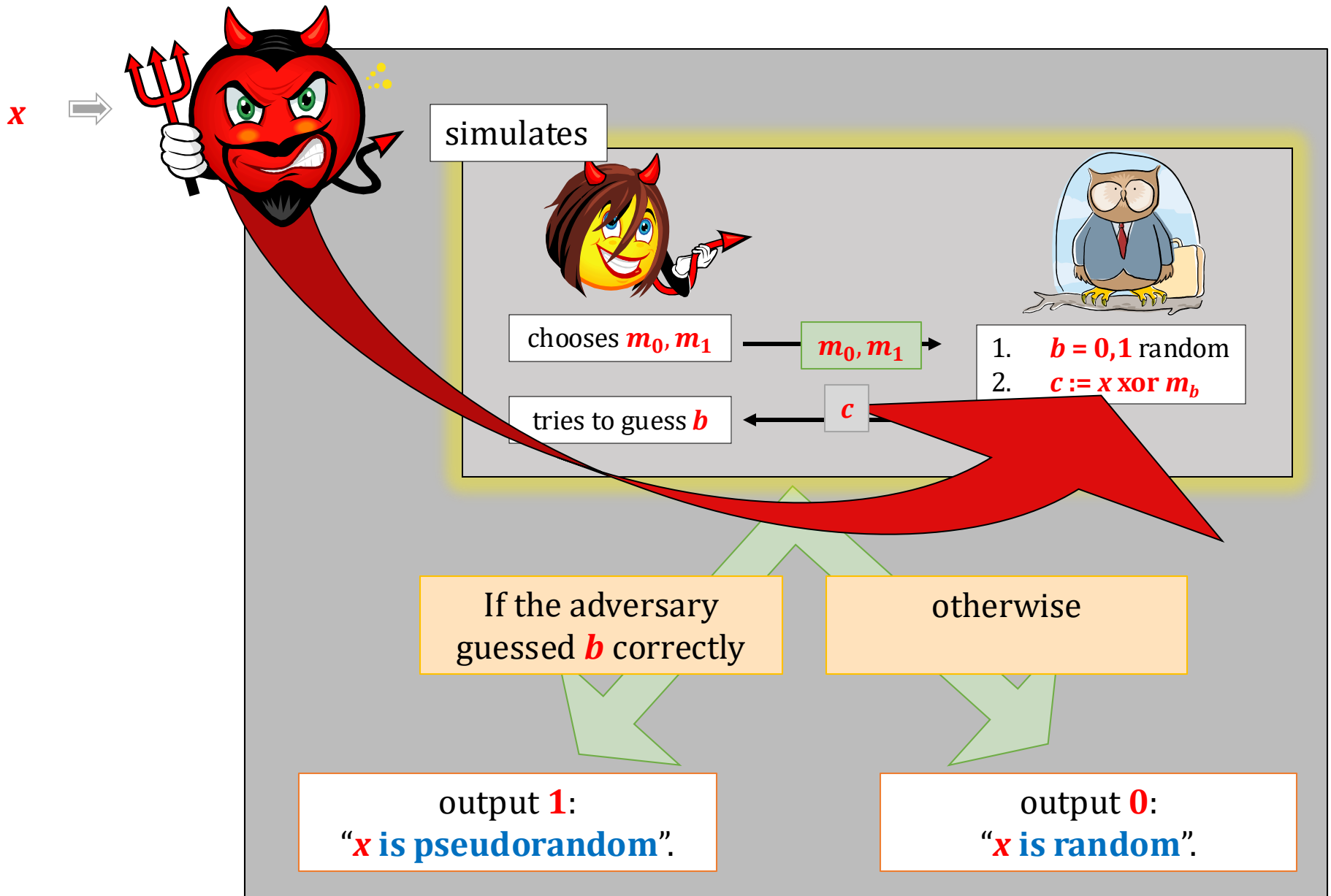
CPA-secure encryption
exists

Proof (sketch)

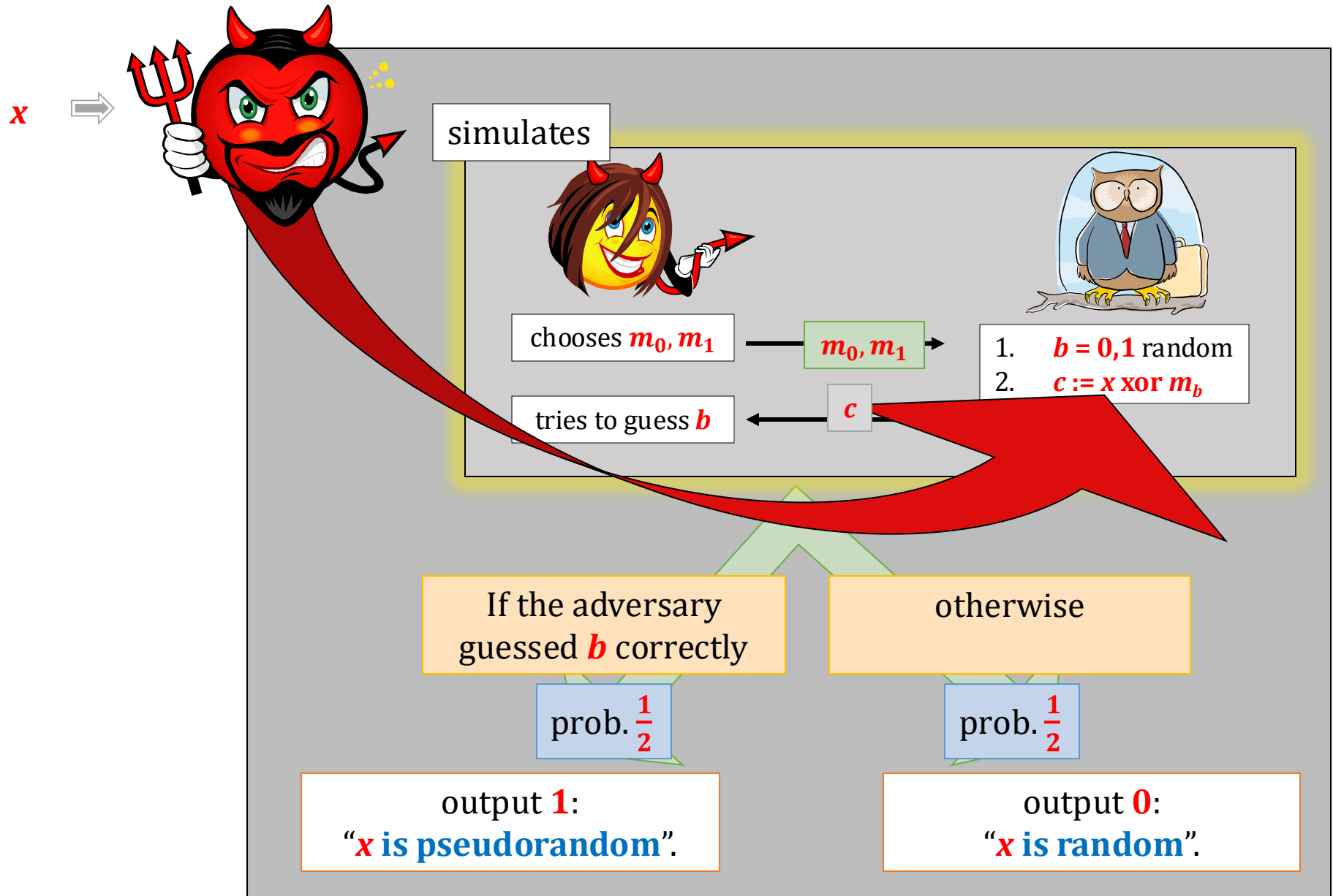
Suppose that it is **not** secure.

Therefore, there exists a poly-time adversary that wins the

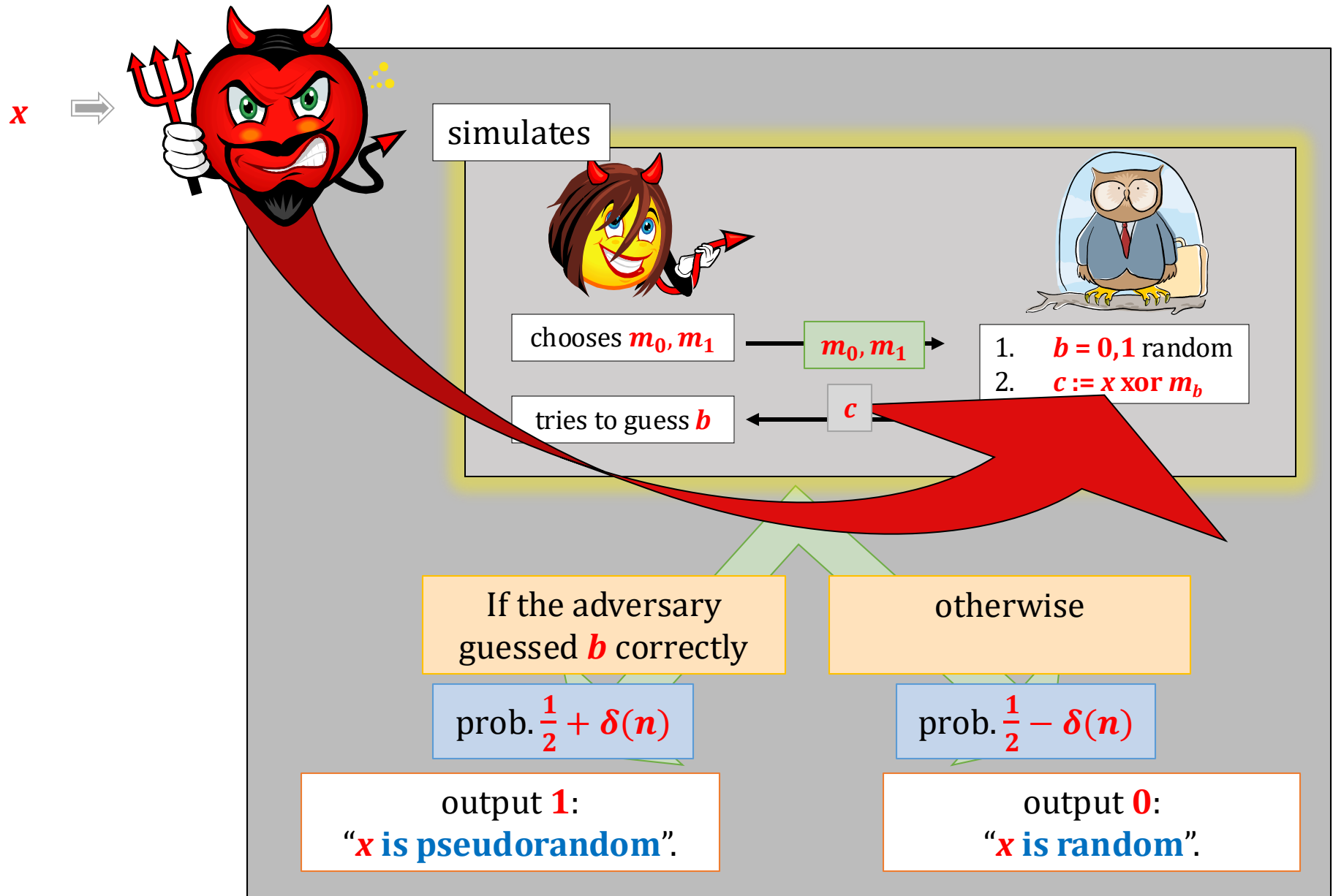
“guessing game” with probability $\frac{1}{2} + \delta(n)$ where $\delta(n)$ is not negligible.



“scenario 0”: x is a random string



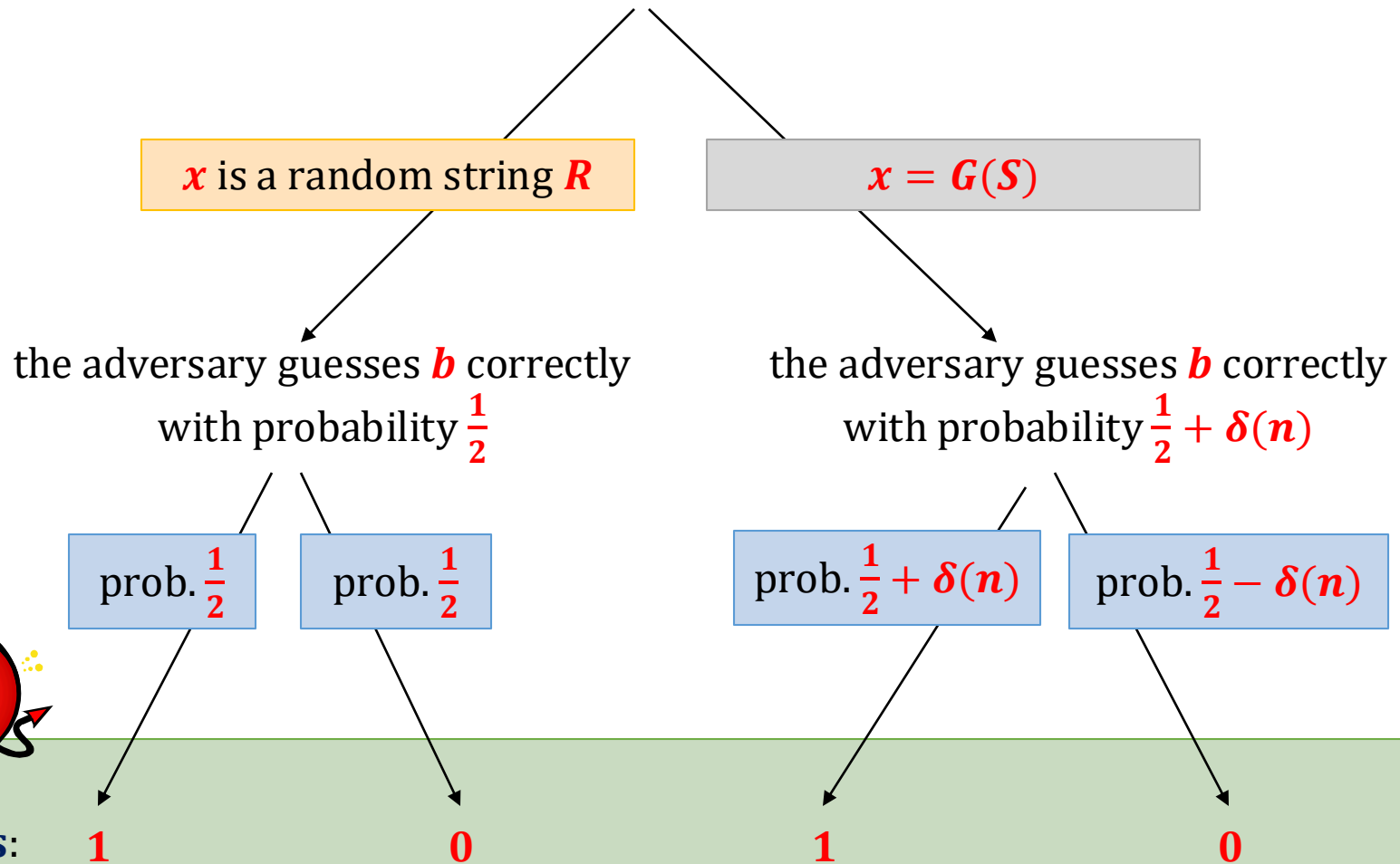
“scenario 1”: $x = G(S)$



Hence



outputs:



$$|P(D(R) = 1) - P(D(G(S)) = 1)| = \left| \frac{1}{2} - \left(\frac{1}{2} + \delta(n) \right) \right| = |\delta(n)|$$

Since δ is not negligible G cannot be a **cryptographic PRG**.

The complexity

The distinguisher



simply simulated

one execution of the adversary



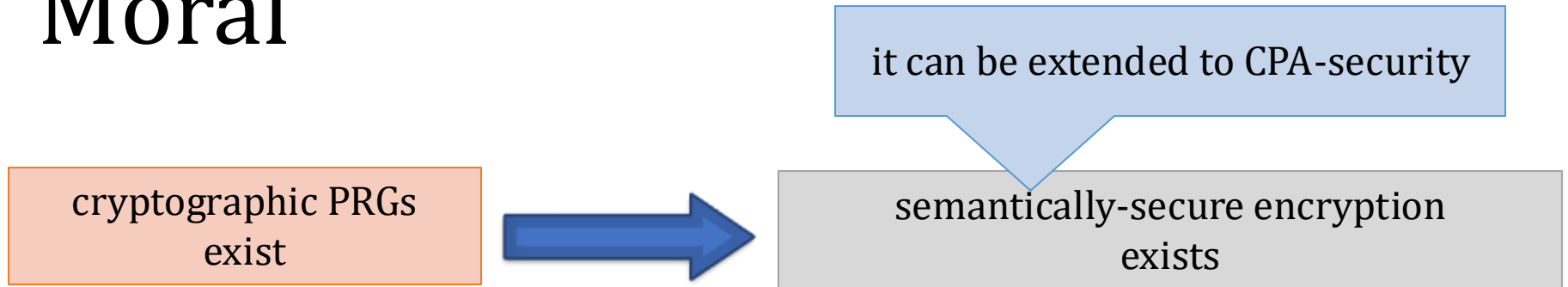
against the oracle



Hence he works in polynomial time.

QED

Moral



To construct secure encryption it suffices to construct a secure PRG.

Moreover, we can also state the following:

Informal remark. The reduction is tight.

A question

What if the distinguisher  needed to simulate **1000** executions of the adversary  ?

An (informal) answer

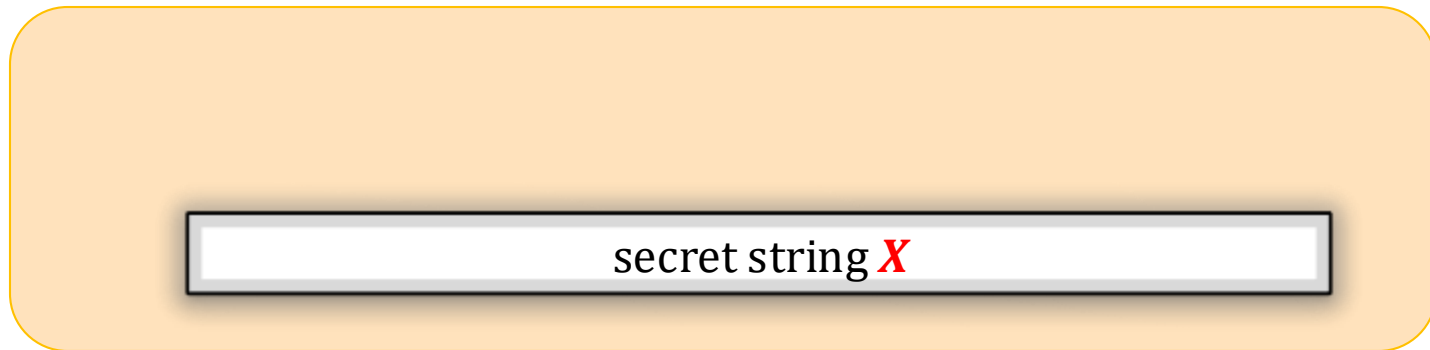
Then, the encryption scheme would be “**1000** times less secure” than the pseudorandom generator.

Why?

To achieve the same result  needs to work **1000** times more than  .

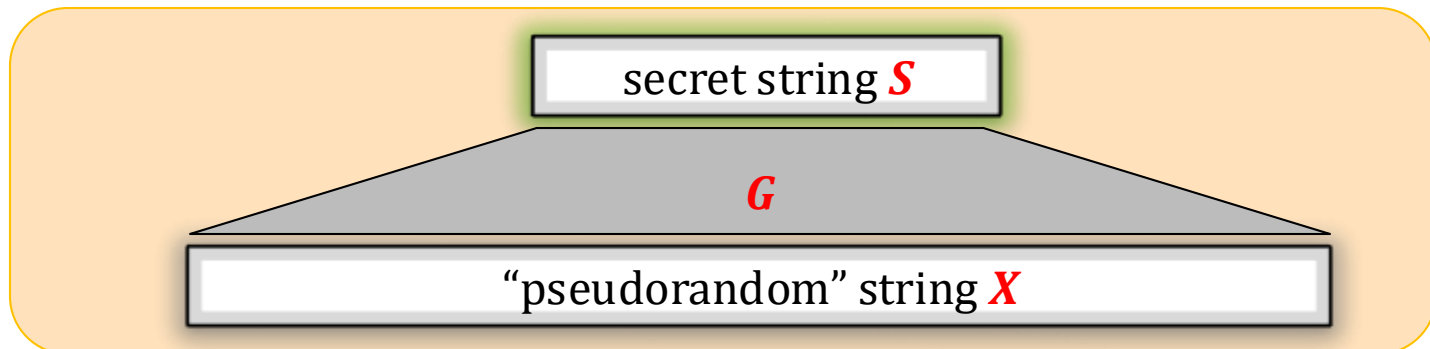
General rule

Take a secure system that uses some long secret string X .



Then, you can construct a system that uses a shorter string S ,
and expands it using a PRG:

$$X = G(S)$$



Constructions of PRGs

A theoretical result

a PRG can be constructed from any **one-way function**
(**very elegant**, **impractical**, **inefficient**)

Based on hardness of some particular computational problems

For example

[Blum, Blum, Shub. *A Simple Unpredictable Pseudo-Random Number Generator*]

(**elegant**, **more efficient**, still **rather impractical**)

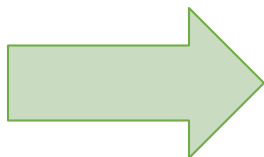
“Stream ciphers”

ugly, **very efficient**, **widely used in practice**

Examples: RC4, Trivium, SOSEMANUK,...

Plan

1. Computational definitions of security
2. If semantically-secure encryption exists, then **P \neq NP**
3. A proof that “the PRGs imply secure encryption”
4. Theoretical constructions of PRGs
5. Stream ciphers
6. Pseudorandom functions and permutations
7. Block ciphers
8. Practical considerations

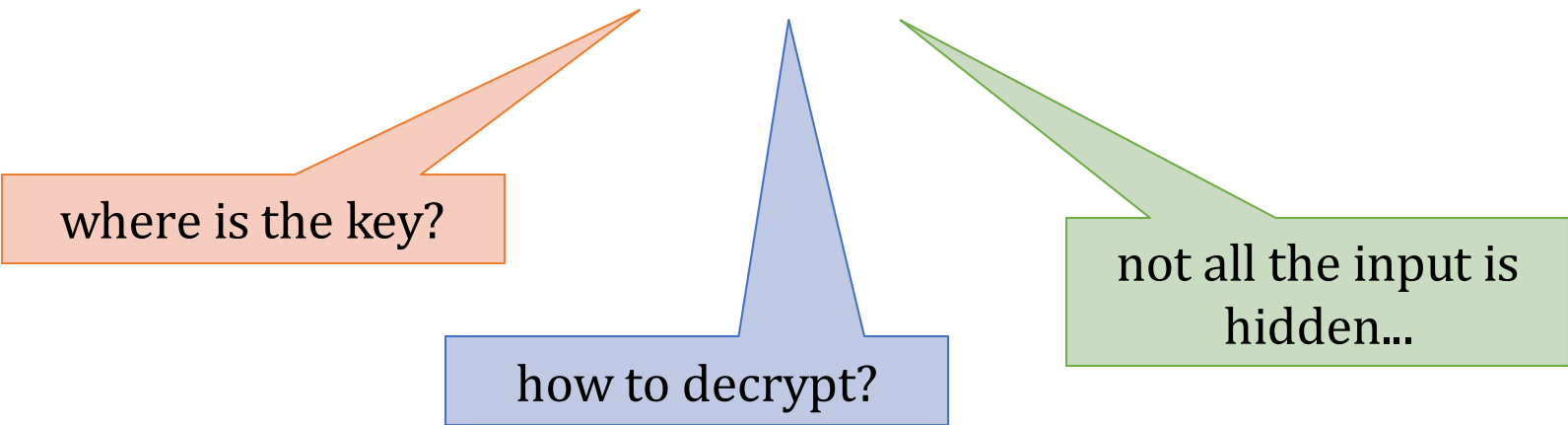


How to encrypt with one-way functions?

Naive (and wrong) idea:

1. Take a one-way function f ,
2. Let a ciphertext of a message M be equal to

$$C := f(M)$$



where is the key?

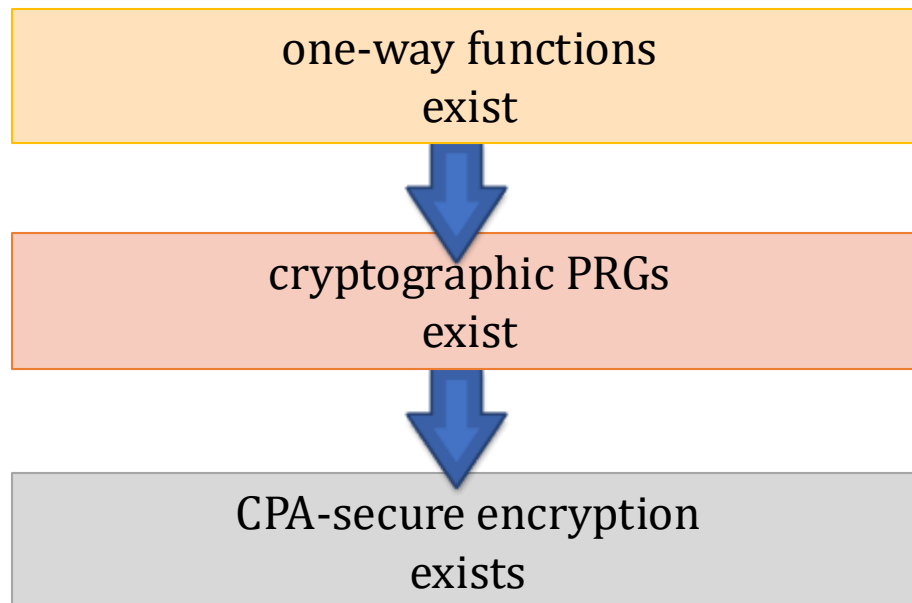
how to decrypt?

not all the input is hidden...

One of the most fundamental results in symmetric cryptography

[Håstad, Impagliazzo, Levin, Luby *A Pseudorandom Generator from any One-way Function*]:

“a PRG can be constructed from any **one-way function**”



Why is this result so important?

It connects **two types of hardness**:

- hardness of **guessing** some value



K ?

- hardness of **distinguishing** two experiments

experiment

0

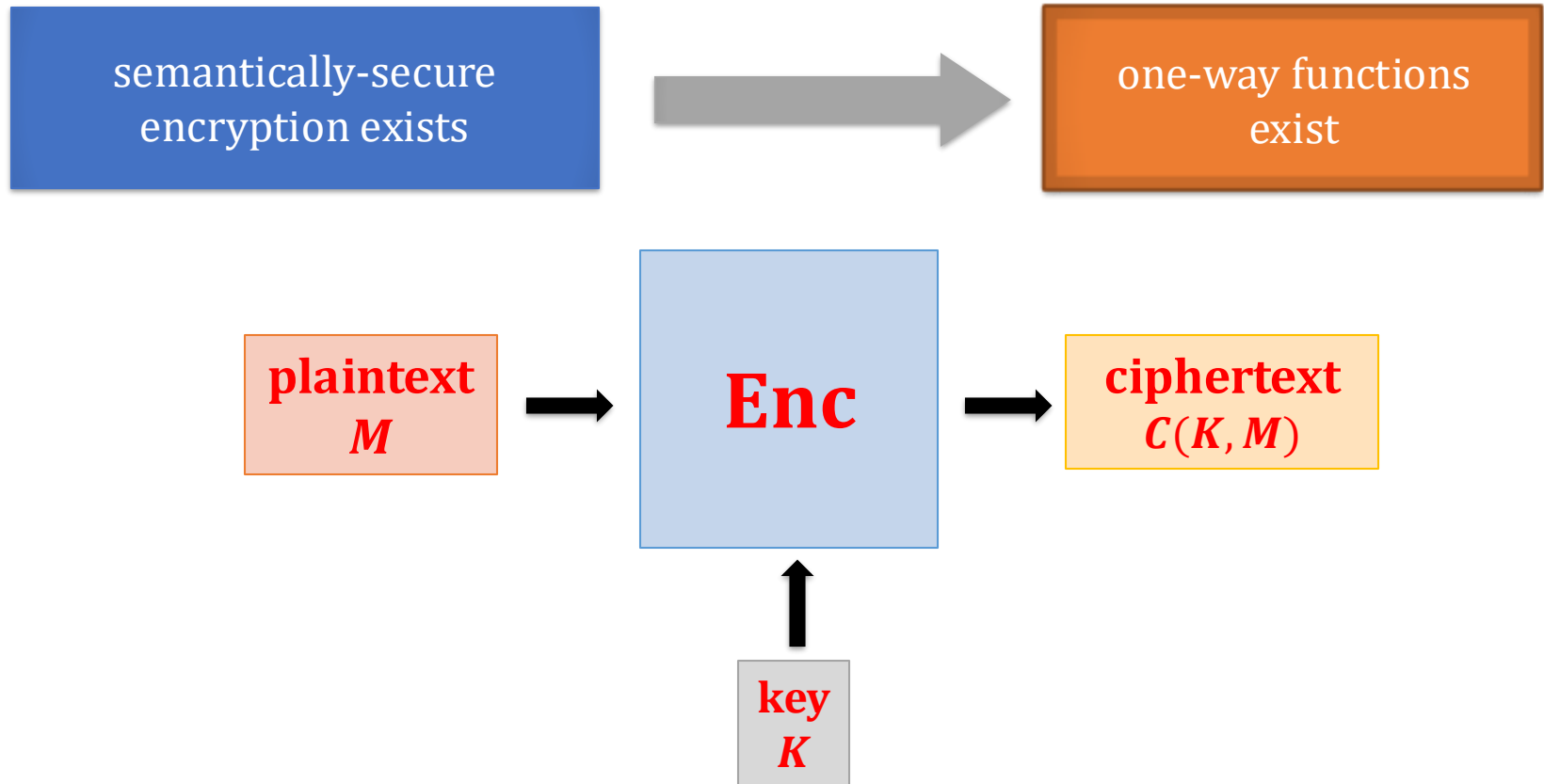


?

experiment

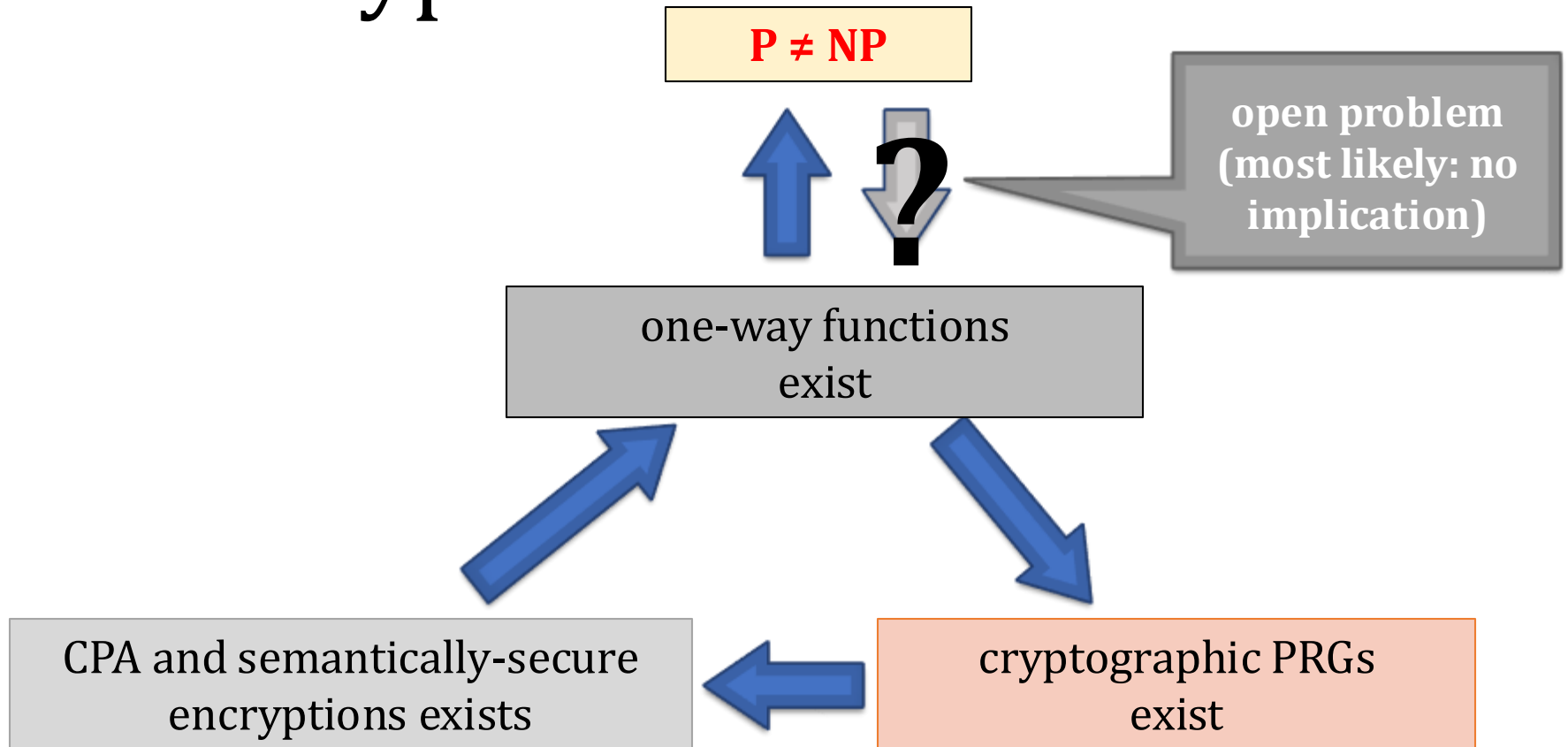
1

The implication also holds in the other direction



$f(K) = \text{Enc}(K, (0, \dots, 0))$ is a one-way function

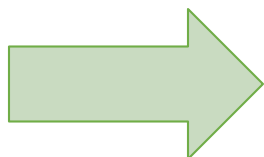
“Minicrypt”



The “world” where the one-way functions exist is called “**minicrypt**”.

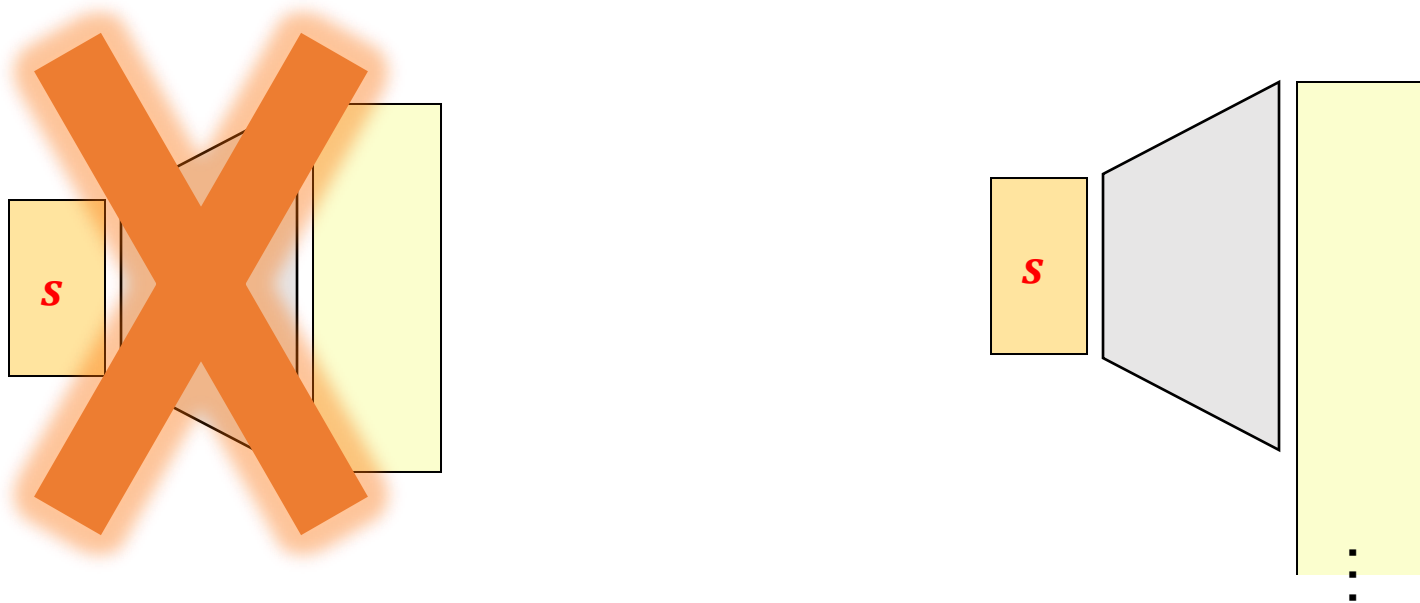
Plan

1. Computational definitions of security
2. If semantically-secure encryption exists, then **P \neq NP**
3. A proof that “the PRGs imply secure encryption”
4. Theoretical constructions of PRGs
5. Stream ciphers
6. Pseudorandom functions and permutations
7. Block ciphers
8. Practical considerations



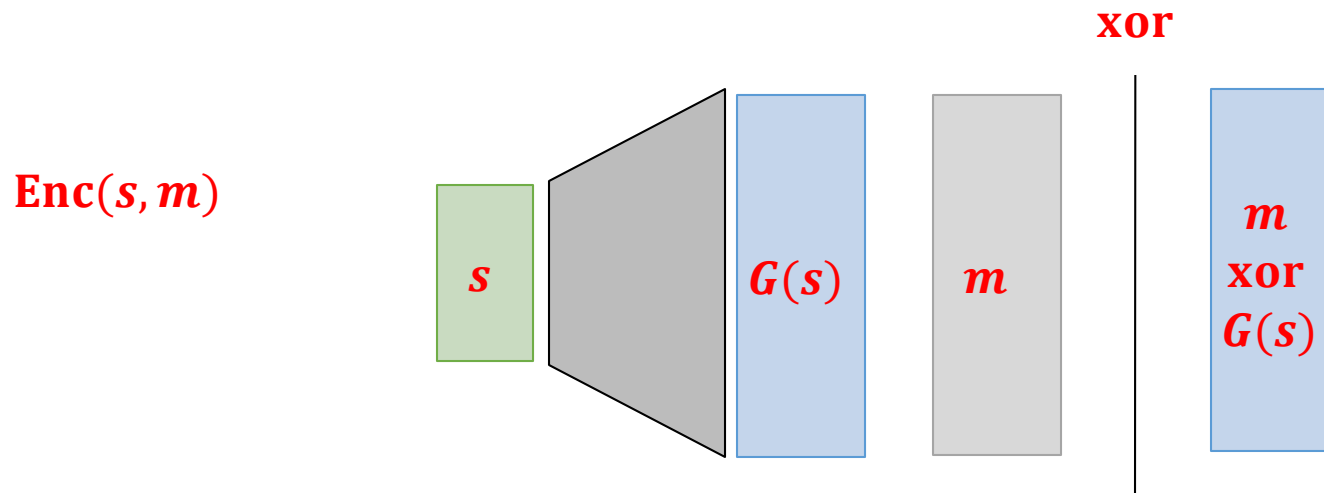
Stream ciphers

The pseudorandom generators used in practice are called **stream ciphers**



They are called like this because their output is an “infinite” **stream** of bits.

How to encrypt multiple messages using pseudorandom generators?



Of course we **cannot** just reuse the same seed
(remember the problem with the one-time pad?)

It is not just a theoretical problem!

Misuse of RC4 in Microsoft Office

[Hongjun Wu 2005]



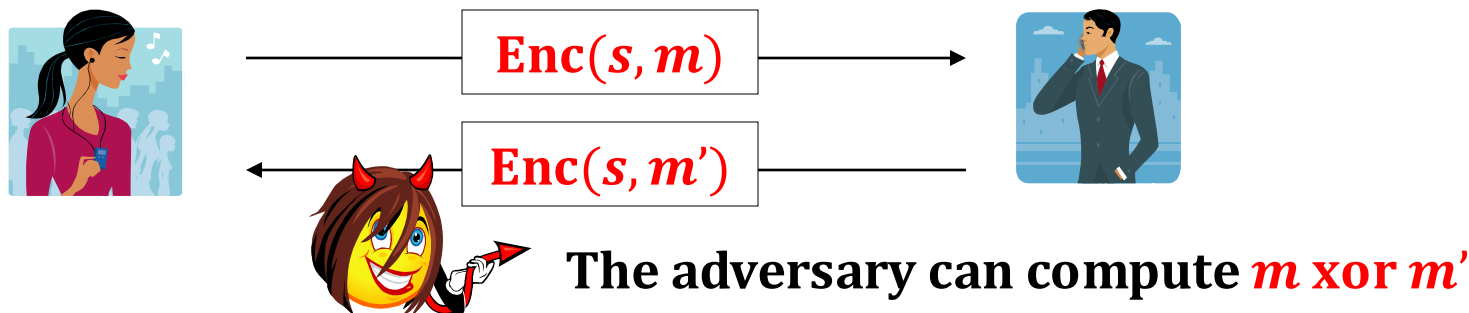
RC4 – a popular PRG/“stream cipher” (now broken)

“Microsoft Strong Cryptographic Provider”
(encryption in Word and Excel, Office 2003)

The key **s** is a function of a **password** and an **initialization vector**.

These values **do not change between the different versions** of the document!

Suppose **Alice** and **Bob** work together on some document:



What to do?

There are two solutions:

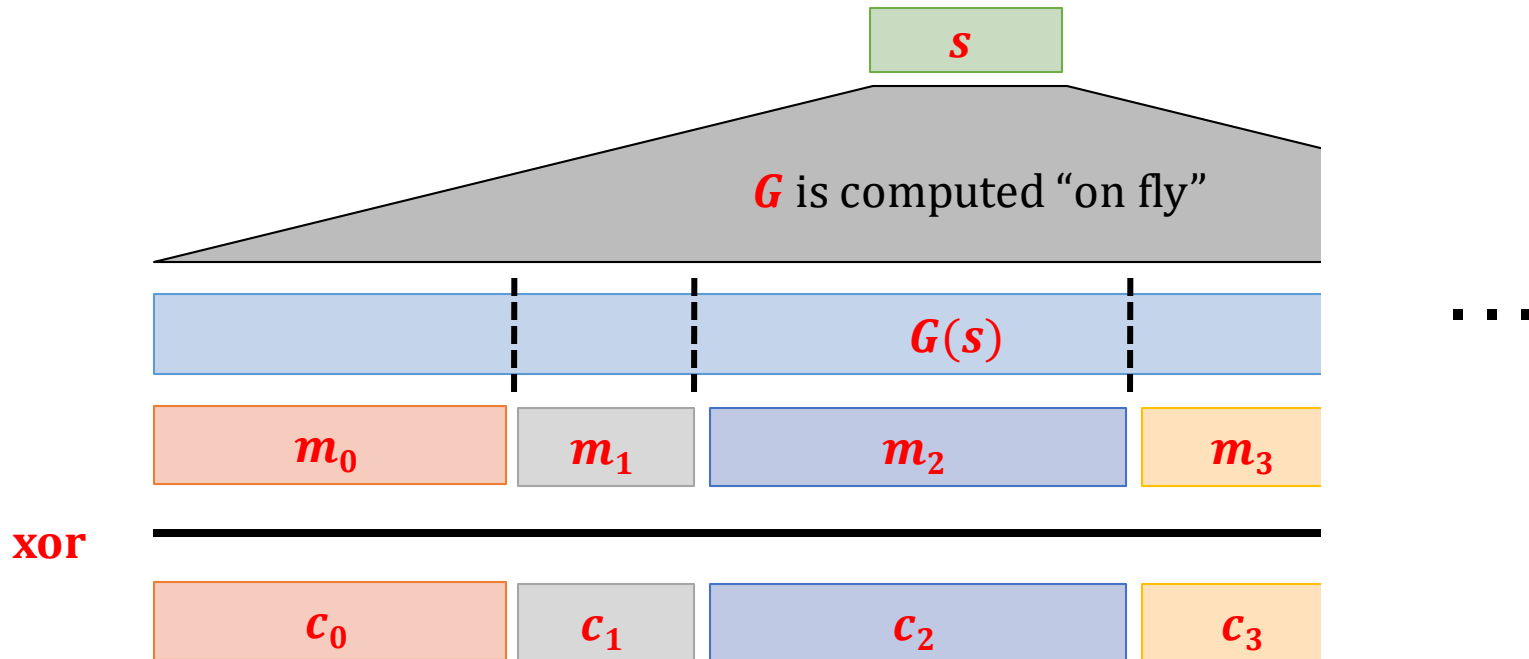
1. The **synchronized mode**
2. The **unsynchronized mode**

How to encrypt several messages

$G: \{0, 1\}^n \rightarrow \{0, 1\}^{\text{very large}}$ – a PRG.

this can be proven to be
CPA-secure

divide $G(s)$ in blocks:



Unsynchronized mode

Idea

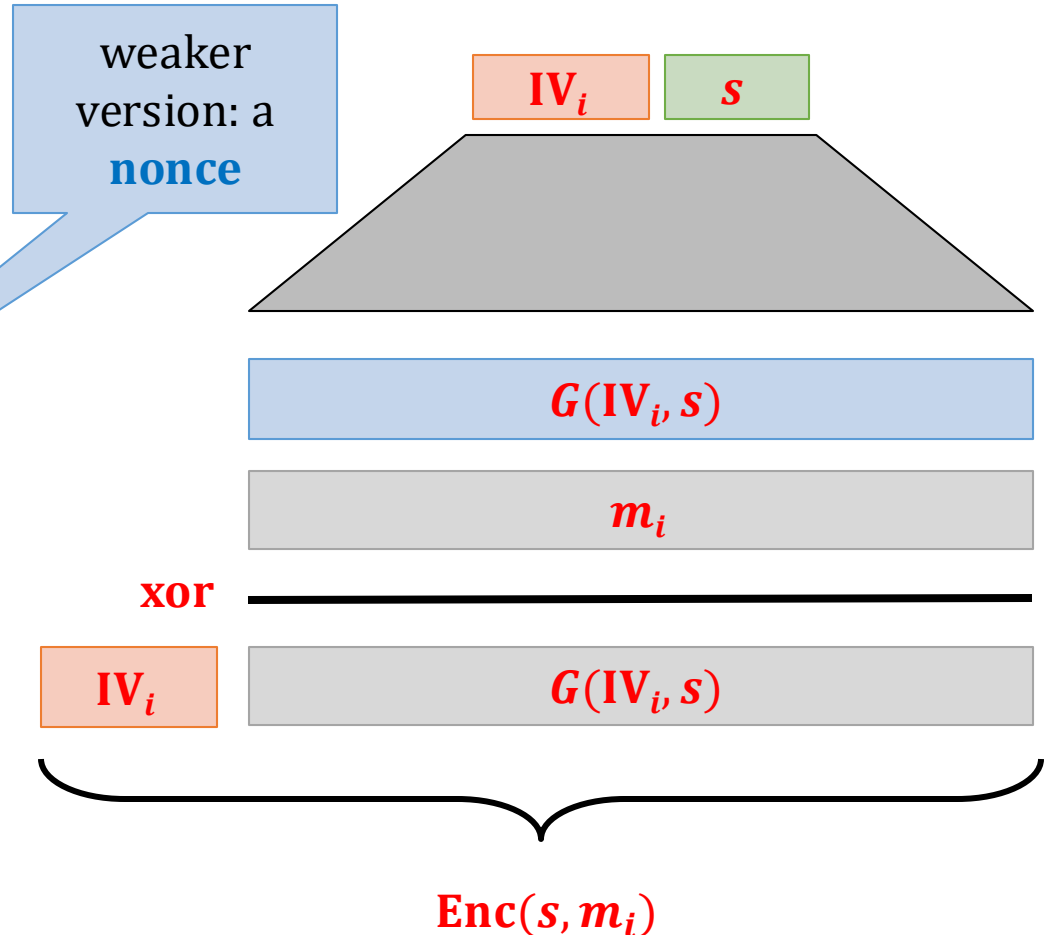
Randomize the encryption procedure.

Assume that G takes as an additional input

an **initialization vector** (IV).

The **Enc** algorithm selects a fresh random IV_i for each message m_i .

Later IV_i is included in the **ciphertext**

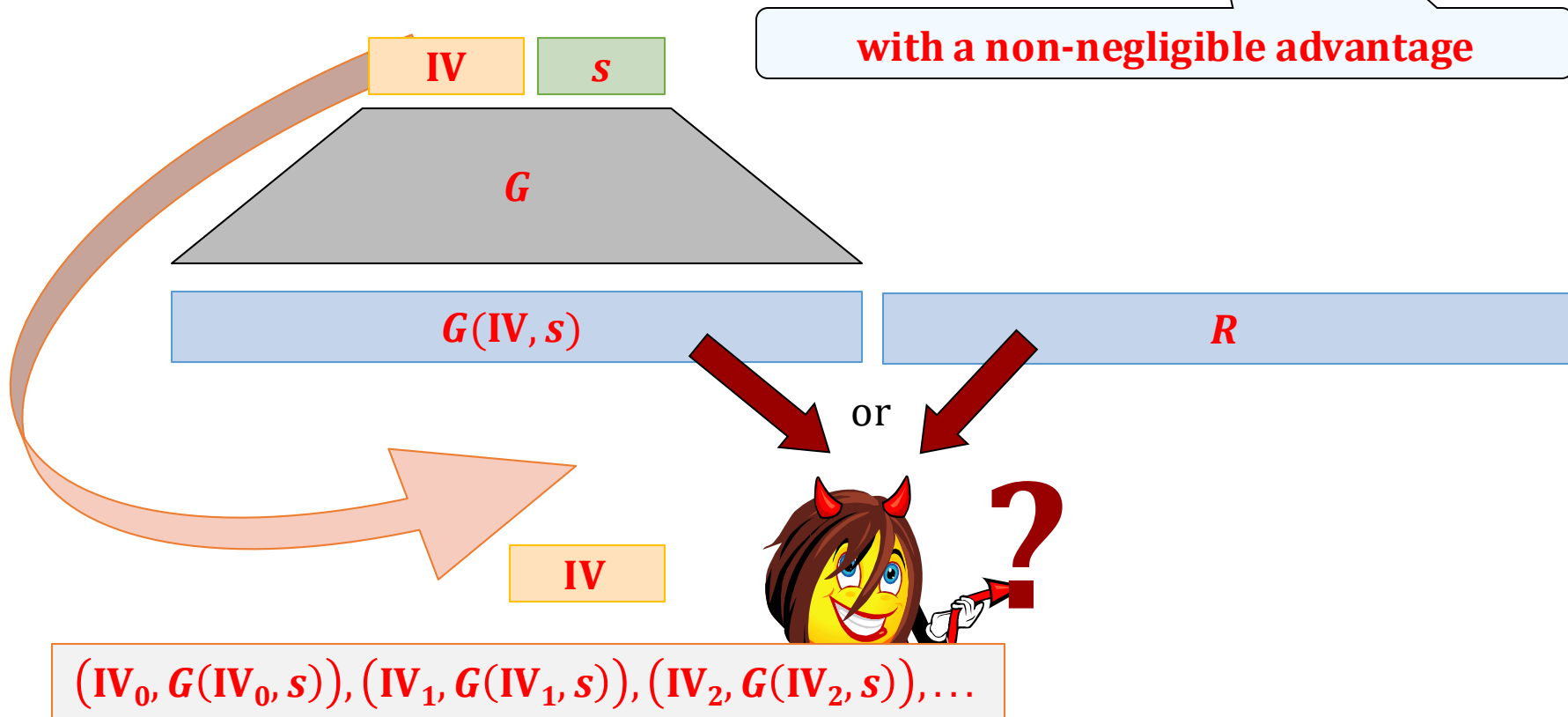


We need an “augmented” PRG

We need a **PRG** such that the adversary cannot distinguish $G(\text{IV}, s)$ from a random string even if she knows IV and some pairs

$$(\text{IV}_0, G(\text{IV}_0, s)), (\text{IV}_1, G(\text{IV}_1, s)), (\text{IV}_2, G(\text{IV}_2, s)), \dots$$

where $s, \text{IV}, \text{IV}_0, \text{IV}_1, \text{IV}_2 \dots$ are random.



How to construct such a PRG?

An **old-fashioned approach**:

1. take a standard **PRG G**
2. set **$G'(IV, s) := G(H(IV, S))$**

often:
just concatenate
 IV and **S**

where **H** is a “hash-function” (we will define cryptographic hash functions later)

A more **modern approach**:

design such a **G** from scratch.

Popular historical stream ciphers


Based on the **linear feedback shift registers**:

- **A5/1** and **A5/2** (used in **GSM**)

Ross Anderson:



completely broken



"there was a terrific row between the NATO signal intelligence agencies in the mid 1980s over whether GSM encryption should be strong or not. The Germans said it should be, as they shared a long border with the Warsaw Pact; but the other countries didn't feel this way, and the algorithm as now fielded is a French design."

- **Content Scramble System (CSS)** encryption



completely broken

Other:

- **RC4**



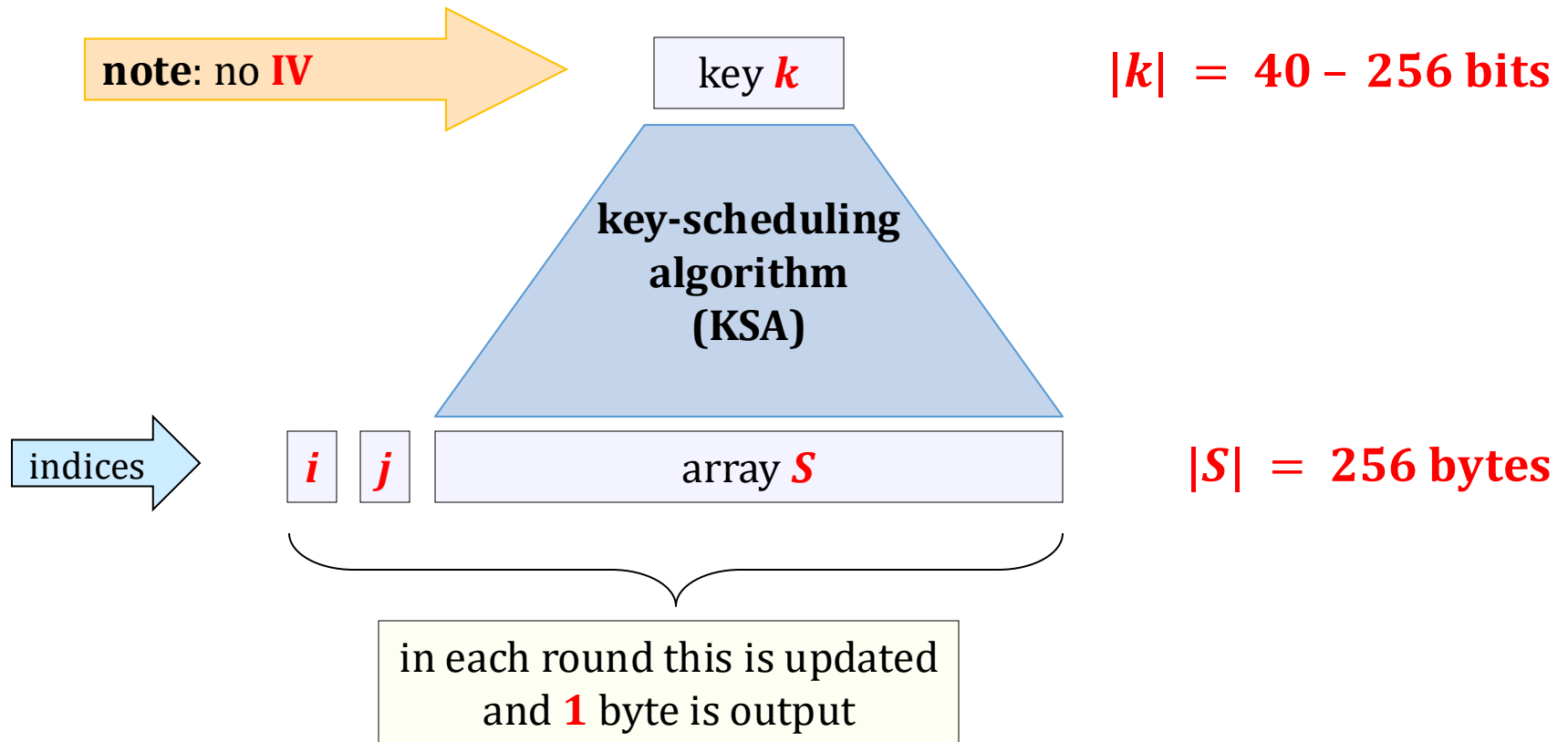
very popular in the past, now broken

RC4

- Designed by **Ron Rivest** (**RSA Security**) in 1987.
RC4 = “**Rivest Cipher 4**”, or “**Ron's Code 4**”.
- Trade secret, but in **September 1994** its description leaked to the internet.
- For legal reasons sometimes it is called: “**ARCFOUR**” or “**ARC4**”.
- Used in **WEP** and **WPA** and **TLS**.
- **Very efficient and simple**, but has serious **security flaws**



RC4 – an overview



(this is called a “**pseudo-random generation algorithm (PRGA)**”)

RC4

KSA

```
for i from 0 to 255
  S[i] := i
end
for j := 0 for i from 0 to 255
  j := (j + S[i] + key[i mod 256]) mod 256
  swap(S[i], S[j])
endfor
```

PRGA

```
i := 0
j := 0
while GeneratingOutput:
  i := (i + 1) mod 256
  j := (j + S[i]) mod 256
  swap(S[i], S[j])
  output S[(S[i] + S[j]) mod 256]
endwhile
```

don't read it!

Problems with RC4

1. Doesn't have a separate **IV**.
2. It was discovered that some bytes of the output are **biased**.
[Mantin, Shamir, 2001]
3. First few bytes of output sometimes **leak some information about the key**
[Fluhrer, Mantin and Shamir, 2001]
Recommendation: discard the first **768-3072** bytes.
4. Other weaknesses are also known...

Use of RC4 in WEP

- **WEP** = “Wired Equivalent Privacy”
- Introduced in **1999**, for a long time widely used to protect **WiFi** communication.
- How **RC4** is used:
 - to get the **seed**, the key **k** is **concatenated** with the **IV**
 - old versions: **$|k| = 40$ bits, $|IV| = 24$ bits**
(artificially weak because of the **US export restrictions**)
 - new versions: **$|k| = 104$ bits, $|IV| = 24$ bits.**

RC4 in WEP – problems with the key length

- **|k| = 40 bits** is **not enough**:
can be cracked using a **brute-force attack**
- **IV** is changed for each packet.
Hence **|IV| = 24 bits** is also not enough:
 - assume that each packet has length **1500 bytes**,
 - with **5Mbps** bandwidth the set of all possible **IVs** will be exhausted in half a day
- Some implementations reset **IV := 0** after each restart – this makes things even worse.

see **Nikita Borisov, Ian Goldberg, David Wagner (2001). "Intercepting Mobile Communications: The Insecurity of 802.11"**

RC4 in WEP – the weak IVs

[Fluhrer, Mantin and Shamir, 2001]

(we mentioned this attack already)

For so-called “**weak IVs**” the key stream **reveals some information about the key.**

In response the vendors started to “filter” the **weak IVs**.

But then **new weak IVs were discovered.**

[see e.g. Bittau, Handley, Lackey *The final nail in WEP's coffin.*]

Another recent attack

“RC4 NOMORE” (www.rc4nomore.com)

[Vanhoeft and Piessens, USENIX Security 2015]

Conclusion: RC4 should not be used in real life.

RC4 forbidden in TLS

Internet Engineering Task Force (IETF)

Request for Comments: 7465

Updates: [5246](#), [4346](#), [2246](#)

Category: Standards Track

ISSN: 2070-1721

A. Popov

Microsoft Corp.

February 2015

Prohibiting RC4 Cipher Suites

Abstract

This document requires that Transport Layer Security (TLS) clients and servers never negotiate the use of RC4 cipher suites when they establish connections. This applies to all TLS versions. This document updates RFCs 5246, 4346, and 2246.

Competitions for new stream ciphers

- **NESSIE** (New European Schemes for Signatures, Integrity and Encryption, 2000 – 2003) project **failed to select** a new stream cipher (all 6 candidates were broken)
(where “*broken*” can mean e.g. that one can distinguish the output from random after seeing **2^{36}** bytes of output)
- **eStream** project (November 2004 – May 2008) chosen a portfolio of ciphers: **HC-128, Grain v1, Rabbit, MICKEY v2, Salsa20/12, Trivium, SOSEMANUK.**

Salsa 20

One of the winners of the **eStream** competition.

Author: Dan Bernstein.

Very efficient both in **hardware** and in **software**.

key k
(size: **256** bits)

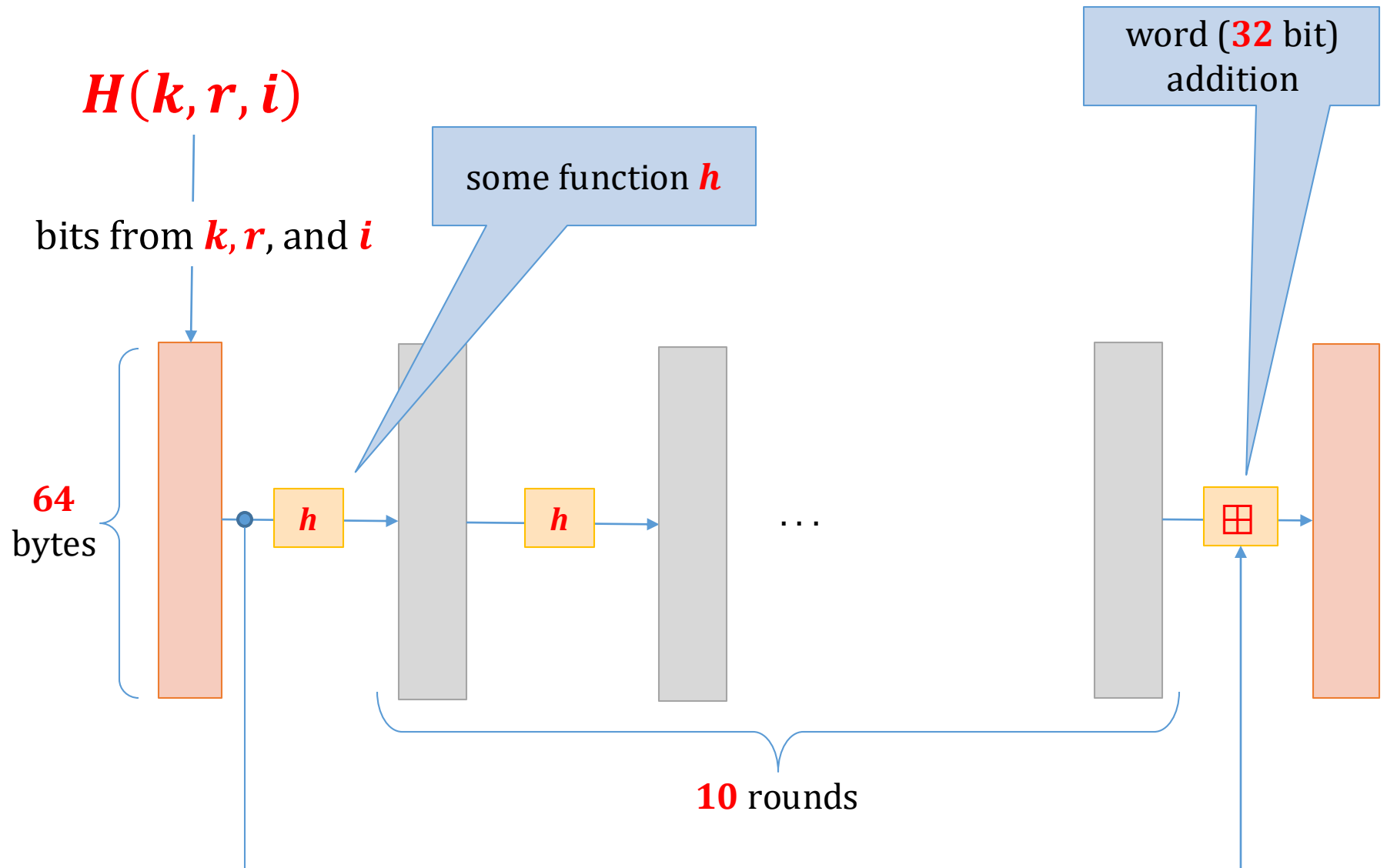
$$\text{Salsa20}(k, r) := H(k, r, 0) || H(k, r, 1) || \dots$$

nonce r
(size: **64** bits)

Note:

- **parallelizable**
- allows “**random access**”

How is H defined?



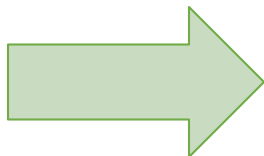
Is there an alternative to the
stream ciphers?

Yes!

the **block ciphers**

Plan

1. Computational definitions of security
2. If semantically-secure encryption exists, then **P \neq NP**
3. A proof that “the PRGs imply secure encryption”
4. Theoretical constructions of PRGs
5. Stream ciphers
6. Pseudorandom functions and permutations
7. Block ciphers
8. Practical considerations



Consider the following (naïve) settings

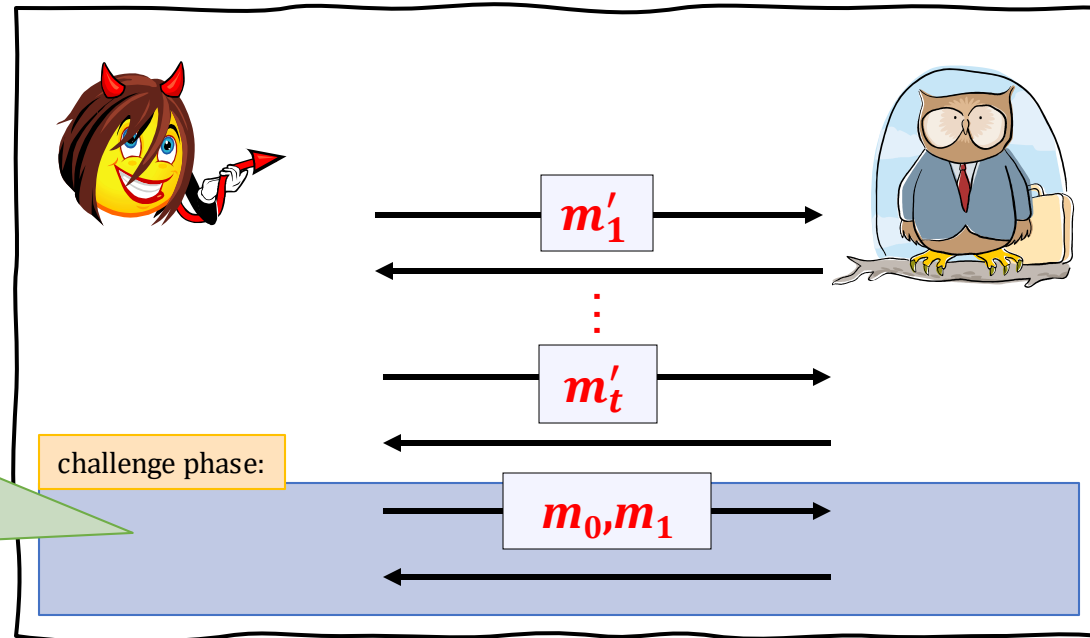
(1)

$$\mathcal{M} = \mathcal{C} = \{0, 1\}^n$$

ciphertexts and messages are just “blocks” of size n .

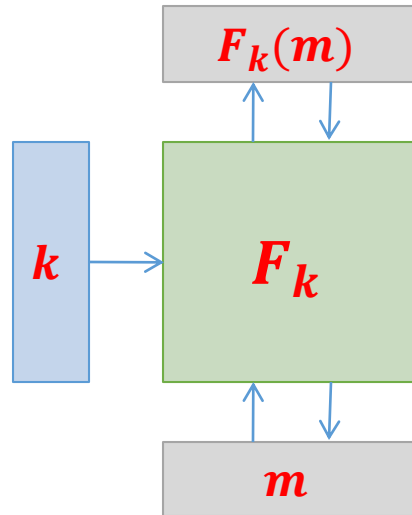
(2)

m_0, m_1 have to be fresh:
 $\{m_0, m_1\} \cap \{m'_1, \dots, m'_t\} = \emptyset$



In this case:

An encryption scheme is just “a **deterministic box that encrypts**”.
Or better: “a **deterministic box that behaves randomly**”.

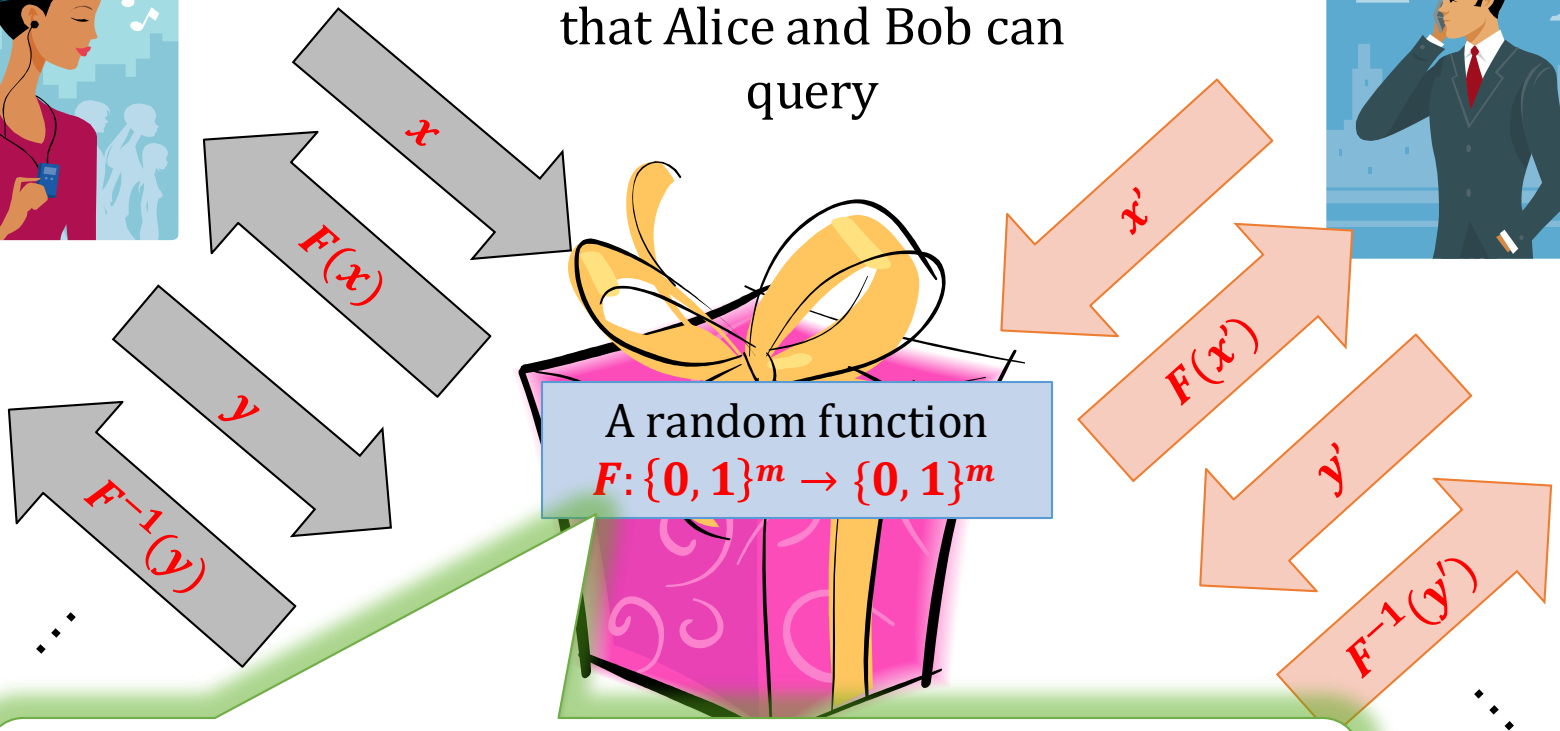


Such boxes have a name:

block ciphers \approx **pseudorandom permutations**

Random permutations

A box with a “random function”
that Alice and Bob can query

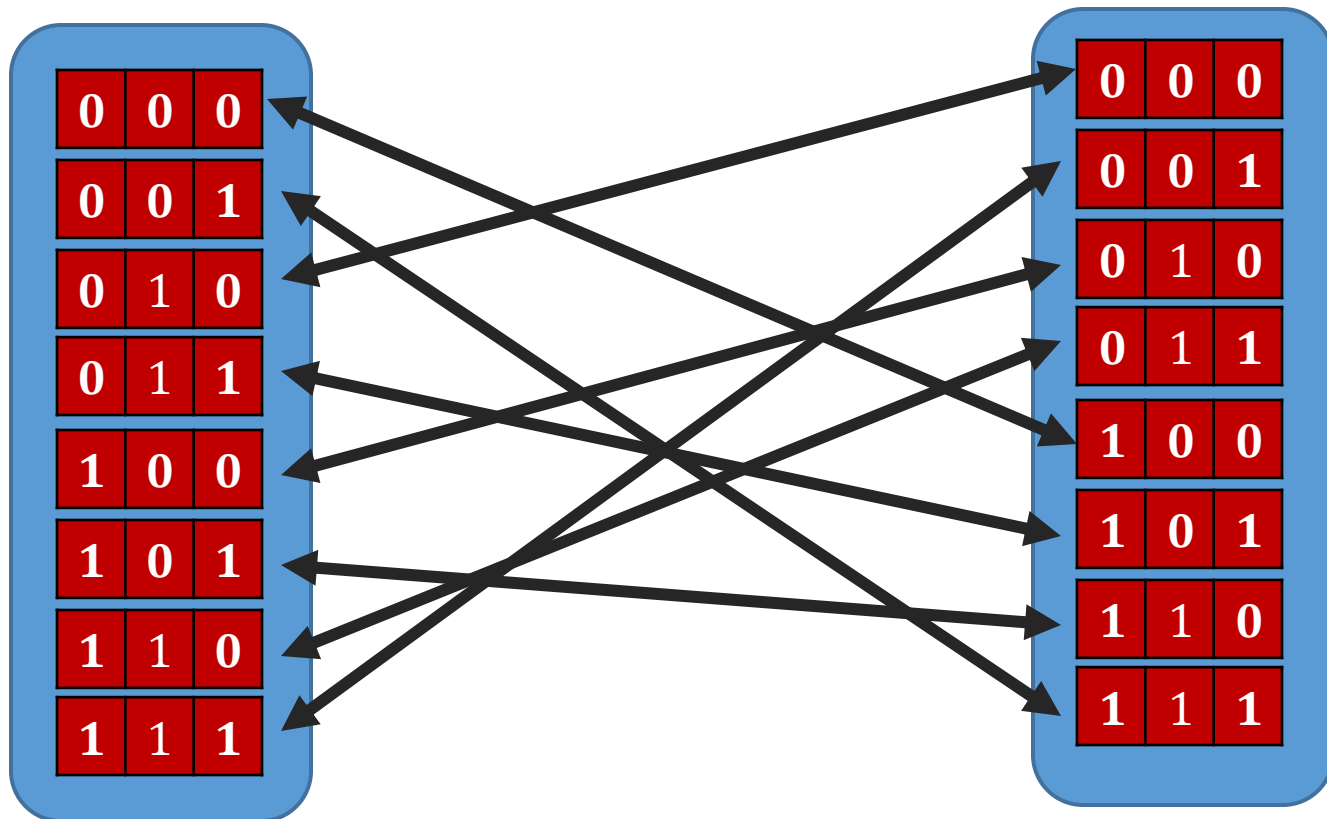


suppose F is a bijection
In other words: it is a **permutation on** $\{0, 1\}^m$

Note

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

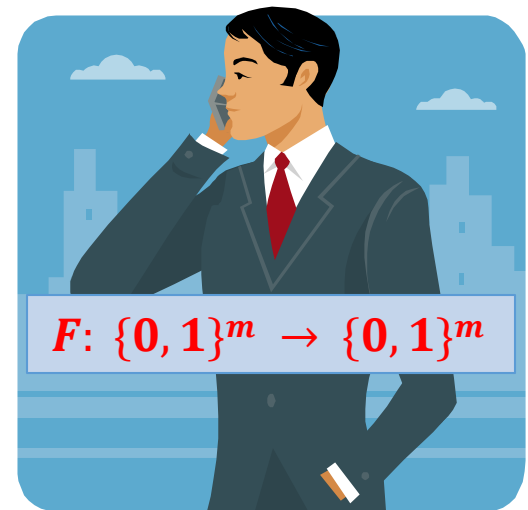
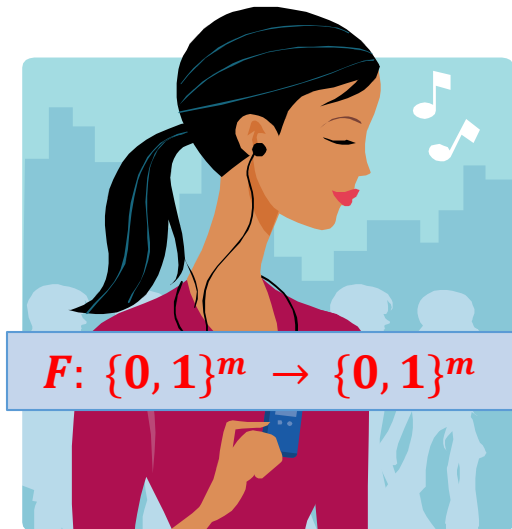
Example:



Can this box be simulated in real life?

Naive solution:

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



Problem:

The number of possible permutations is $(2^m)!$

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:

YES, it is possible! (under certain assumptions)

objects like these are called

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

Keyed permutations

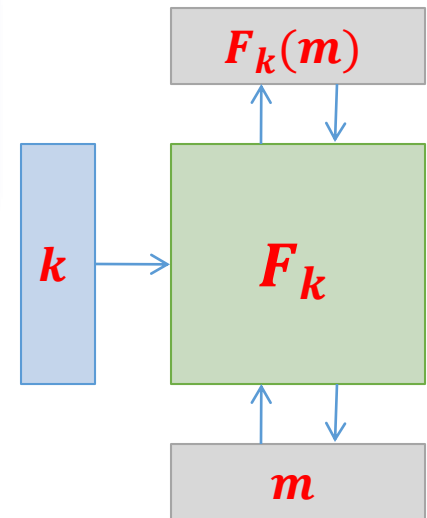
For a partial function

$F: \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^*$
let $F_k(m)$ denote $F(k, m)$.

A **keyed-permutation** is a function

$F: \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ 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

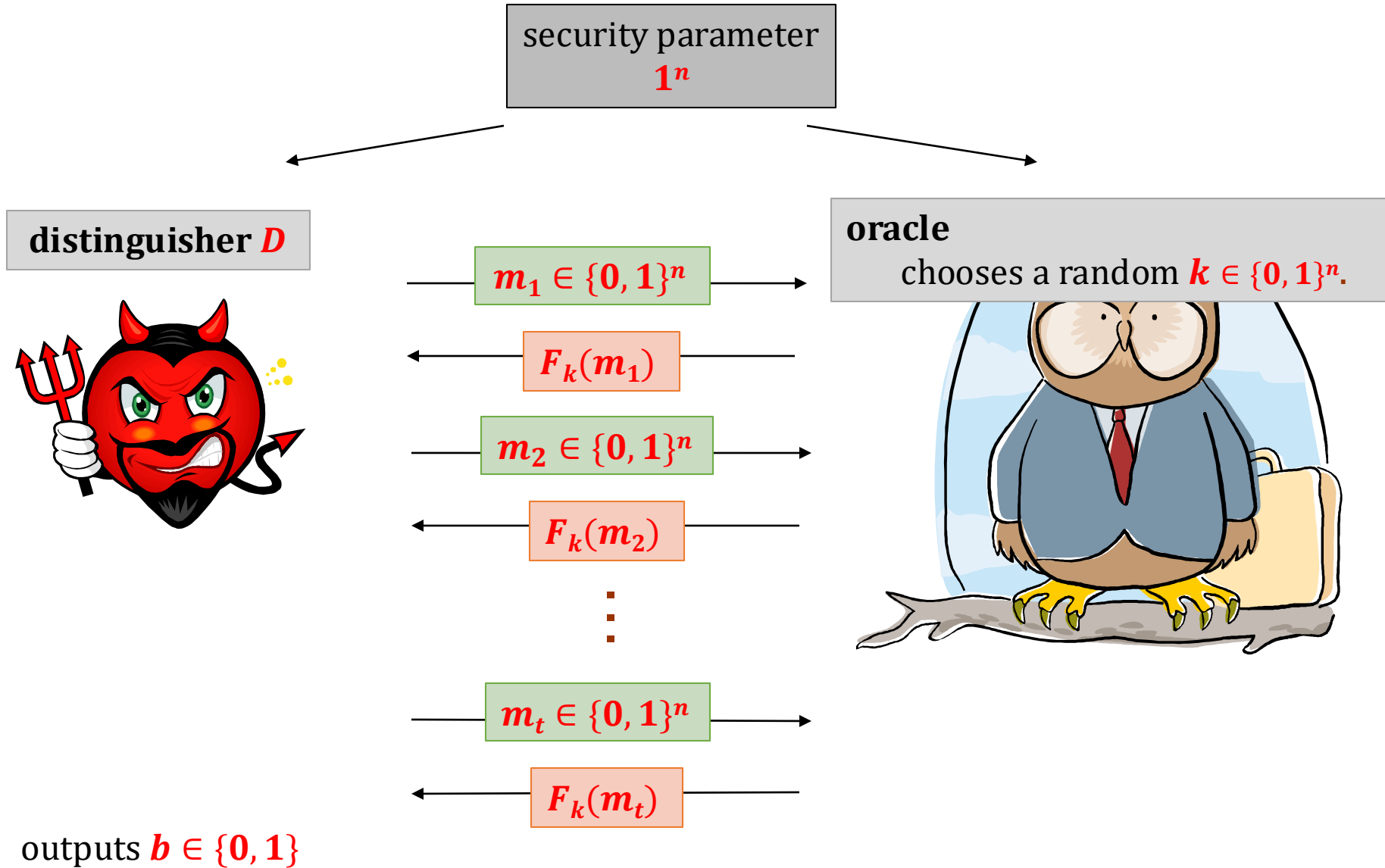
Intuition:

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

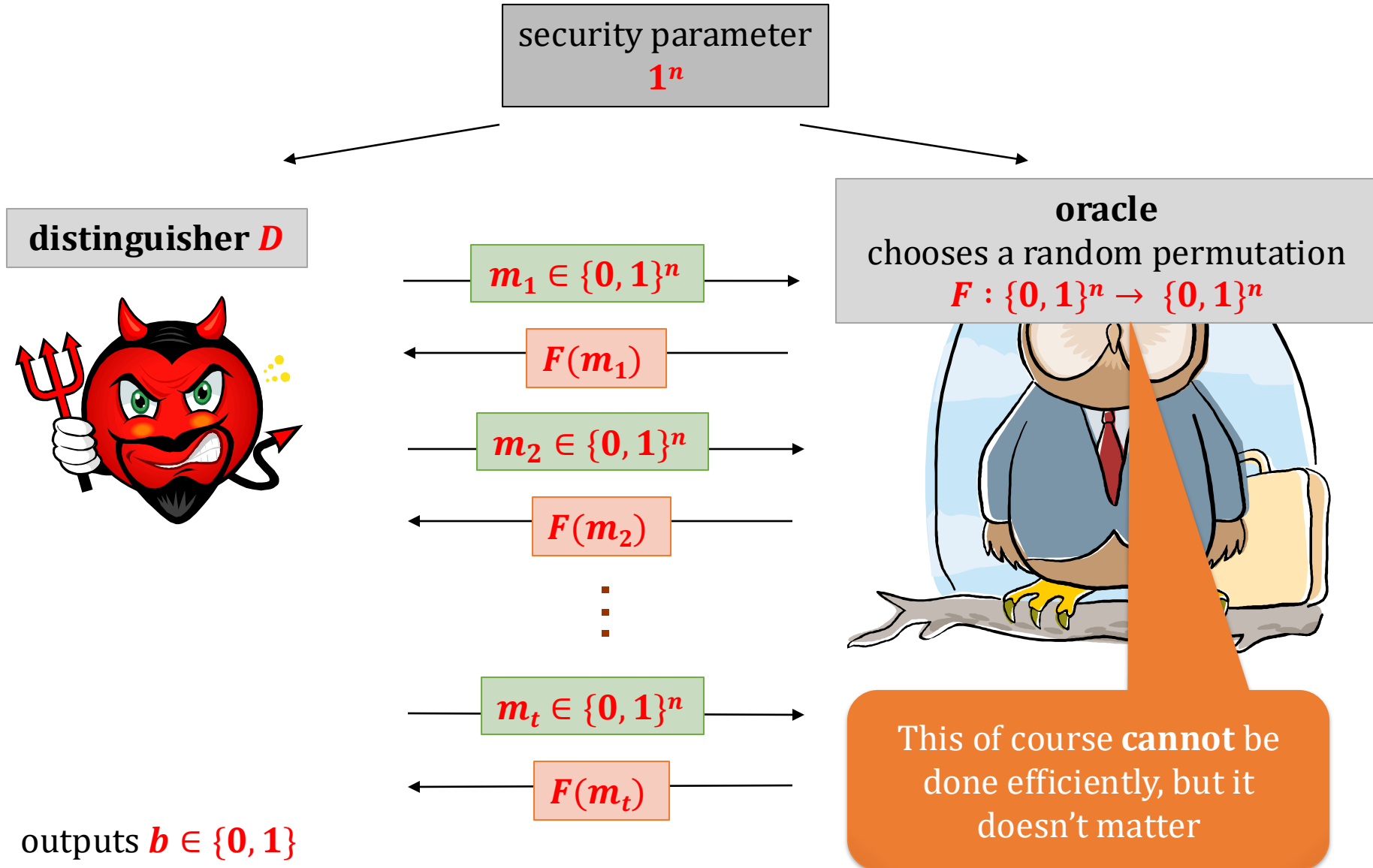


This has to be formalized

Scenario 0



Scenario 1



Pseudorandom permutations – the definition

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

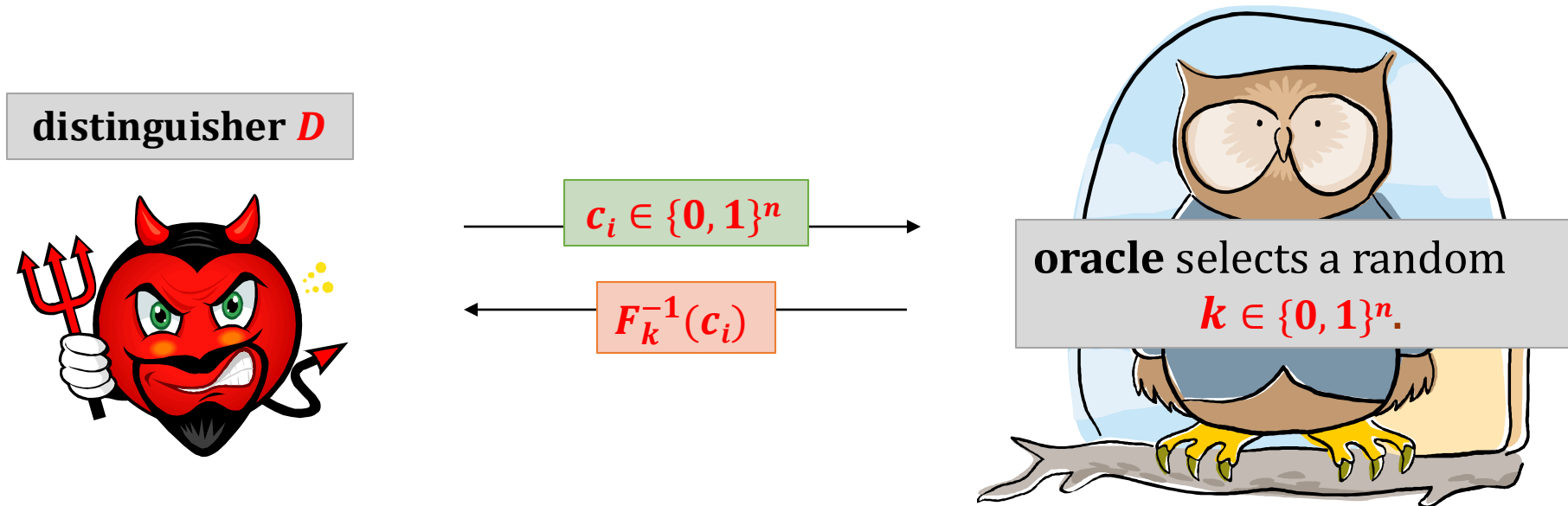
any **polynomial-time randomized distinguisher** D cannot distinguish **scenario 0** from **scenario 1** with a **non-negligible advantage**.

That is:

$|P(D \text{ outputs "1" in scenario 0}) - P(D \text{ outputs "1" in scenario 1})|$
is negligible in n .

Strong pseudorandom permutations

Suppose we allow the distinguisher to **additionally** ask the oracle for inverting F :



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.

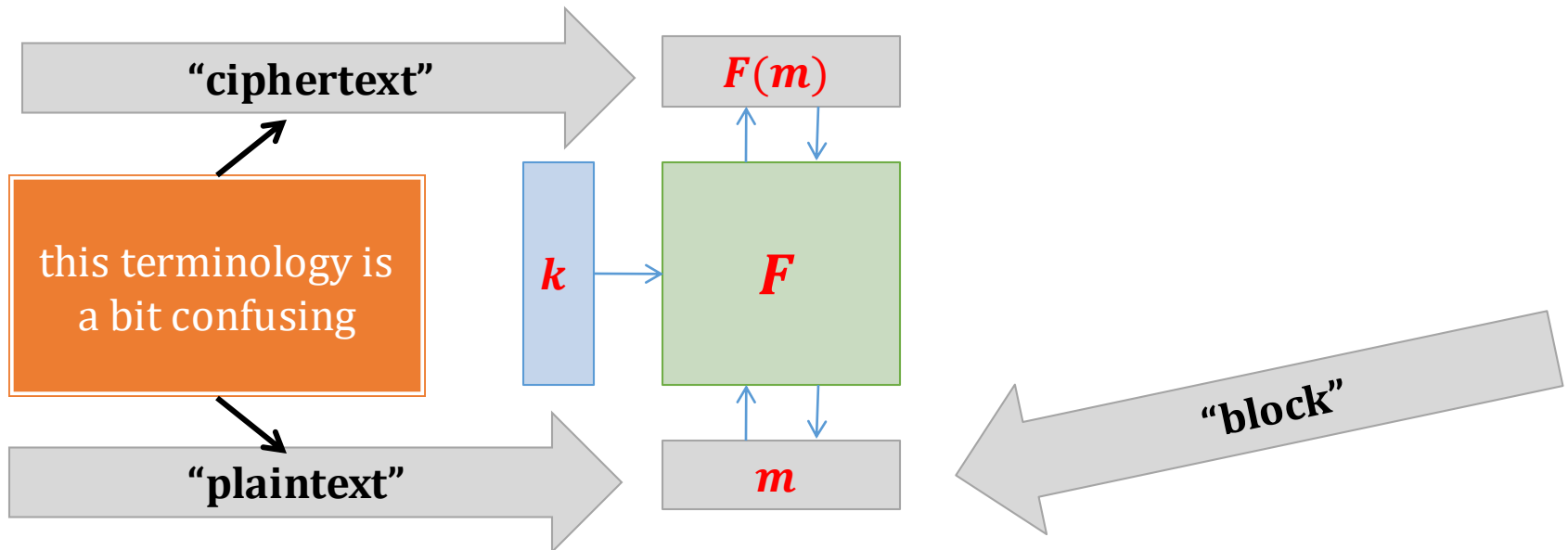
Terminology

Before we had:

stream ciphers \approx **pseudorandom generators**

Similarly:

block ciphers \approx **pseudorandom permutations**



Another way to look at the stream ciphers :

m is a parameter



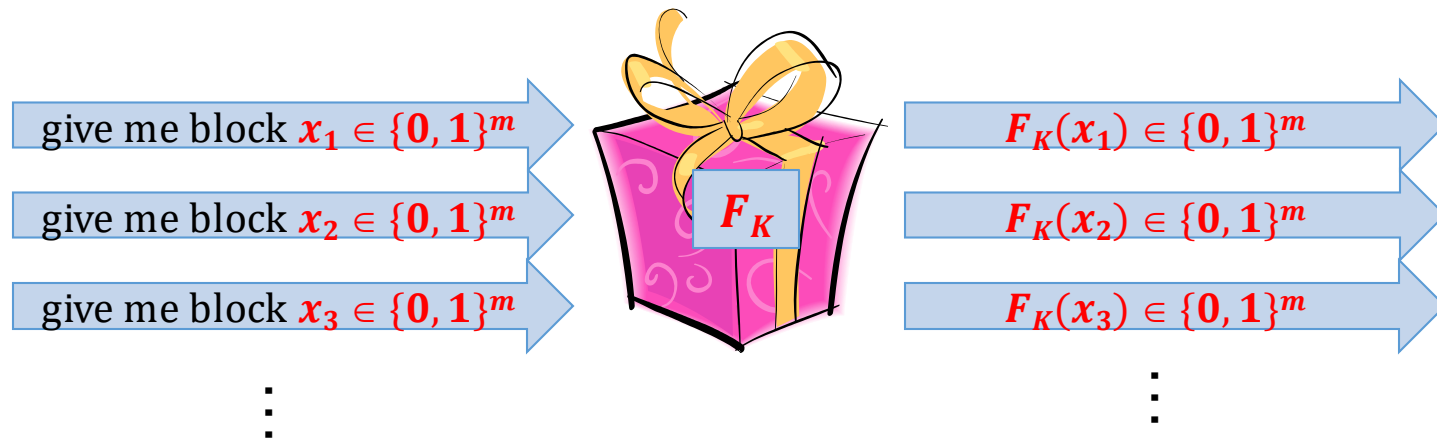
Requirement:

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

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

Block ciphers:

m is a parameter

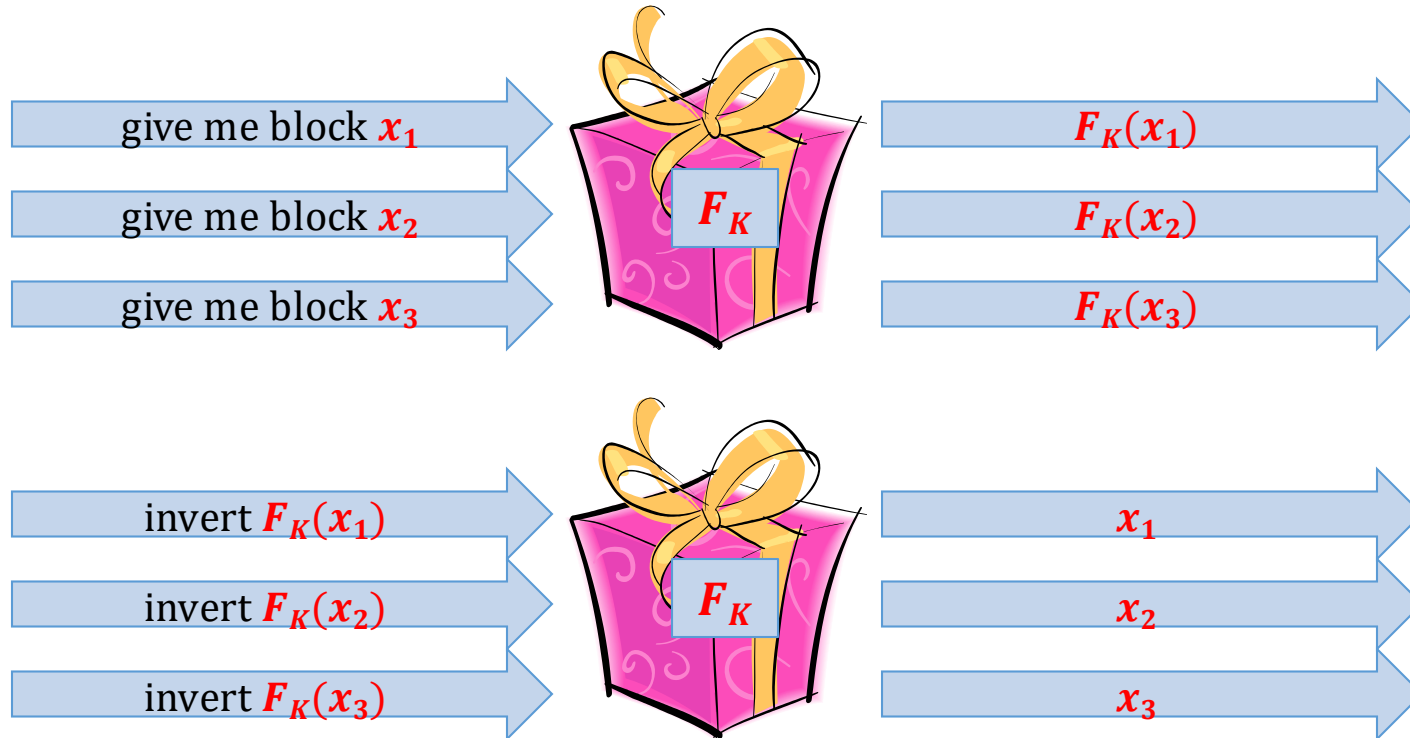


Requirement:

for x_1, x_2, x_3, \dots chosen adversarially

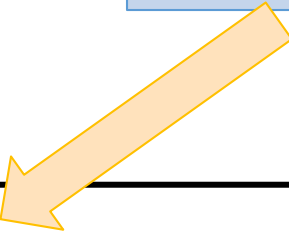
$F_K(x_1), F_K(x_2), F_K(x_3), \dots$
has to “look random” if K is random and secret.

An 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**.

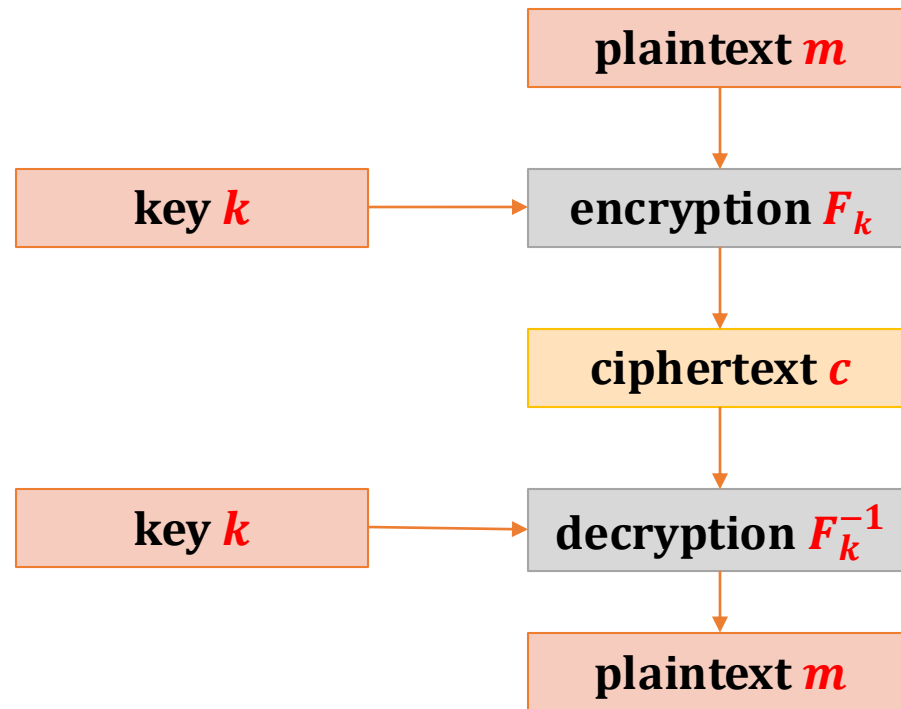


	key length	block length
DES (1976) (Data Encryption Standard)	56	64
IDEA (1991) (International Data Encryption Algorithm)	128	64
AES (1998) (Advanced Encryption Standard)	128, 192 or 256	128

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

How to encrypt using the block ciphers?

A naive (wrong) idea: 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. Computational definitions of security
2. If semantically-secure encryption exists, then **P \neq NP**
3. A proof that “the PRGs imply secure encryption”
4. Theoretical constructions of PRGs
5. Stream ciphers
6. Pseudorandom functions and permutations
7. Block ciphers
 1. Modes of operation
 2. Popular construction paradigms
 3. Cascade ciphers
8. Practical considerations



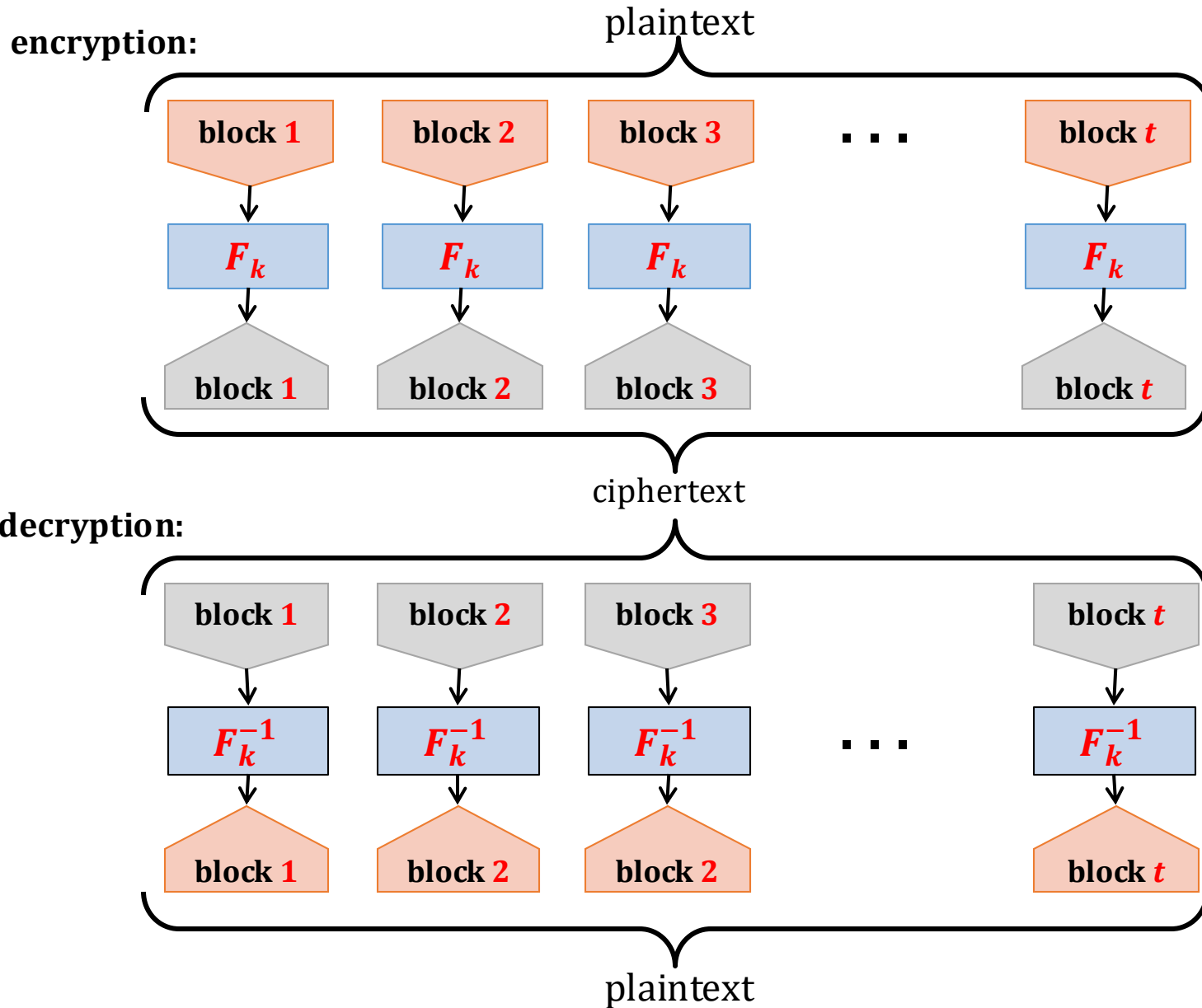
Block cipher modes of operation

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



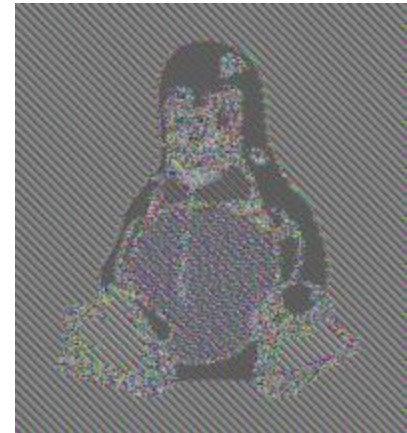
This mode was used in the past.

It is not secure, and should not be used.

Example:

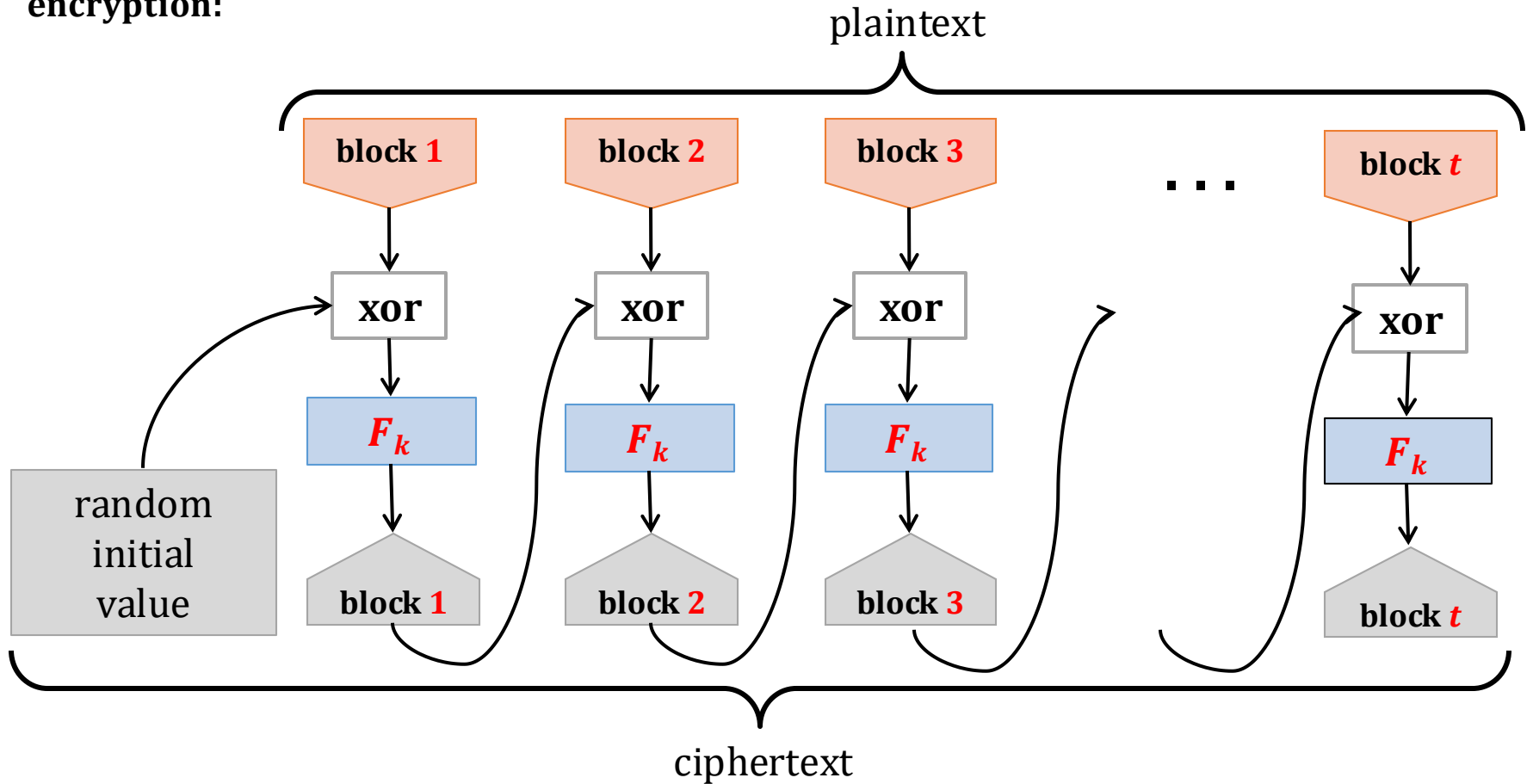


ECB



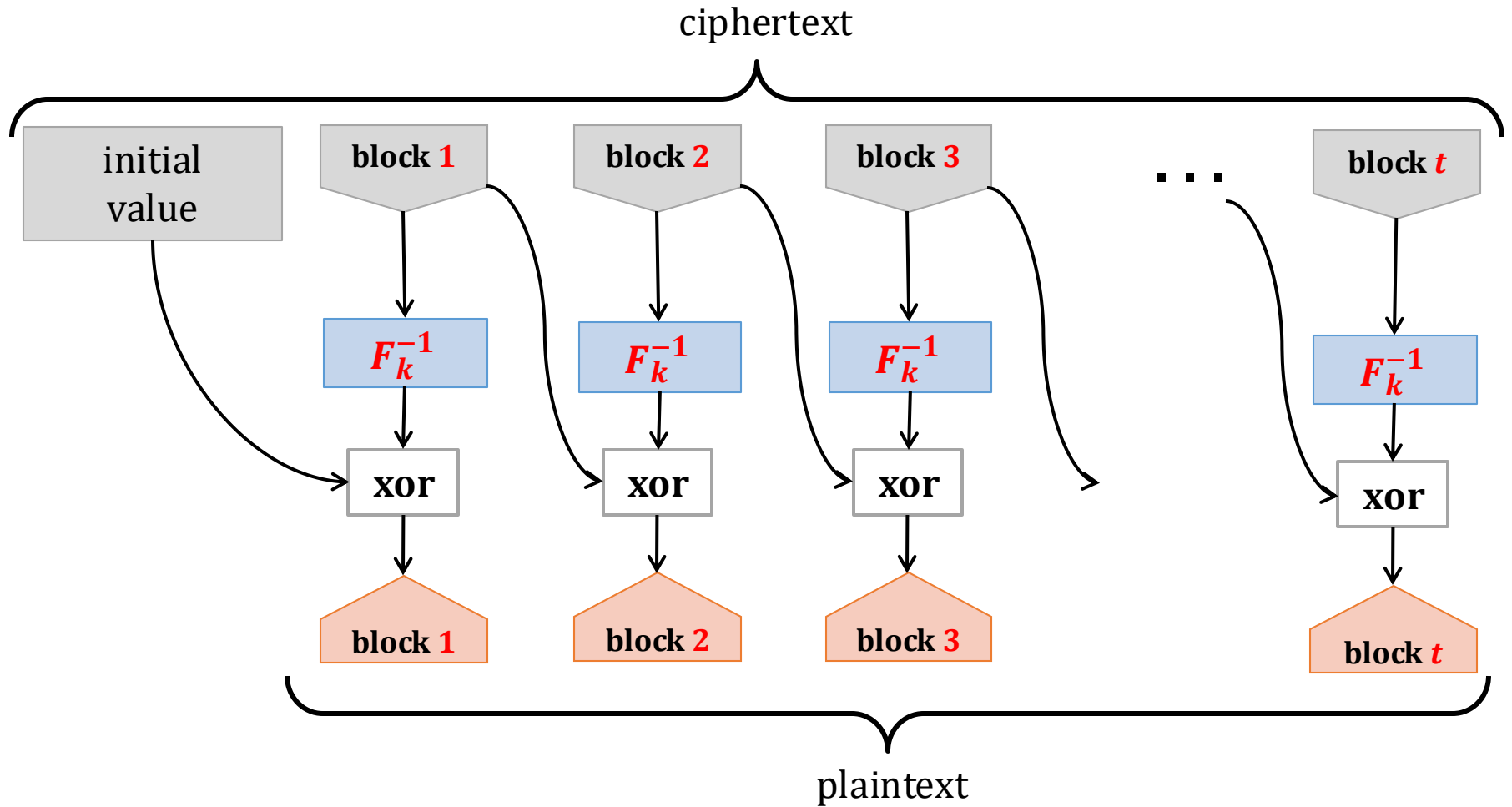
Cipher-Block Chaining (CBC)

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)



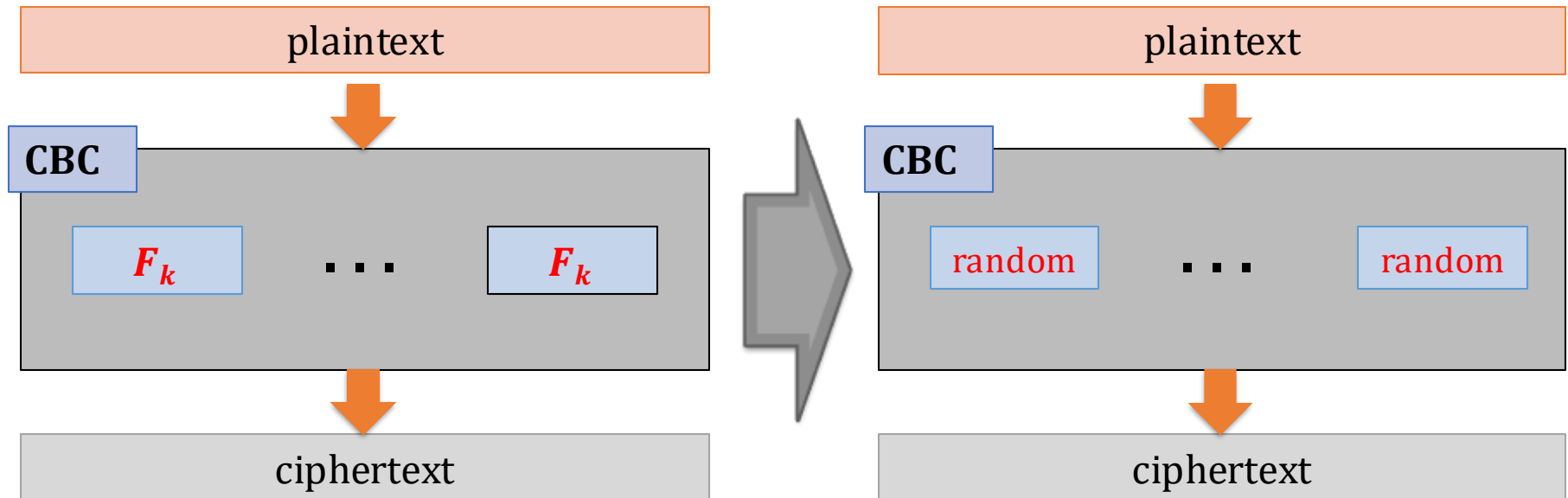
CBC mode is secure

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

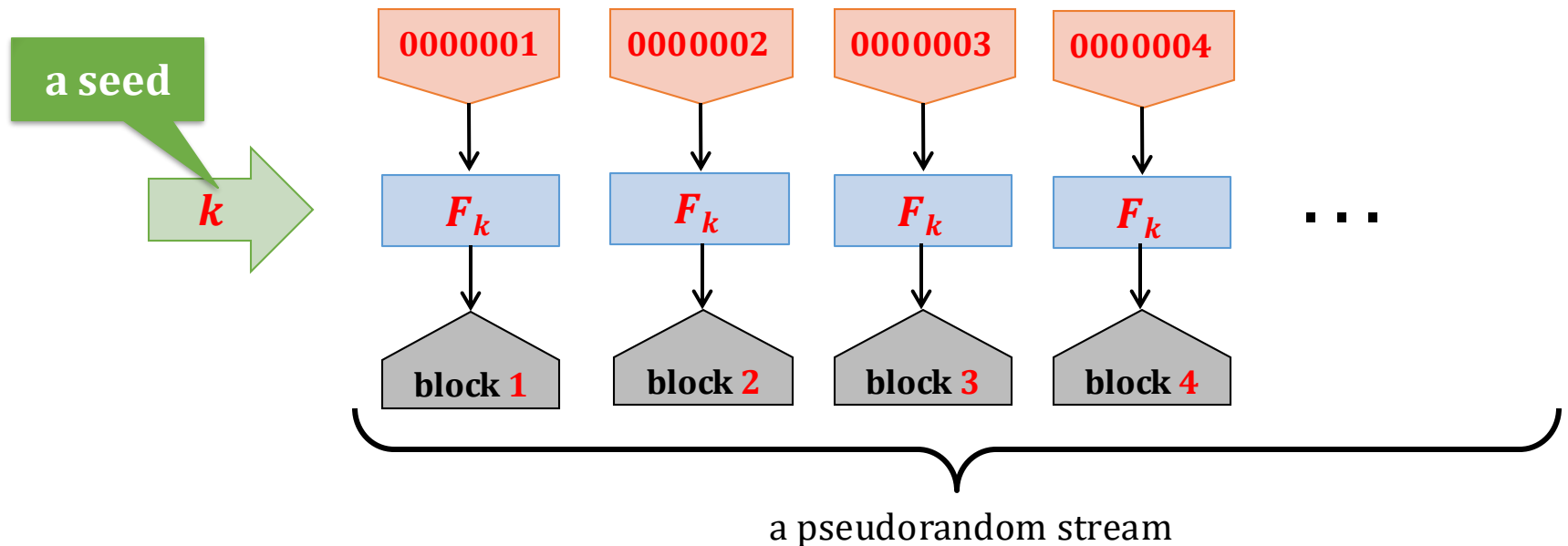
[M. Bellare, A. Desai, E. Jorjipii and P. Rogaway 1997]

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 distinguisher.)



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

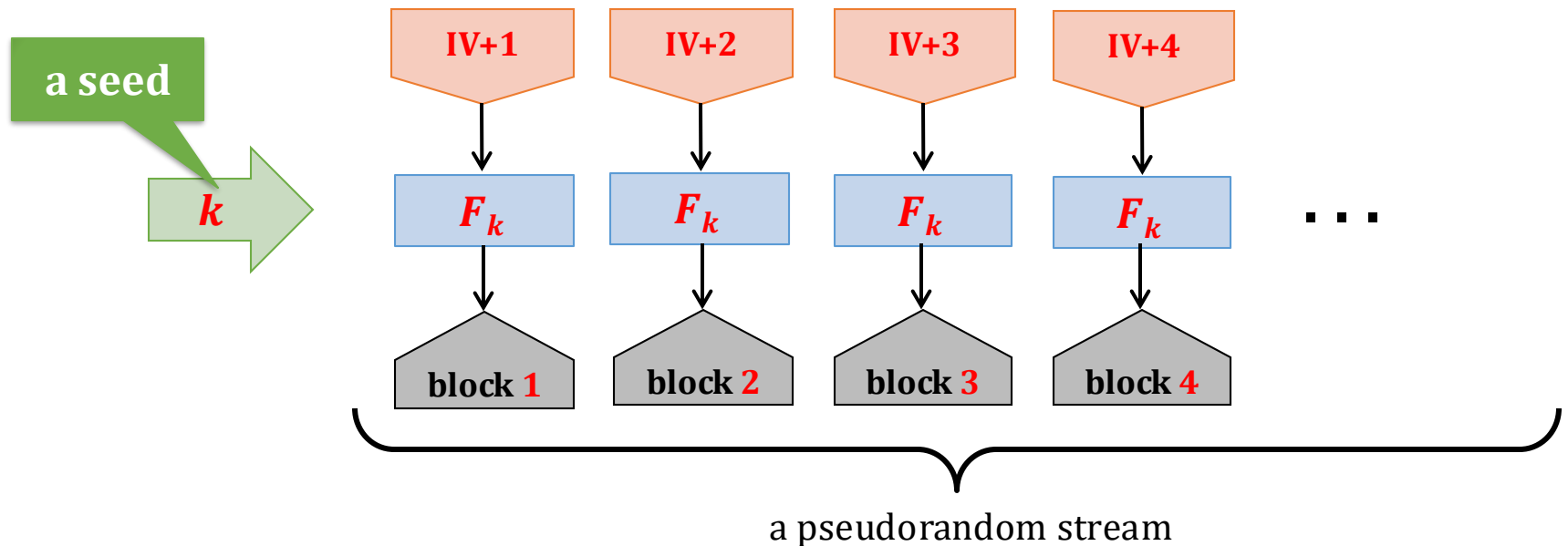


$$G(k) := F_k(1) \parallel F_k(2) \parallel F_k(3) \parallel \dots$$

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

How to “randomize” this?

take some random **IV**



$$G(k, IV) := F_k(IV + 1) || F_k(IV + 2) || F_k(IV + 3) || \dots$$

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 not self-synchronizing)



Can encryption be parallelized?

Yes



Can decryption be parallelized?

Yes

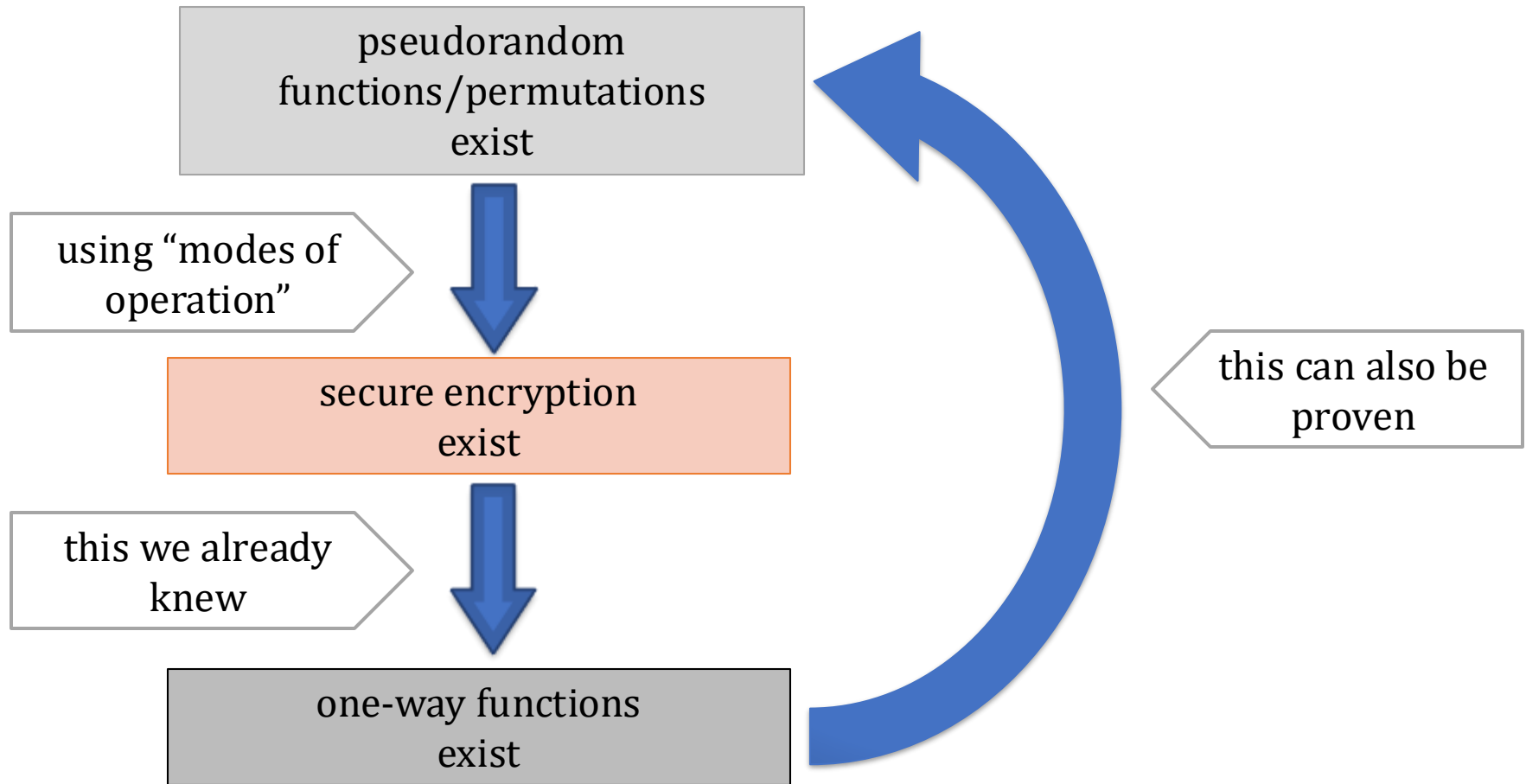


What if one bit of plaintext is changed?

Only one block needs to be recomputed

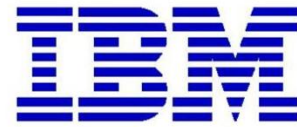


One more member of minicrypt!



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. Computational definitions of security
2. If semantically-secure encryption exists, then **P \neq NP**
3. A proof that “the PRGs imply secure encryption”
4. Theoretical constructions of PRGs
5. Stream ciphers
6. Pseudorandom functions and permutations
7. Block ciphers
 1. Modes of operation
 2. Popular construction paradigms
 3. Cascade ciphers
8. Practical considerations



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).

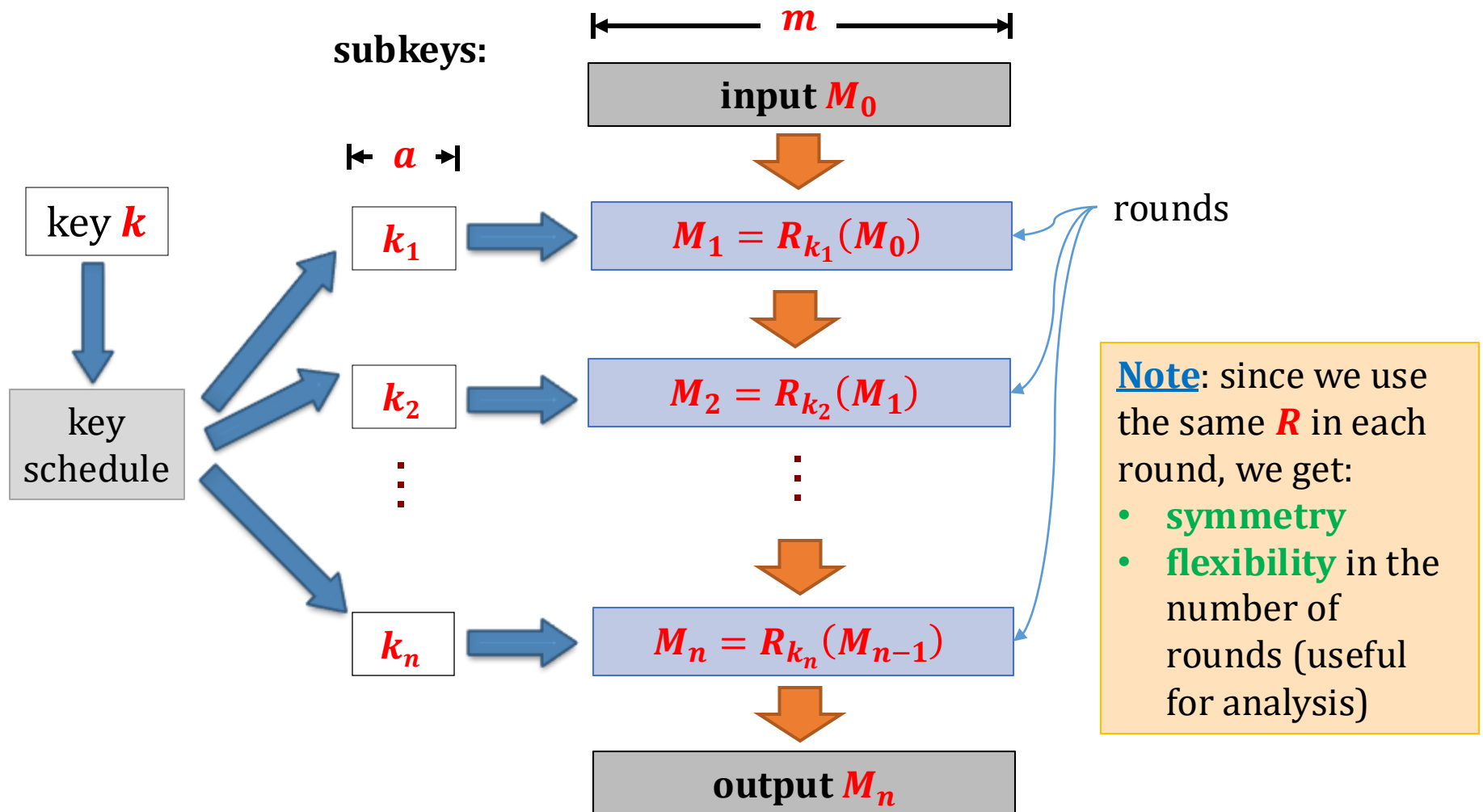
Block ciphers –advanced security requirements

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

A very popular paradigm: iterated ciphers

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

Typically we write the first argument in a subscript.



Popular types of iterated ciphers

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

Plan

1. Computational definitions of security
2. If semantically-secure encryption exists, then
P \neq NP
3. A proof that “the PRGs imply secure encryption”
4. Theoretical constructions of PRGs
5. Stream ciphers
6. Pseudorandom functions and permutations
7. Block ciphers
 1. Modes of operation
 2. Popular construction paradigms
 1. Feistel ciphers
 2. Substitution-permutation networks
 3. Cascade ciphers
8. Practical considerations



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,...

DES (Digital Encryption Standard)

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



History 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 too short (only **56** bits).
- Unclear role of **NSA** in the design
 - hidden **backdoor**?
 - **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 almost **5 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

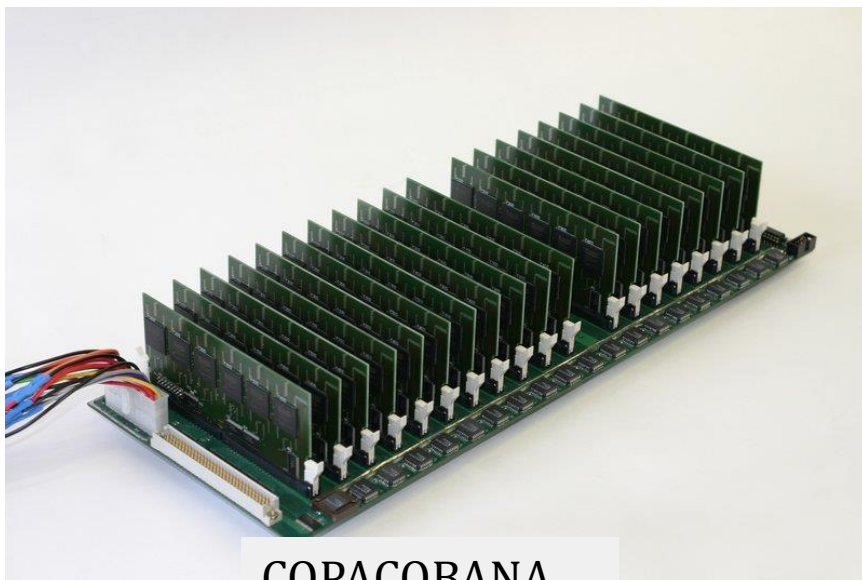
The **United States Senate Select Committee on Intelligence** (1978):

"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."

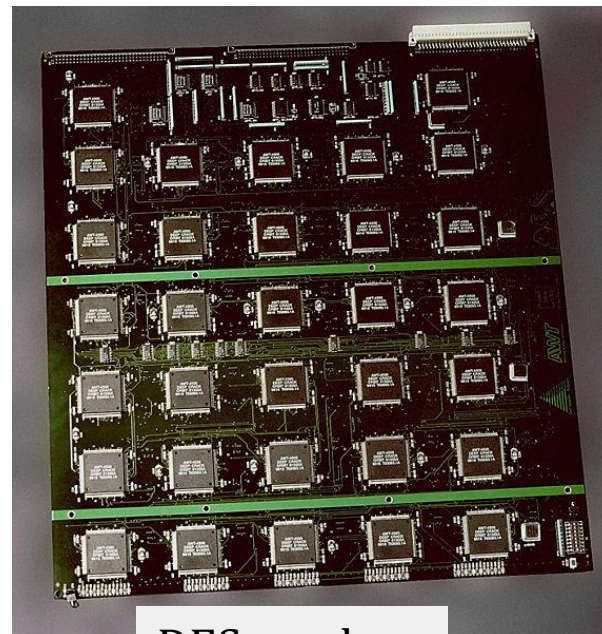
"**NSA** did not tamper with the design of the algorithm in any way. **IBM** invented and designed the algorithm, made all pertinent decisions regarding it, and concurred that the agreed upon key size was more than adequate for all commercial applications for which the **DES** was intended."

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**
DESHALL 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\$**
- **2000s**
COPACOBANA (the Cost-Optimized Parallel COde Breaker) breaks **DES** in **1 week** and costs **10,000\$**



COPACOBANA



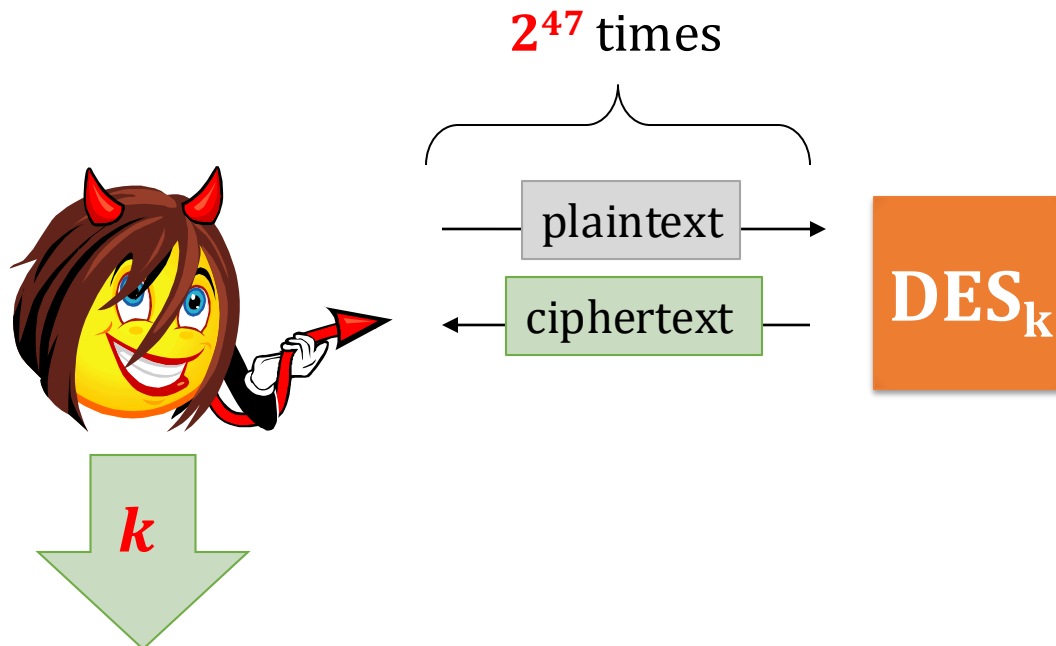
DES-cracker

Theoretical attacks on DES – differential cryptoanalysis

Biham and Shamir (late 1980s):

differential cryptoanalysis

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



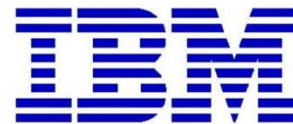
Not very practical...

Differential cryptoanalysis – an interesting observation

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

Moral

NSA and **IBM** knew it!





Don Coppersmith, IBM

"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."

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 cryptanalysis

Matsui (early 1990s):

linear cryptanalysis

uses a **known-plaintext attack**

2^{43} (plaintext, ciphertext) pairs

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

Let's now discuss in detail how **DES** is built.

Feistel network

subkeys:

k

key
schedule

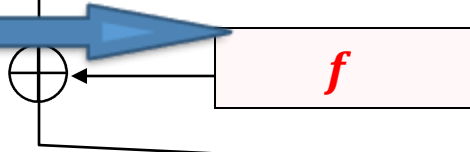
k_1

k_2

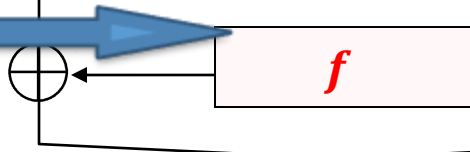
k_n

$m/2$ $m/2$

L_0 R_0



L_1 R_1



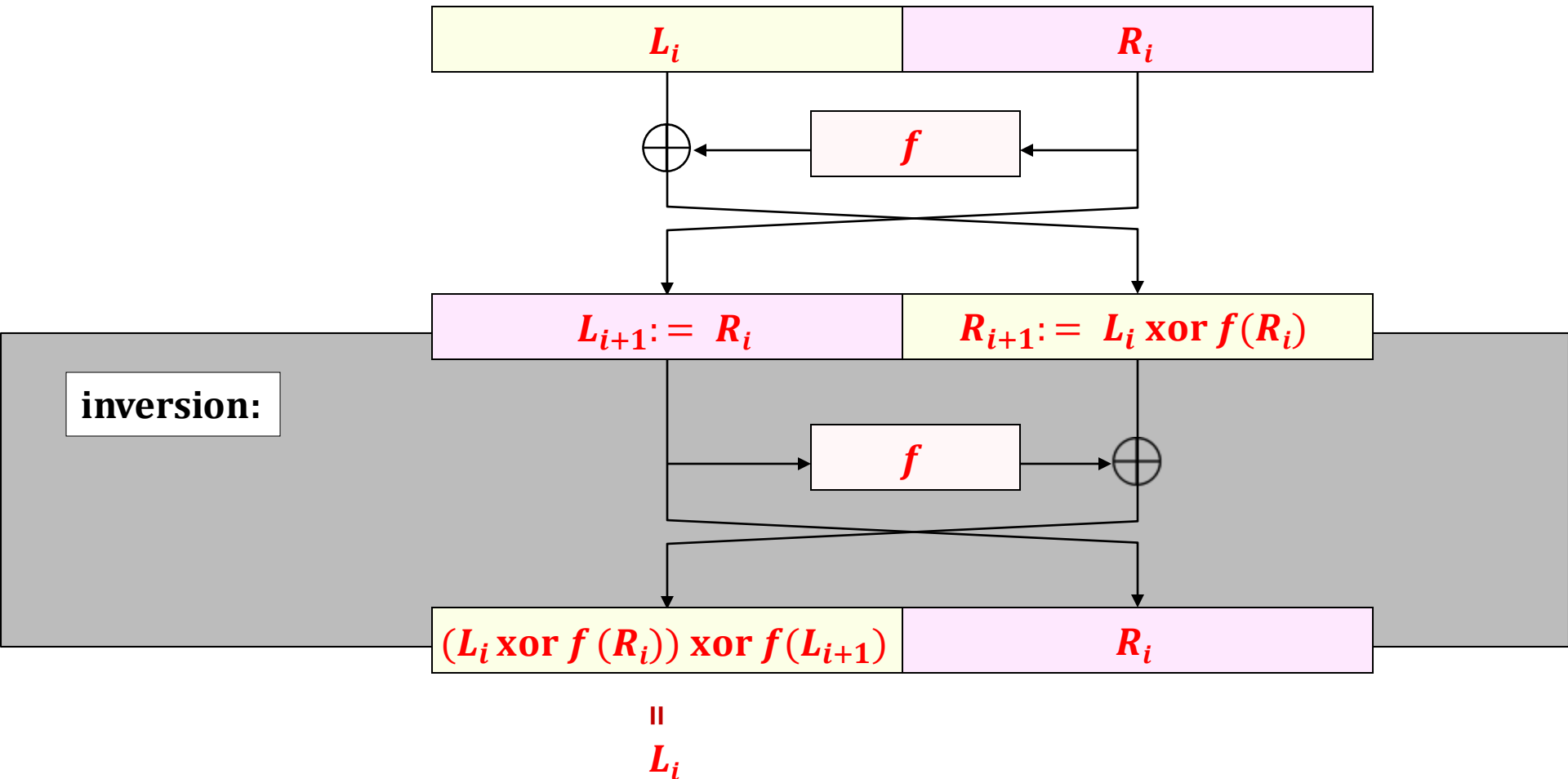
L_n R_n

n "Feistel rounds"

here no twist

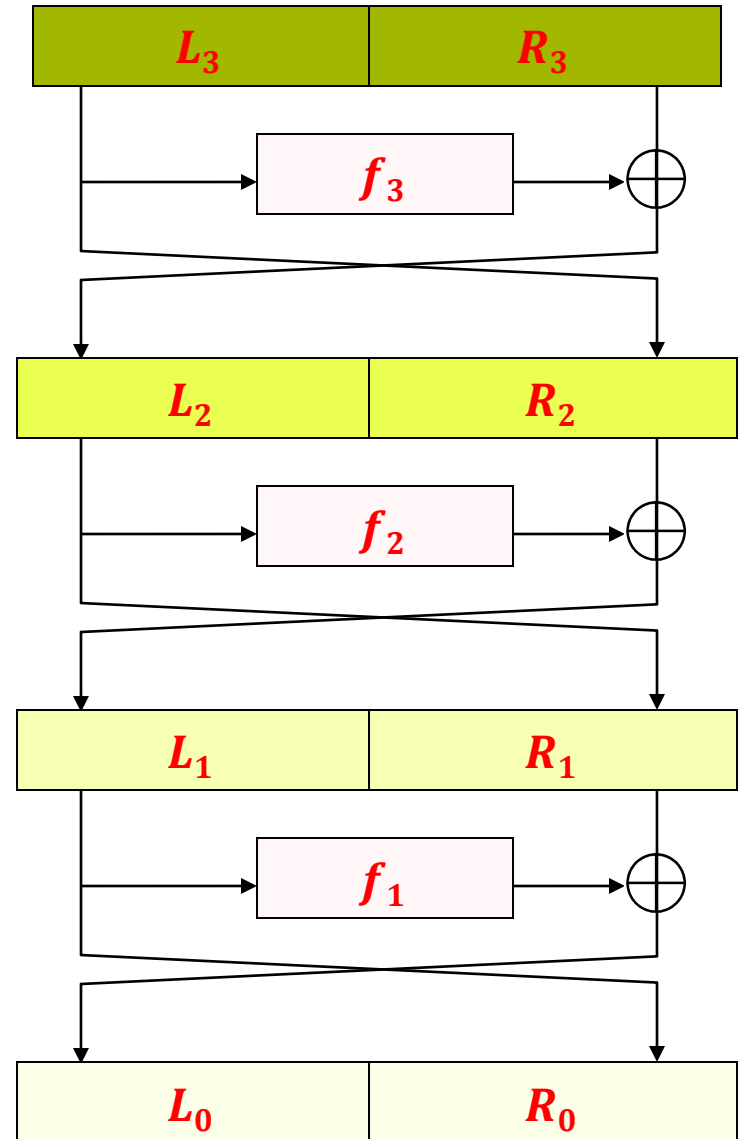
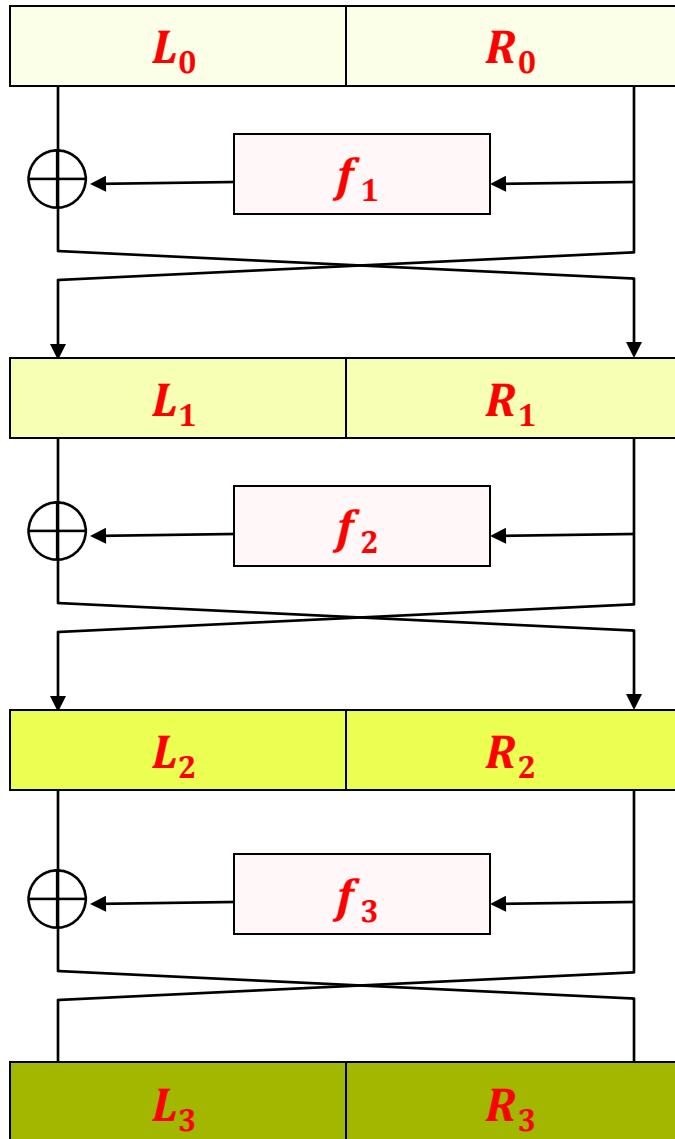
A nice property of Feistel rounds

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

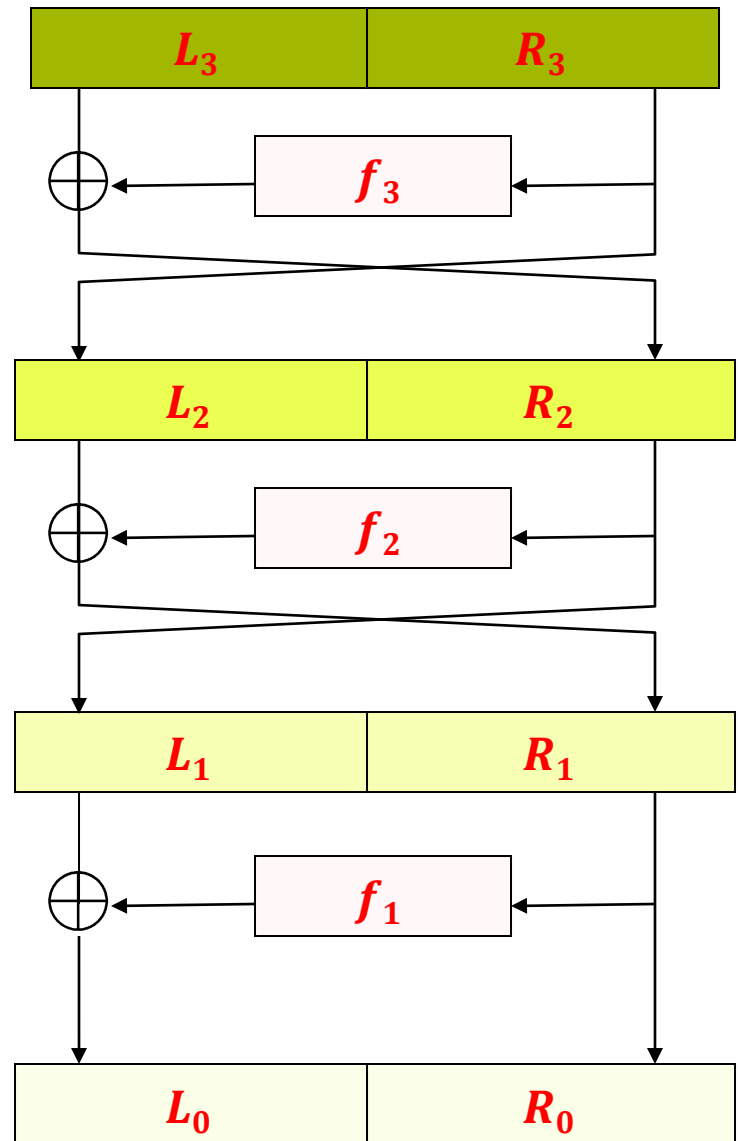
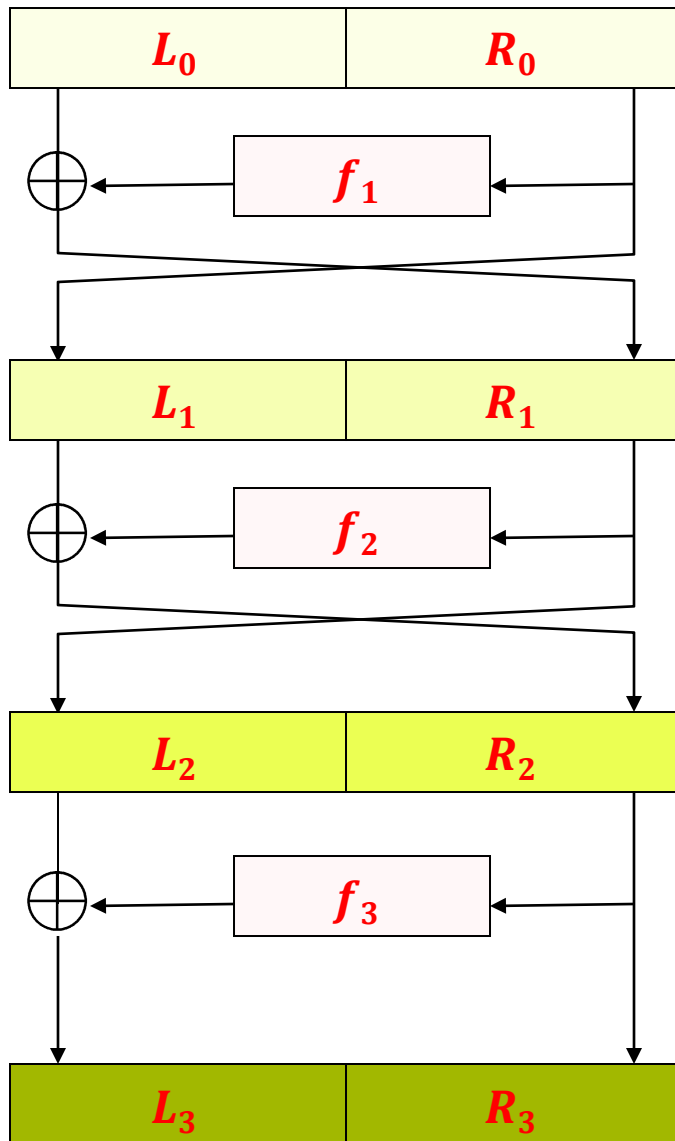


Hence: the Feistel network can be “inverted”!

Example: 3 round Feistel network

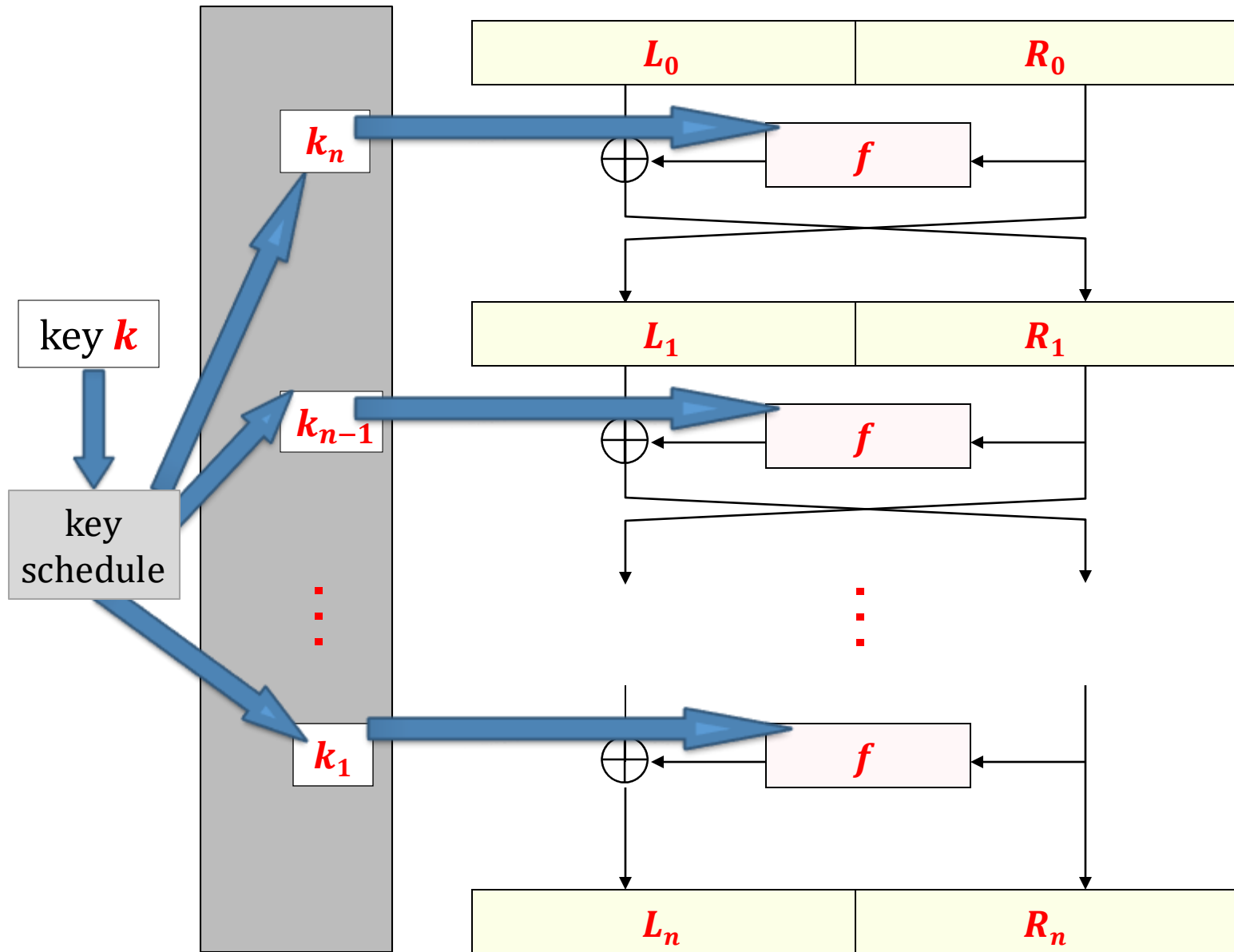


Without a “twist” in the last round:



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**.

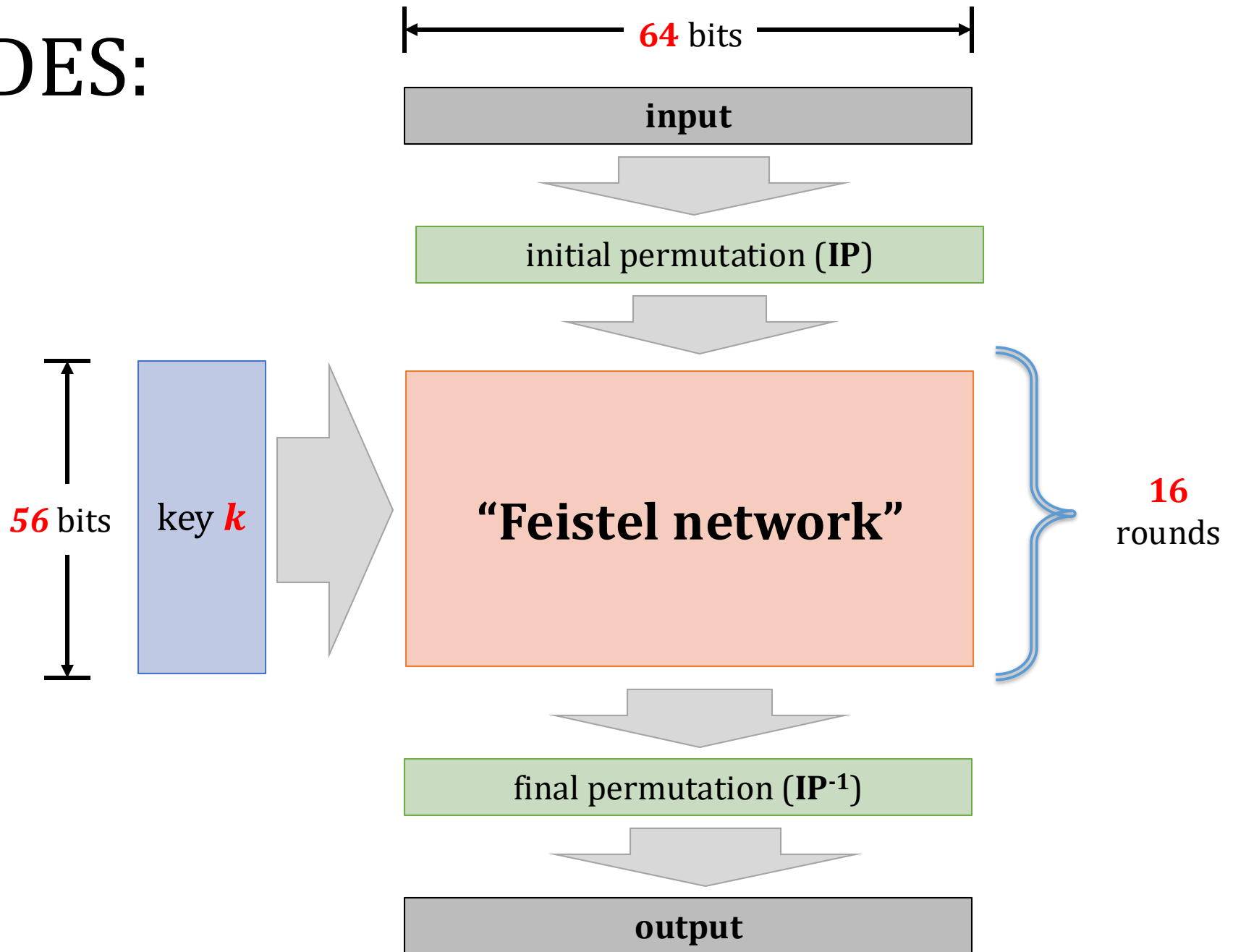
see **M. Luby and C. Rackoff. "How to Construct Pseudorandom Permutations and Pseudorandom Functions." In *SIAM J. Comput.*, vol. 17, 1988, pp. 373-386.**

How is the Feistel network used in DES?

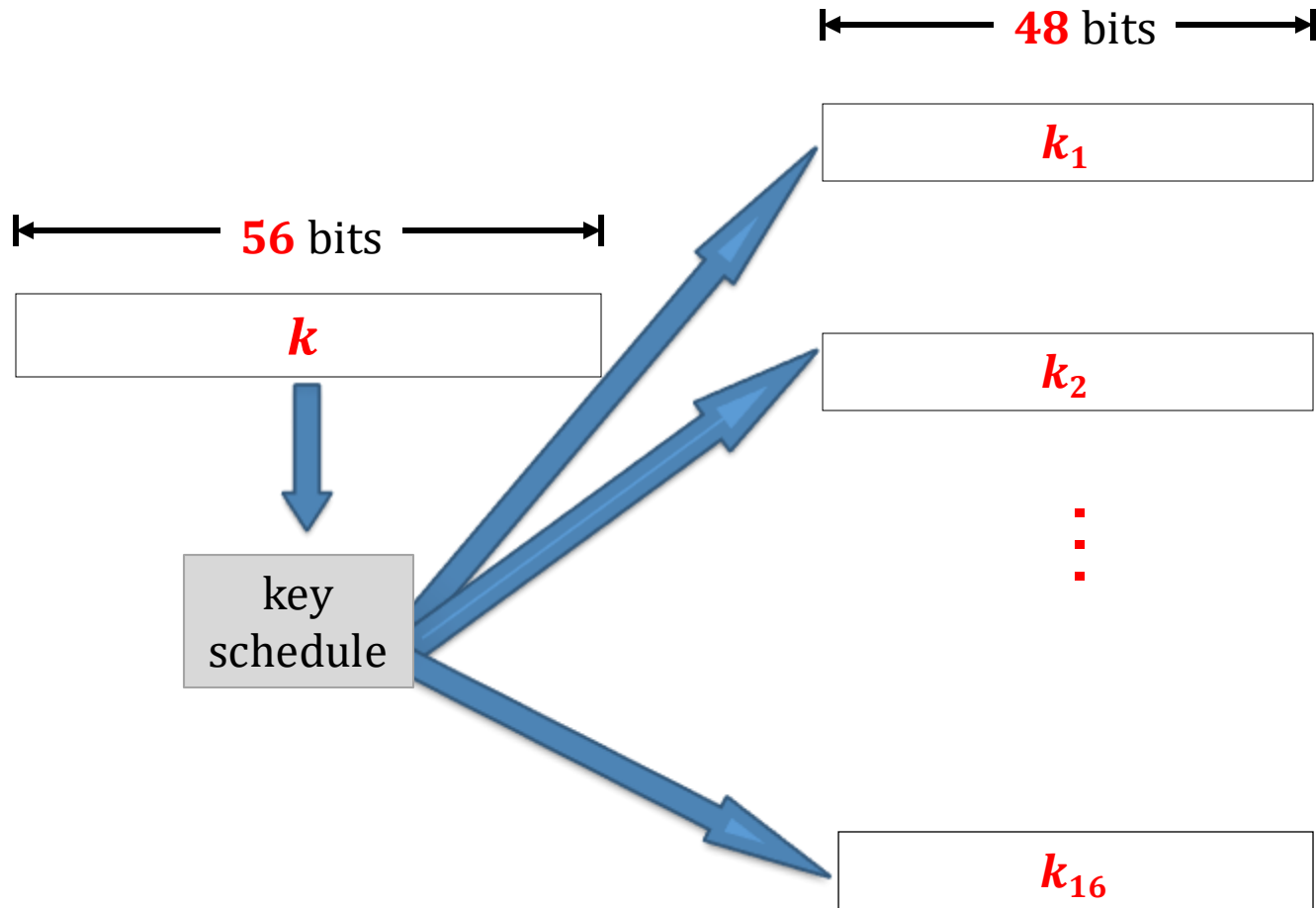
The following needs to be described:

1. The concrete parameters
2. The key schedule algorithm.
3. The functions *f*.

DES:

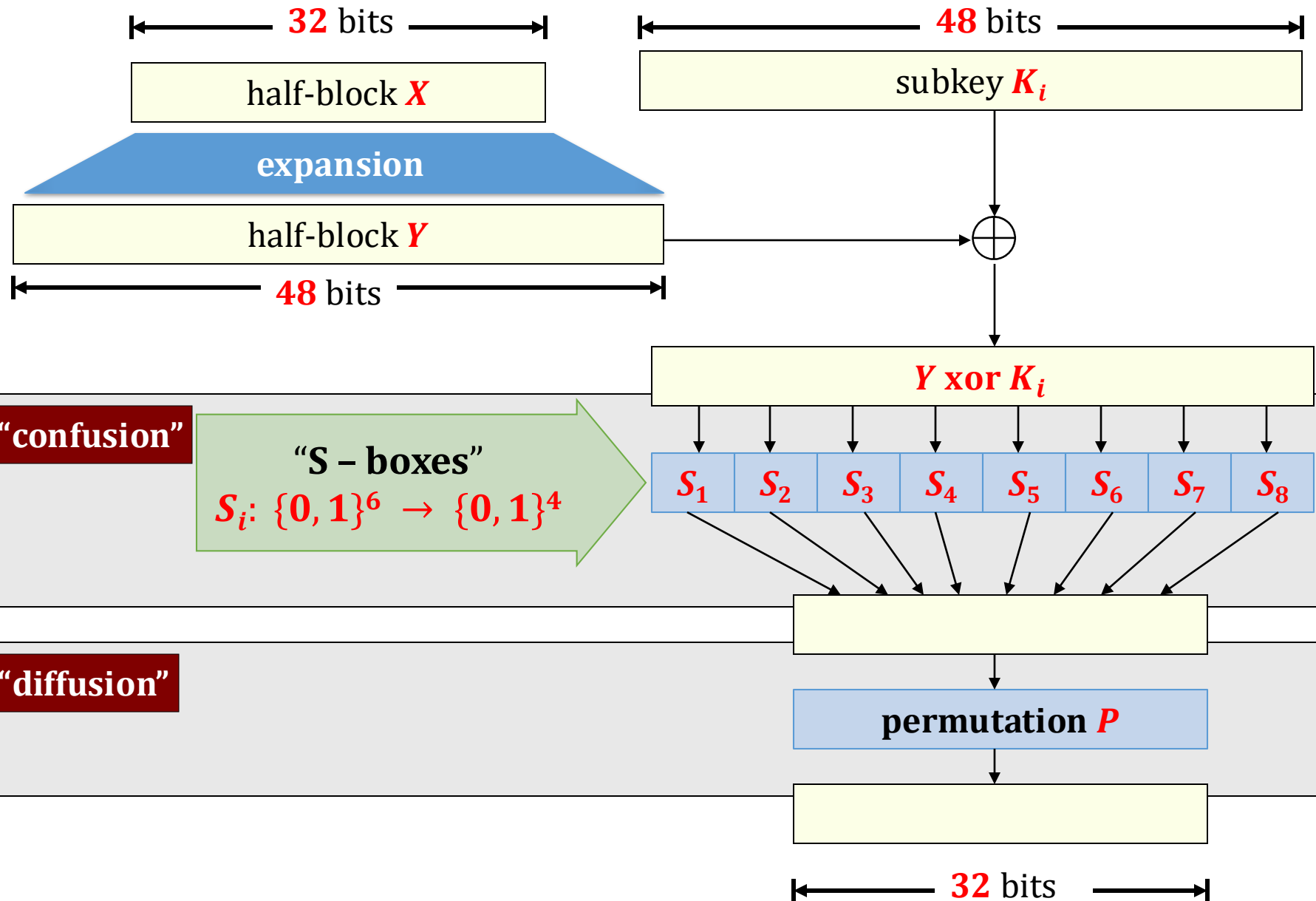


DES key schedule

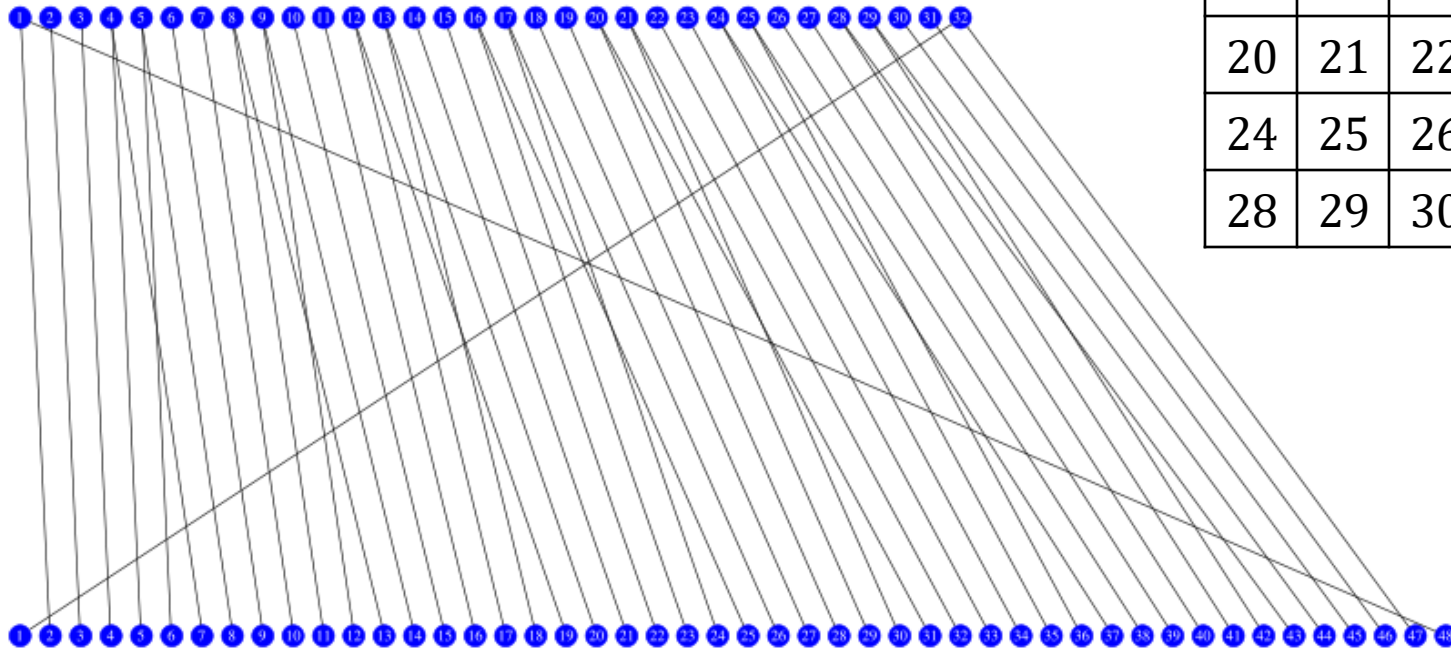


each subkey k_i consists of some bits of k (we skip the details)

function f :



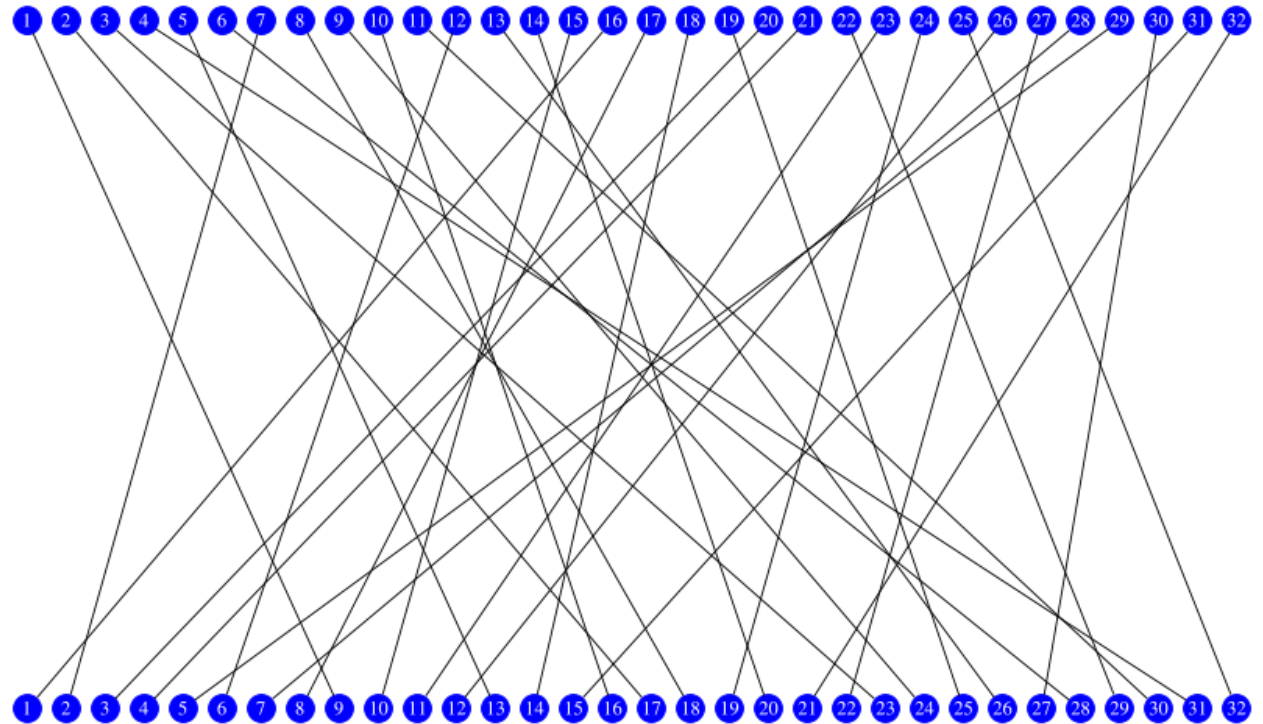
The expansion function



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 input bit is held constant.

Properties of S-boxes [2/2]

- For any S-box S , it holds that

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

differ in at least two bits.

- For any S-box S , it holds that

$$S[x] \neq S[x \oplus 11rs00]$$

for any binary values r and 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 that for any nonzero 6-bit value α , and for any 4-bit value β , that the number of solutions (for x) to the equation

$$S[x] \oplus S[x \oplus \alpha] = \beta$$

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.

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

Yes! (substitution-permutation networks)

Plan

1. Computational definitions of security
2. If semantically-secure encryption exists, then
P \neq NP
3. A proof that “the PRGs imply secure encryption”
4. Theoretical constructions of PRGs
5. Stream ciphers
6. Pseudorandom functions and permutations
7. Block ciphers
 1. Modes of operation
 2. Popular construction paradigms
 1. Feistel ciphers
 2. Substitution-permutation networks
 3. Cascade ciphers
8. Practical considerations



Substitution-permutation networks

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



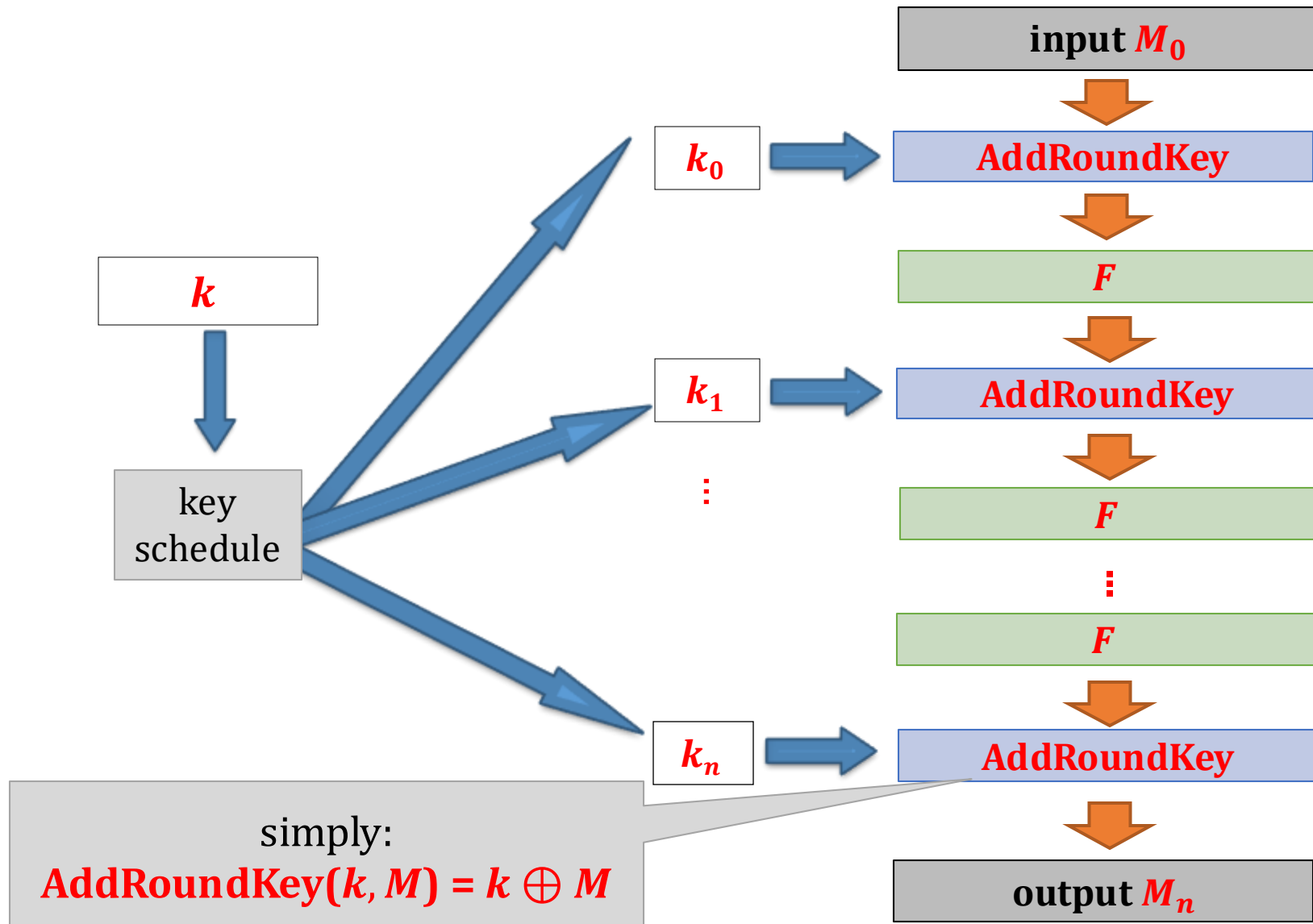
Used in **AES (Rijndael), 3-Way, SAFER, SHARK, Square...**

Advanced Encryption Standard (AES)

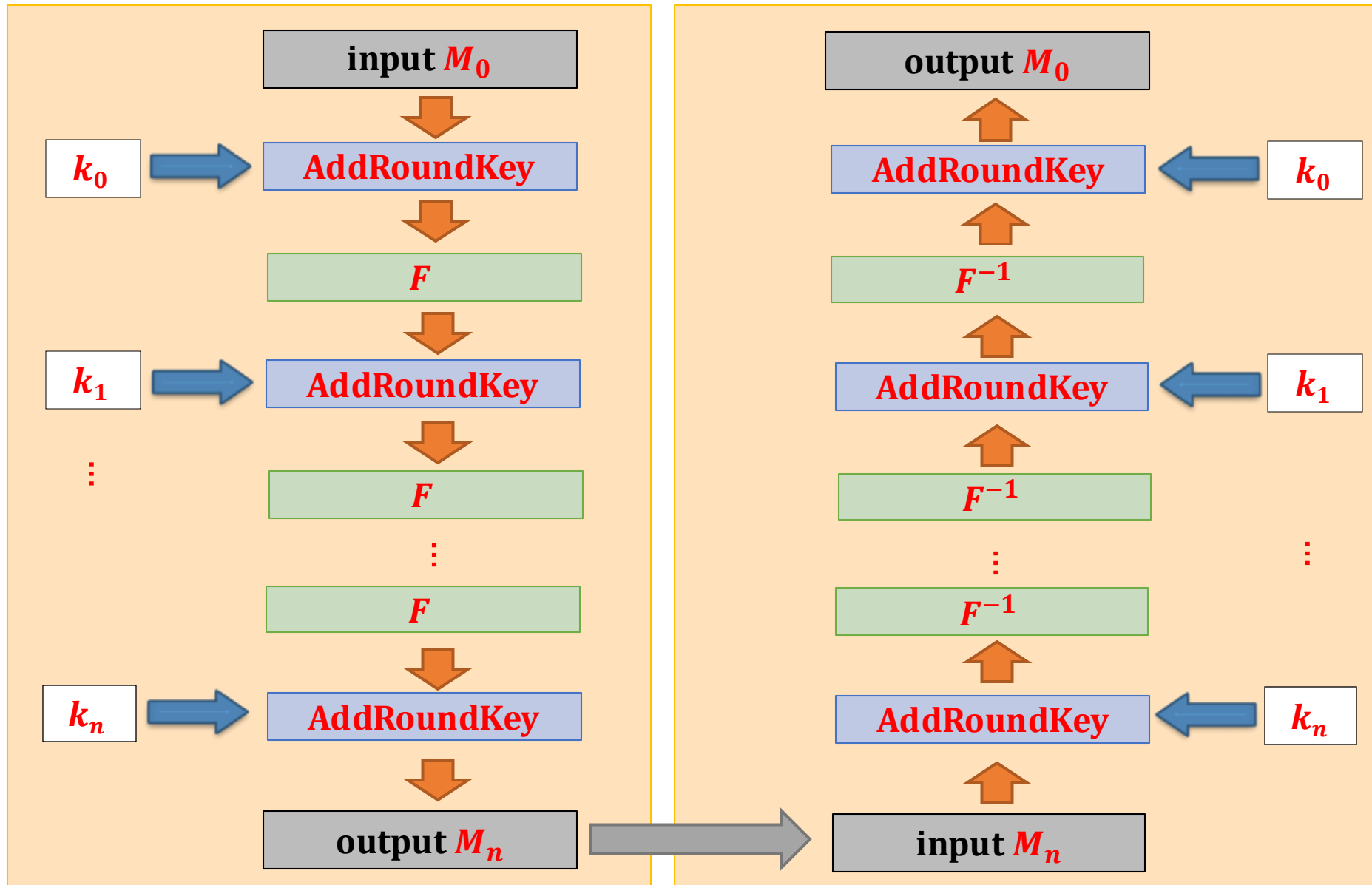
- Competition for **AES** announced in **January 1997** by the **US National Institute of Standards and Technology (NIST)**
- **15** ciphers submitted
- **5** finalists: **MARS, RC6, Rijndael, Serpent**, and **Twofish**
- **October 2, 2000**: **Rijandel** selected as the winner.
- **November 26, 2001**: **AES** becomes an official standard.
- Authors : **Vincent Rijmen, Joan Daemen** (from Belgium)
- **Key sizes**: **128, 192** or **256** bit, **block size**: **128** bits



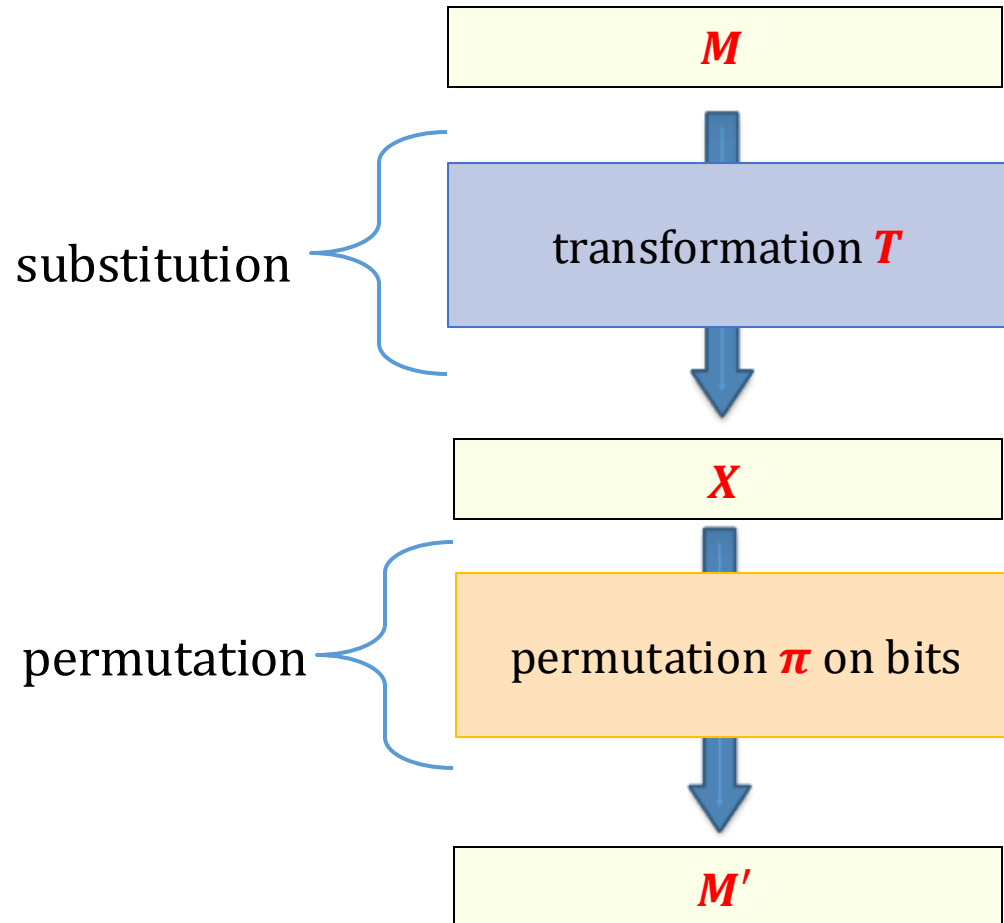
Substitution-permutation networks



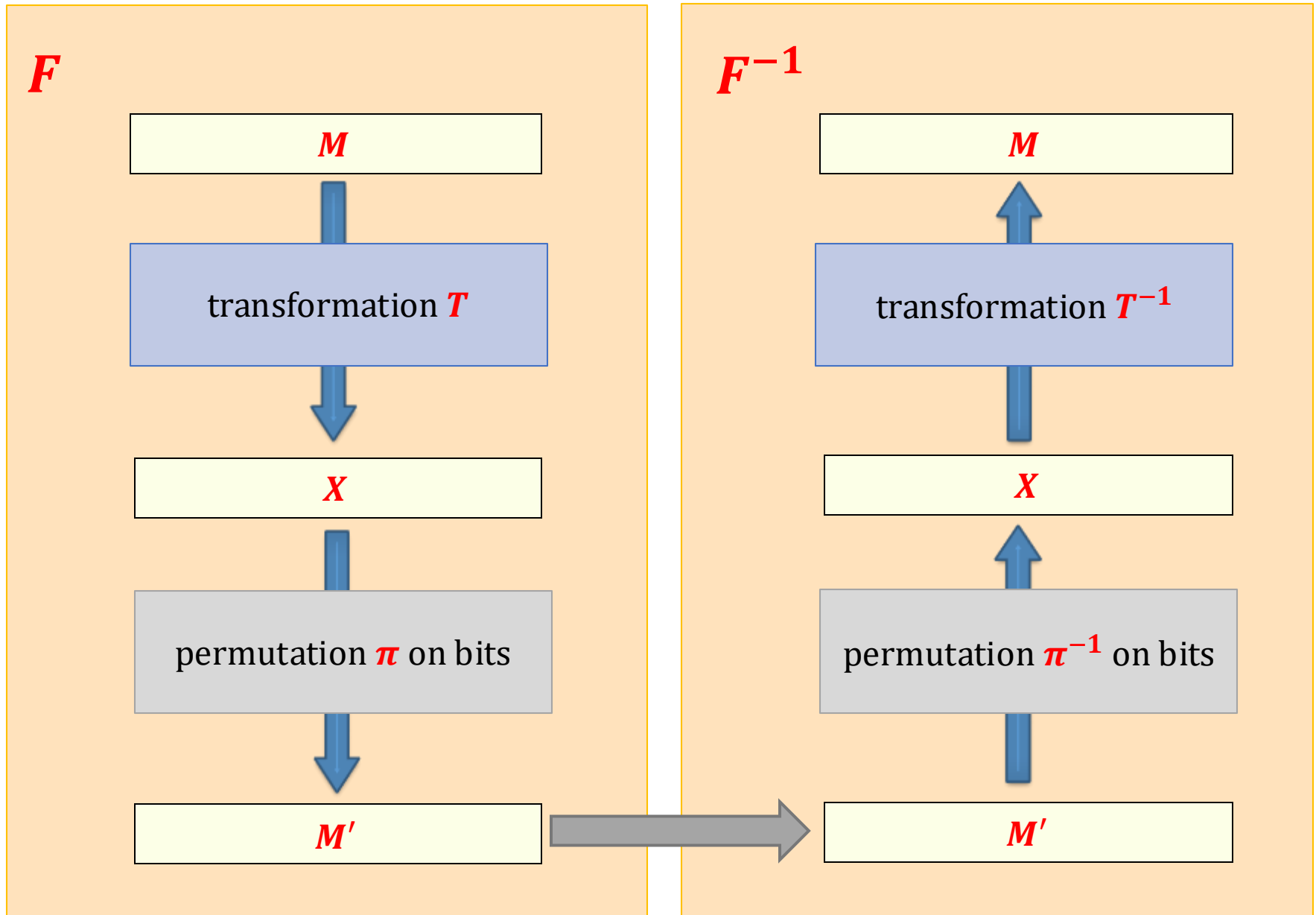
To invert: invert the order and apply F^{-1} instead of F .



A construction of F

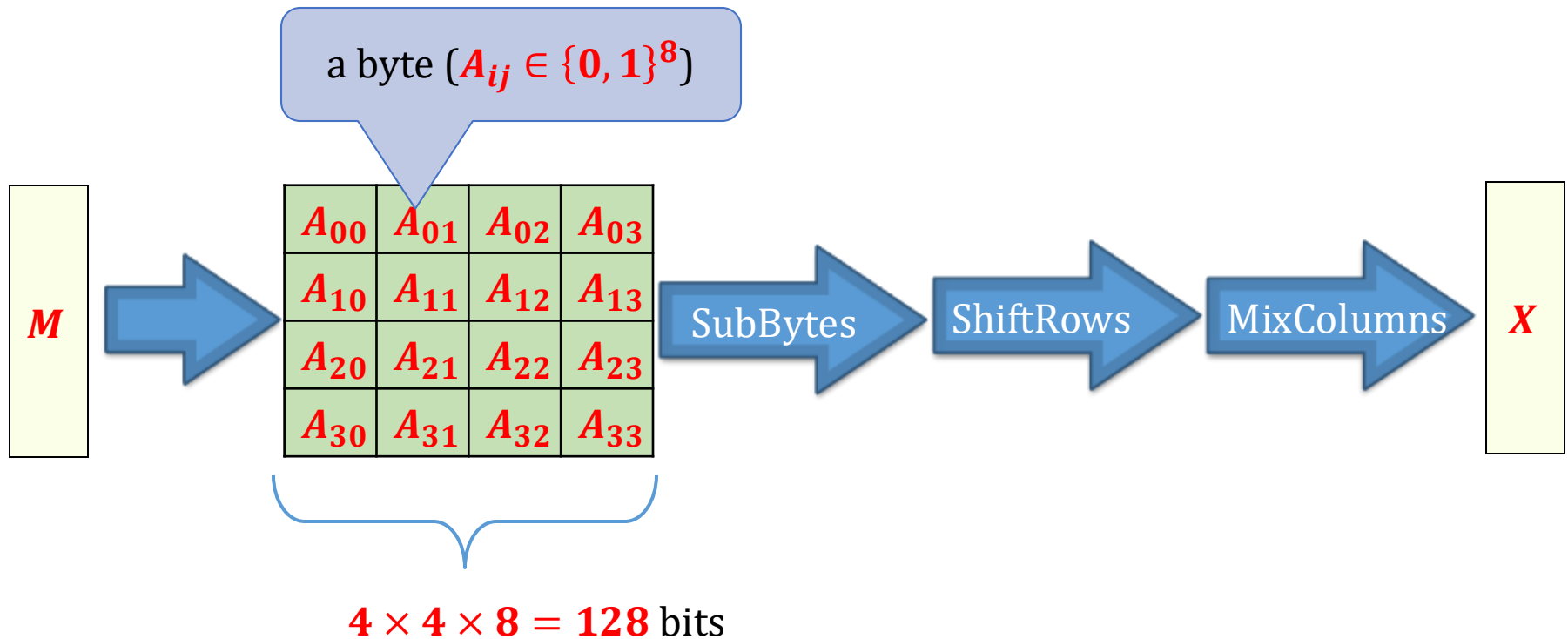


How to construct F^{-1} ?



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 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 1 ,
 $g \cdot \cdots \cdot g$ (n times) is denoted with g^n
 $(g^{-1})^n$ is denoted with g^{-1} .

Fields

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

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

How to define a “field over bytes”?

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

where

$$\begin{aligned} &(\mathbf{a}_1, \dots, \mathbf{a}_n) + (\mathbf{b}_1, \dots, \mathbf{b}_n) \\ &= (\mathbf{a}_1 \oplus \mathbf{b}_1, \dots, \mathbf{a}_n \oplus \mathbf{b}_n) \end{aligned}$$



xor

Extremely efficient to implement.

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

“Galois fields” $\text{GF}(2^n)$:

Represent each $(a_0, \dots, a_{n-1}) \in \{0, 1\}^n$ as a polynomial A over \mathbb{Z}_2 of degree $n - 1$.

$$A(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$$

Note: if (b_0, \dots, b_{n-1}) is represented as a polynomial $B(x) = b_0 + b_1x + \dots + b_{n-1}x^{n-1}$, 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:

$$A(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}$$

$$B(x) = b_0 + b_1x + \cdots + b_{n-1}x^{n-1}.$$

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

How to reduce this degree?

do not exists non-constant polynomials q_0, q_1 such that $p = q_0 \cdot q_1$

Take an **irreducible polynomial** p of degree n , and compute

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

Then C is a polynomial of degree $n - 1$.

Write $C(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$.

Define: $(a_0, \dots, a_{n-1}) \times (b_0, \dots, b_{n-1}) = (c_0, \dots, c_{n-1})$

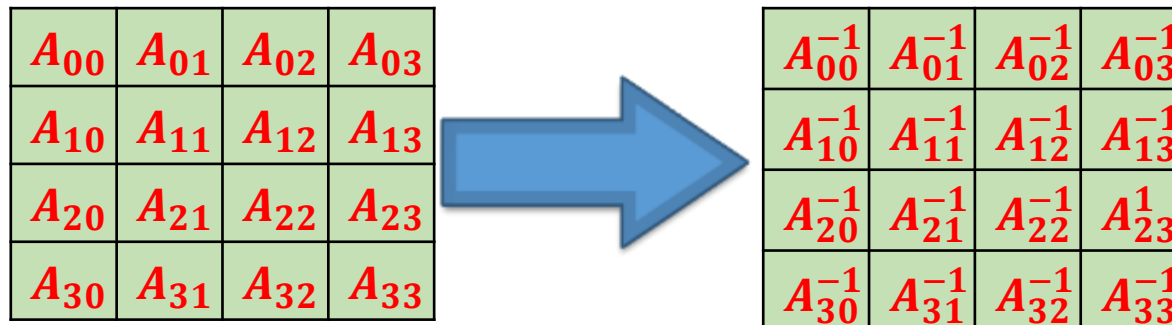
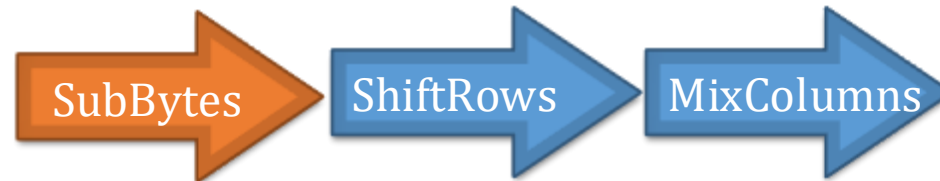
Fact from algebra: this defines a field.

AES field

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

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

First step of SubBytes



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

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

Another observation

We can look at \mathbf{Z}_2^n as a **linear space**.

AES defines the following affine transformation:

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

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

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

Advantage: **SubBytes** is not an operation only in $\mathbf{GF}(2^n)$.

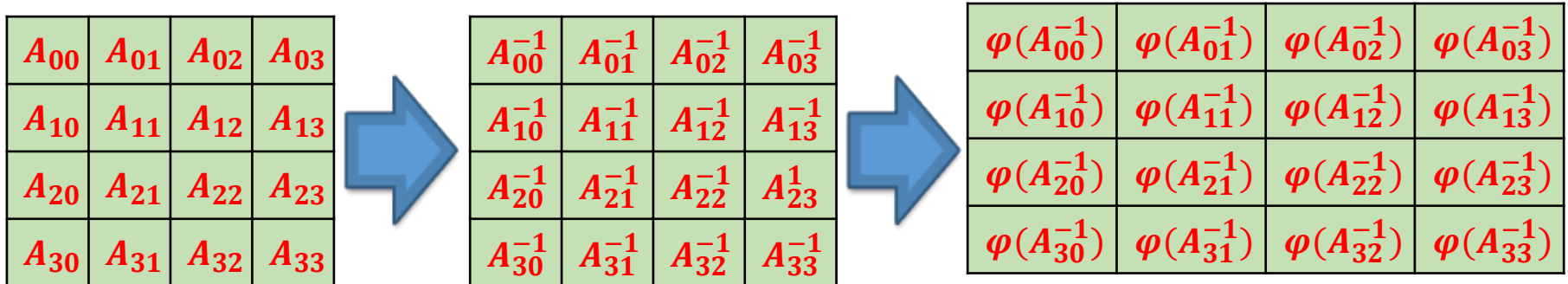
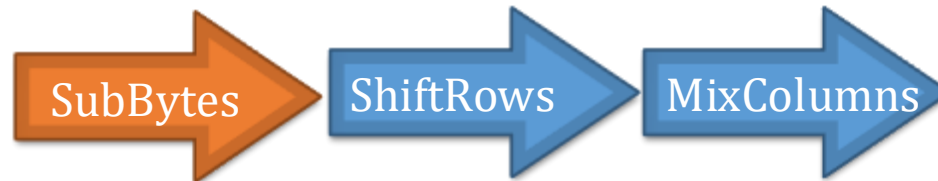
The constant vector is chosen in such a way that there are no

fixpoints $\varphi(X) = X$

anti-fixpoints $\varphi(X) = \overline{X}$

φ is invertible.

Complete SubBytes



Observe that

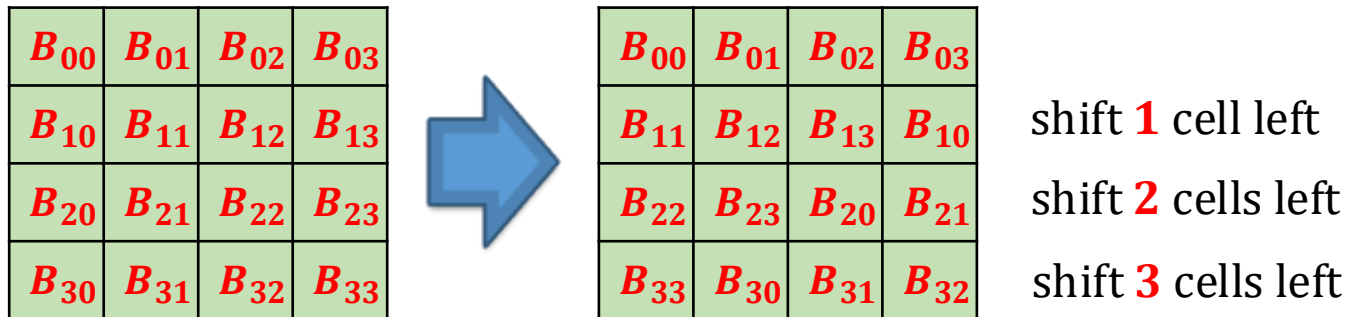
$$A_{ij} \mapsto A_{ij}^{-1} \mapsto \varphi(A_{ij}^{-1})$$

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

ShiftRows

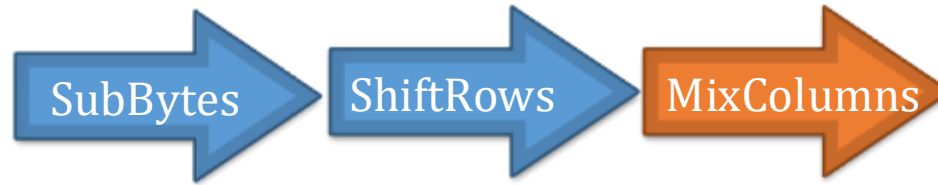


Cyclic shifts of rows:



Clearly: ShiftRows is invertible.

MixColumns



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

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}

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}

M :

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

bytes treated
as polynomial
coefficients

*

C_{02}
C_{12}
C_{22}
C_{32}

=:

C'_{02}
C'_{12}
C'_{22}
C'_{32}

Clearly M is invertible, so the whole operation also is.

AES construction – more details

Concrete parameters:

key size: 128, 192 or 256 bit,

block size: 128 bits

We omit the description of the key schedule.

Security:

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

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

Plan

1. Computational definitions of security
2. If semantically-secure encryption exists, then **P \neq NP**
3. A proof that “the PRGs imply secure encryption”
4. Theoretical constructions of PRGs
5. Stream ciphers
6. Pseudorandom functions and permutations
7. Block ciphers
 1. Modes of operation
 2. Popular construction paradigms
 3. Cascade ciphers
8. 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!

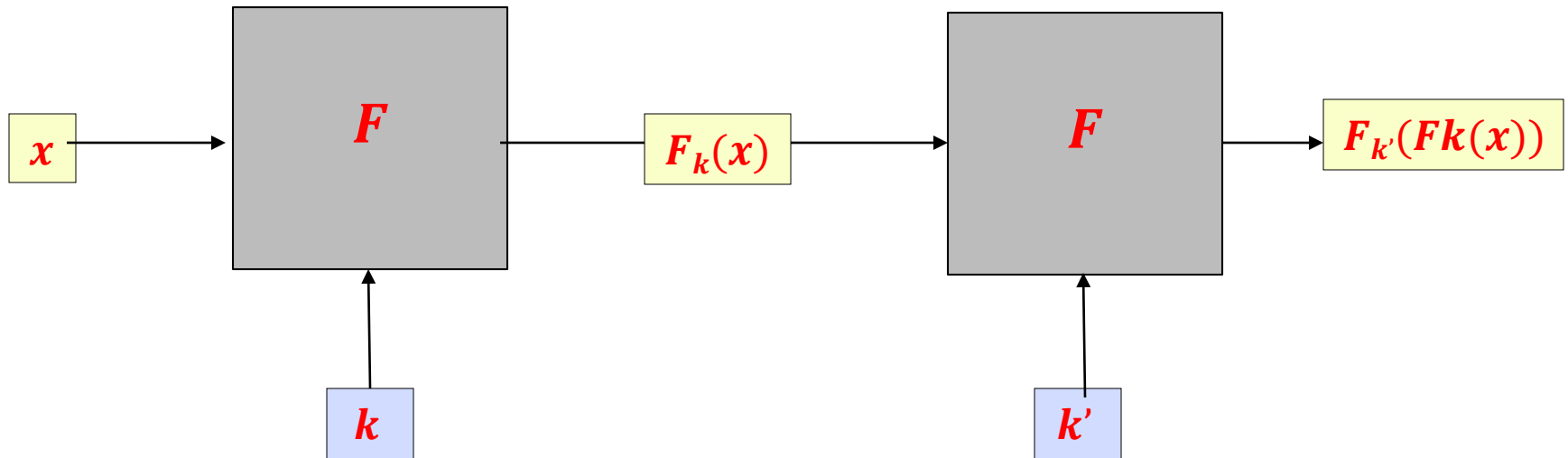
We now describe it in an abstract way (for any block cipher ***F***)

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

n = block length = key length

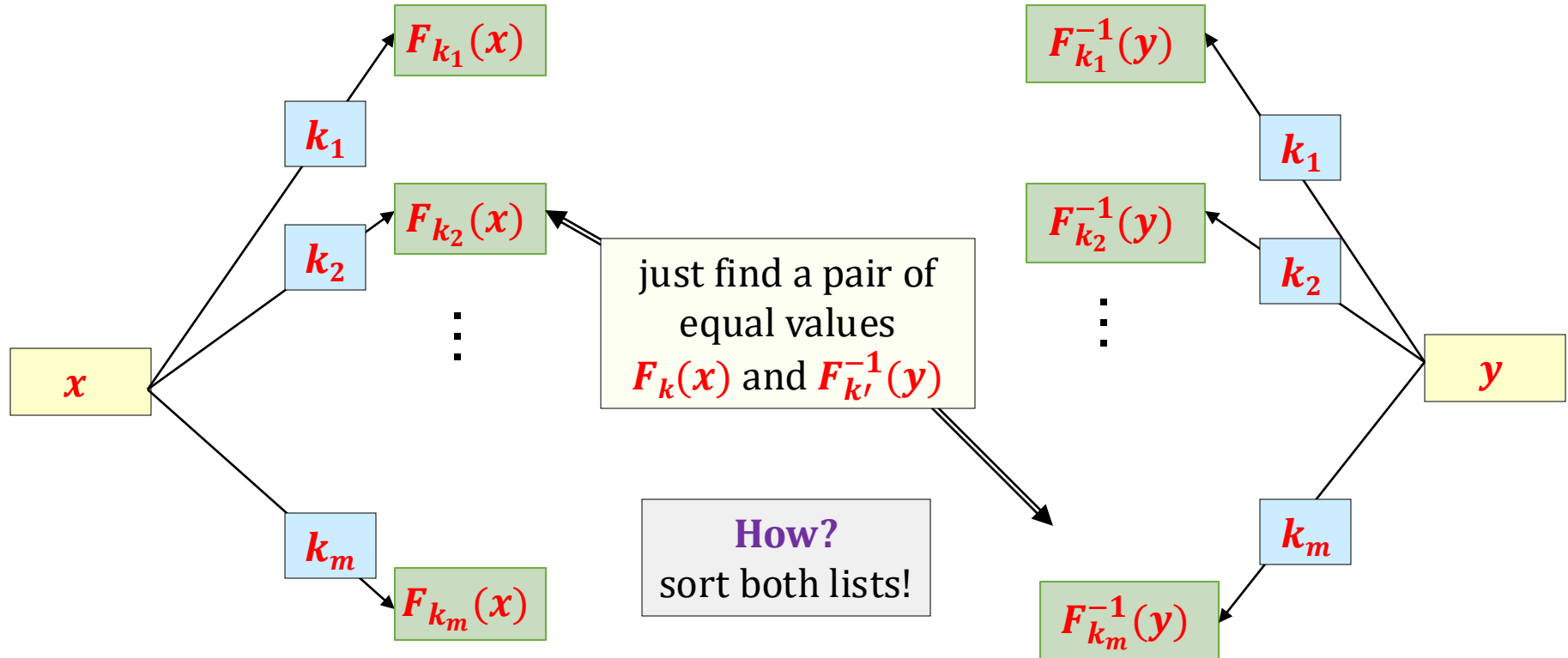
Double encryption can be broken using

- time $O(2^n)$,
- space $O(2^n)$,
- 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 (k, k') such that for some z we have
 $(z, k) \in L$ and $(z, k') \in L'$.
5. Output S .

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

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

P (a random pair (k, k') 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

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

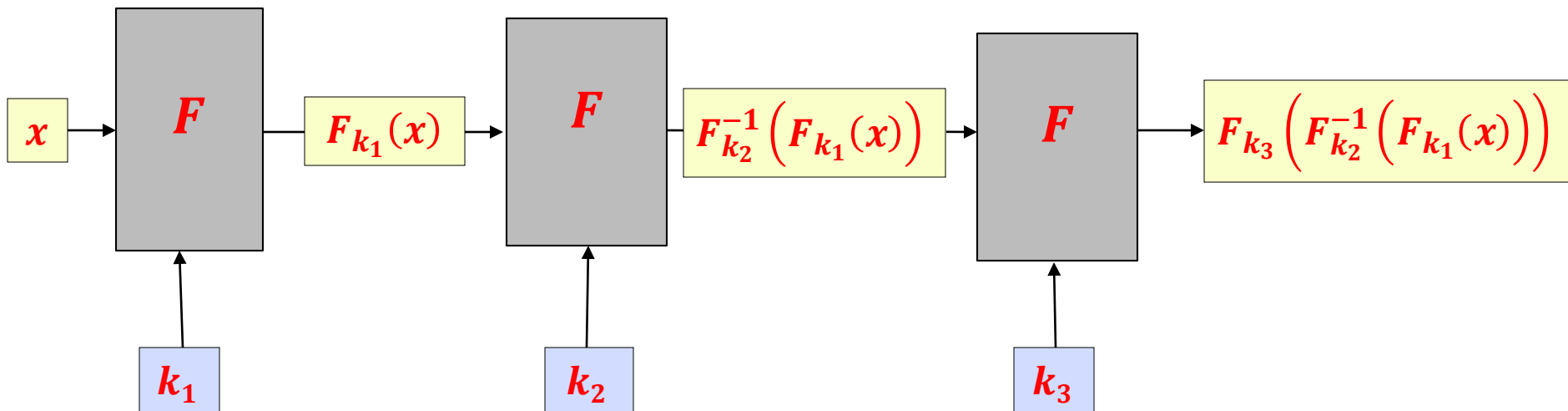
Hence, the expected number of “false positives” is around

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

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. Computational definitions of security
2. If semantically-secure encryption exists, then **P \neq NP**
3. A proof that “the PRGs imply secure encryption”
4. Theoretical constructions of PRGs
5. Stream ciphers
6. Pseudorandom functions and permutations
7. Block ciphers
8. Practical considerations



Benchmarks

	Algorithm	MiB/Second	Cycles Per Byte
stream {	Salsa20	643	2.7
	Sosemanuk	727	2.4
block {	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
	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.

©2025 by Stefan Dziembowski. Permission to make digital or hard copies of part or all of this material is currently granted without fee *provided that copies are made only for personal or classroom use, are not distributed for profit or commercial advantage, and that new copies bear this notice and the full citation.*