# Lecture 2
# Symmetric Encryption I


**Stefan Dziembowski**

www.crypto.edu.pl/Dziembowski

**University of Warsaw**


**12.10.16** **version 1.0**

# Plan

1. If semantically-secure encryption exists then **P ≠ NP**
2. A proof that “the PRGs imply secure encryption”
3. Theoretical constructions of PRGs
4. Stream ciphers

# From the last lecture: semantic security

security parameter
$1^n$

adversary
(polynomial-time probabilistic Turing machine)

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
   $c := \mathrm{Enc}(k,m_b)$

$c$

has to guess $b$

Alternative name: **has indistinguishable encryptions**

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

# Is it possible to prove security?

Bad news:

**Theorem**

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

**then**

$\mathbf{P \neq NP}$

**Intuition**: if **P = NP** then the adversary can guess the key...

# Proof [1/5]

**(Enc,Dec)** – an encryption scheme.
For simplicity suppose that **Enc** is deterministic

Consider the following language:

$$L = \{(c,m) : \textbf{there exists } k \textbf{ such that } c = \mathrm{Enc}(k,m)\}$$

$L$ is a language of all pairs $(c,m)$, where $c$ can be a ciphertext of $m$

Clearly $L$ is in **NP**.

$k$ is the **NP**-witness

# Proof [2/5]

Suppose **P=NP**.

Therefore there exists a poly-time machine $M_L$ such that:


**"yes"** – if there exists $k$ such that $c$ **=** **Enc**($k$,$m$)

**"no"** – otherwise

# Proof [3/5]

$L$ is a language of all pairs $(c,m)$, where $c$ can be a ciphertext of $m$

Suppose **P = NP** and hence $L$ is poly-time decidable.

chooses random $m_0, m_1$ such that $|m_i| = n+1$

$m_0, m_1$ →

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

← $c$

$c$, $m_0$ → $M_L$ → yes/no →

If $(c, m_0) \in L$ then output **0**
else output **1**

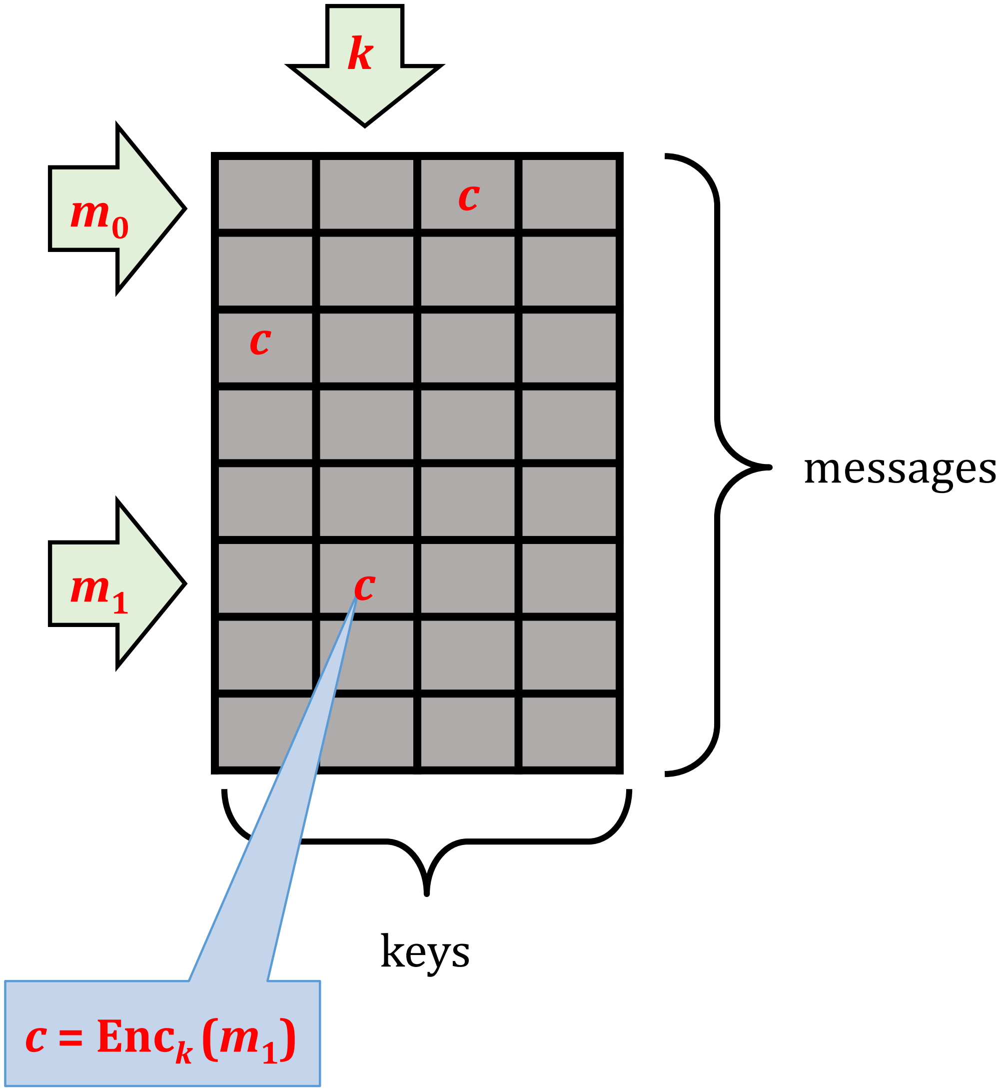
**Observation**

The adversary guesses incorrectly if $b=1$ and there exists $k'$ such that

$$\mathrm{Enc}(k', m_0) = \mathrm{Enc}(k, m_1)$$

What is the probability that this happens?

# Proof [4/5]



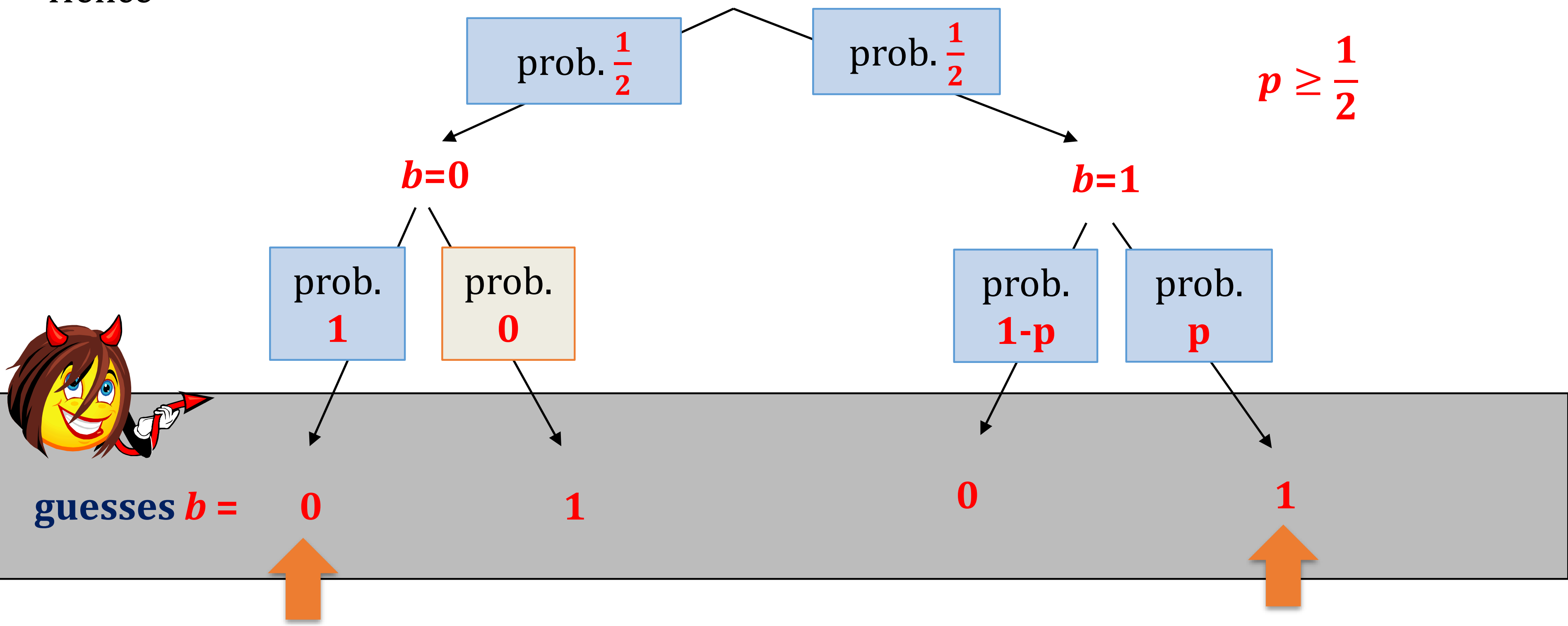

From the correctness of encryption:
$c$ can appear in each column **at most once**.

Hence the probability $p$ that it appears in a randomly chosen row is at most:

$$|\mathcal{K}| \, / |\mathcal{M}| = \frac{1}{2}$$

# Proof [5/5]

Hence




probability of a correct guess:

$$\frac{1}{2} + \frac{p}{2} \geq \frac{3}{4}$$

Hence **(Enc,Dec)** is not secure.

**QED**

**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 **<u>a lower bound on the computational complexity of some problem.</u>**

In the "asymptotic setting" that would mean that
at least
we show that **P ≠ NP**.

Does the implication in the other direction hold?
(that is: does **P ≠ 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 ≠ 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**:
“decisional Diffie-Hellman assumption”
“strong RSA assumption”

Some “computational assumption **A**” holds

interesting only if this is “far from trivial”

then scheme **X** is secure.

in this we have to “believe”

the rest is provable

# Example

Suppose that $G$ is a "cryptographic pseudorandom generator"

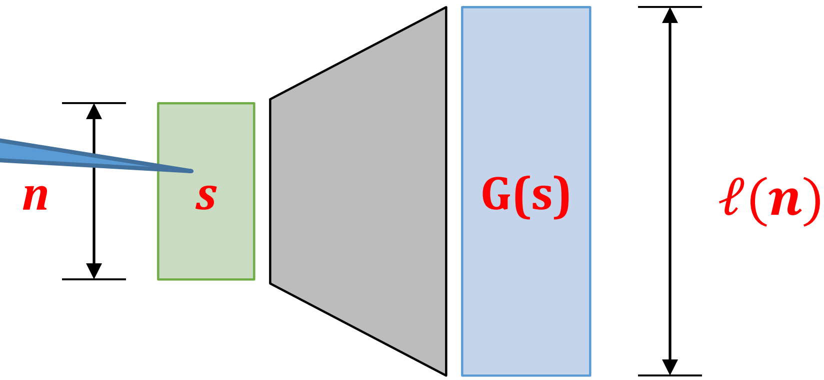
we can construct a secure encryption scheme based on $G$

# Plan

1. If semantically-secure encryption exists then **P ≠ NP**
2. A proof that "the PRGs imply secure encryption"
3. Theoretical constructions of PRGs
4. Stream ciphers

# Pseudorandom generators

“seed”




“expansion factor”

**Definition**

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

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

for every $n$

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

we have

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

and for a random $s$ the value $G(s)$ “**looks random**”.

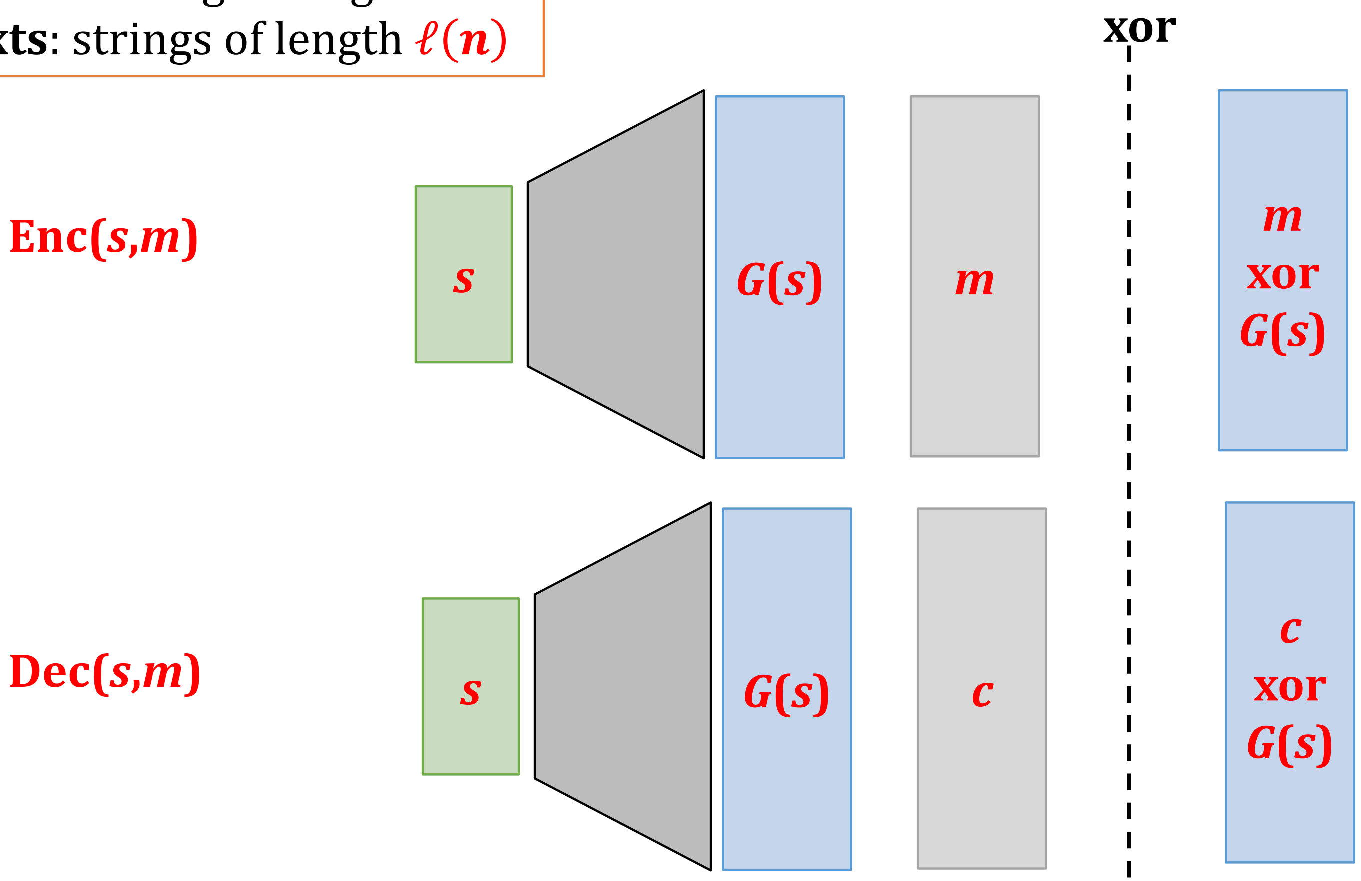
this has to be formalized

# 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 $\boldsymbol{n}$
**Plaintexts**: strings of length $\ell(\boldsymbol{n})$



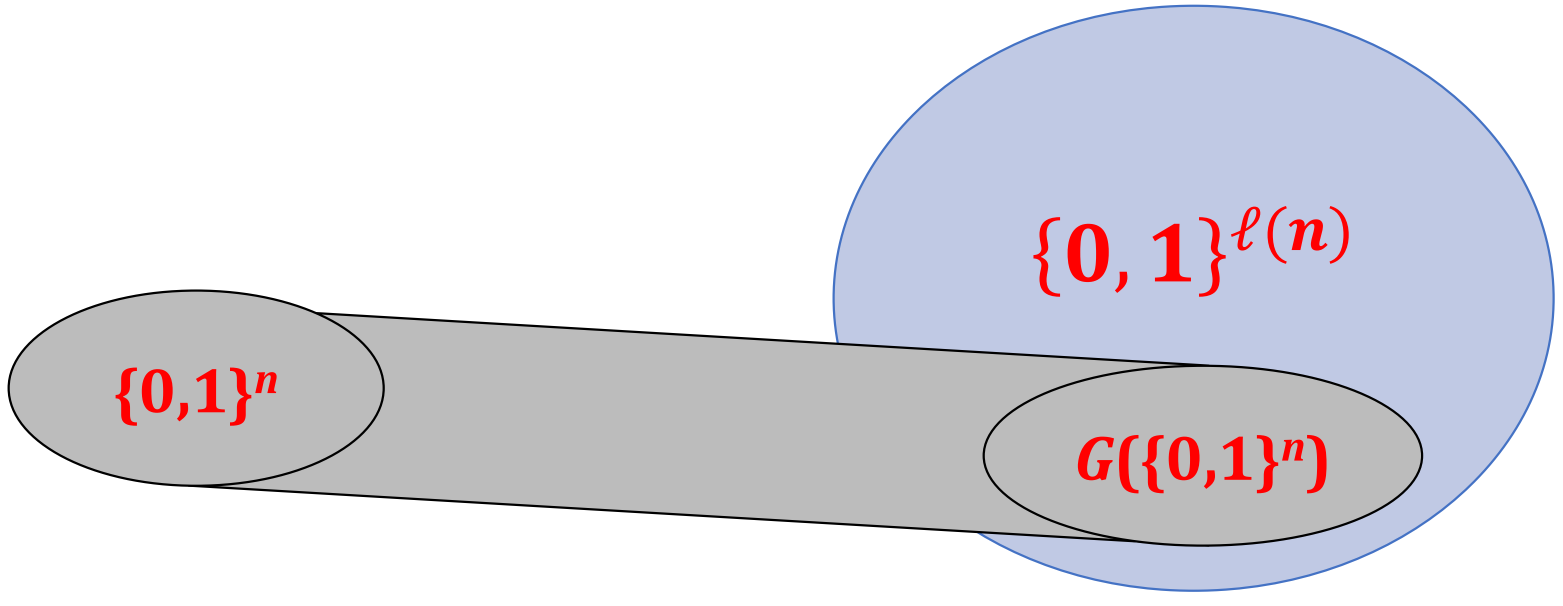

If we use a “normal PRG” – this idea doesn’t work
We have to use the **cryptographic PRGs.**

# "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

- $\boldsymbol{X_0 \in Z_m}$ – the key
- for $\boldsymbol{n = 1, 2, \ldots}$ let

$$\boldsymbol{X_{n+1} := (a \cdot X_n + c) \bmod m}$$

**output**: $\boldsymbol{Y_1, Y_2, \ldots}$ where

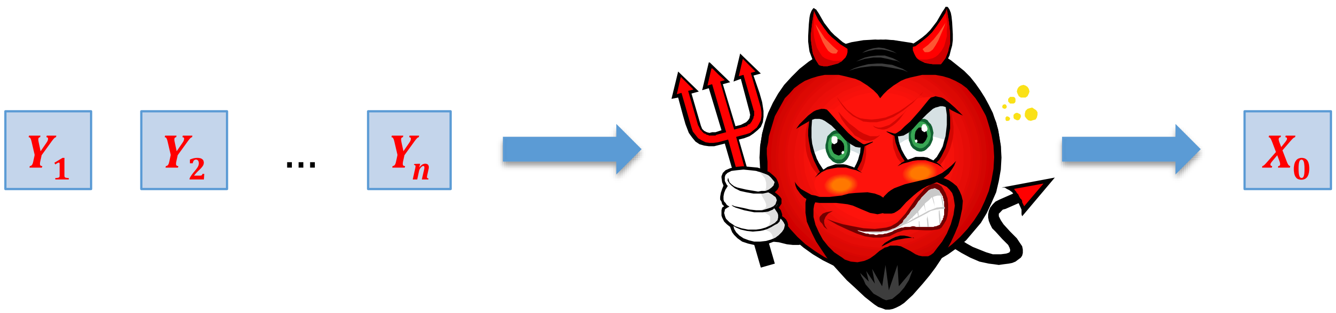
$\boldsymbol{Y_i}$ = first $\boldsymbol{t}$ bits of each $\boldsymbol{X_i}$

---

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

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

# How to break an LRS



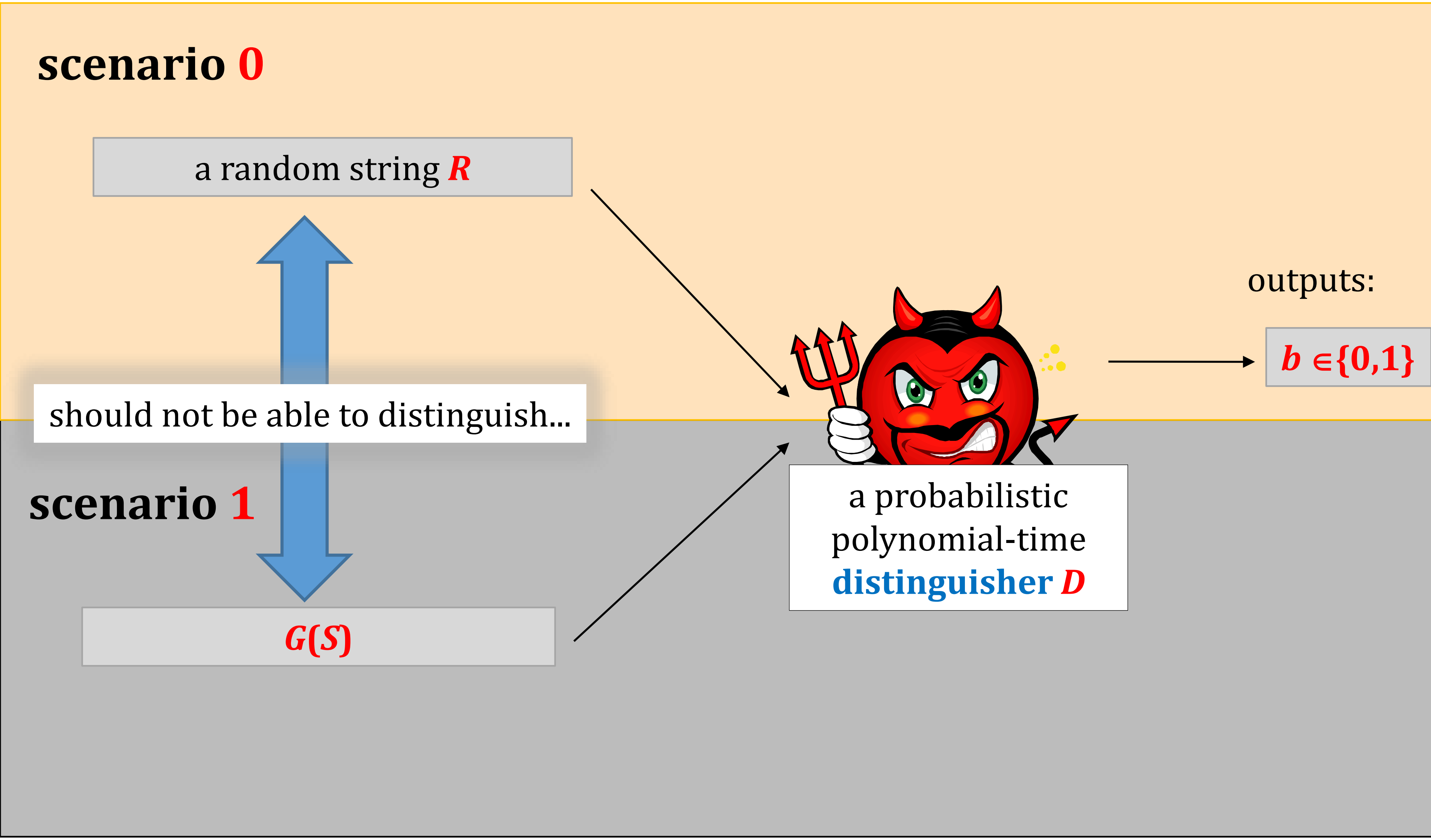

Solve linear equations with “partial knowledge” (because you only know only first $t$ bits)

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



# Cryptographic PRG

a random string $R$

or

$G(S)$ (where $S$ random)

outputs:

$0$ 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

$$\left|P(D(R)=1) - P\big(D\big(G(S)\big)=1\big)\right|$$

is negligible in $n$.

# Constructions

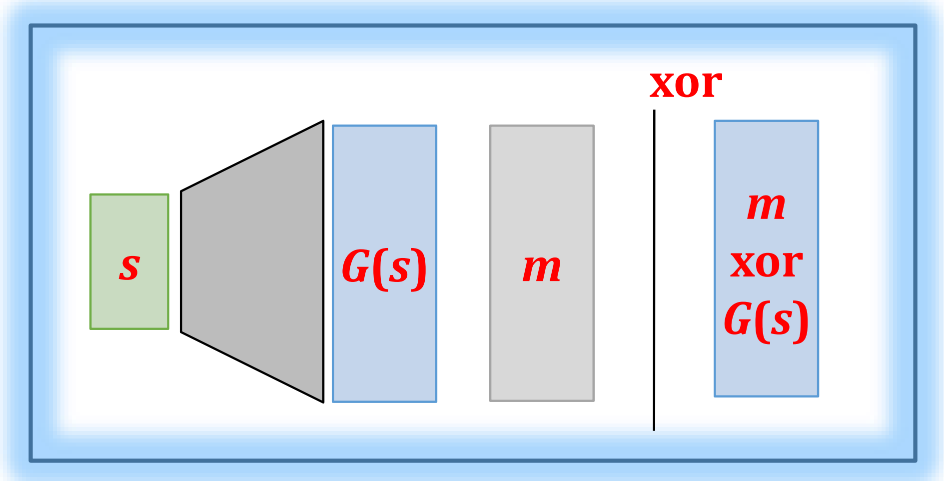
There exists constructions of cryptographic pseudorandom-generators, that are **conjectured** to be secure.

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 CPA-secure.



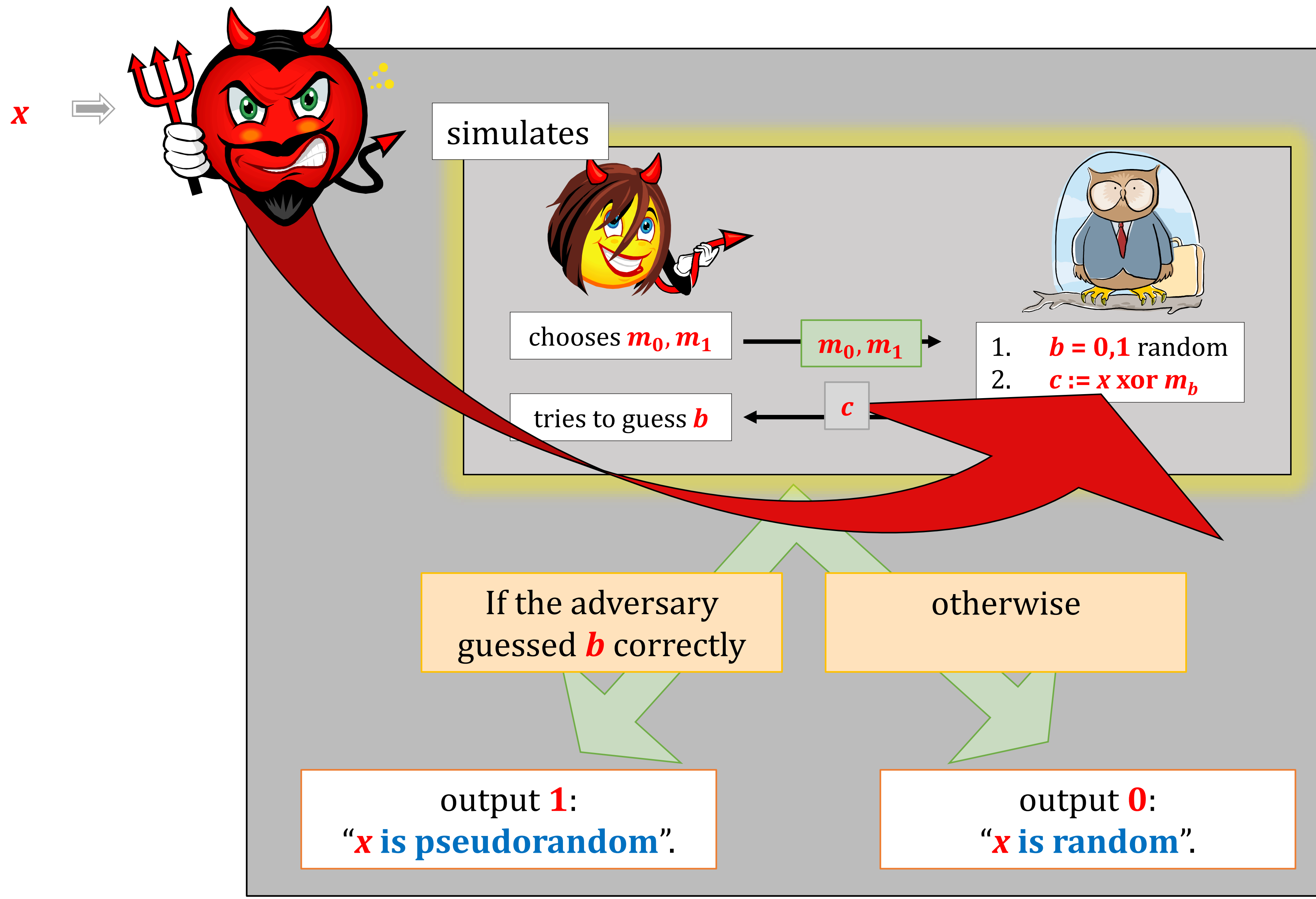

cryptographic PRGs exist → CPA-secure encryption exists

**<u>Proof</u>** (sketch)

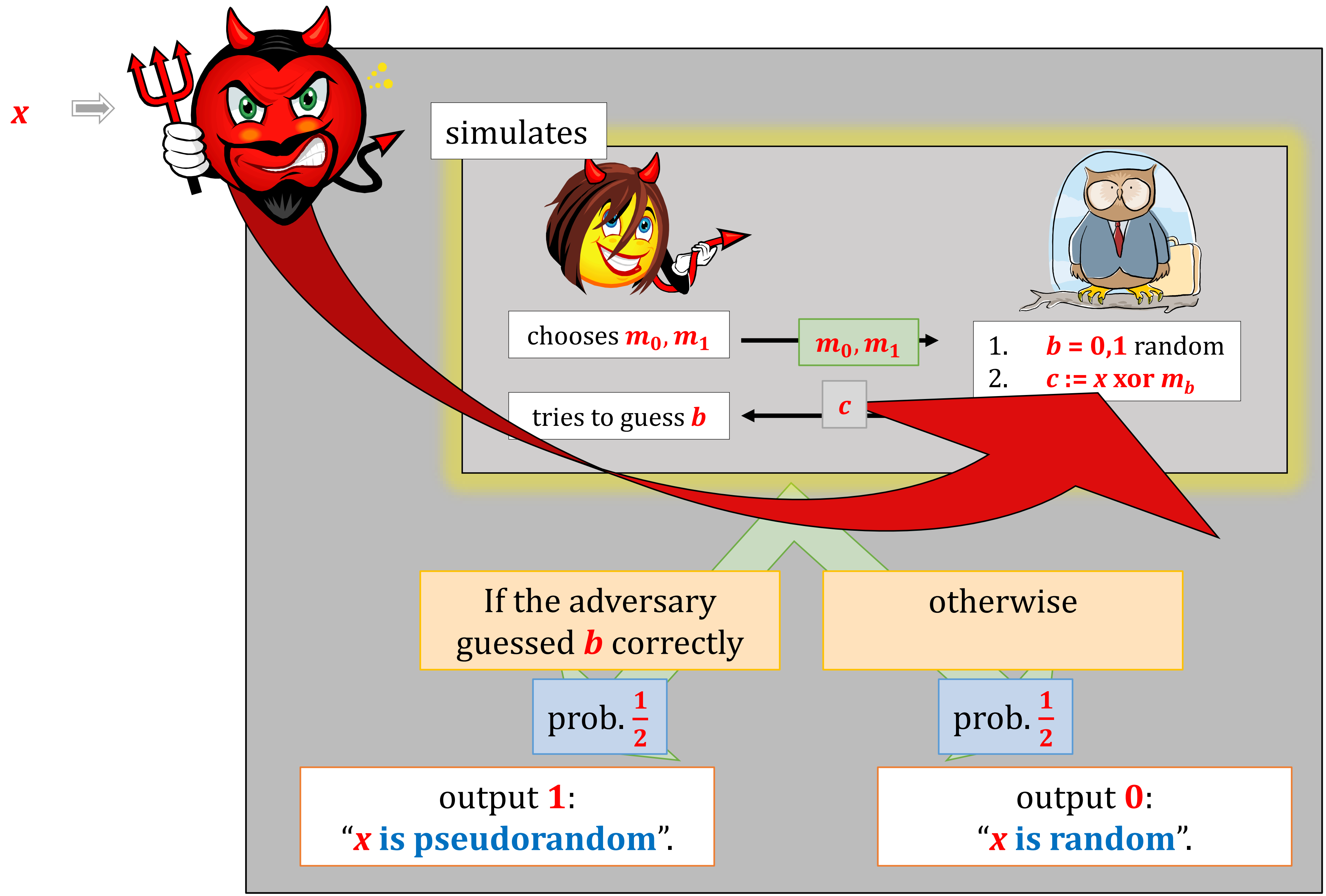
Let us concentrate on the **one message case** (i.e. semantic security).

Suppose that it is **not** secure.

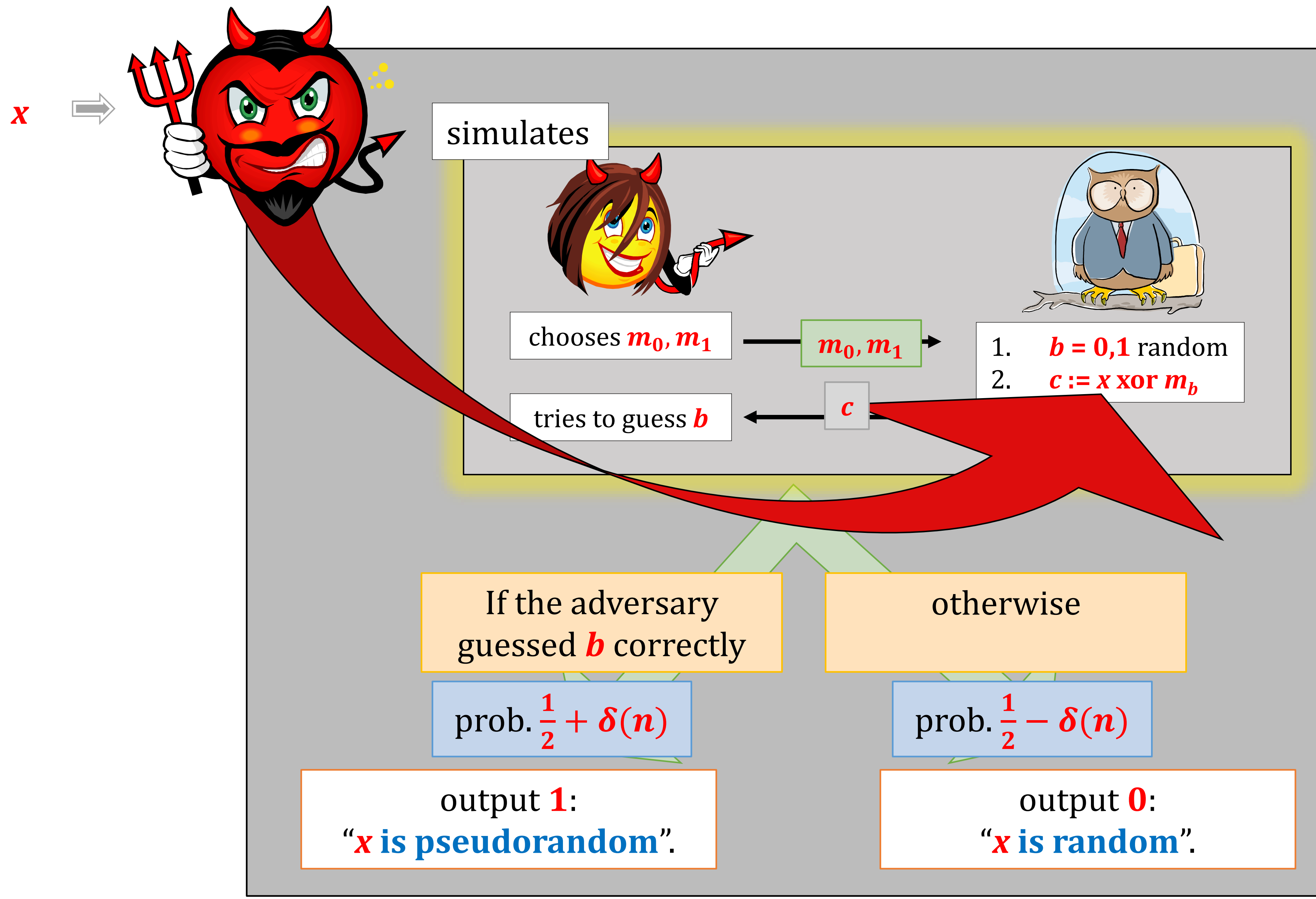
Therefore there exists an poly-time adversary that wins the "guessing game" with probability $\frac{1}{2} + \delta(n)$ where $\delta(n)$ is not negligible.


$x$
simulates
chooses $m_0, m_1$
$m_0, m_1$
1. $b = 0,1$ random
2. $c := x$ xor $m_b$
$c$
tries to guess $b$
If the adversary guessed $b$ correctly
otherwise
output 1: "$x$ is pseudorandom".
output 0: "$x$ is random".

# “scenario 0”: $x$ is a random string



# "scenario 1": $x = G(S)$



Hence

$x$ is a random string $R$

$x = G(S)$

the adversary guesses $b$ correctly with probability $\frac{1}{2}$

the adversary guesses $b$ correctly with probability $\frac{1}{2} + \delta(n)$

prob. $\frac{1}{2}$

prob. $\frac{1}{2}$

prob. $\frac{1}{2} + \delta(n)$

prob. $\frac{1}{2} - \delta(n)$

**outputs**: $\mathbf{1}$ $\mathbf{0}$ $\mathbf{1}$ $\mathbf{0}$

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

Since $\delta$ 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

it can be extended to CPA-security

cryptographic PRGs exist

semantically-secure encryption exists

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.

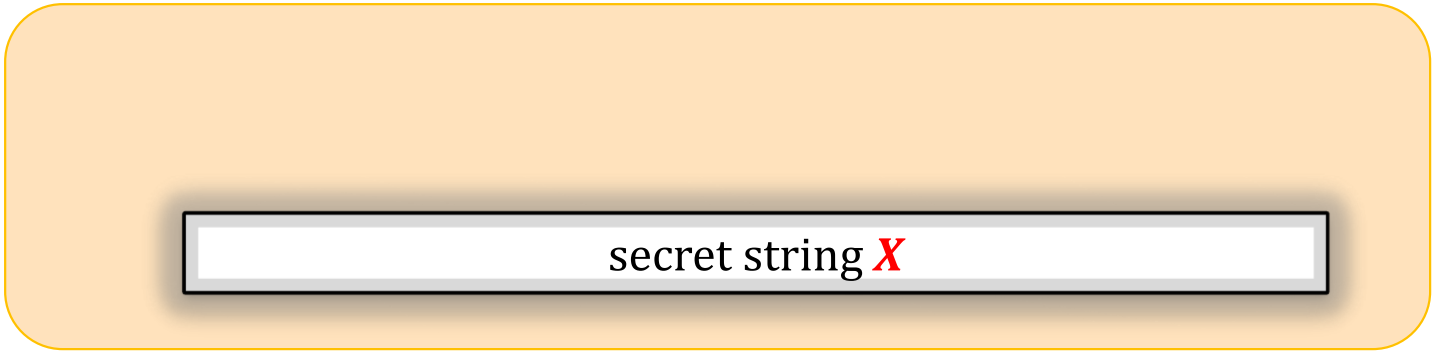
# Why?

To achieve the same result  needs to work **1000** times

more than  .

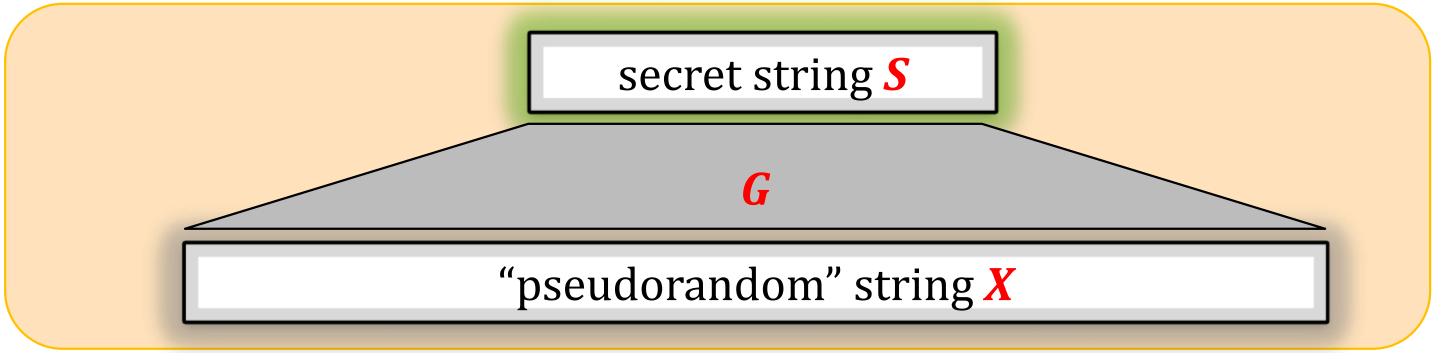
# General rule

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

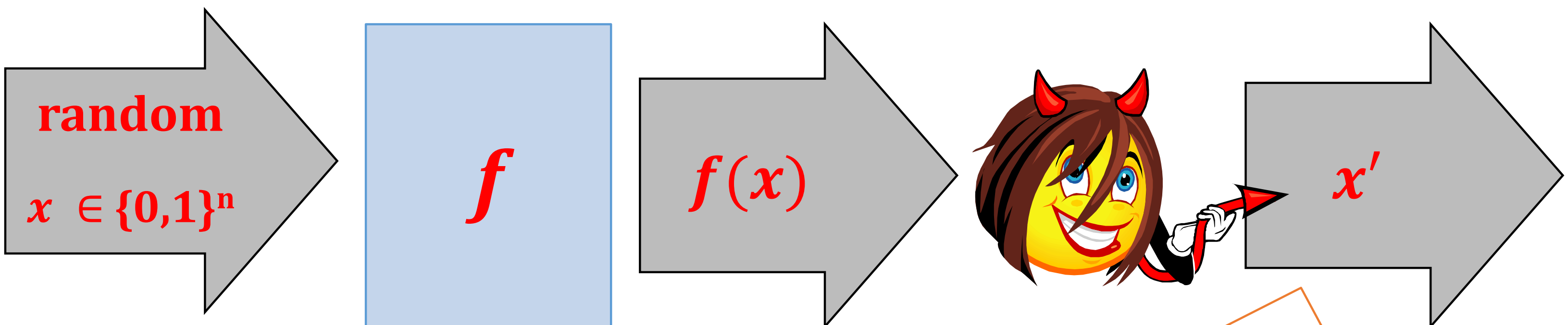
1. If semantically-secure encryption exists then **P ≠ NP**
2. A proof that "the PRGs imply secure encryption"
3. Theoretical constructions of PRGs
4. Stream ciphers

# One-way functions

A function

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

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”.

# 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 $f: \{0,1\}^* \to \{0,1\}^*$ is **one-way** if

$\forall$ **polynomial-time Turing Machine $M$** **P ($M$ wins) is negligible**

# 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.

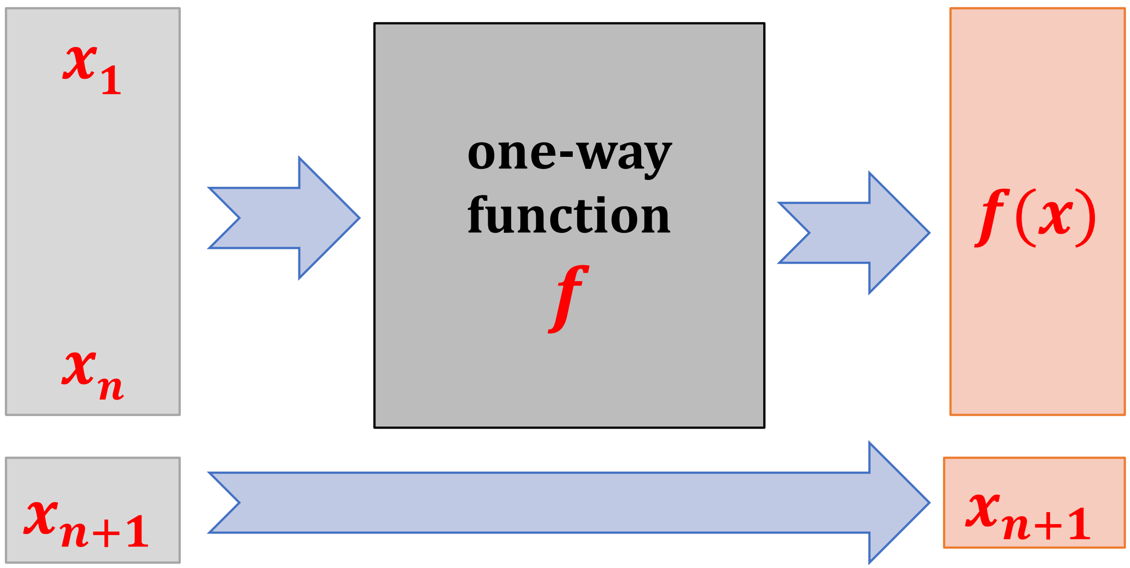
**Example**:

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

this function is defined on
**primes × primes**,
not on
$\{0,1\}^*$
but it's just a technicality

# One way functions **do not** "hide all the input"

Example:




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

# How to encrypt with one-way functions?

Naive (and wrong idea):

1. Take a one-way function $\boldsymbol{f}$,
2. Let a ciphertext of a message $\boldsymbol{M}$ be equal to

$$\boldsymbol{C} := \boldsymbol{f}(\boldsymbol{M})$$

where is the key?

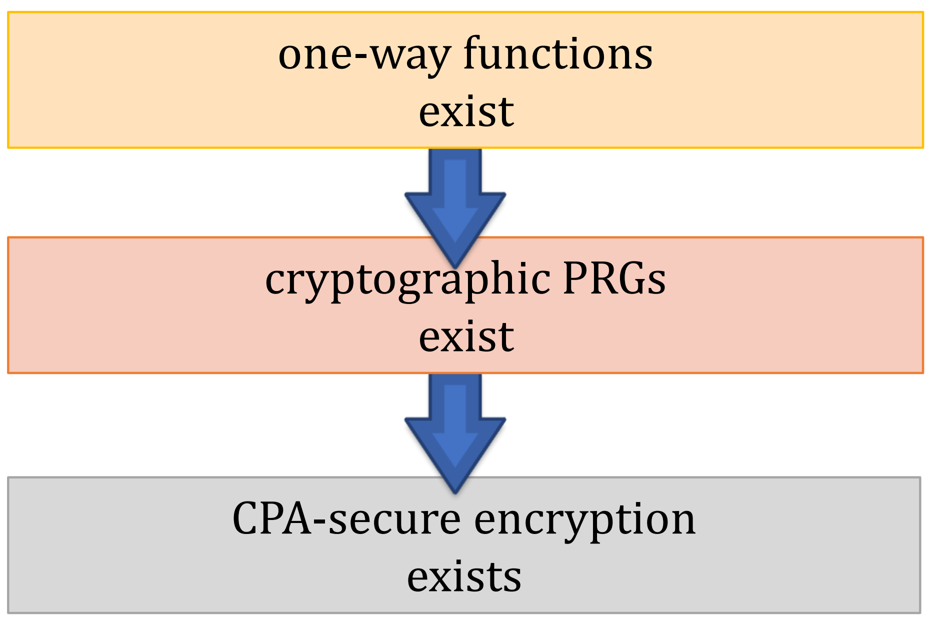
how to decrypt?

not all the input is hidden...

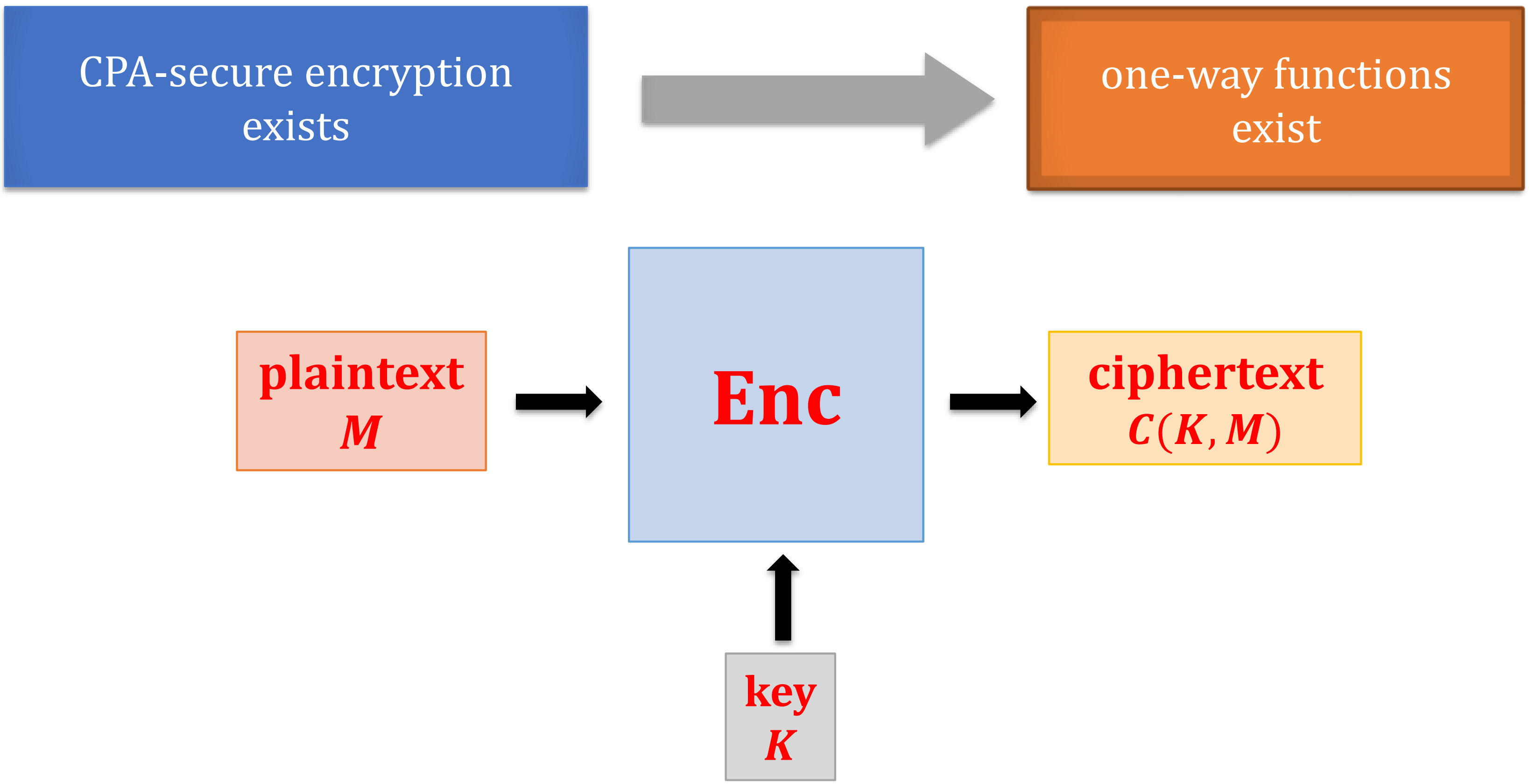
# One of the most fundamental results in the symmetric cryptography

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

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

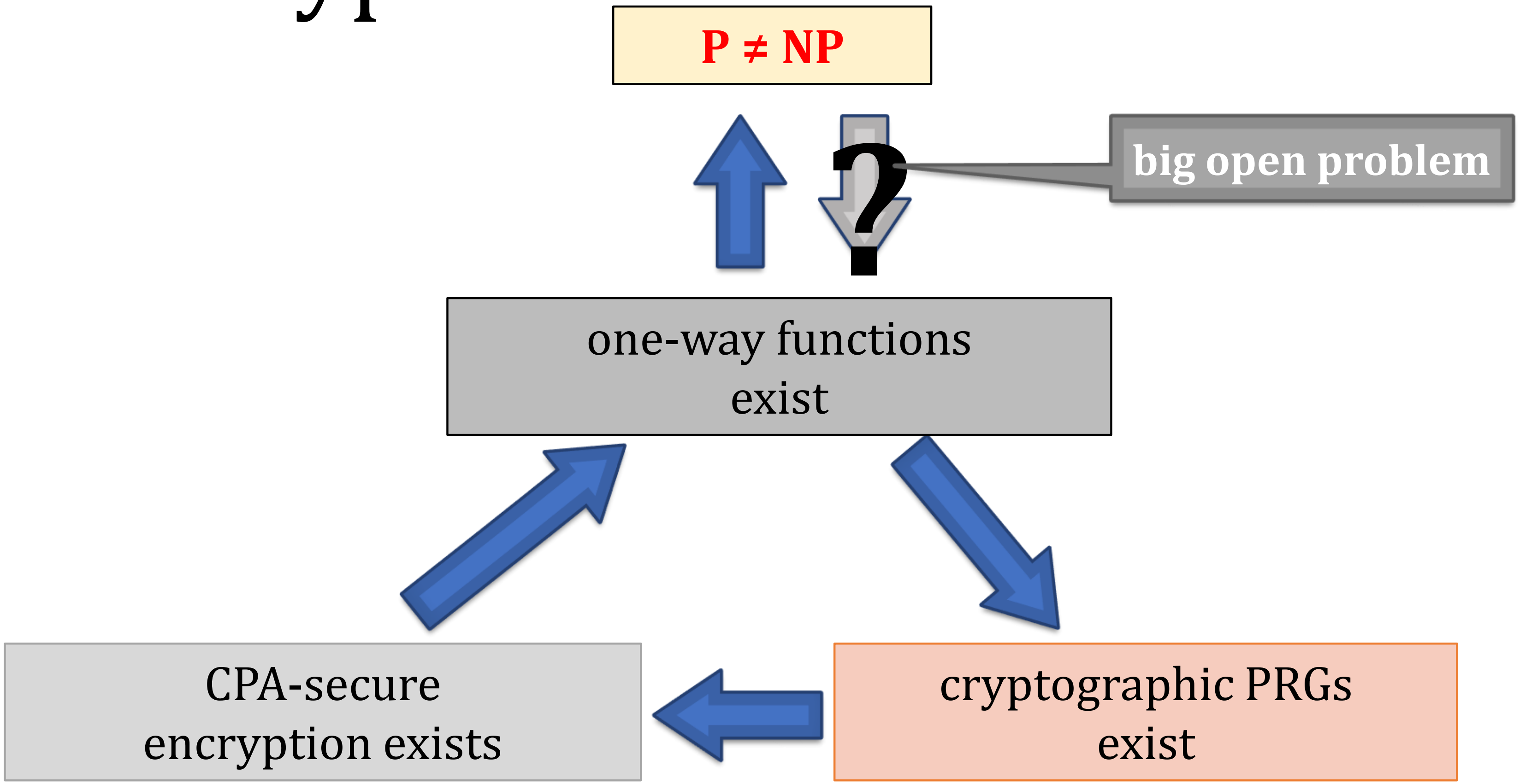


# The implication also holds in the other direction




$\boldsymbol{f(K) \;=\; \mathbf{Enc}(K, (0, \ldots, 0))}$ is a one-way function

# “Minicrypt”



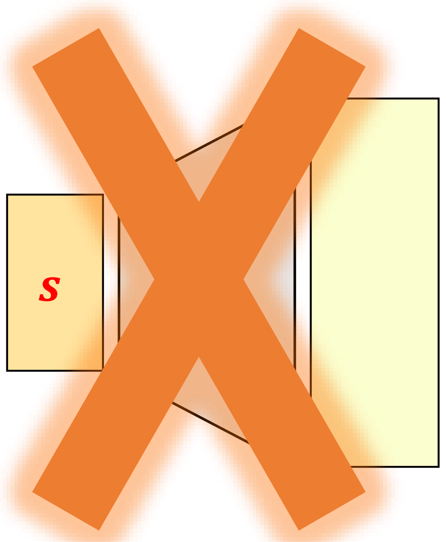

The “world” where the one-way functions exist is called “**minicrypt**”.

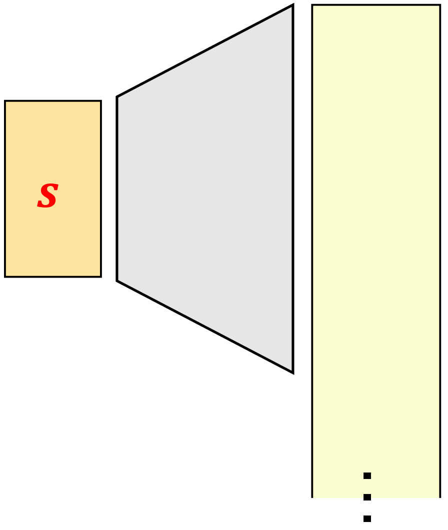
# Plan

1. If semantically-secure encryption exists then **P ≠ NP**
2. A proof that "the PRGs imply secure encryption"
3. Theoretical constructions of PRGs
4. Stream ciphers

# Stream ciphers

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

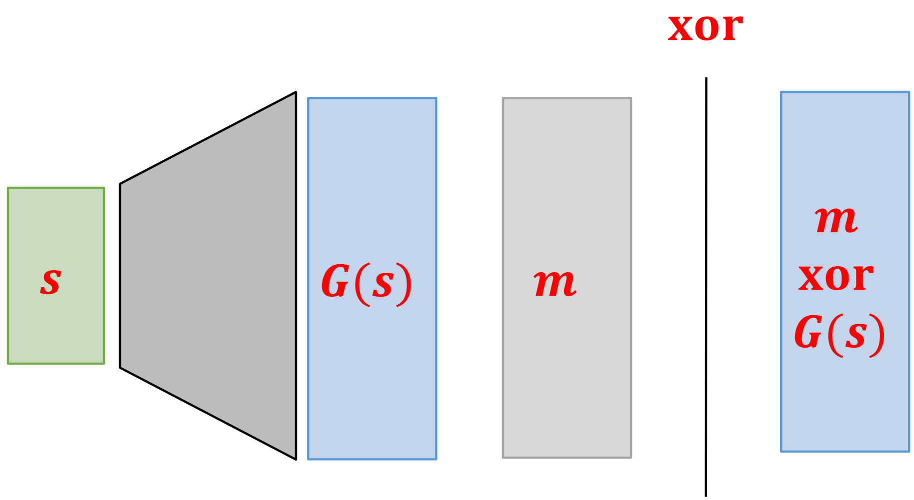



They are called like this because their output is an "infinite" **stream** of bits.

# How to encrypt multiple messages using pseudorandom generators?

$\mathbf{Enc}(s, m)$




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

**It is <u>not</u> just a theoretical problem!**

# Misuse of RC4 in Microsoft Office
## [Hongjun Wu 2005]

**RC4** – a popular PRG (or a “stream cipher”)

> **“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:

$\mathbf{Enc}(s, m)$ →

← $\mathbf{Enc}(s, m')$

**The adversary can compute $m$ xor $m'$**

# What to do?

There are two solutions:

1. The **synchronized mode**

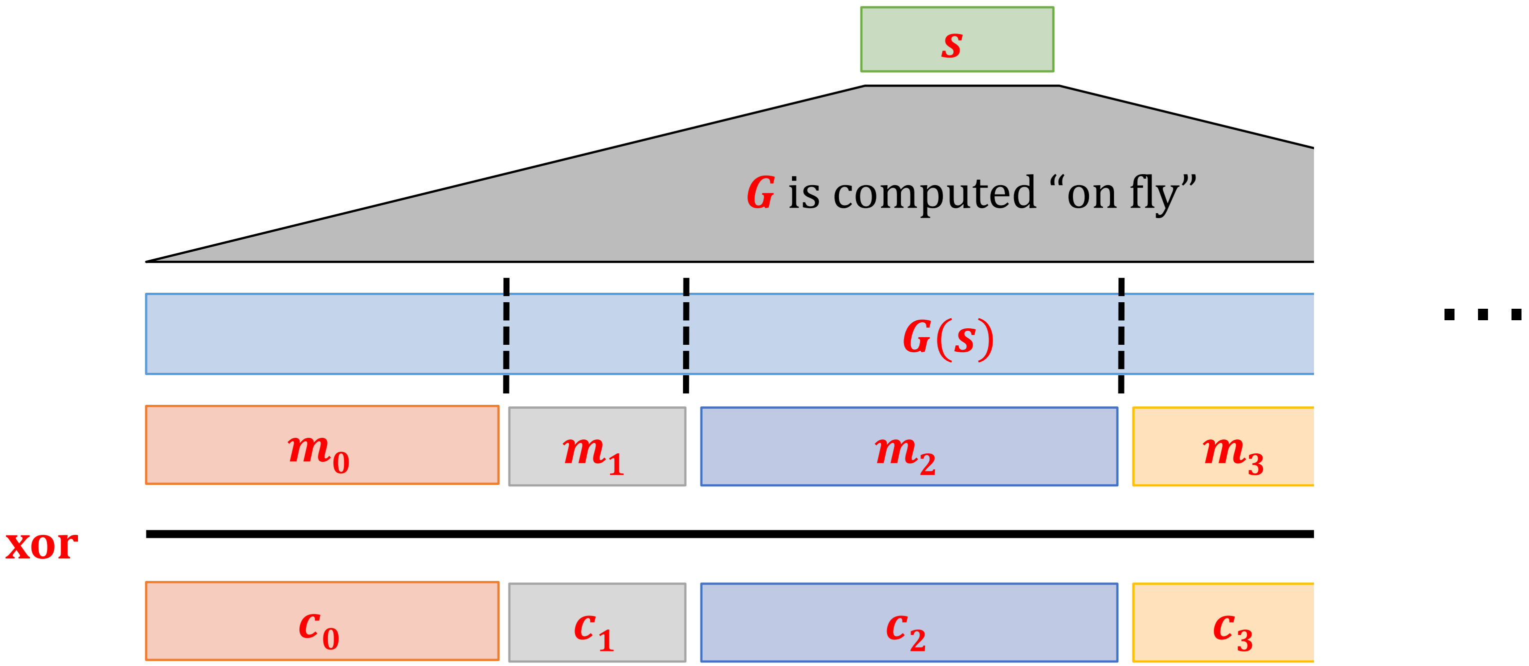
2. The **unsynchronized mode**

# How to encrypt several messages

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

this can be proven to be CPA-secure

divide $\boldsymbol{G(s)}$ in blocks:



# Unsynchronized mode

## Idea

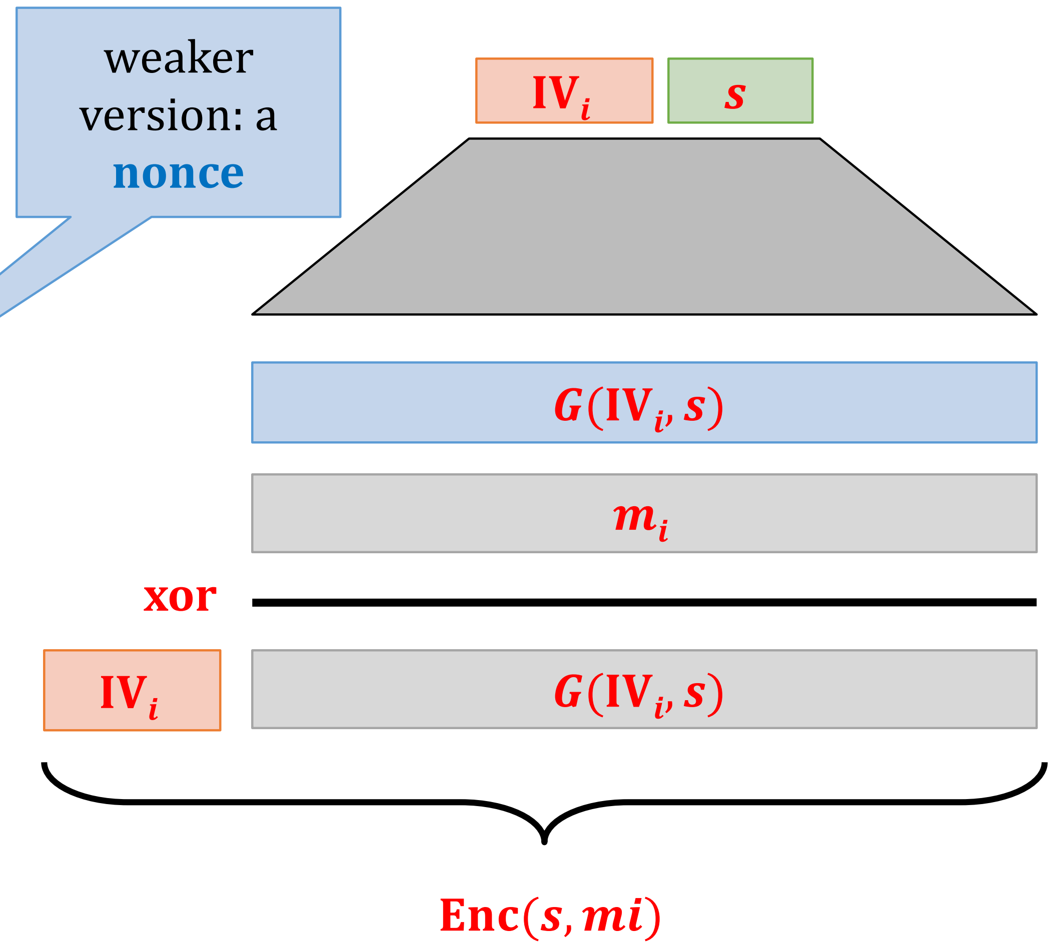
**Randomize the encryption procedure.**

Assume that $G$ takes as an additional input

an **initialization vector** (**IV**).

The **Enc** algorithm selects a fresh random $\mathbf{IV}_i$ for each message $\mathbf{m}_i$.

Later $\mathbf{IV}_i$ is included in the **ciphertext**



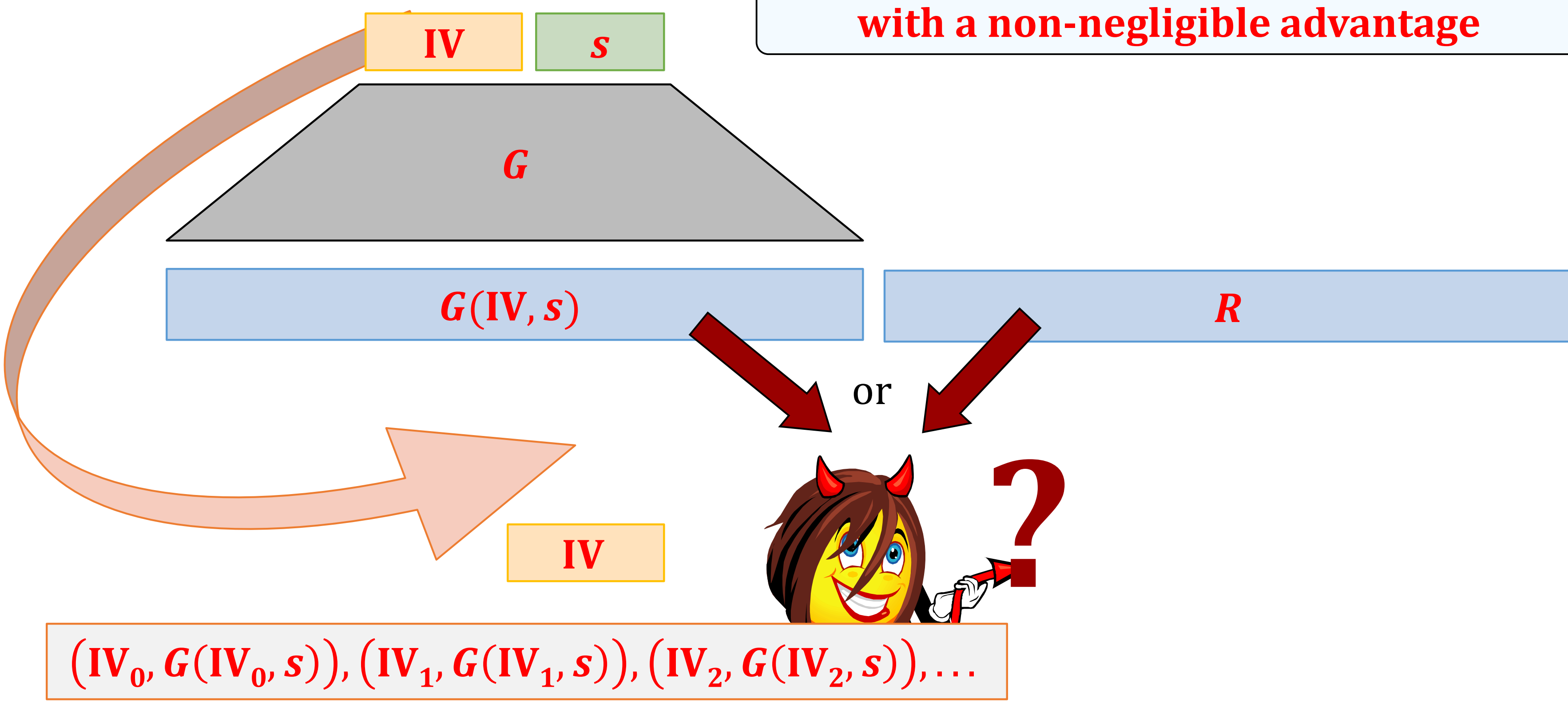
# We need an "augmented" PRG

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

$$\left(\mathrm{IV}_0, G(\mathrm{IV}_0, s)\right), \left(\mathrm{IV}_1, G(\mathrm{IV}_1, s)\right), \left(\mathrm{IV}_2, G(\mathrm{IV}_2, s)\right), \ldots$$

where $s, \mathrm{IV}, \mathrm{IV}_0, \mathrm{IV}_1, \mathrm{IV}_2 \ldots$ are random.

**with a non-negligible advantage**



# How to construct such a PRG?

An **old-fashioned approach**:

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

**often**:
**just concatenate**
**IV and $\boldsymbol{S}$**

where $\boldsymbol{H}$ is a "hash-function" (we will define cryptographic hash functions later)

A more **modern approach**:

design such a $\boldsymbol{G}$ 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:

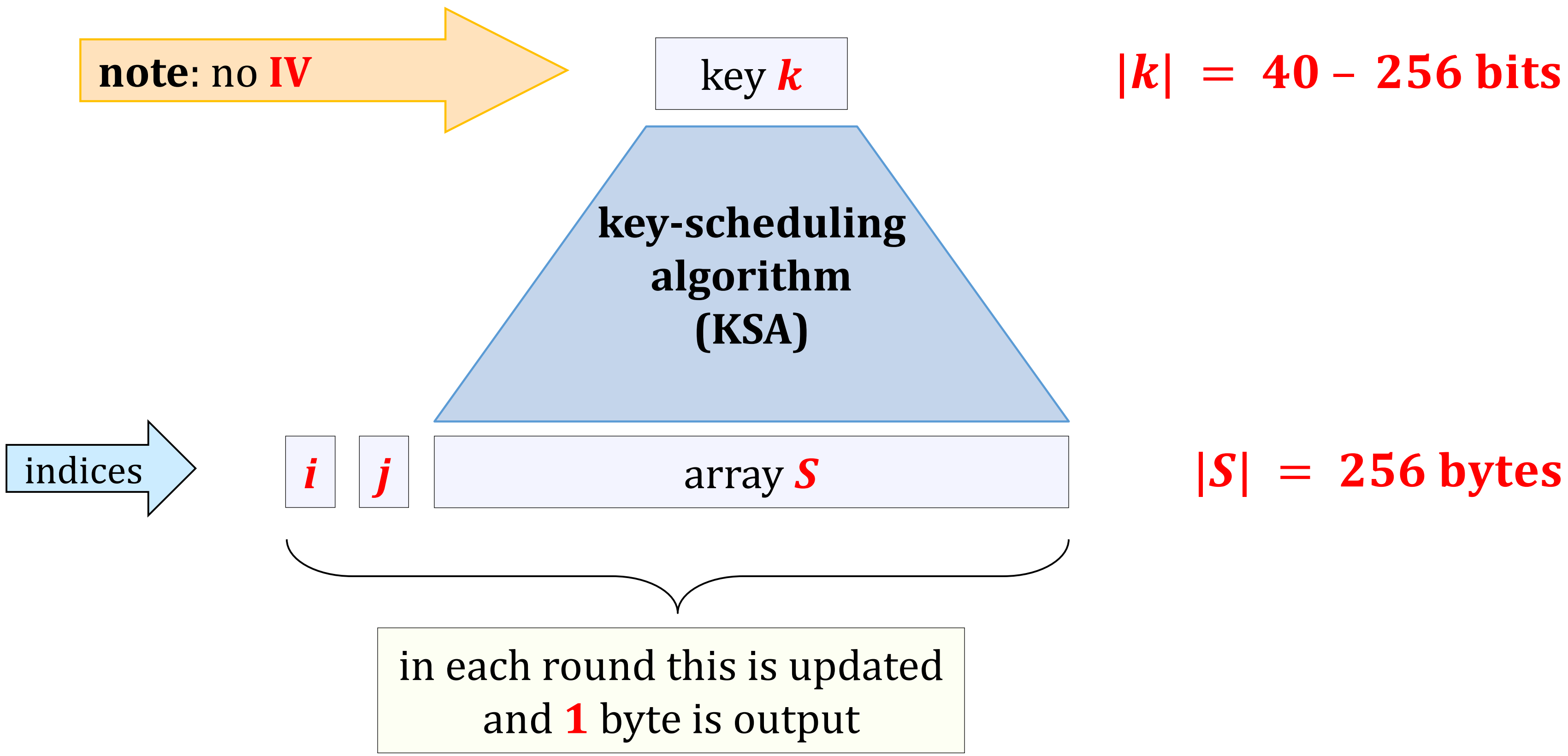
- **RC4**

very popular, but has some security weaknesses

# 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 some **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 keylength]) mod 256
        swap(S[i],S[j])
endfor
```

**don't read it!**

**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
```

# 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**, still widely used to protect **WiFi** communication.

- How **RC4** is used:

  to get the **seed**, the key $\boldsymbol{k}$ 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.*]

# These attacks are practical!

[Fluhrer, Mantin and Shamir, 2001] attack:


Using the **Aircrack-ng** tool one can break WEP in 1 minute (on a normal PC)

[see also: Tews, Weinmann, Pyshkin
*Breaking 104 bit WEP in less than 60 seconds,* 2007]

# How bad is the situation?

**RC4** is still rather secure if used in a correct way.

**Example**:

Wi-Fi Protected Access (**WPA**) – a successor of **WEP**:

several improvements (e.g. **128-bit key** and a **48-bit IV**).

# 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 $\mathbf{2^{36}}$ 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.

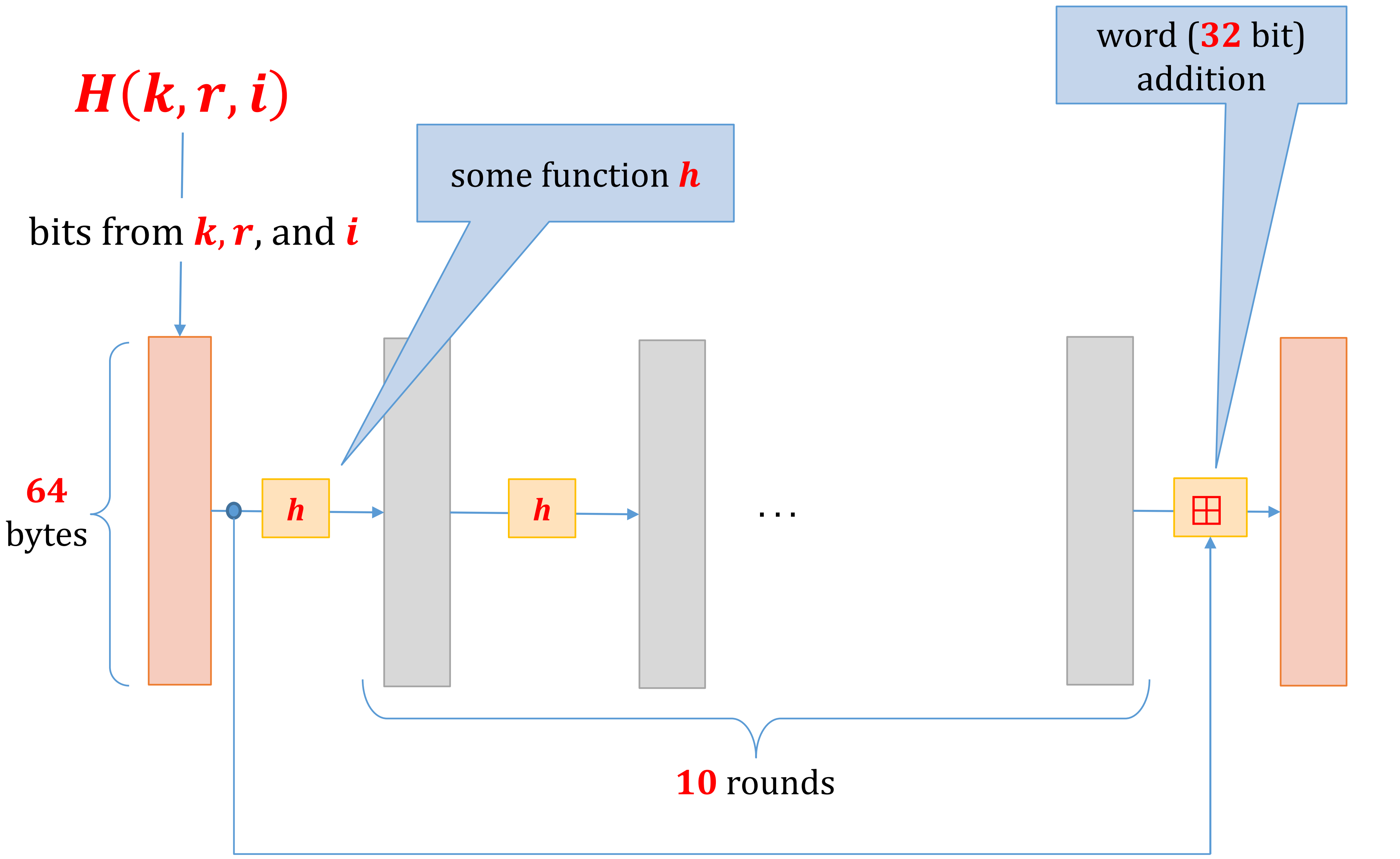
**Very efficient** both in **hardware** and in **software**.

**key** $\boldsymbol{k}$
(size: **256** bits)

$$\mathbf{Salsa20}(\boldsymbol{k}, \boldsymbol{r}) := \boldsymbol{H}(\boldsymbol{k}, \boldsymbol{r}, \mathbf{0}) \| \boldsymbol{H}(\boldsymbol{k}, \boldsymbol{r}, \mathbf{1}) \| \cdots$$

**nonce** $\boldsymbol{r}$
(size: **64** bits)

# How is $H$ defined?



# Benchmarks

| Algorithm | MiB/Second | Cycles Per Byte | Microseconds to Setup Key and IV | Cycles to Setup Key and IV |
|---|---|---|---|---|
| **Salsa20/12** | 643 | 2.7 | 0.483 | 884 |
| **Sosemanuk** | 727 | 2.4 | 1.240 | 2269 |
| **RC4** | 126 | 13.9 | 2.690 | 4923 |

https://www.cryptopp.com/benchmarks.html
*"All were 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. x86/MMX/SSE2 assembly language routines were used for integer arithmetic, AES, VMAC, Sosemanuk, Panama, Salsa20, SHA-256, SHA-512, Tiger, and Whirlpool"*

# Is there an alternative to the stream ciphers?

Yes!

the **block ciphers**

©2016 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*.