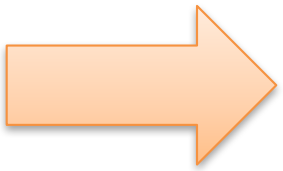
Information Security

Symmetric Cryptography 1

(Pseudorandomness, Stream Cipher)

Srđan Čapkun

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: IND-CPA

(Enc,Dec) – an encryption scheme

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 $c := Enc(k, m_b)$

has to guess **b**

← **c**

Security definition:

We say that **(Enc,Dec)** is **semantically-secure** if any **polynomial time** adversary guesses **b** correctly with probability at most **0.5 + ε(n),** where **ε** is negligible.

Is it possible to have provable IND-CPA encryption?

(…without relying on a "hardness" assumption!)

Bad news:

Theorem

If IND-CPA encryption exists (with **|k| < |m|**)

then

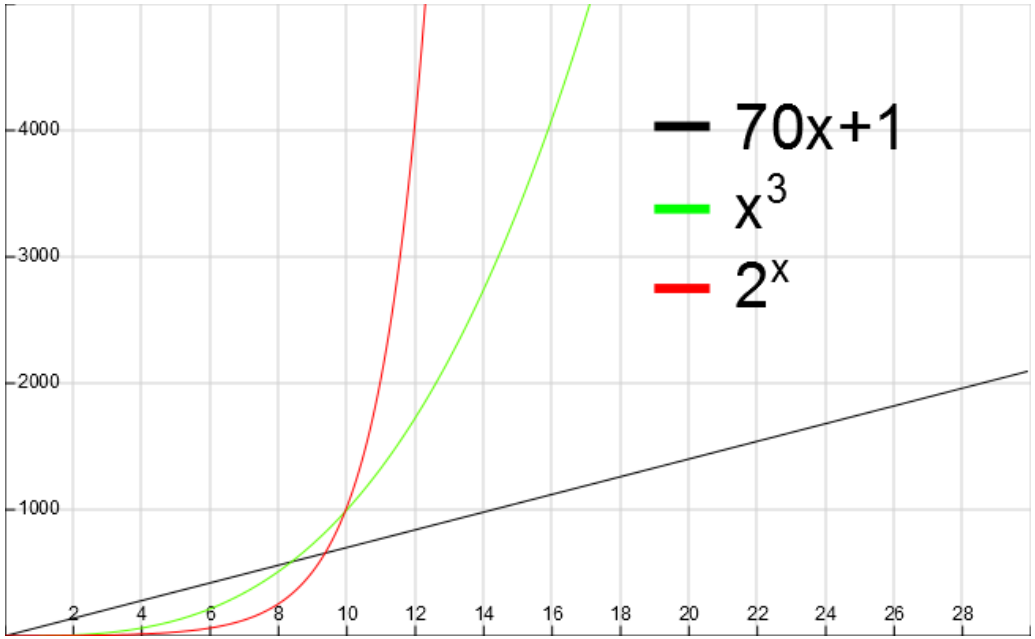
P ≠ NP

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

P vs. NP

- P: set of problems that can be **solved** in polynomial time
  - Polynomial in the size of the input
    - E.g., input size n → time $3n^2+$
- NP: set of problems that can be **verified** in polynomial
  - Given an answer
  - Example **subset-sum** problem
    - Given a set of n integers, is there a subset that sums to zero?
    - Hard to solve (no poly-time algorithm exists)
      - Known algorithms run in $O(2^{n/2})$
    - Easy to verify

Polynomila vs. Exponential



Why is this a problem?

- P vs. NP (P = NP or P ≠ NP) one of oldest unsolved problems in computer science
- Many people have unsuccessfully tried to solve it
- If provable IND-CPA encryption **(Enc_k, Dec_k)** proves P≠NP

→ gives evidence that might be actually quite difficult to find **(Enc_k, Dec_k)**

  – Without hardness assumption!

Proof [1/5]

Let **(Enc,Dec)** be an IND-CPA encryption scheme.
For simplicity suppose that **Enc** is deterministic.

Consider the following language:

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

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

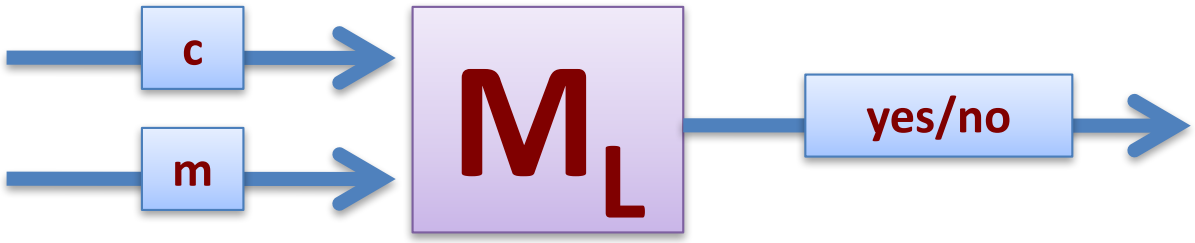
Clearly **L** is in **NP**. (**k** is the **NP**-witness)

NP-Turing machine guesses k

Proof [2/5]

Suppose **P=NP**.

Therefore, there exists a deterministic poly-time Turing machine $\mathbf{M_L}$ 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.**

|k|<|m|

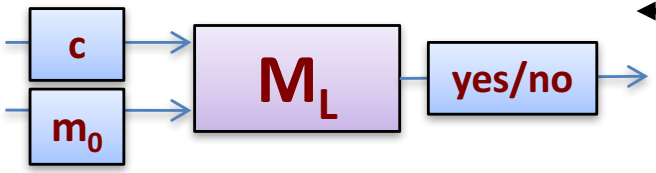
Suppose **P = NP** and hence **L** is poly-time decidable.

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

m_0, m_1

1. selects **k** randomly
2. chooses a random **b = 0,1**
3. calculates **c := Enc(k,m_b)**

c



If "yes"/ **(c, m_0) ϵ L** then output **0**
else output **1**

Observation

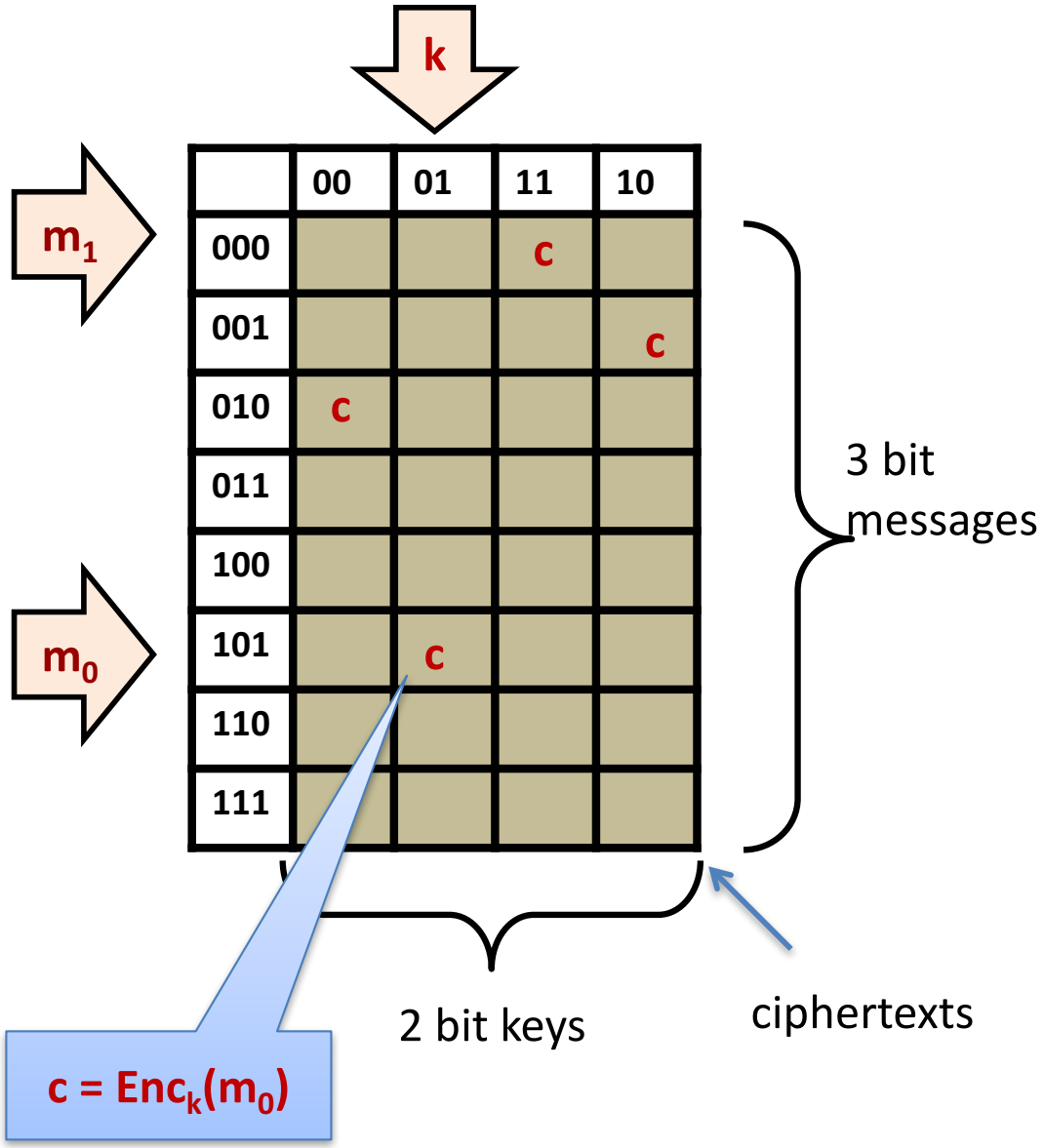
The adversary guesses incorrectly, if **b=1**, and there exists **k'** such that

Enc(k,m_0) = Enc(k',m_1)

What is the probability **p** that this happens?

Proof [4/5]

Example: 2 bit key, 3 bit message



From the correctness of encryption: $c = Enc_k(m_0)$ can appear in each column **at most once**. (Otherwise, same **c** with same **k** decrypts to different **m**.)

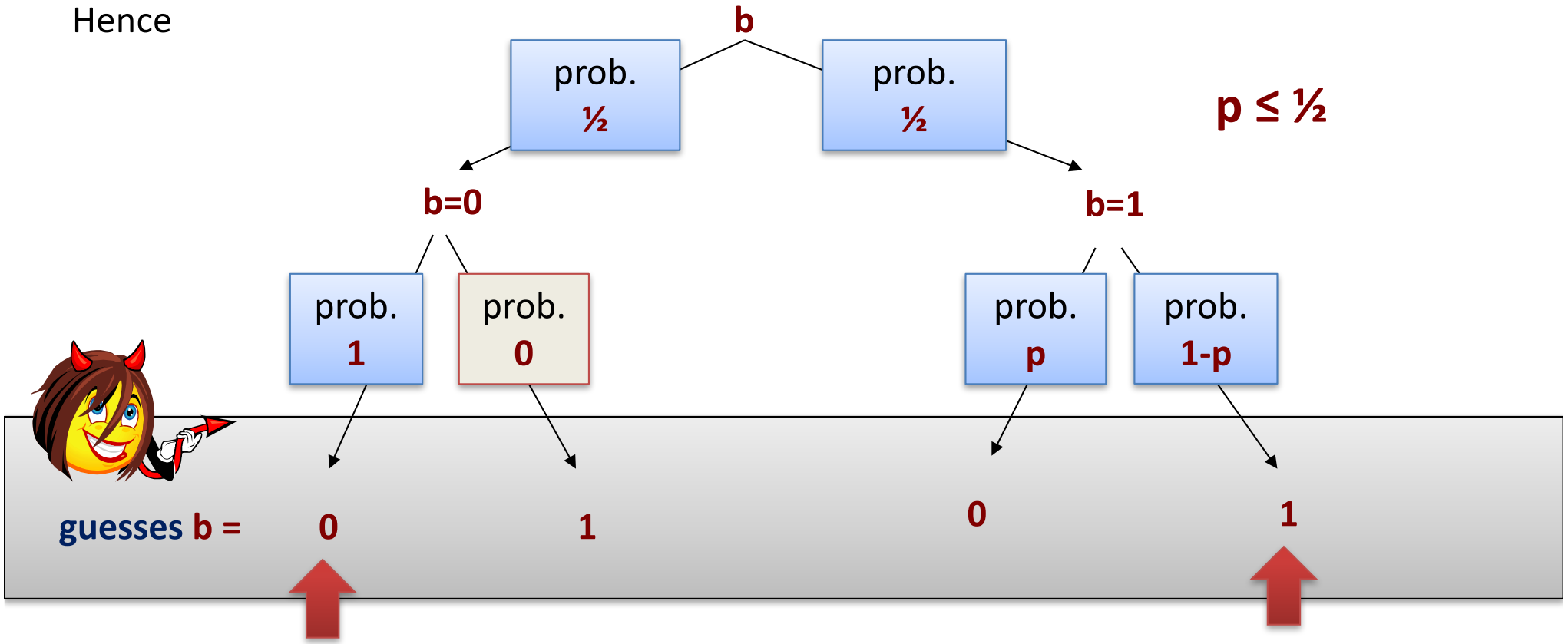
There are **|K|** columns.
→ **c** can be up to **|K|** times in the table.

Hence, the probability **p** that **c** appears in a randomly chosen row is **at most**:

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

Proof [5/5]

Hence



probability of a **correct guess**:

½ + ½ (1-p) ≥ ¾ ← non-negligibly larger than 1/2

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

Conclusion

- Finding IND-CPA encryption (without hardness assumption) is at least as difficult as proving 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.

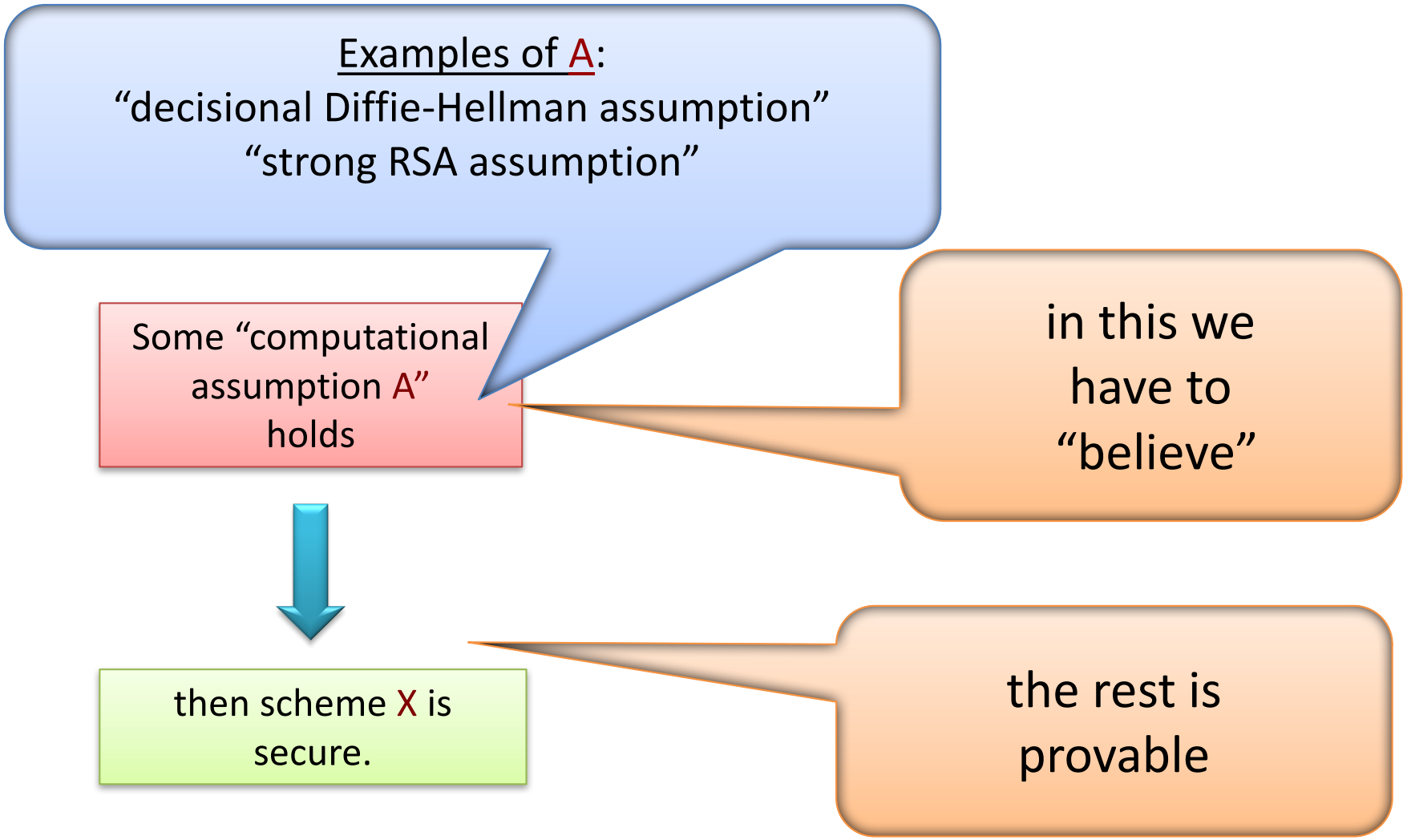
Suppose that some scheme **Y** is secure

↓

then scheme **X** is secure.

Modern cryptography

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

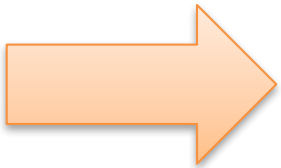


Example

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

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

Plan

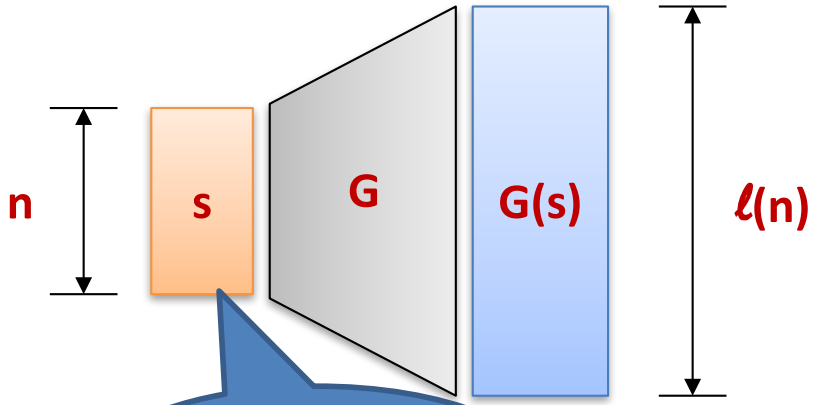


1. If semantically-secure encryption exists then **P ≠ NP**
2. A proof that "the PRGs imply secure encryption"
3. Theoretical constructions of PRGs
4. Stream ciphers

PRG: Pseudo Random Generator

- You all know PRG!
- C(++): random()
- Java: Java.util.Random, nextBytes()
- Python: random.py, getrandbits()
- There is an input “seed” involved
  - randomize(), Random.setseed(), random.seed()
  - Seed is really random (system time)
- …a little bit more formal?

Pseudorandom generators



"expansion factor"

Definition

ℓ – 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**".

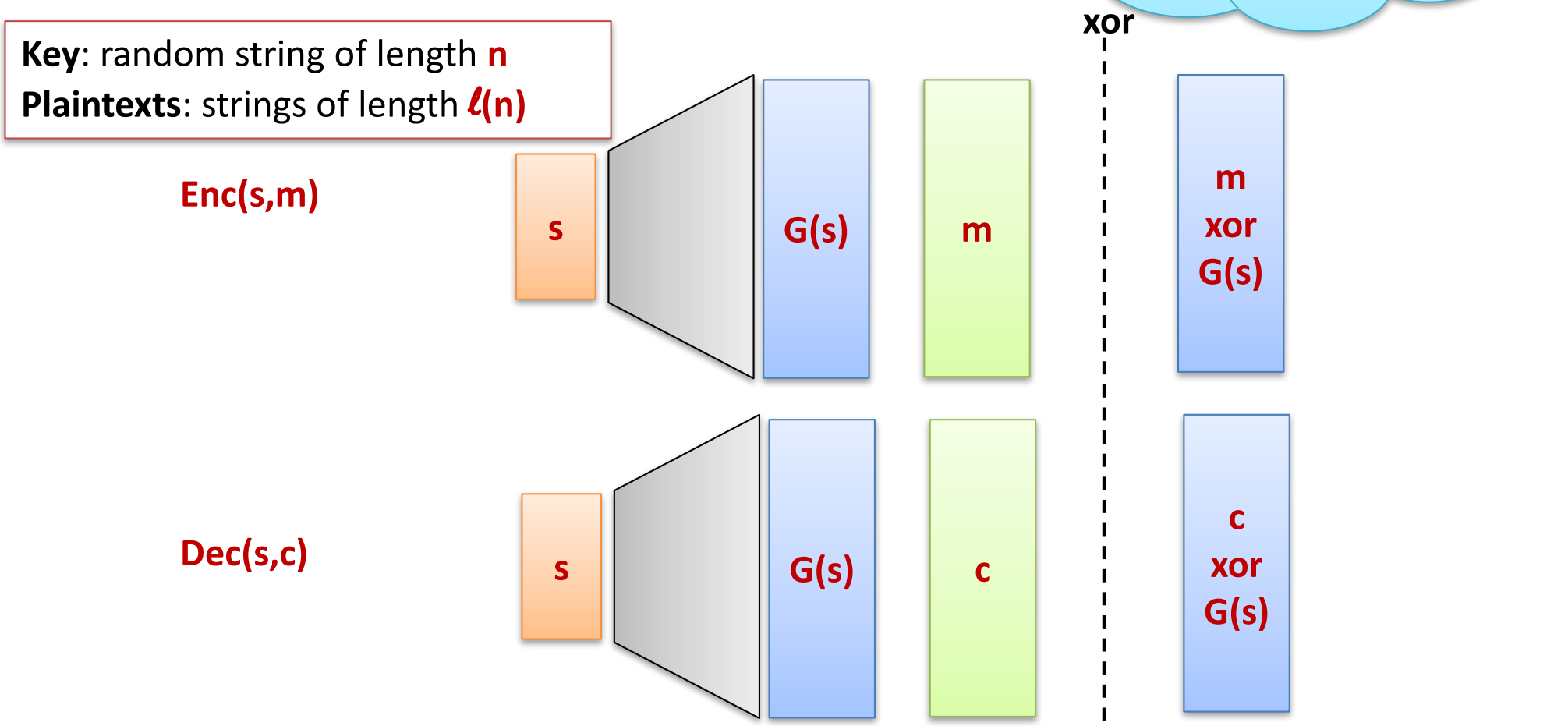
this has to be formalized

Idea: use PRG for encryption

Use PRGs to "shorten" the key in one time pads
→ Practical One-Time Pads!

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

Key: random string of length **n**
Plaintexts: strings of length **ℓ(n)**



If we use a "normal PRG" (Java/ANSI C random function) – this idea doesn't work
We have to use **cryptographic PRGs.**

Meaning of: “Looks random”?

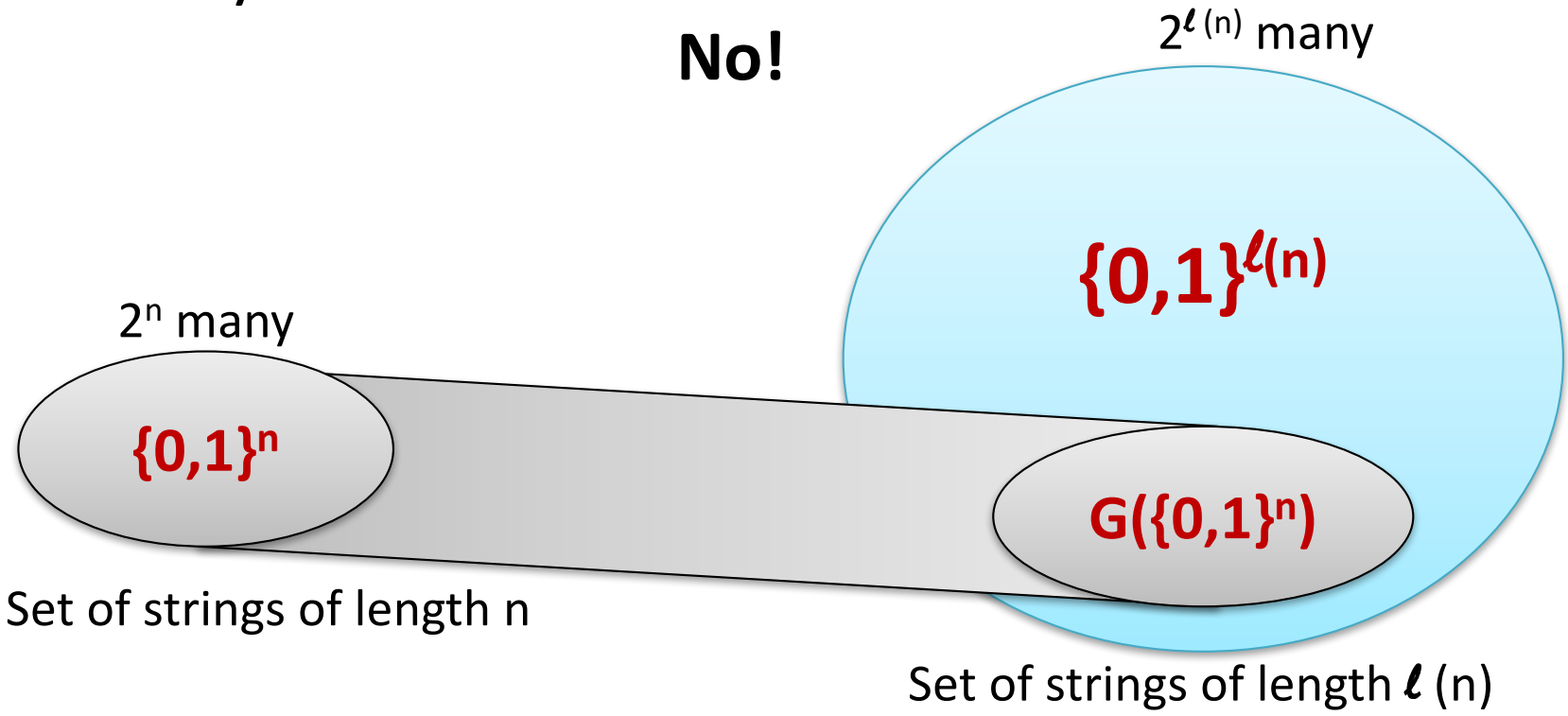
Suppose $s \in \{0,1\}^n$ is chosen randomly.

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 with high probability**.

(…2^n vs $2^{\ell(n)}$ problem does not matter)

Golomb's Postulates ("Axioms"), 1967

- 1: In **G(s)**, difference between number of 1s and 0s must be ≤1.
- Definition "**run**": a (sub)sequence of only 1s or 0s
- 2: In **G(s)**, length i runs occurs with $P=2^{-i}$. For each i, there are the same number of 0- and 1-runs
  - half of runs have length 1, one fourth 2, one eight length 3 …
- 3: {G(s),G(s)>>1,G(s)>>2…}'s autocorrelation C(i) constant for all i

$$C(i) = \frac{\#match(i) - \#diff(i)}{\ell(n)} = \frac{1}{\ell(\mathrm{n})} \sum_{j=1}^{\ell(n)} (-1)^{G(s)_j} (-1)^{G(s)_{j+i}} = \begin{cases} 1 \text{ if } i = 0 \\ c \text{ if } i \neq 0 \end{cases}$$

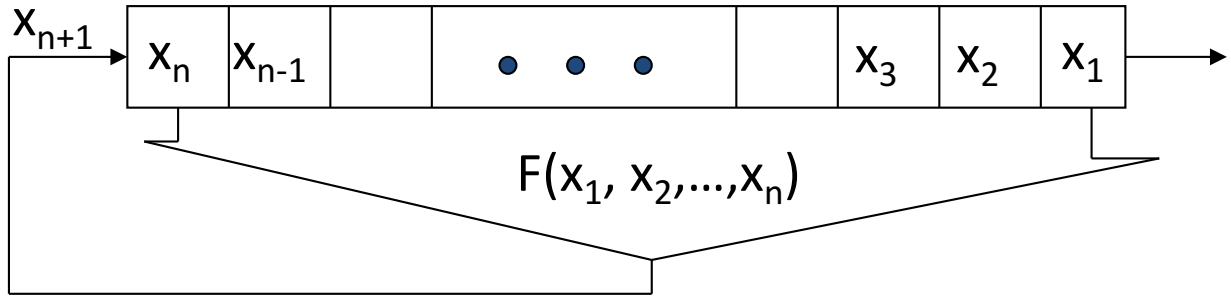
Example

1000111101011100

- G1, G2, G3?

Warning!

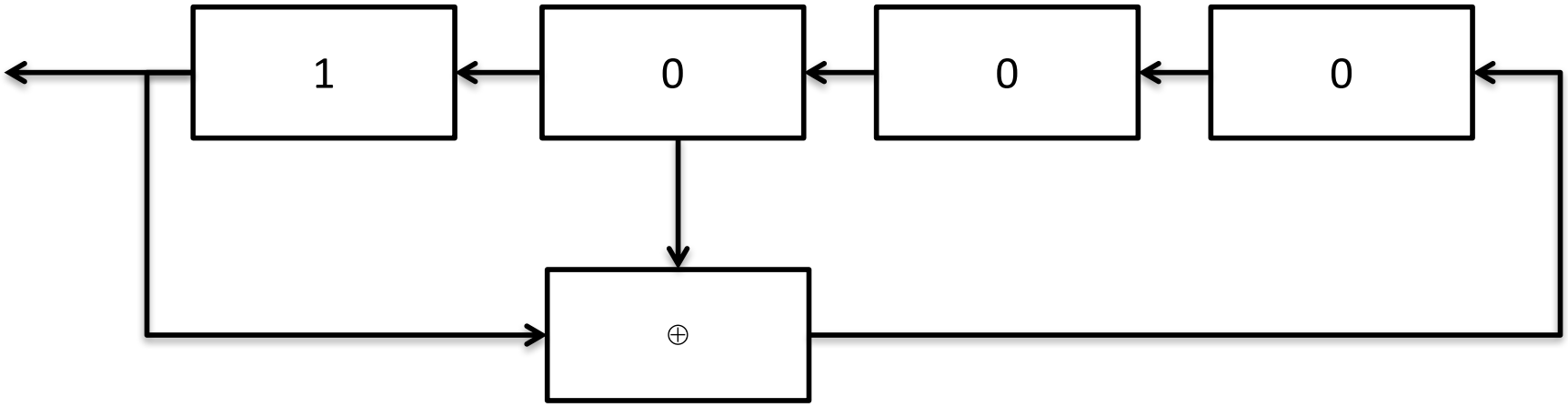
- Golomb or other (FIPS-140) not sufficient
- Famous example: Linear Feedback Shift Register



- LFSRs with proper F ("taps") pass tests
  - However, **predictable** if 2•n output bits are known
- Are postulates even necessary?!

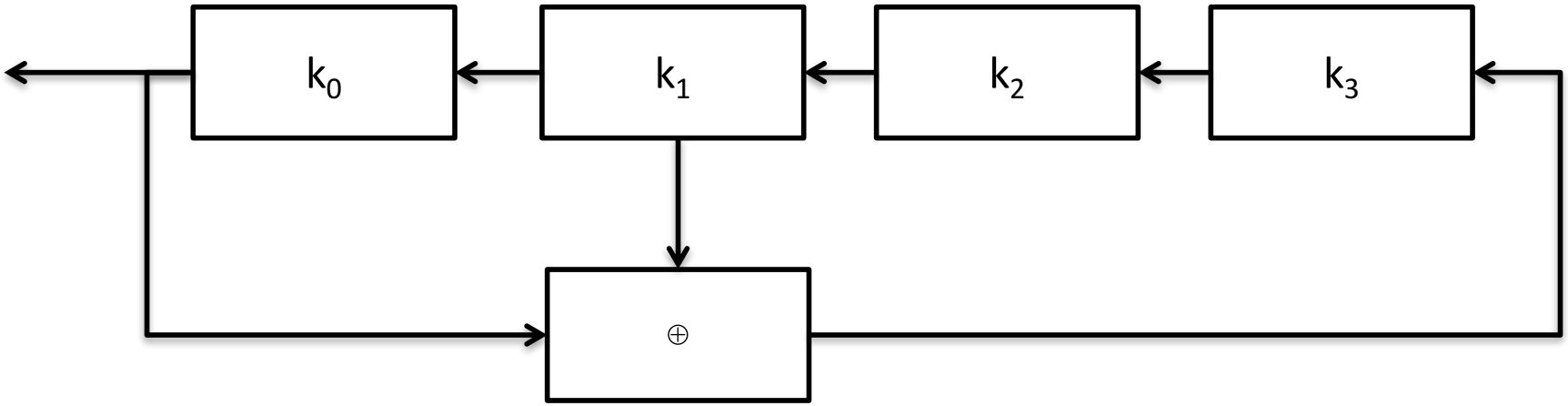
We need something stronger!

Example



- Seed is 1000
- Output is 100010011010111 …
- Repeats every 2^4-1 bits

Example



- At every iteration
  - k_0 is output
  - $K_1,\ldots,k_{m-1}$ are shifted
  - The new value of "k_m" is computed as $k_m = \sum_{j=0}^{m-1} c_j k_{j+1}$
  - Example
    - $k_5 = 1 \cdot k_0 + 1 \cdot k_1 + 0 \cdot k_2 + 0 \cdot k_3 \bmod 2$

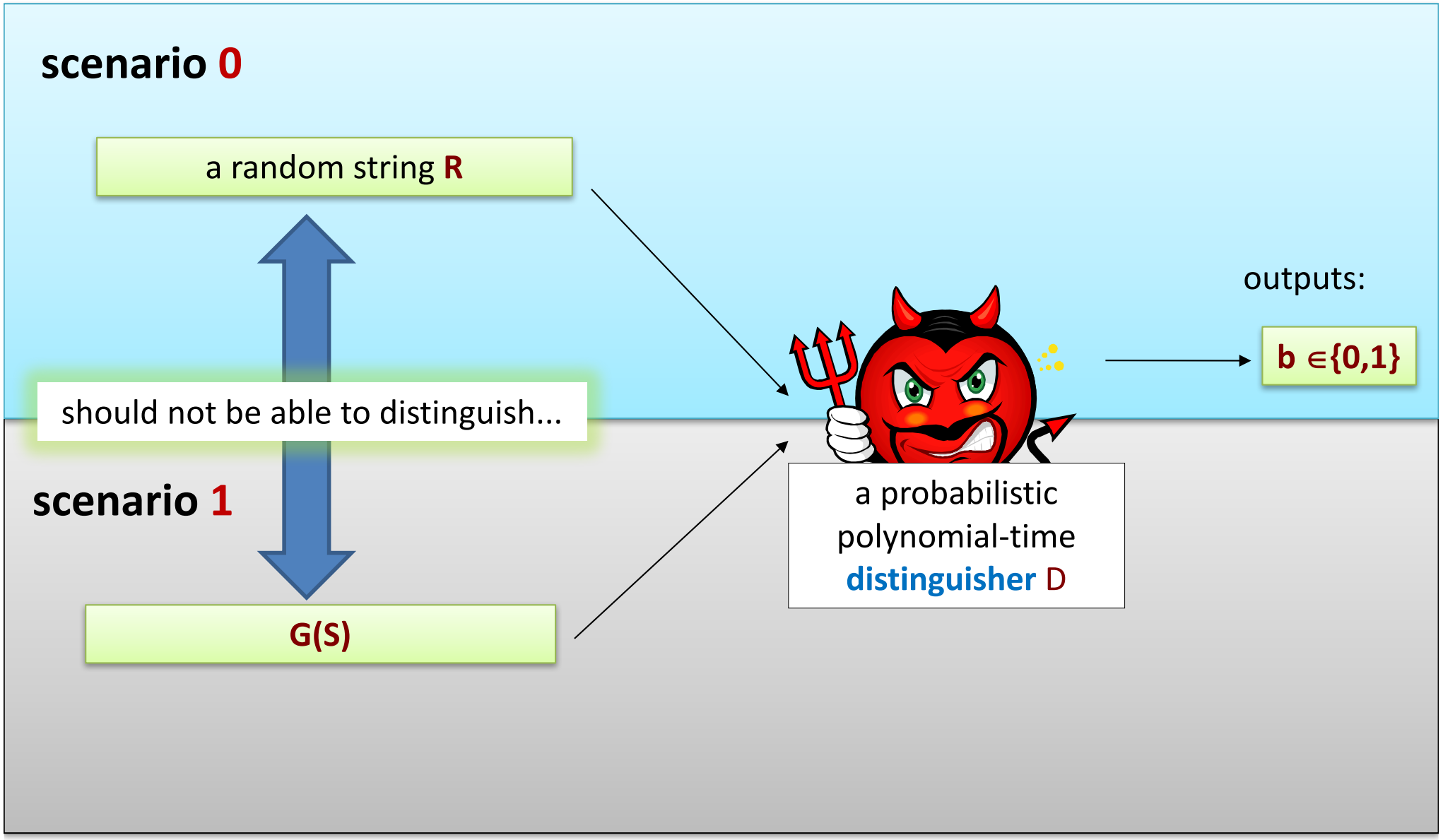
Breaking LFSR

- The LFSR is $k_m = \sum_{j=0}^{m-1} c_j k_{j+1}$
  - m linear equations with $c_0,\ldots,c_{m-1}$

$$k_4 = c_3k_3 + c_2k_2 + c_1k_1 + c_0k_0 \; mod \; 2$$
$$k_5 = c_3k_4 + c_2k_3 + c_1k_2 + c_0k_1 \; mod \; 2$$
$$k_6 = c_3k_5 + c_2k_4 + c_1k_3 + c_0k_2 \; mod \; 2$$
$$k_7 = c_3k_6 + c_2k_5 + c_1k_4 + c_0k_3 \; mod \; 2$$

  - Knowing 2m output bits ($k_0,\ldots,k_7$) one can compute the “taps”

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 (security) 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**.

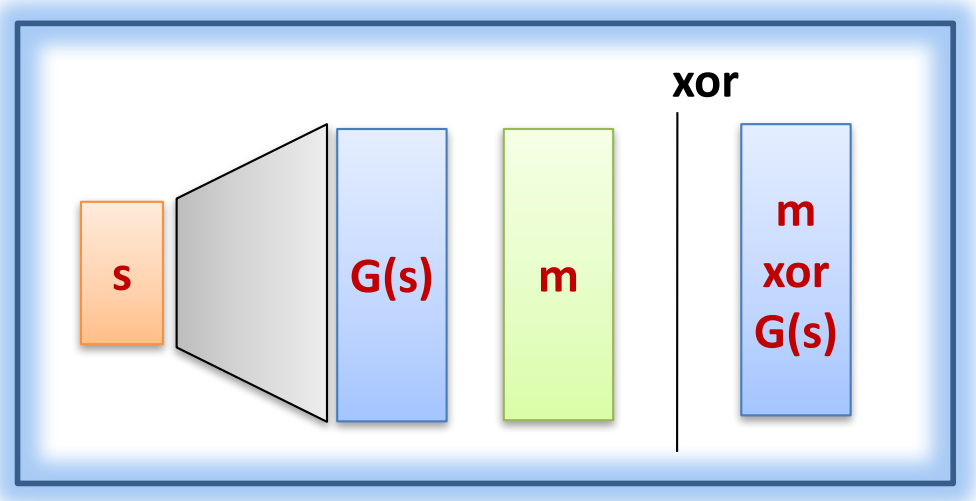
"D's **behavior** does not depend on the input."

Constructions

- There exists constructions of PRGs, conjectured to be secure
  - We will discuss later…
  - For now: **assume** there is a PRG
- Can we have secure encryption? Yes!
  - We will prove this now…

Theorem

If G is a **cryptographic PRG**,
then the encryption scheme constructed before is computationally-secure.



cryptographic PRGs exist → computationally-secure encryption exists

Proof (sketch)

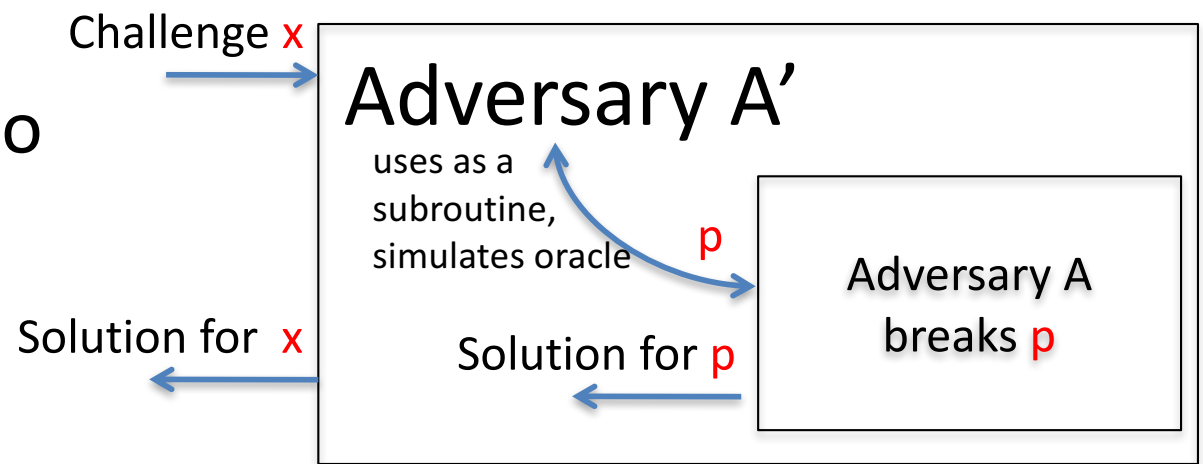
Let us concentrate on the **one message case**.

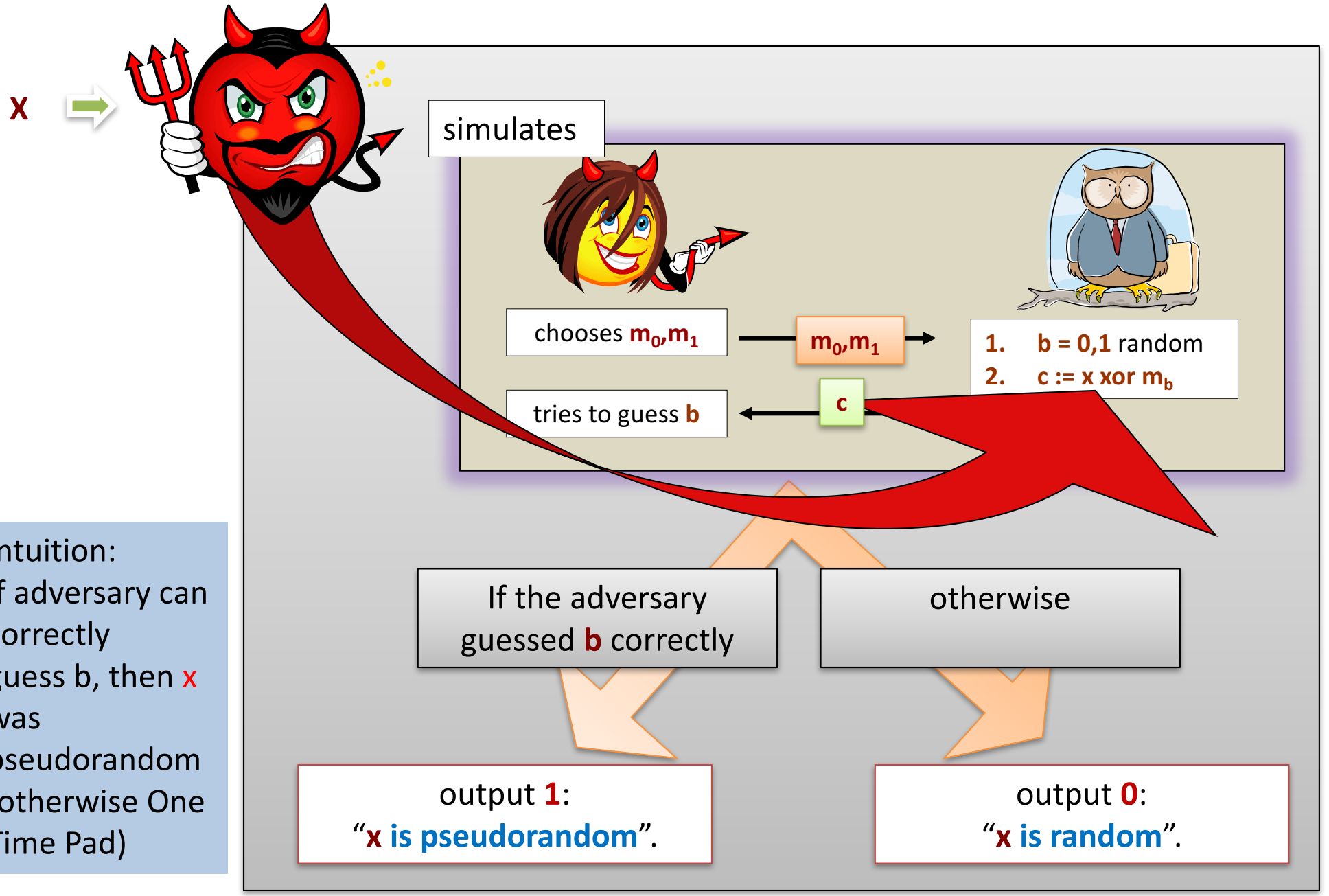
Suppose that it is **not** secure.

Therefore, there exists a poly-time **adversary** that wins the "guessing game" with probability **0.5 + ε(n)**, where **ε(n)** is not negligible.

Standard Technique: Proofs by Reduction

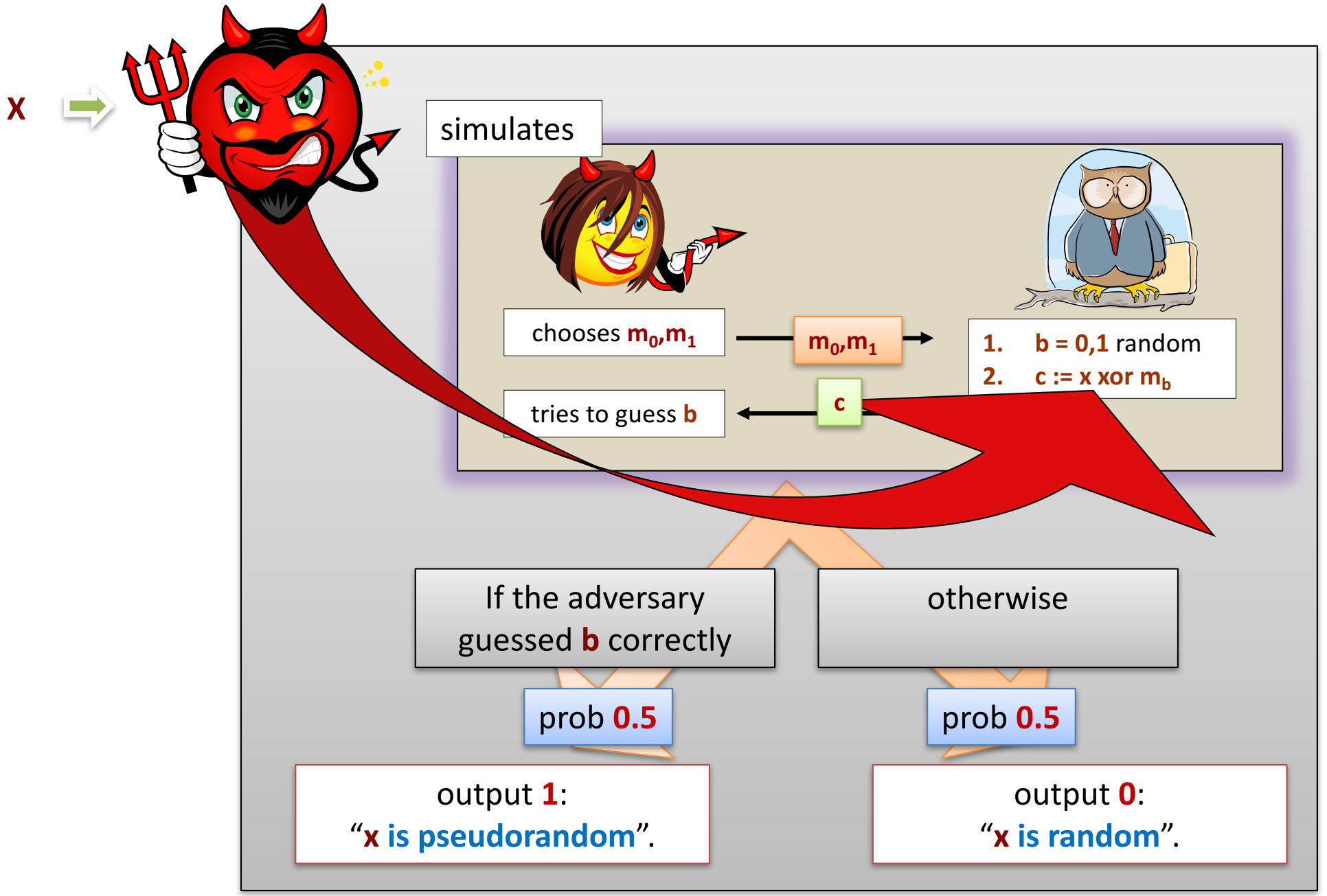
- Prove security of new mechanism p, based on old, known secure mechanism x
  - Assume existence of **adversary A** (we don't know how A works!) that wins game instances of p
  - Construct adversary **A'** to solve instances of old problem – using **A** as **subroutine**
  - A' simulates the oracle for A in x
  - A' **transforms** instances of **x** to instances of **p**



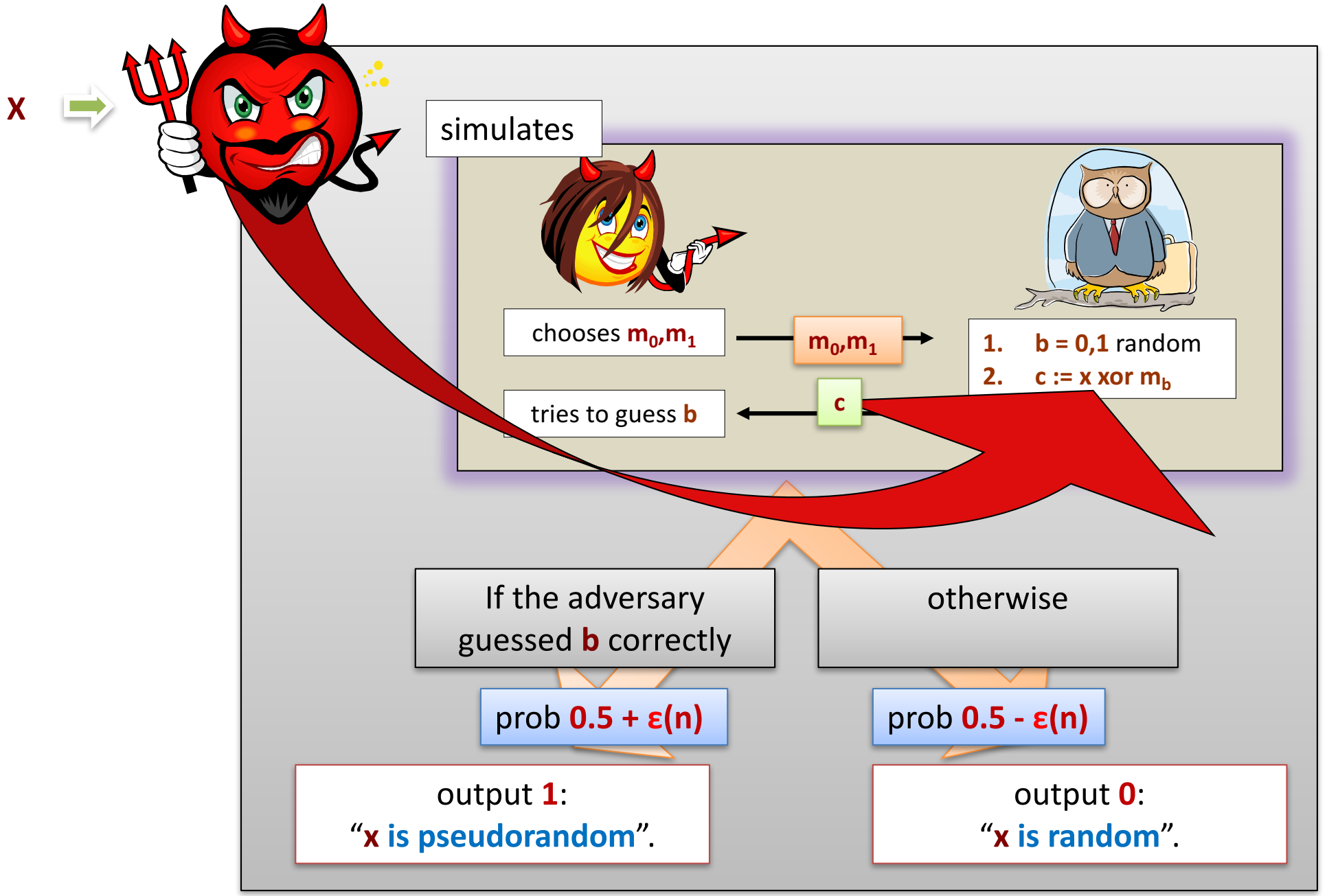


Intuition:
If adversary can correctly guess b, then x was pseudorandom (otherwise One Time Pad)

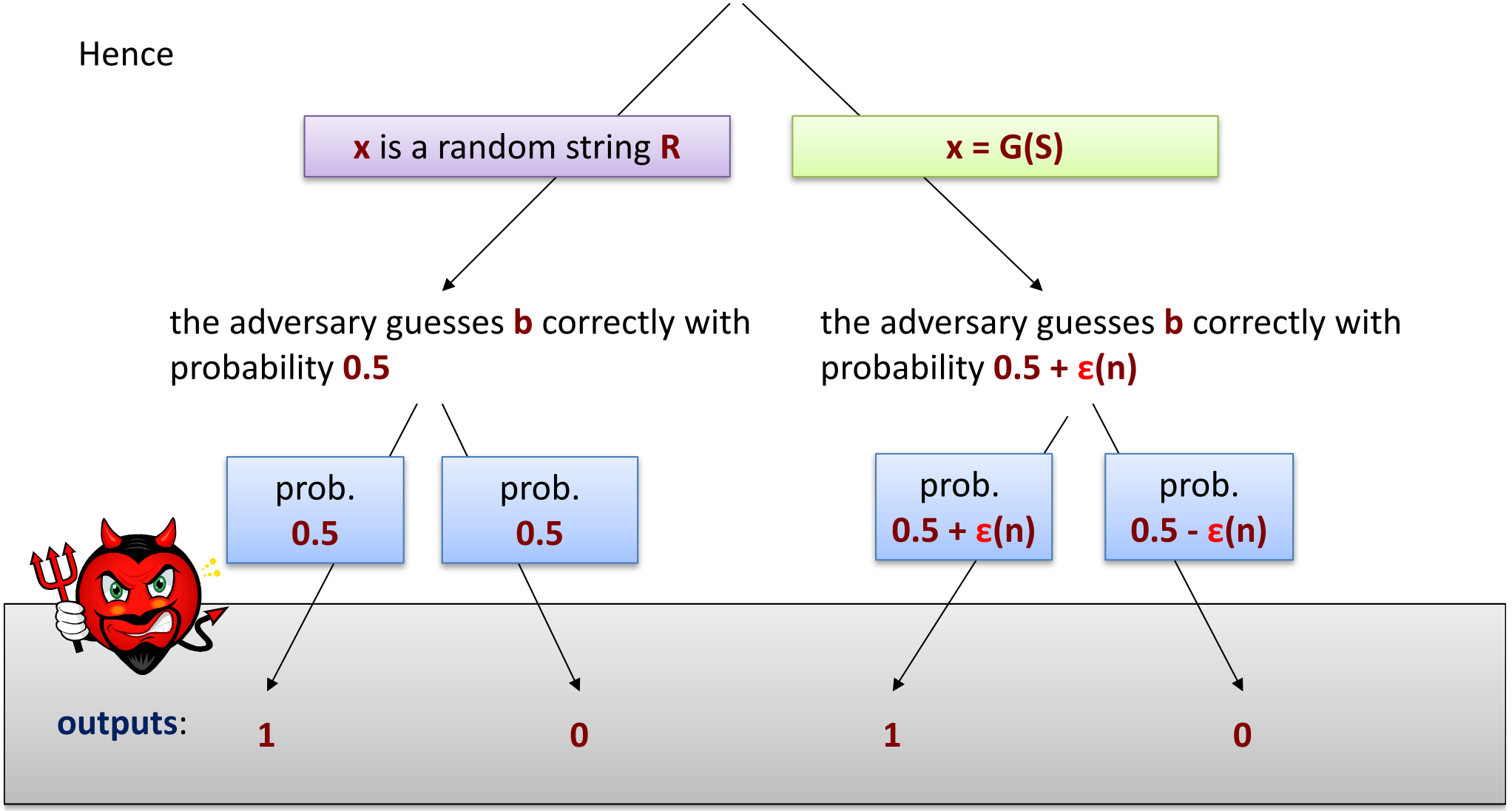
“scenario 0”: x is a random string



“scenario 1”: x = G(S)



Hence



$$\left| P(D(R) = 1) - P(D(G(S)) = 1) \right| = \left| 0.5 - (0.5 + \varepsilon(n)) \right| = \varepsilon(n)$$

Since ε is not negligible, G cannot be a **cryptographic PRG**

The complexity

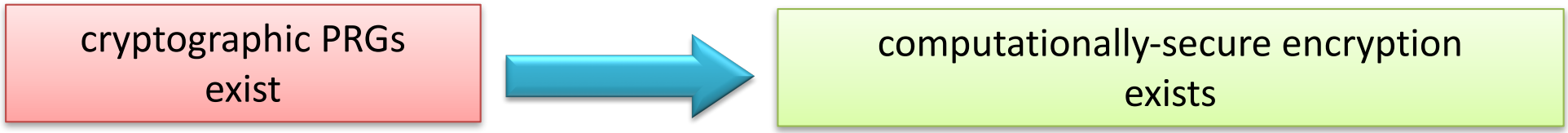
The distinguisher  simply simulated

one execution of the adversary 

against the oracle  .

Hence, he works in polynomial time.

Moral



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

Moreover, we can also state the following:

Important (but informal) Remark. The reduction is tight.

A question

What if the distinguisher  needed to simulate

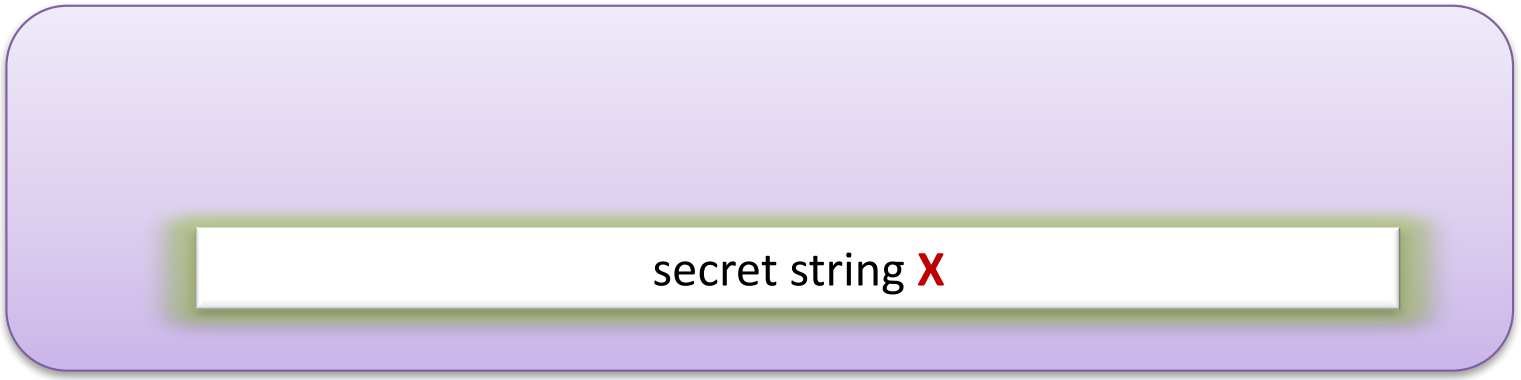
1000 executions of the adversary  ?

An (informal) answer

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

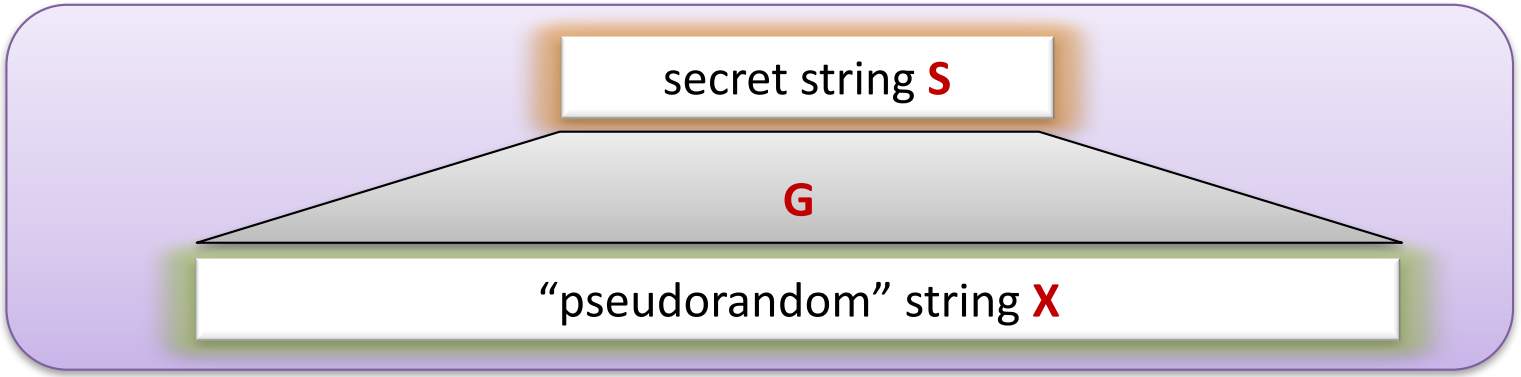
General rule

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"

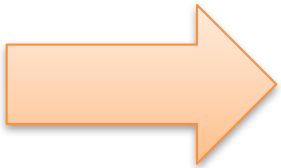
ugly, **very efficient**, **widely used in practice**

Examples: RC4, Trivium, SOSEMANUK,...

End of Class

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

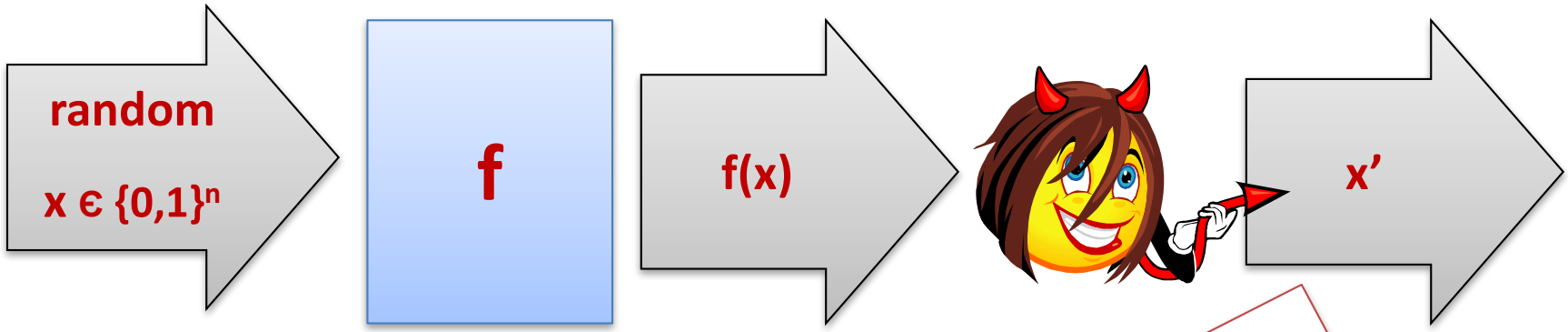


PRGs with "One-way functions"

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”.

More formally...

experiment **(machine M, function f)**

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

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

$\forall$ **probabilistic polynomial-time** Turing Machine M

P (M wins**)=ε(n)** is **negligible** in **n**

Example of (candidate for) a one-way function

If **P=NP**, then **one-way functions do not exist**. Why?

So, no function can **easily** be proven to be one-way.

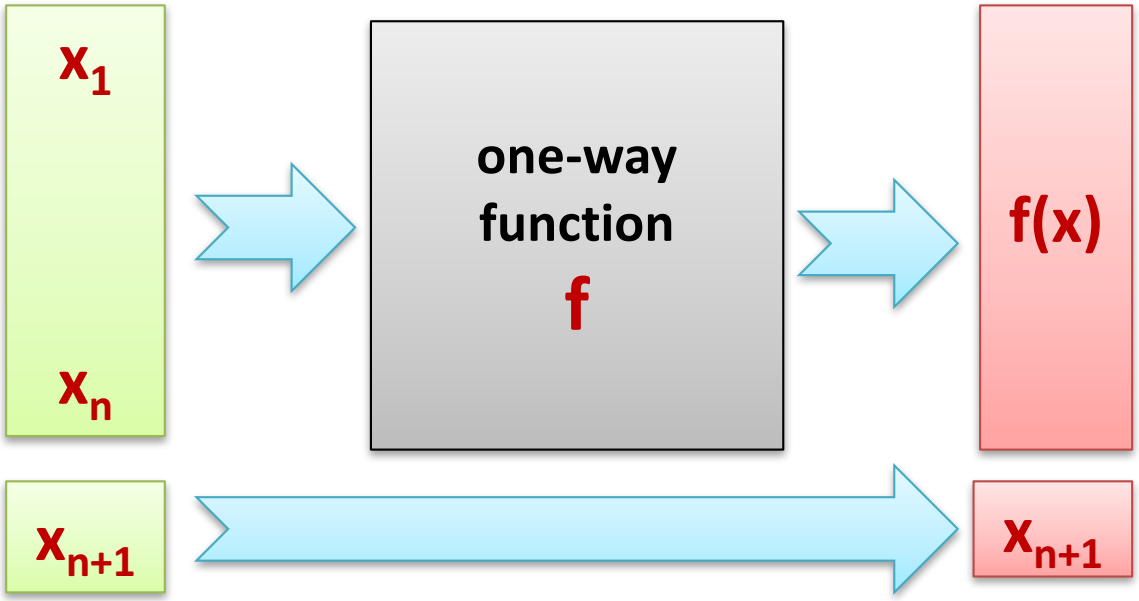
But there exist candidates based on assumptions. Example:

$f(p,q) = pq$

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

This bit is leaked.

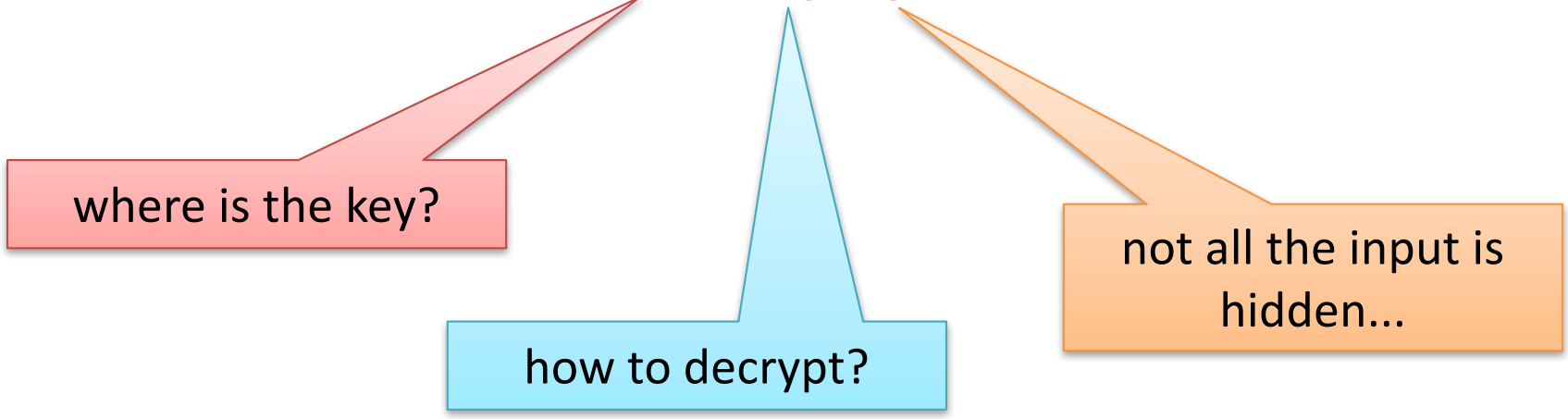
But these are not.

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

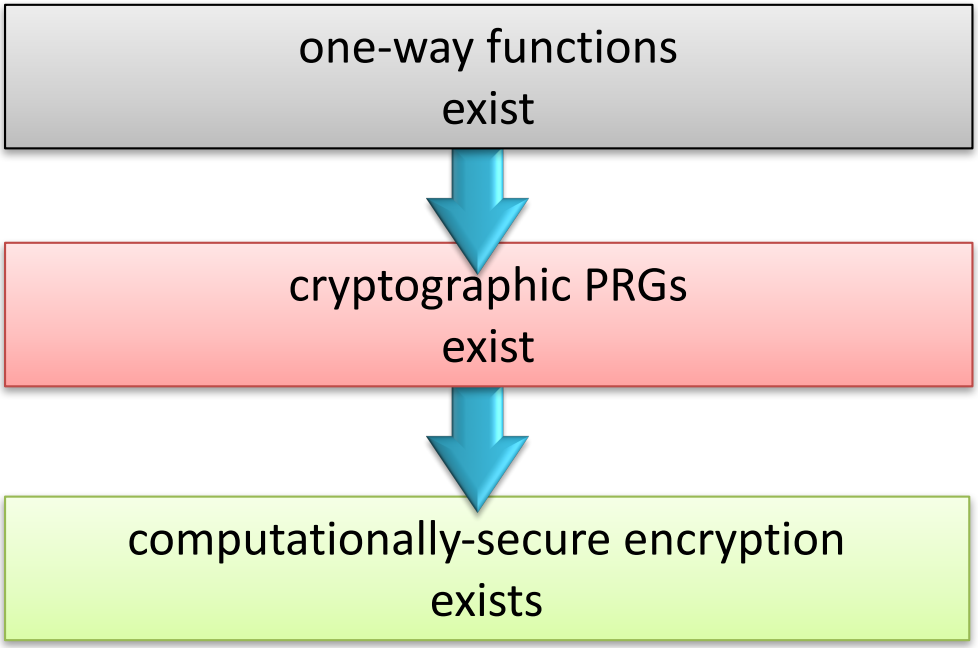
c := f(m)



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**"



Sneak Peek 1:
PRG from one-way **permutation**

- "Hard core" bit **hc** of **f(x)**
  - (Informal): "**1 bit** property of **x**" that is difficult to compute for adversary
- Examples HC properties (predicates)
  - Given **f(x)**, compute Least Significant Bit (**LSB**) of **x**
    - **$f(x) = x^2 \bmod n=p\bullet q$**, **LSB(x)** is HC bit for **f**, **hc(x)=LSB(x)**
  - Given **$f(x_1,\ldots,x_n)=x_2,\ldots,x_n$**, **$hc(x)=x_1$** is HC bit.

But f is not one way!

But not for other f!

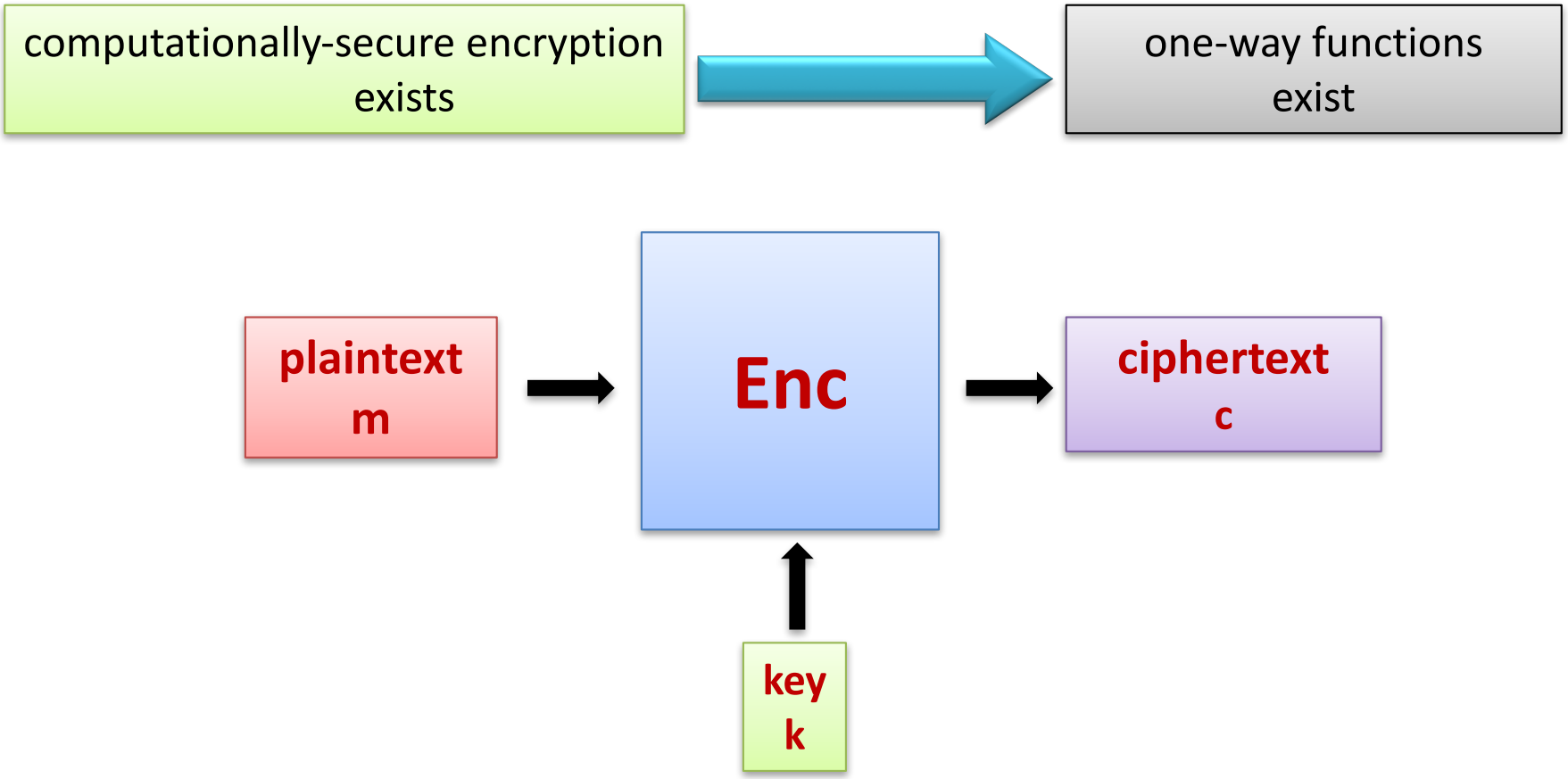
$\forall$ **probabilistic polynomial-time** Turing Machine M

$$P[M(f(x)) = hc(x)] \leq \frac{1}{2} + \varepsilon(n)$$

Construction
PRG from one-way **permutation**

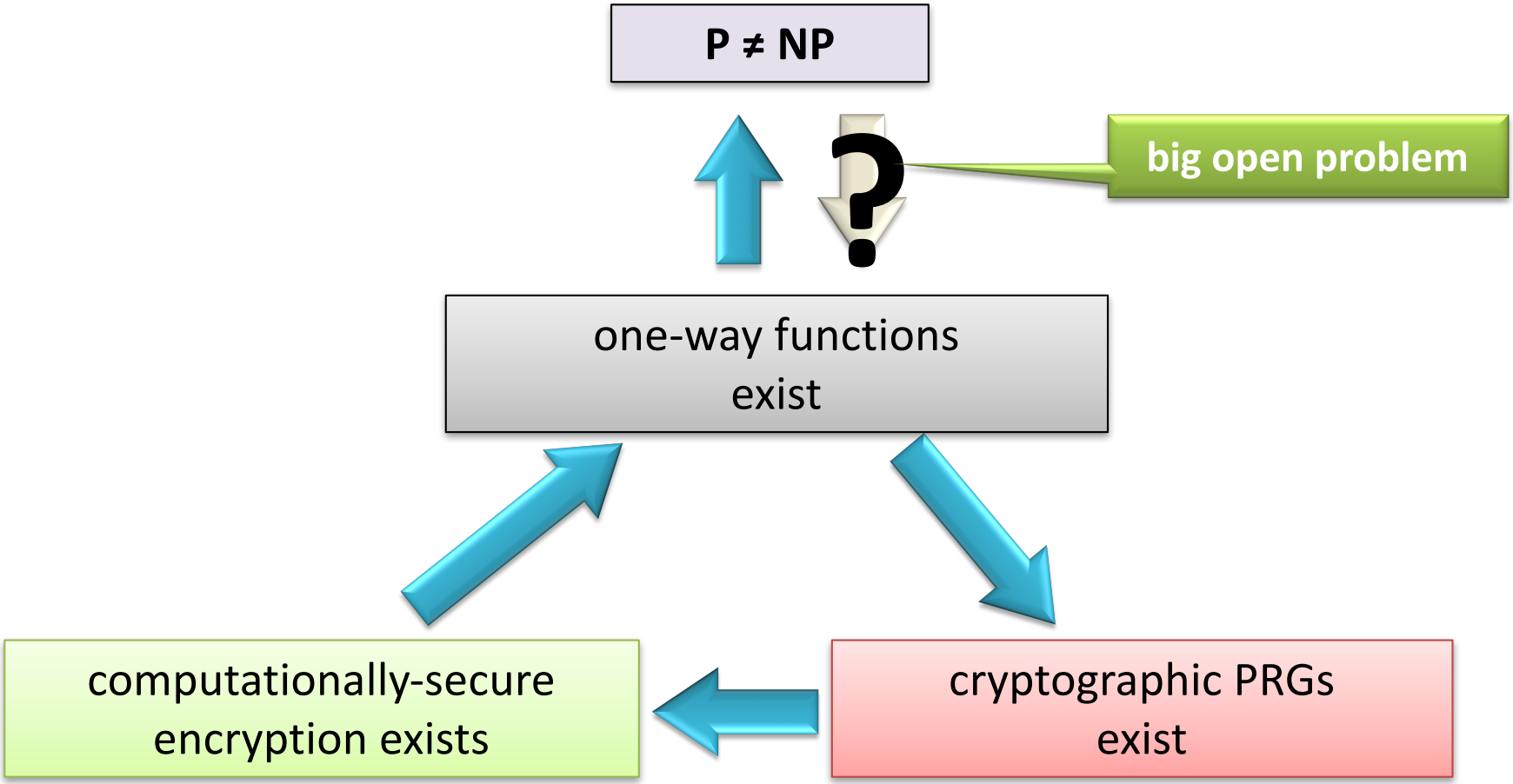
- HC bit for any **f**: assume **f** one-way
  - **r** is random, **|x|=|r|**
  - Hard-core bit for **f**: $hc(x) = \bigoplus_{i=1}^{|x|} r_i \cdot x_i$
- **G: {0,1}$^{|x|}$→{0,1}$^{|x|+1}$**
  G(x)="f(x)|| hc(x)" is **PRG** with **ℓ(|x|)=|x|+1**
- Extension to arbitrary **ℓ(|x|)** possible
  - Iteration
    1. Compute **G(x)**, **|G(x)|=|x|+1**, output last bit
    2. Use first **|x|** bits as **x'**, compute **G(x')=f(x')||hc(x')**, output last bit
    3. Goto 2

The implication also holds in the other direction



$f(k) = Enc_k(0,\ldots,0)$ is a one-way function

Summary


P ≠ NP
?
big open problem
one-way functions exist
computationally-secure encryption exists
cryptographic PRGs exist

Sneak Peek 2: Blum Blum Shub PRG

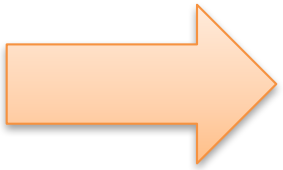
- Compute **G(s)** iteratively
  - Select **$n=p \bullet q$**, and primes **$p,q=3 \bmod 4$**
  - **x_0 = seed s**, co-prime to n, **$|s| \leq |n|$**
  - Iterate state: **$x_i = x_{i-1}^2 \bmod n$**
  - After each iteration: output **x_i**'s least significant bit
- Example: p=11, q=19, n=209, s=2
  - Internal State: 4, 16, 47, 119, 158, 93, 80, 130, 180, 5
  - Output?

Sneak Peek 2: Blum Blum Shub PRG

- Security assumption: **factoring** large **n** difficult
  - Proof "involving"
  - Extracting more than 1 bit possible, but dangerous
- Real world example: generate $\ell(|n|)=2^{23}$ bits (1MB)
  - Distinguisher security $\varepsilon=1\%$ in 2^{115} steps
  - Requires **|n|=7680 bits**!
- Too slow for encryption, good for key generation

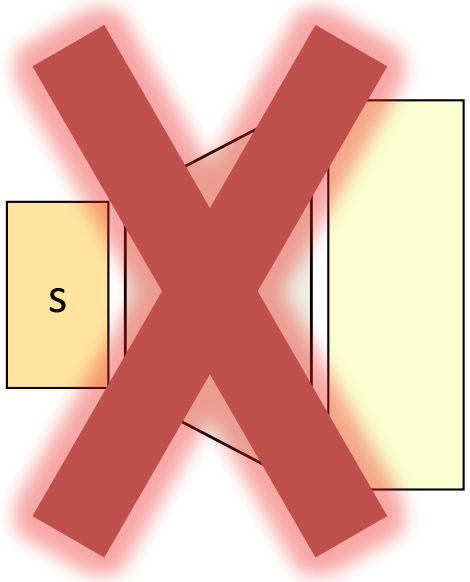
Plan

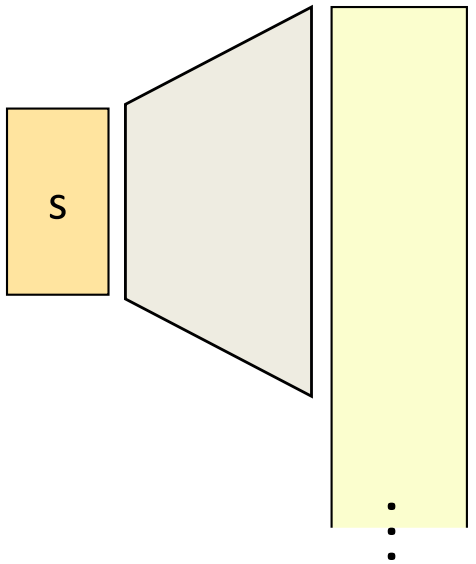
1. If semantically-secure encryption exists then **P ≠ NP**
2. A proof that “the PRGs imply secure encryption”
3. Theoretical constructions of PRGs
4. Stream ciphers



Stream ciphers

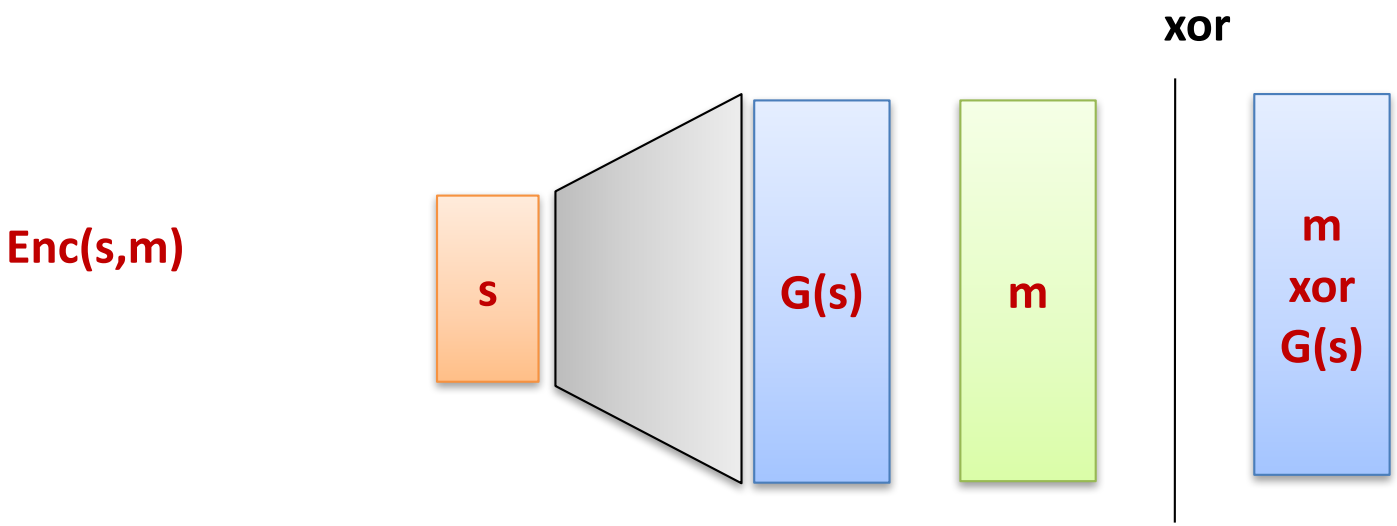
The pseudorandom generators used in practice are called **stream ciphers**





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

How to encrypt multiple messages using pseudorandom generators?



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]

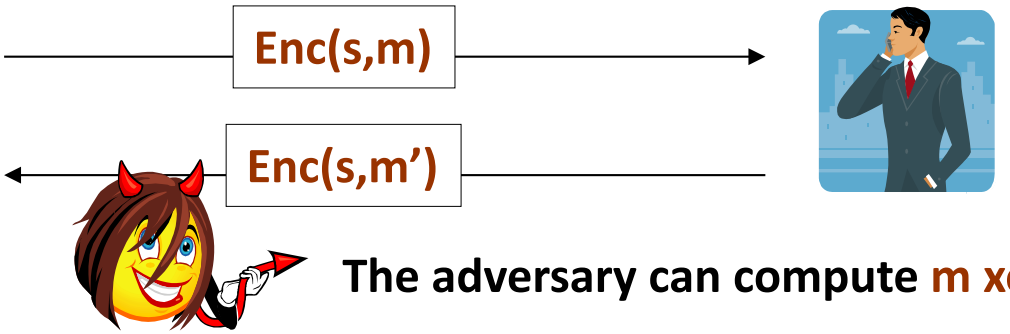
RC4 – a popular PRG (or a "stream cipher")

> **"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:



What can you learn about m, m'?

Multiple Messages: What to do?

There are two solutions:

1. The **synchronized mode**

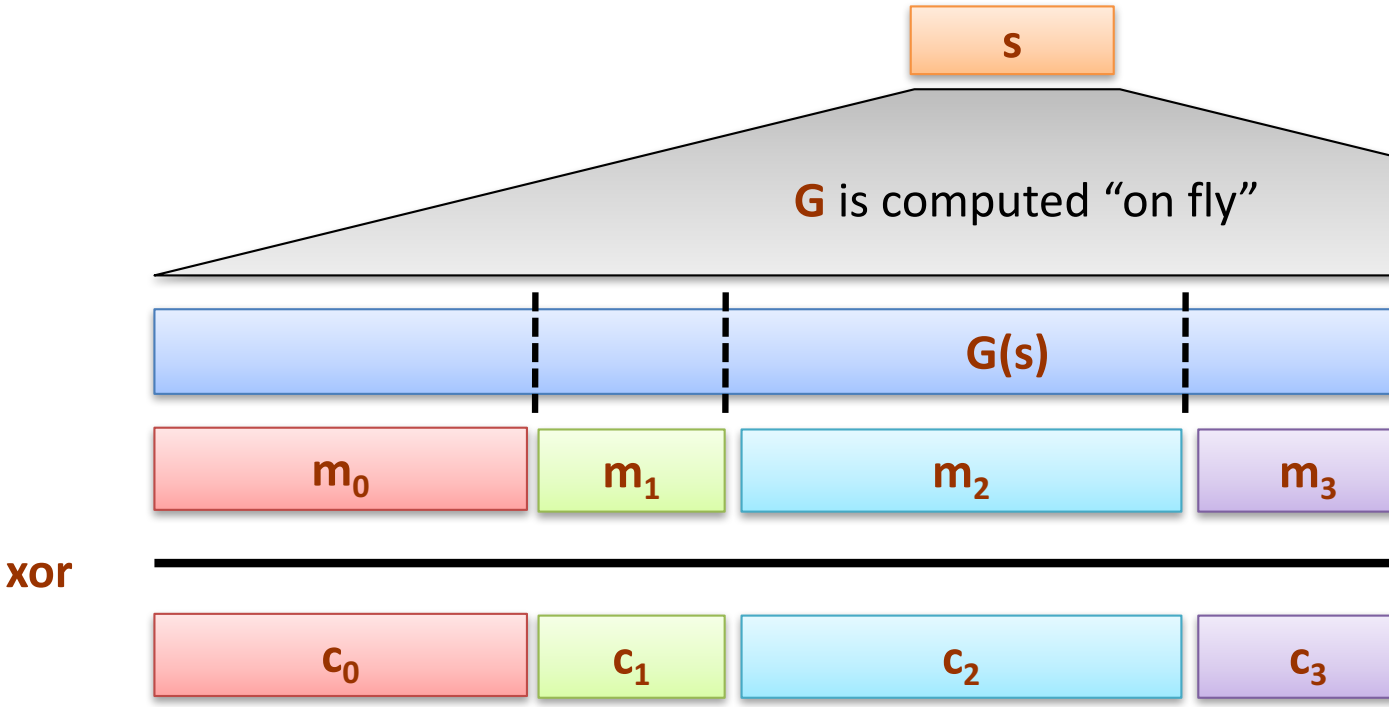
2. The **unsynchronized mode**

Synchronized mode

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

This can be proven to be CPA-secure

divide **G(s)** in blocks:



. . .

In practice: difficult to keep "state"…

Unsynchronized mode

Idea

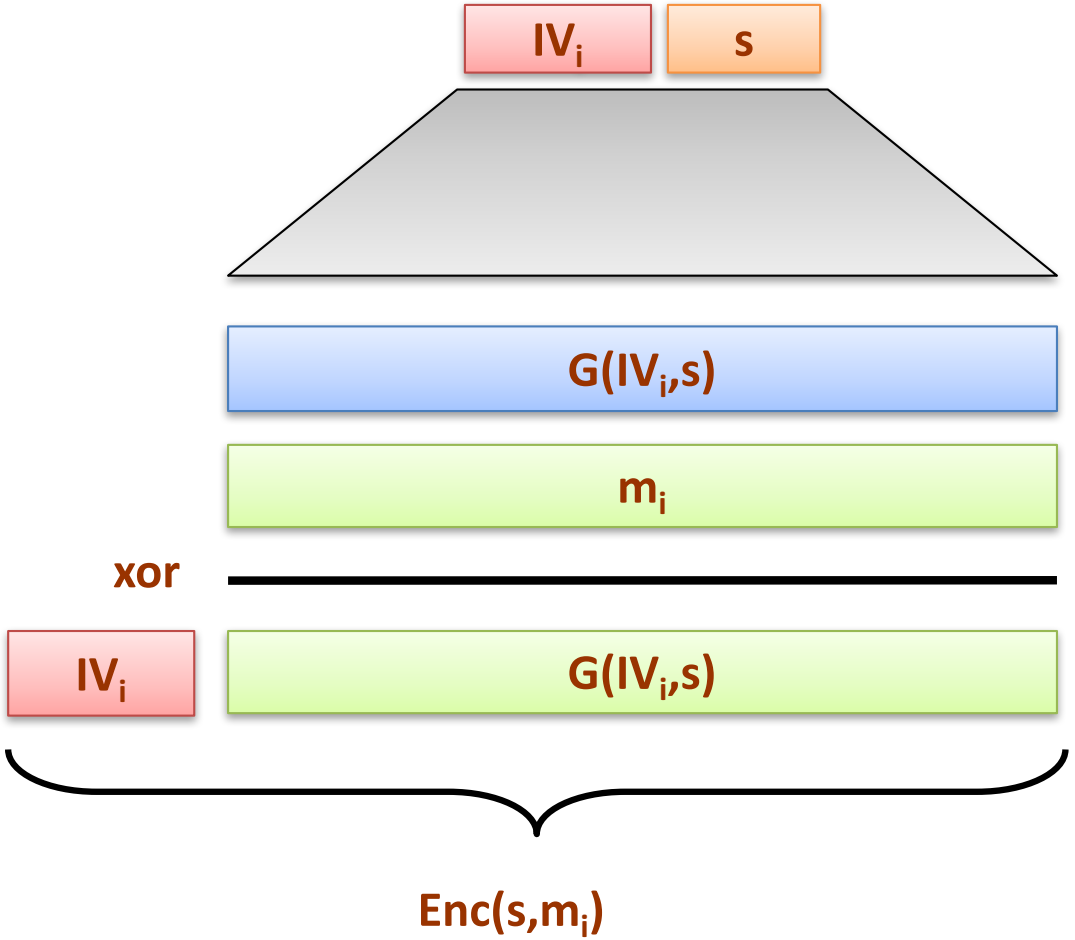
Randomize the encryption procedure.

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**



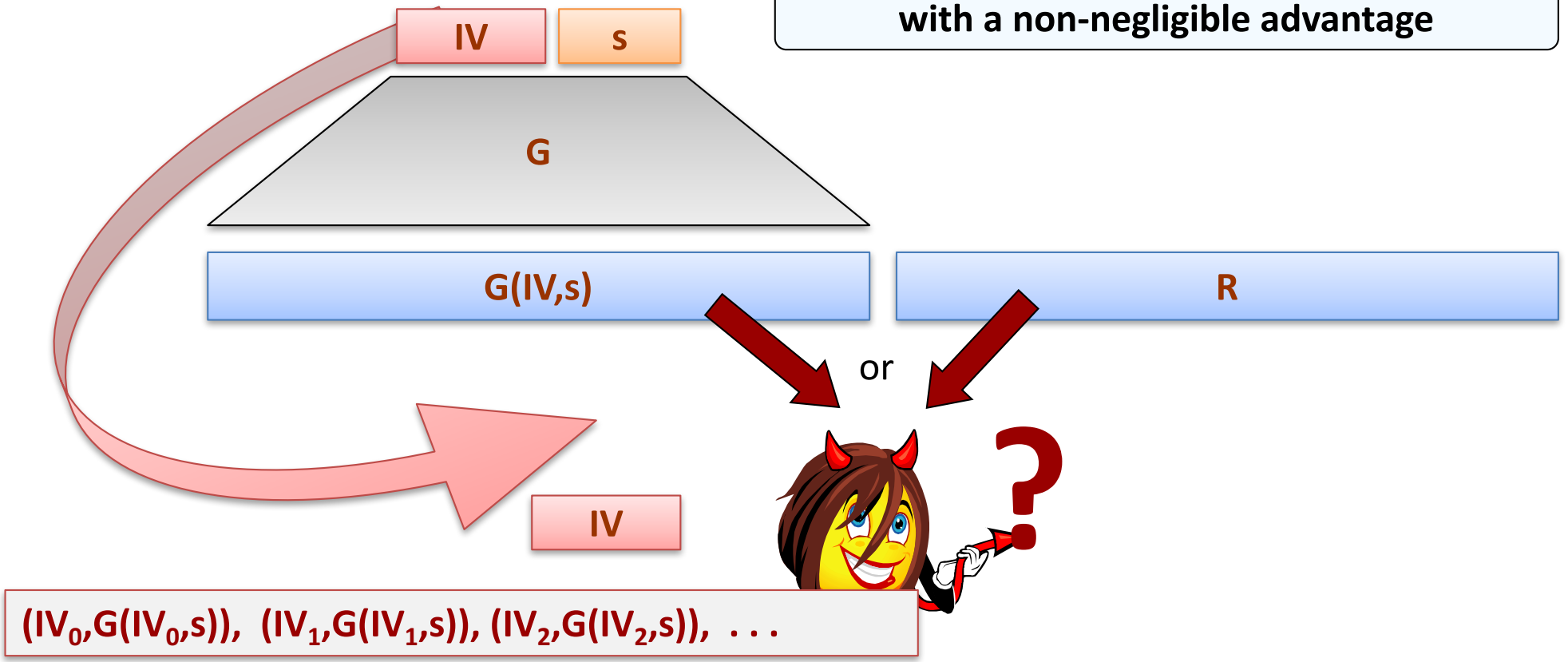
We need an "augmented" PRG

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

$(IV_0, G(IV_0,s)),\ (IV_1, G(IV_1,s)),\ (IV_2, G(IV_2,s)),\ \ldots$

where $s, IV, IV_0, IV_1, IV_2 \ldots$ are random.

with a non-negligible advantage



How to construct such a PRG?

- An **old-fashioned approach**:

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

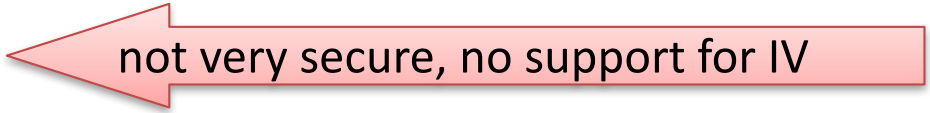
often:
just concatenate
IV and s

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

- A more **modern approach**:

  design such a **G** from scratch.

Popular stream ciphers

- **RC4** 

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

- …

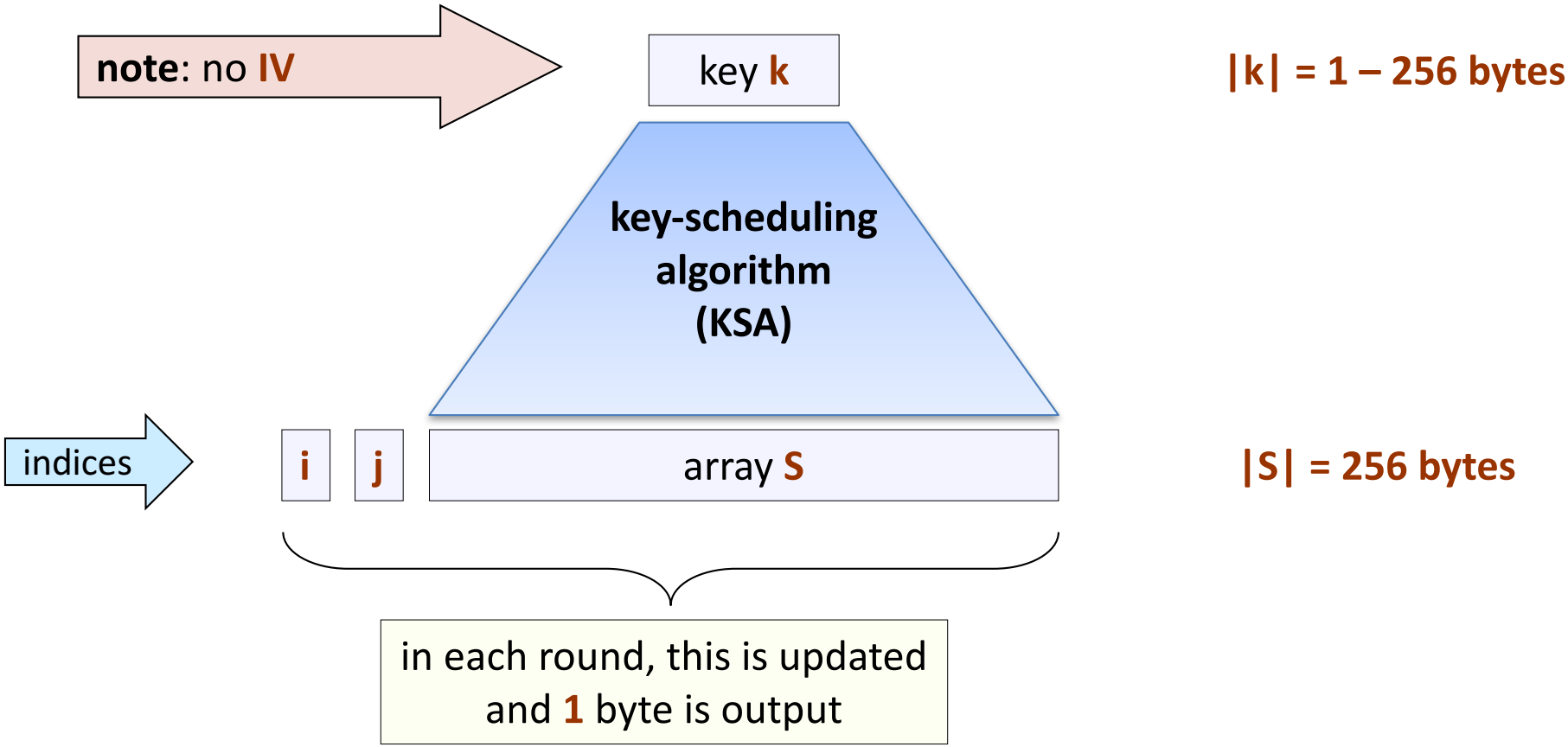
Competitions for new stream ciphers

- **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) announced a portfolio of ciphers: **HC-128, Grain v1, Rabbit, MICKEY v2, Salsa20/12, Trivium, SOSEMANUK**.

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 (trade mark) reasons sometimes it is called: "**ARCFOUR**" or "**ARC4**".
- Used in **WEP**, **WPA**, and **TLS**.
- **Very efficient and simple**, but has some **security flaws**

RC4 – an overview



RC4

KSA

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

Now, S contains all 256 bytes in random order

PRG

```
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
```

Remember: There is neither a security intuition, nor a proof! Don't try to "read".

Problems with RC4

1. Does not 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.
   (RC4-dropN)

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 **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**).

Is there an alternative to stream ciphers?

Yes!

block ciphers

End of Class