
CSE 3400/CSE 5850 - Introduction to Cryptography and
Cybersecurity / Introduction to Cybersecurity

Lecture 3

Encryption – Part II
(and Pseudo-randomness)

Ghada Almashaqbeh

UConn

Adapted from the textbook slides

Outline

- One time pad (OTP) encryption.
- Pseudorandom number generators (PRGs).
- Pseudorandom number functions (PRFs).
- Encryption schemes from PRGs and PRFs.

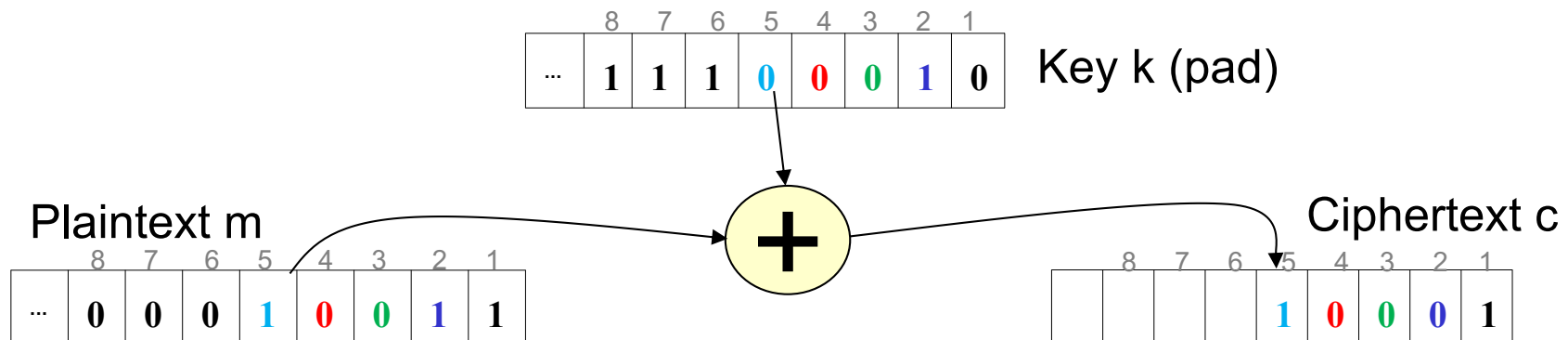
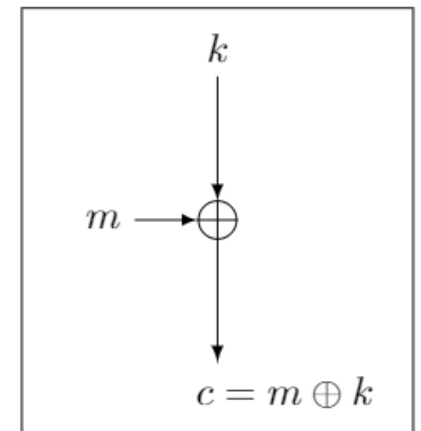
We can apply generic, exhaustive attacks to every cryptosystem. So, is breaking just a question of resources?

Can encryption be secure unconditionally – even against attacker with unbounded time and storage?

Yes it can!

One-Time-Pad (OTP)

- To encrypt message m , compute the bitwise XOR of the key k with the message m :
 - $E_k(m)=c$ where $c[i] = k[i] \oplus m[i]$
- To decrypt ciphertext c , compute the bitwise XOR of the key with the ciphertext:
 - $D_k(c)=m$ where $m[i] = k[i] \oplus c[i]$



One-Time-Pad: Example, Properties

$k = 11001$

$m = 10011$

$c = 01010$

$k = 11001$

$c = 01010$

$m = 10011$

- Correctness: $k \oplus c = k \oplus (k \oplus m) = (k \oplus k) \oplus m = 0 \oplus m = m$
- **Very simple, and efficient... but:**
 - Stateful encryption (must remember the keys, or a counter of the key bits, used so far to avoid using them again)
 - Size of key must be (at least) equal to the message size.
 - Key cannot be reused for several encryptions (one time!).
- Shannon [1949; simplified]: OTP is unconditionally secure, and for every unconditionally-secure cipher, $|k| \geq |m|$
 - Proofs of these claims? See cryptography course / books 😊

To go around the above limitations: we assume attackers are computationally limited

Recall: Unconditional vs. Computational Security

- Unconditional security
 - No matter how much computing power is available, the cipher cannot be broken
 - Computational security
 - The cost of breaking the cipher exceeds the value of the encrypted information
 - The time required to break the cipher exceeds the useful lifetime of the information
 - *So it deals with Probabilistic Polynomial Time (PPT) attackers.*
-

Looking ahead: Stream Ciphers vs. Block Ciphers

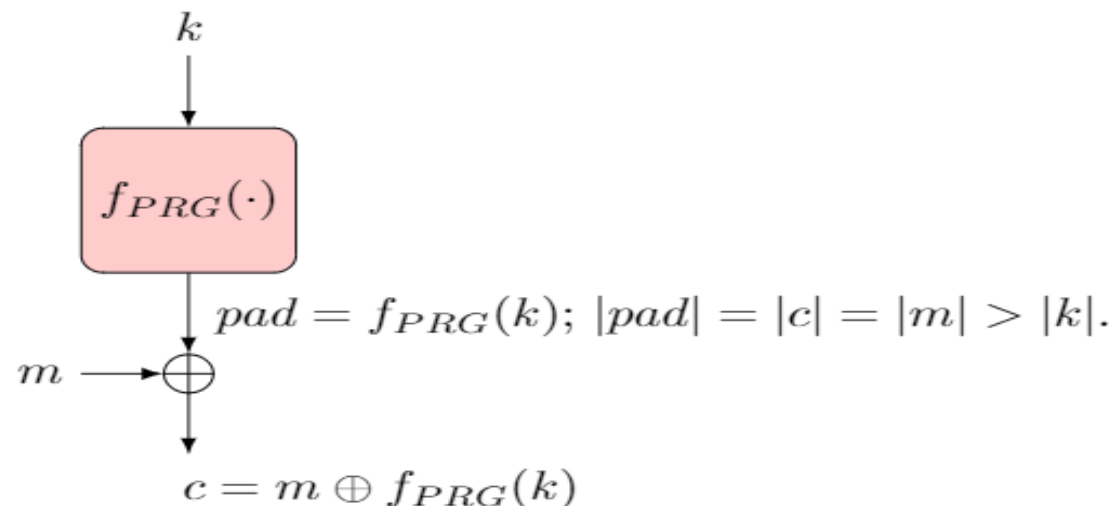
- Stream cipher
 - Encrypts a message bit by bit (stream of bits).
 - Inherently stateful; needs to keep track of the location of last encrypted bit.
- Block cipher
 - Encrypts a block (string) of bits all at once.
 - Can be stateless or stateful

Can we do computationally-secure
variant of OTP, with ‘short key’
($|k| \ll |m|$) ?

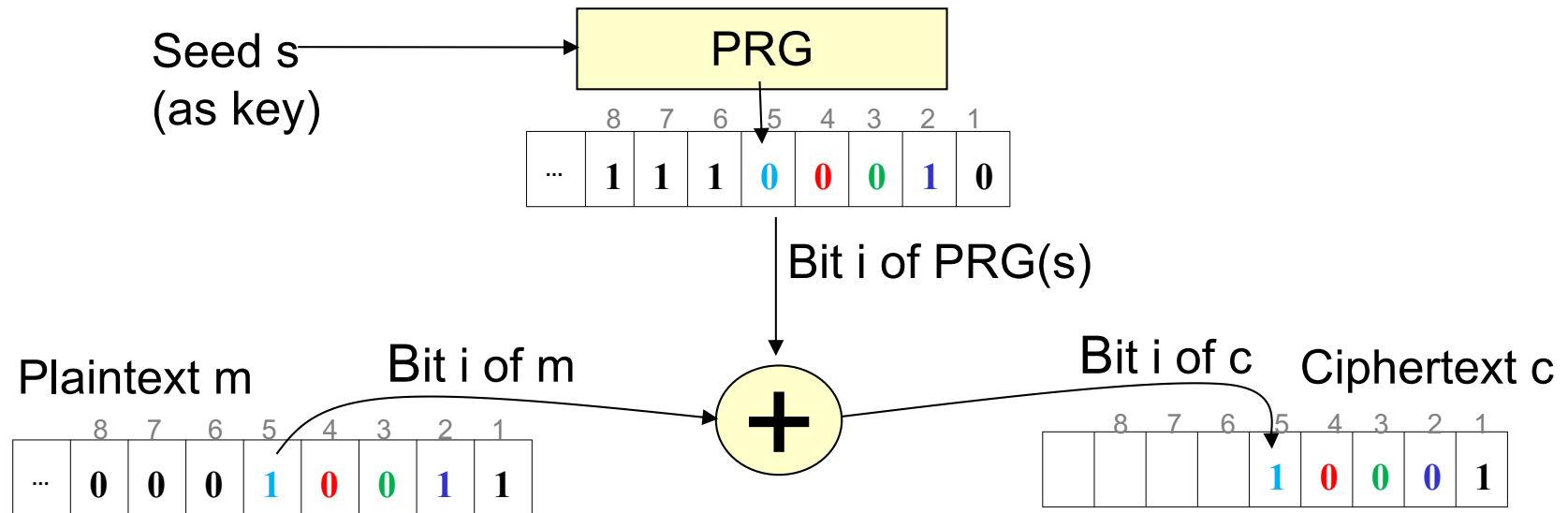
Yes, using pseudorandom number
generators (PRGs)!

PRG Stream Cipher

- Idea: `similar' to OTP, but with bounded-length key k
- How?
 - Use a pseudorandom generator $f_{PRG}(\cdot)$
 - $f_{PRG}(k)$ outputs a long stream of bits (longer than $|k|$)
 - This stream is `indistinguishable from a random' bit-stream
 - What is this `indistinguishability' requirement??
 - This is related to the famous Turing Test!

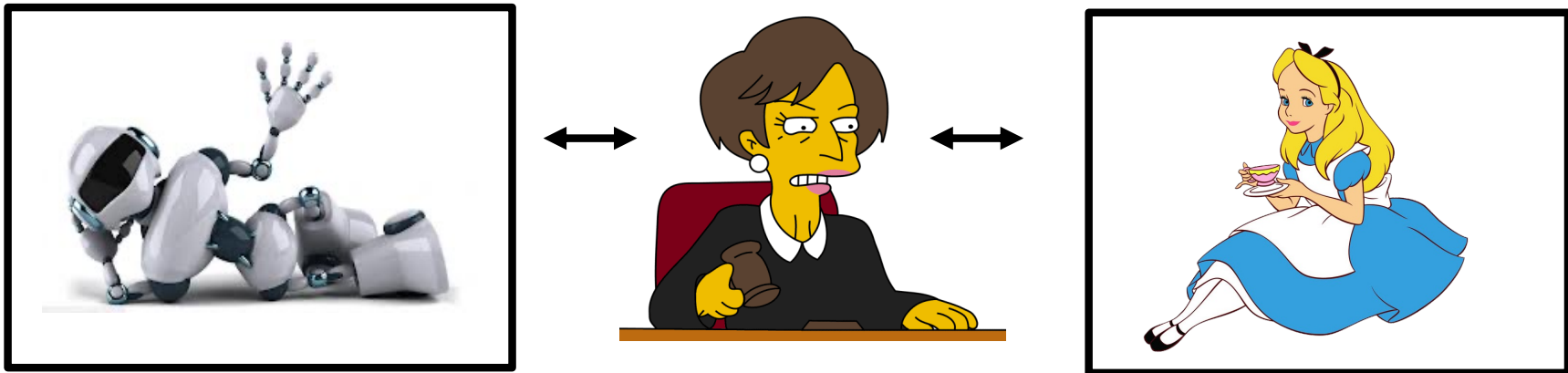


PRG Stream Cipher - Example



The Turing Test [1950]

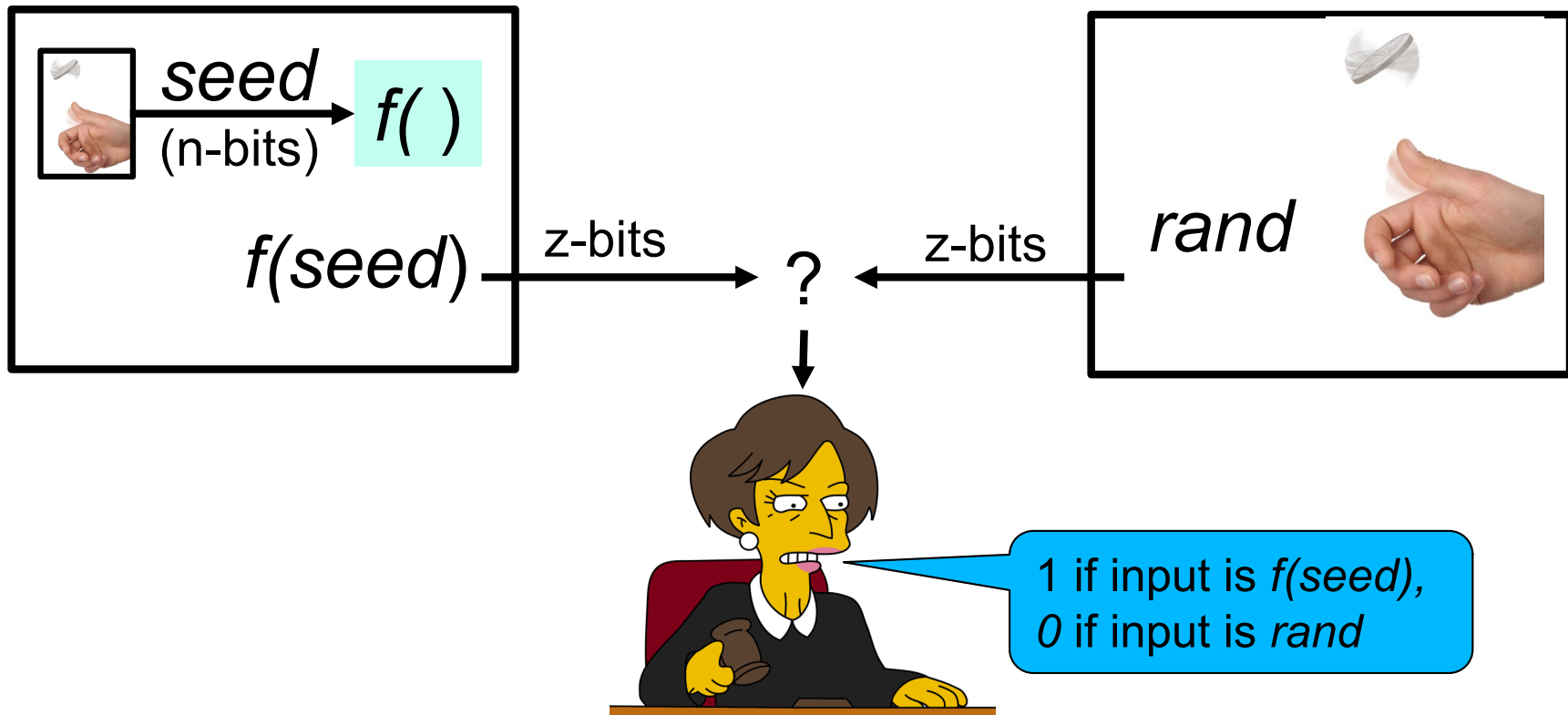
- ❑ Defined by Alan Turing
- ❑ Machine M is intelligent if an evaluator cannot *distinguish* between M and a human
 - ❑ Only textual communication, to avoid 'technicalities'



- ❑ If M is 'intelligent', then the judge will only be able to make a random guess
 - ❑ I.e., probability of distinguishing would be (at most) $\frac{1}{2}$

The PRG Indistinguishability Test

- Consider function f that maps n -bits to z -bits ($z > n$)
- Let $seed$ and $rand$ be random strings such that $|seed|=n$, $|rand|=l$
- f is a PRG if no **efficient** distinguisher D can tell which is which.
 - i.e., cannot output 1 given $f(seed)$ and 0 given $rand$ with **non-negligible advantage**.



Recall: An Efficient (PPT) Algorithm

- ❑ An algorithm A is efficient if its running time is bounded by some polynomial in the length of its inputs.
- ❑ *PPT (Probabilistic Polynomial Time)* is the set of all randomized efficient algorithms
- ❑ Examples: Given n bit input x and y (i.e., $n = |x| = |y|$), is there an efficient algorithm that:
 - ❑ Finds xy (multiplication)?
 - ❑ Finds the factors of x ?

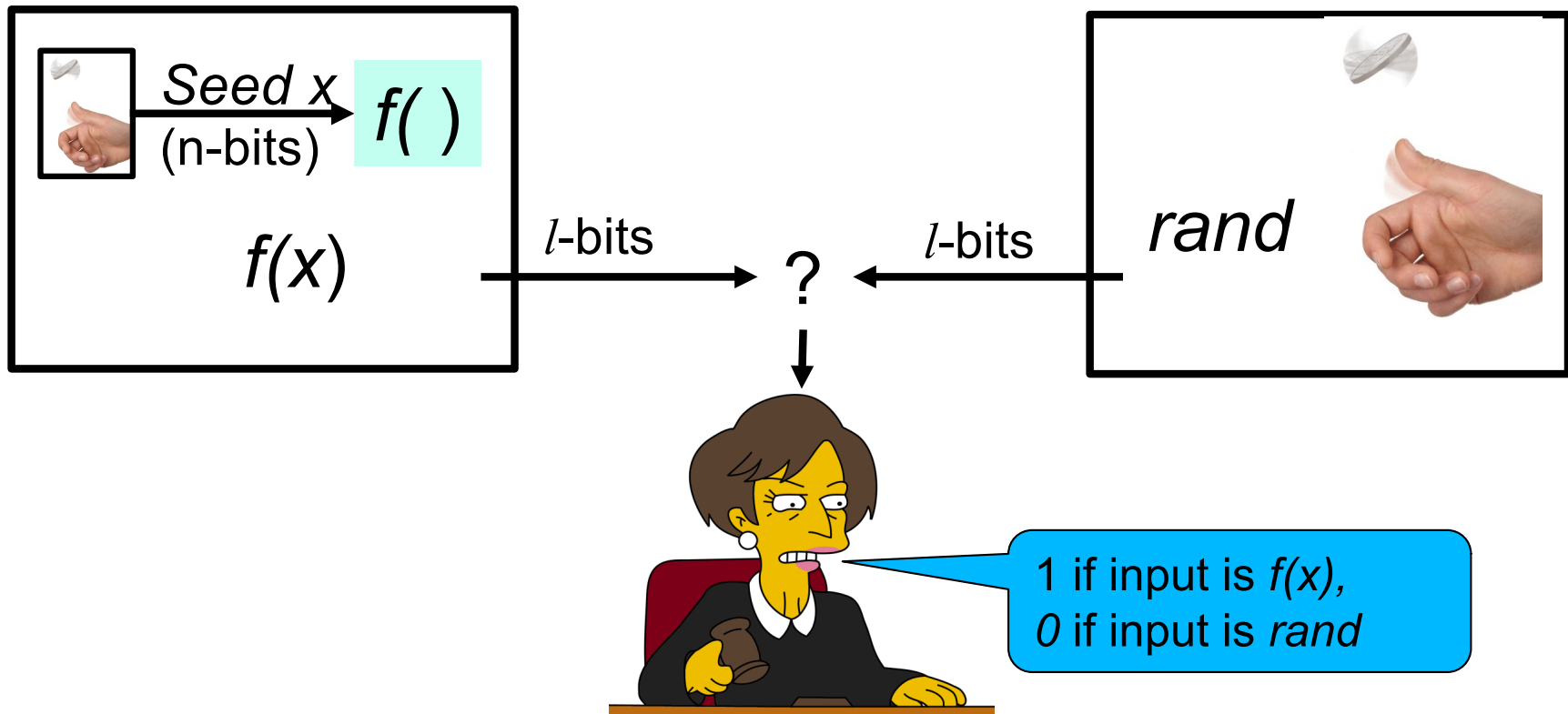
Recall: Negligible Functions

- Informally, a negligible function $\varepsilon(n)$ converges to zero as n approaches infinity.
 - Example: $\varepsilon(n) = \frac{1}{2^n}$
- Useful propositions:
 - If $\varepsilon_1(n)$ and $\varepsilon_2(n)$ are negligible, then $\varepsilon_3(n) = \varepsilon_1(n) + \varepsilon_2(n)$ is also negligible.
 - For any polynomial $p(n)$ and negligible function $\varepsilon(n)$, the function $\varepsilon_4(n) = p(n) \cdot \varepsilon(n)$ is also negligible.

The PRG Advantage

- ❑ A random guess is correct half of the time
- ❑ A distinguisher in the PRG game will have **an advantage**:

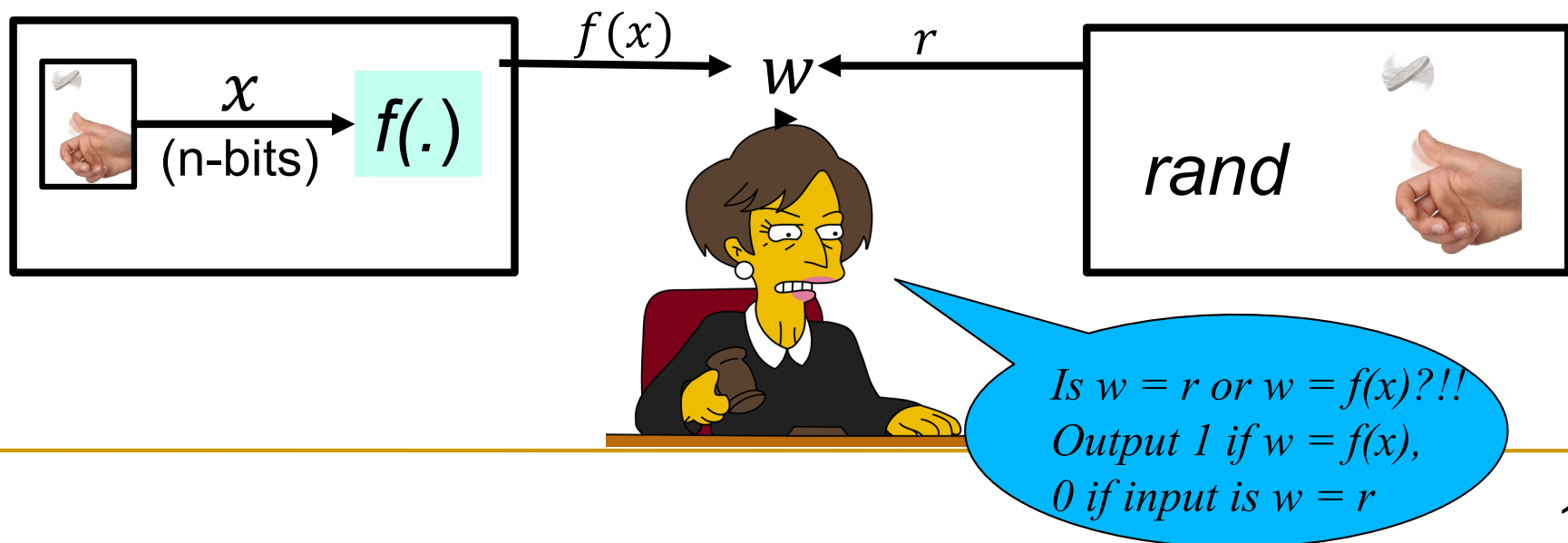
$$\varepsilon_{D,f}^{PRG}(n) \equiv \Pr_{x \xleftarrow{\$} \{0,1\}^n} [D(f(x)) = 1] - \Pr_{r \xleftarrow{\$} \{0,1\}^l} [D(r) = 1]$$



Pseudo-Random Generator: Definition

A PRG is an efficiently-computable function $f \in PPT$, which is length-increasing $((\forall x) |f(x)| > |x|)$, and whose output is indistinguishable from random, i.e. its advantage is negligible.

$$\varepsilon_{D,f}^{PRG}(n) \equiv \Pr_{x \xleftarrow{\$} \{0,1\}^n} [D(f(x)) = 1] - \Pr_{r \xleftarrow{\$} \{0,1\}^l} [D(r) = 1]$$

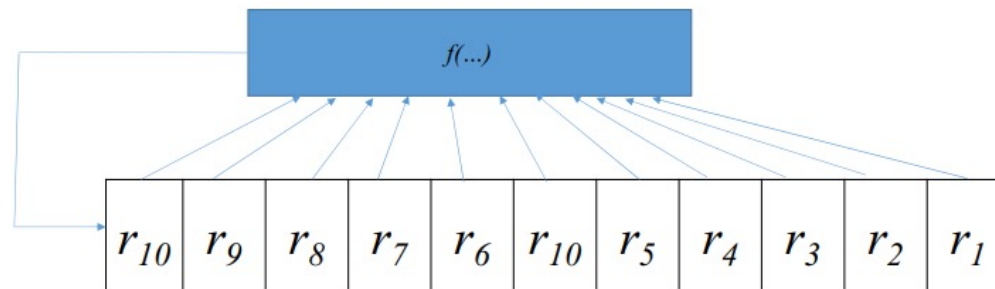


Exercise

- ❑ Let s be the seed and $f(s)$ be a PRG, are the following PRGs?
 - ❑ $g(s) = 1 || f(s)$
 - ❑ $q(s) = (\text{parity of } s) || f(s)$
 - ❑ $w(s) = \sim f(s)$
 - ❑ $\sim$ is the bitwise complement or negation

Many PRG proposals I

- Often based on Feedback Shift Registers
 - Easy construction for efficient hardware implementations.
 - Linear feedback (LFSR), or non-linear feedback function ($f(\dots)$ in the figure, e.g., XOR all previous bits to produce the next one).
 - LFSR is easily predictable (not a secure PRG)



Many PRG proposals II

- In practice, attacks on PRGs (or constructions that use PRGs) are often caused by an incorrect use of a PRG.
 - Example: a PRG-based OTP encryption scheme with a fixed PRG seed.
 - What is wrong with this construction?

Example: Misusing Stream-Cipher

MS-Word 2002 uses RC4 as a PRG for encryption:

$\text{PAD} = \text{RC4}(\text{password})$

Save ciphertext = $\text{PAD} \oplus \text{Document}$

The Problem: same pad used to encrypt when the document is modified

Attacker gets (where $d1$ and $d2$ are two different versions of the document d): $c1 = \text{PAD} \oplus d1$, $c2 = \text{PAD} \oplus d2$

Enough redundancy in English to decrypt!

[Mason et al., CCS'06]

Cryptography is bypassed more often than broken!!

Provably-Secure PRG?

- ❑ f is a secure PRG $\rightarrow$ no PPT distinguisher
 - ❑ But given s , it is trivial to identify $f(s)$
- ❑ This means that the PRG problem **is** in NP
 - ❑ NP: in PPT, if given a 'hint' – e.g., s ...
- ❑ So a provable secure PRG $\rightarrow P \neq NP$
 - ❑ The 'holy grail' of the theory of complexity
- ❑ Thus, we prove that a given PRG construction is secure, if <assumption>
 - ❑ The paradigm of proof by reduction

Provably-Secure PRG : by reduction

- ❑ Construct PRG f from g assumed to be X
 - ❑ X is some hard problem (or a hardness assumption)
 - ❑ Known (or believed) to be hard to be broken.
- ❑ Reduction: if g is secure given X $\Rightarrow f$ is a secure PRG
 - ❑ Basic method of theory of cryptograph
 - ❑ You will study it in a theory cryptography course.
 - ❑ Many such PRG constructions.

PRG by reduction (informal) – An Example

Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$ be a secure PRG. Is $f' : \{0, 1\}^{n+1} \rightarrow \{0, 1\}^{n+2}$, defined as $f'(b \parallel x) = b \parallel f(x)$ where $b \in \{0, 1\}$ is a random bit, also a secure PRG?

Hints:

- intuitively, is f' a secure PRG? Why?
- How does security of f' rely on the security of f ?

Stream-Cipher Like but Stateless Encrypt?

- PRG-based stream ciphers are stateful.
 - Need to remember how many bits (or bytes) were already encrypted, and how many bits (or bytes) of PRG output have been used so far.
- Can secure encryption be **stateless**?

Yes it can!

In three steps (or versions):

1. Use **less** state
2. Use **no** state with a random function
3. Use **no** state, but with **pseudo-random function**

First, what's a ('truly') random function f ?

- Fix domain D , usually binary strings: $\{0,1\}^l$
- Fix range R , usually binary strings: $\{0,1\}^n$
- For each value x in D , randomly select a value y in R
- $f(x) = y$
- Example:

Domain D
 $\{0,1\}^2$

	$f()$
00	
01	
10	
11	

Range R $\{0,1\}^5$



What's a ('truly') random function?

- Fix domain D, usually binary strings: $\{0,1\}^l$
- Fix range R, usually binary strings: $\{0,1\}^n$
- For each value x in D, randomly select a value y in R
- $f(x) = y$
- Example:

Domain D
 $\{0,1\}^2$

	f()	Range R $\{0,1\}^5$
00	01101	
01	11010	
10	01101	
11	11101	



What's a ('truly') random function?

- Another example:
- Domain D : integers
- Range R : bits $\{0,1\}$
- For each integer i , randomly select a bit $f(i)$
- Example:

Domain:
integers

i	$f(i)$
1	
2	
3	
4	
5	
6	
...	...

Range: bits $\{0,1\}$



What's a ('truly') random function?

- Another example:
- Domain D : integers
- Range R : bits $\{0,1\}$
- For each integer i , randomly select a bit $f(i)$
- Example:

Domain:
integers

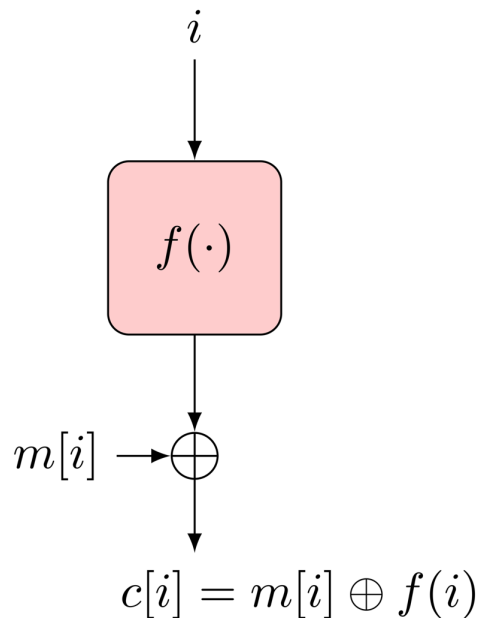
i	$f(i)$
1	0
2	1
3	1
4	0
5	0
6	1
...	...

Range: bits $\{0,1\}$



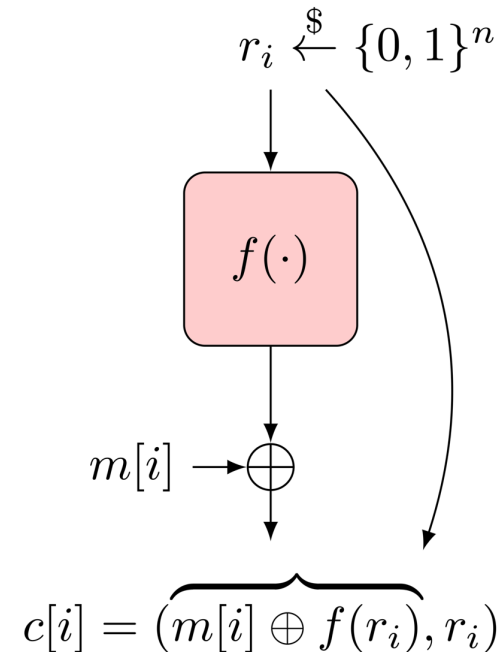
Random Function-Based Encryption (stream cipher)

Stateful (counter-based) Design



- **Stateful**
- No extra random bits required
- $|\text{ciphertext}| = |\text{plaintext}|$

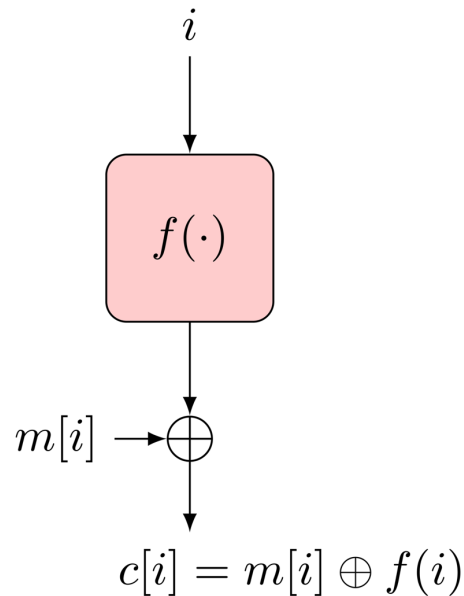
Stateless Design



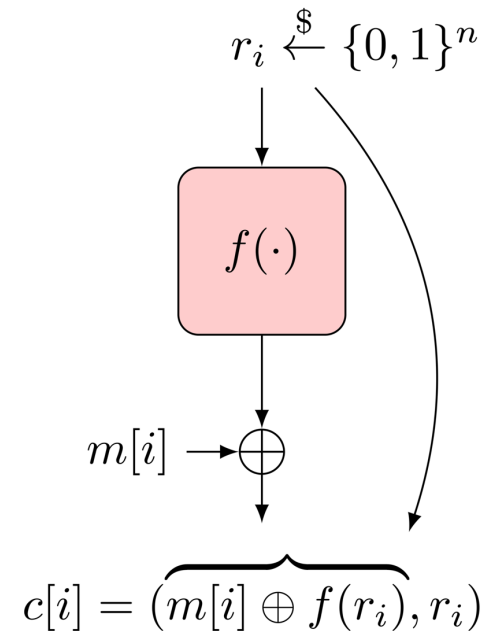
- **Stateless**
- n random bits per plaintext bit
- $|\text{ciphertext}| = (n + 1) \cdot |\text{plaintext}|$

Random Function-Based Encryption (stream cipher)

Stateful (counter-based) Design



Stateless Design

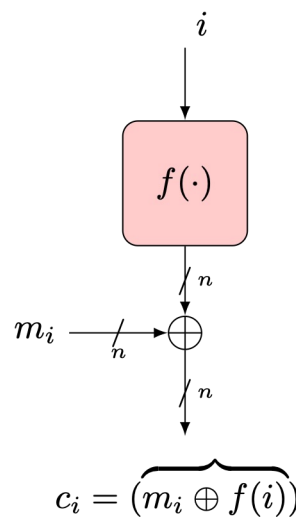


Drawbacks:

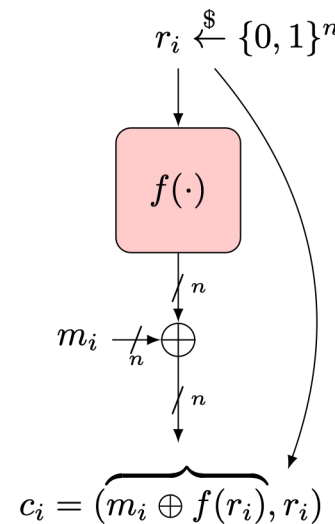
- Require (true) random function (impractical)
- One random function invocation per bit (computational overhead)

Reduce Overhead: Blockciphers

- **Optimization:** operate in blocks (say of n bits)
 - f be random function that produce n -bits random strings
 - m_i be i -th block of n -bits of plaintext
 - c_i be i -th block of n -bits of ciphertext



Stateful (counter-based) Design

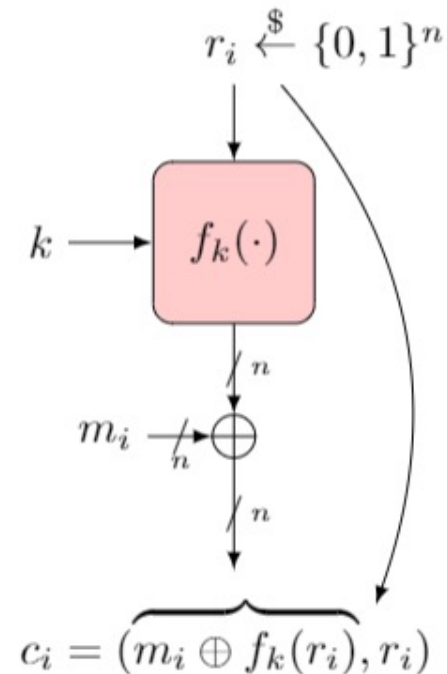
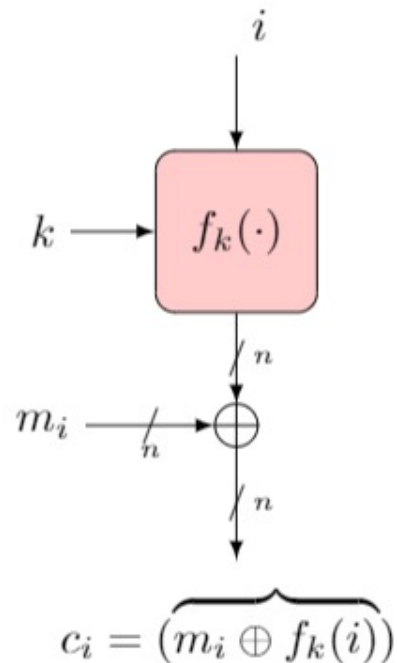


Stateless Design

- **Challenge:** sharing such random function f (both sender and recipient must have it)!!
 - Size of table? 2^n entries of n bits each...
- **Idea:** use **pseudo-random function (PRF)** instead!

Encryption with PRF

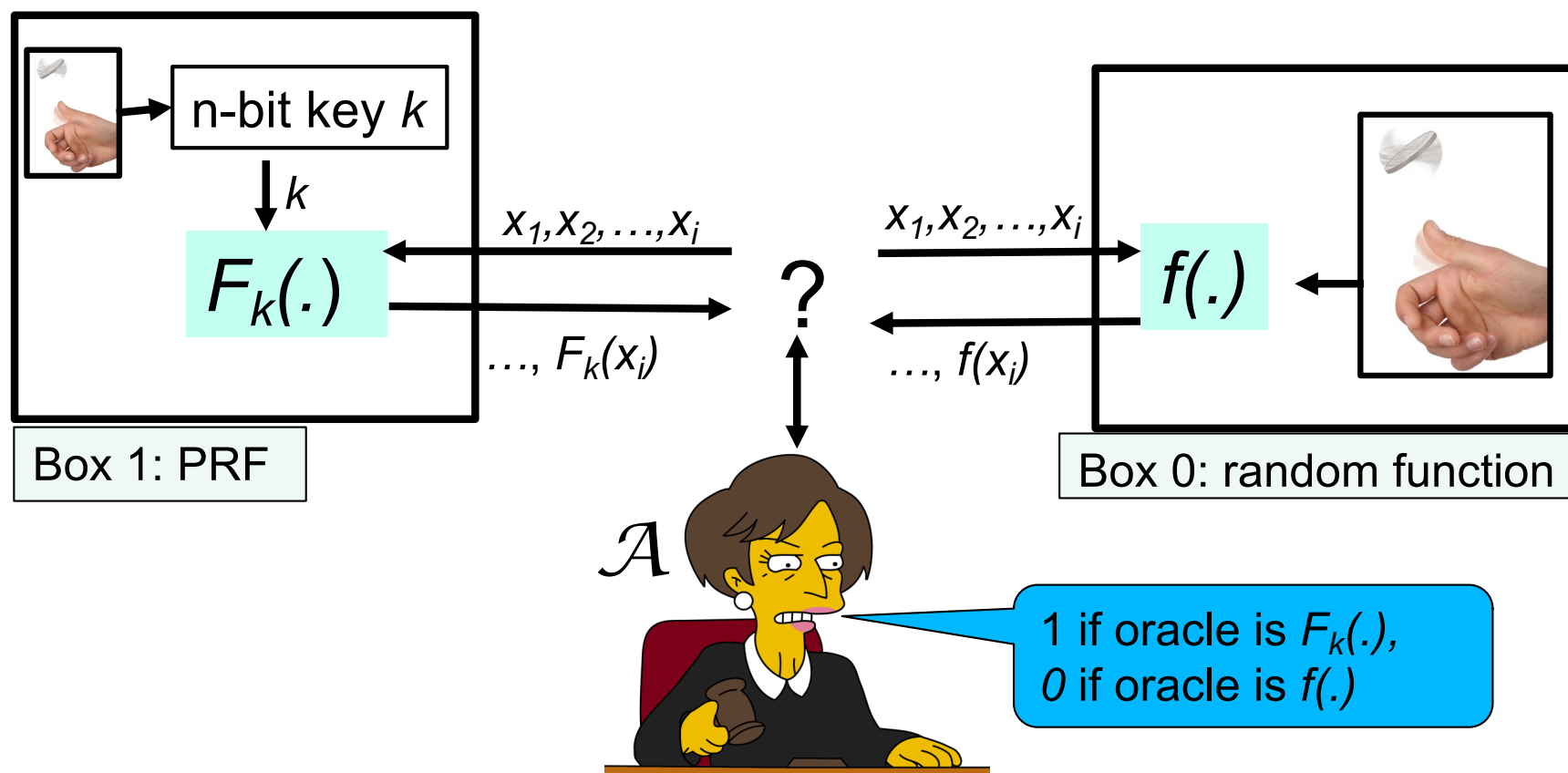
- Operate in blocks (say of n bits)
- Use Pseudo-Random Function (PRF) $f_k(\cdot)$ that outputs n bits
 - Efficient , compact
 - Keyed with a secret key k .



But what is a PRF ?

The PRF Indistinguishability Test

- F is a PRF from domain D to range R , if no distinguisher $\mathcal{A}$:
 - Outputs 1 (signaling PRF) given oracle access to $F_k(.)$ (for random n -bits key k), and
 - Outputs 0 (signaling random) given oracle access to $f(.)$, a random function (from D to R)



PRF Definition

- A PRF is ‘as secure as a random function’
 - Against efficient adversaries (PPT), allowing negligible advantage
 - Yet practical, even efficient
- Formally, a PRF F_k is:

Definition A pseudorandom function (PRF) is a polynomial-time computable function $F_k(x) : \{0,1\}^* \times D \rightarrow R$ s.t. for all PPT algorithms $\mathcal{A}$, $\varepsilon_{\mathcal{A},F}^{PRF}(n) \in NEGL$, i.e., is negligible, where the advantage $\varepsilon_{\mathcal{A},F}^{PRF}(n)$ of the PRF F against adversary $\mathcal{A}$ is defined as:

$$\varepsilon_{\mathcal{A},F}^{PRF}(n) \equiv \Pr_{k \xleftarrow{\$} \{0,1\}^n} [\mathcal{A}^{F_k}(1^n)] - \Pr_{f \xleftarrow{\$} \{D \rightarrow R\}} [\mathcal{A}^f(1^n)] \quad (2.29)$$

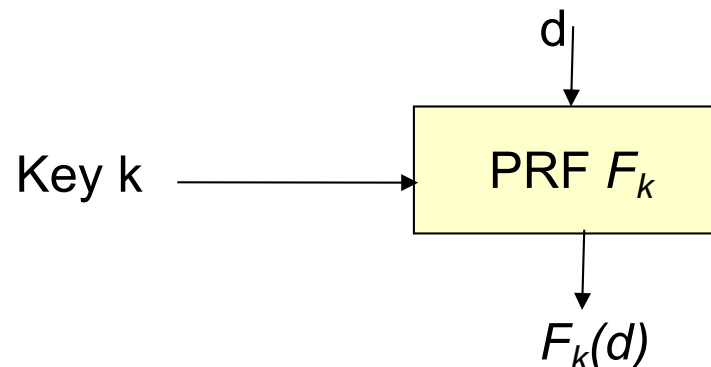
The probabilities are taken over random coin tosses of $\mathcal{A}$, and random choices of the key $k \xleftarrow{\$} \{0,1\}^n$ and of the function $f \xleftarrow{\$} \{D \rightarrow R\}$.

Constructing a PRF

- ❑ Heuristics: efficient, not proven secure
 - ❑ Construct PRF from PRG
 - ❑ Provably secure - if PRG is secure (reduction)
 - ❑ But many PRG calls for each PRF computation
 - ❑ → Not deployed in practice since it is inefficient.
 - ❑ Provable secure PRF without assumptions?
 - ❑ If exists, would imply that $P \neq NP$. Why?
 - ❑ Recall that P : problems solvable in polynomial time and NP : same, but given also any 'hint' (e.g. key k)
 - ❑ Given the key k , it is trivial to identify the PRF
 - ❑ Do you see how this enables $\mathcal{A}$ to win all the time?
-

PRF Applications

- PRFs have many more applications:
 - Encryption, authentication, key management, ...
- Example: derive independent key for each day d
 - Easy, with PRF and single shared key k
 - Key for day d is $k_d = F_k(d)$
 - Exposure of keys of Monday and Wednesday does not expose key for Tuesday
 - Similarly: separate keys for different goals, e.g., encryption and authentication



Examples

- ❑ Let F_k be a secure PRF, are the following functions secure PRFs and why?
 - ❑ $F'_k(x) = F_1^n(x) \parallel F_k(x)$
 - ❑ $F''_k(x) = F_k(x) \parallel \text{lsb}(F_k(x))$
 - ❑ lsb is the least significant bit

Covered Material From the Textbook

- ❑ Chapter 2:
 - ❑ Section 2.5
 - ❑ Section 2.6
 - ❑ Section 2.7

Thank You!

