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?

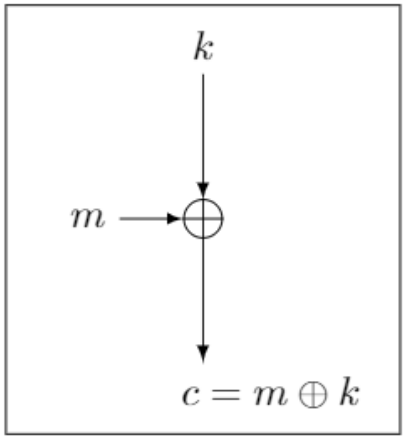
Can encryption be secure unconditionally – even against attacker with unbounded time and storage?

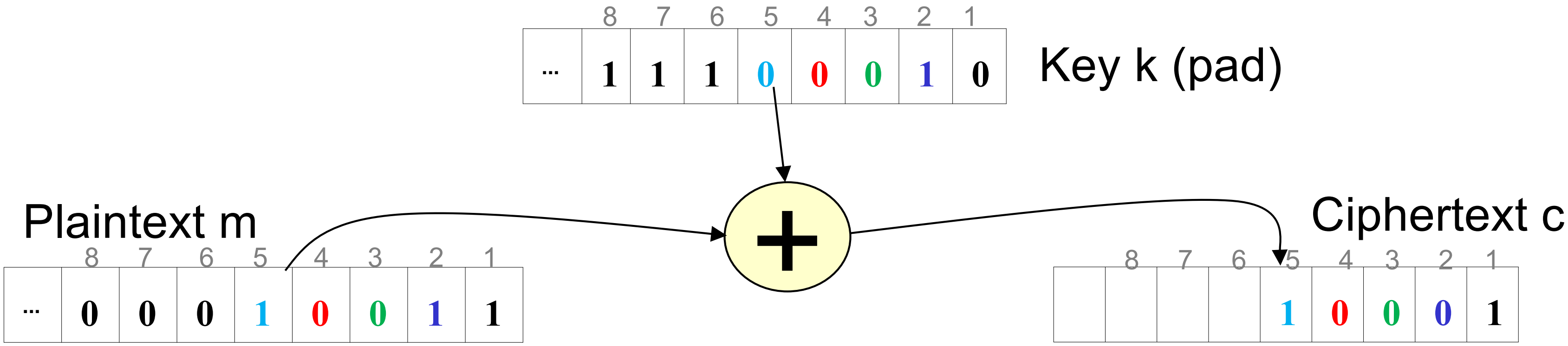
Yes it can!

One-Time-Pad (OTP)

[Frank Miller, 1882] and
[Vernham (and Mauborgne?), 1919]

- 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]





Key k (pad)

	8	7	6	5	4	3	2	1
…	1	1	1	0	0	0	1	0

Plaintext m

	8	7	6	5	4	3	2	1
…	0	0	0	1	0	0	1	1

Ciphertext c

	8	7	6	5	4	3	2	1
				1	0	0	0	1

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 crypto 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

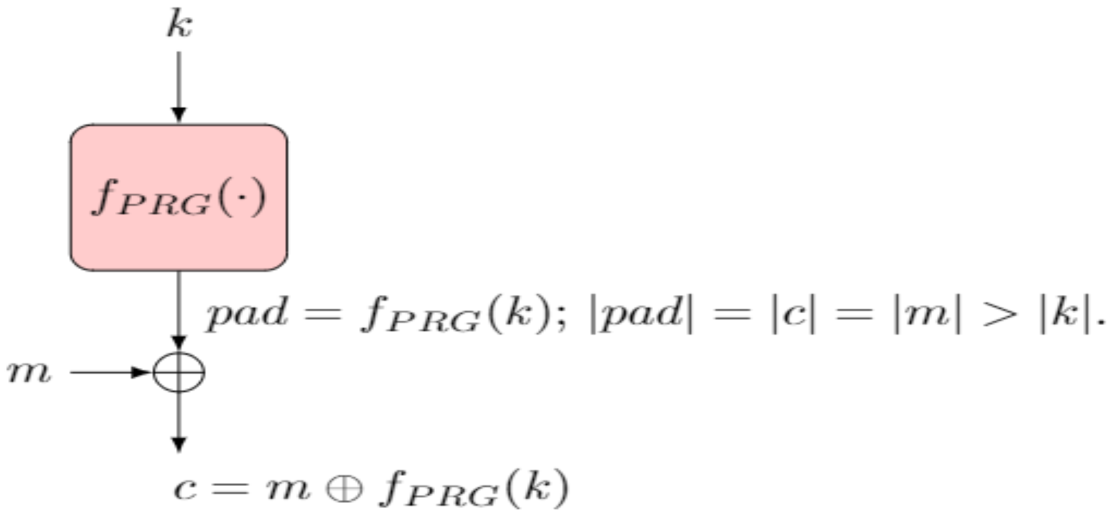
- Stream cipher
  - Encrypts a message bit by bit (stream of bits).
  - Inherently stateful; needs to keep track of the location of last encrypted bit.
- Block cipher
  - Encrypts a block (string) of bits all at once.
  - Can be stateless or stateful

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

Yes, using pseudorandom number generators (PRGs)!

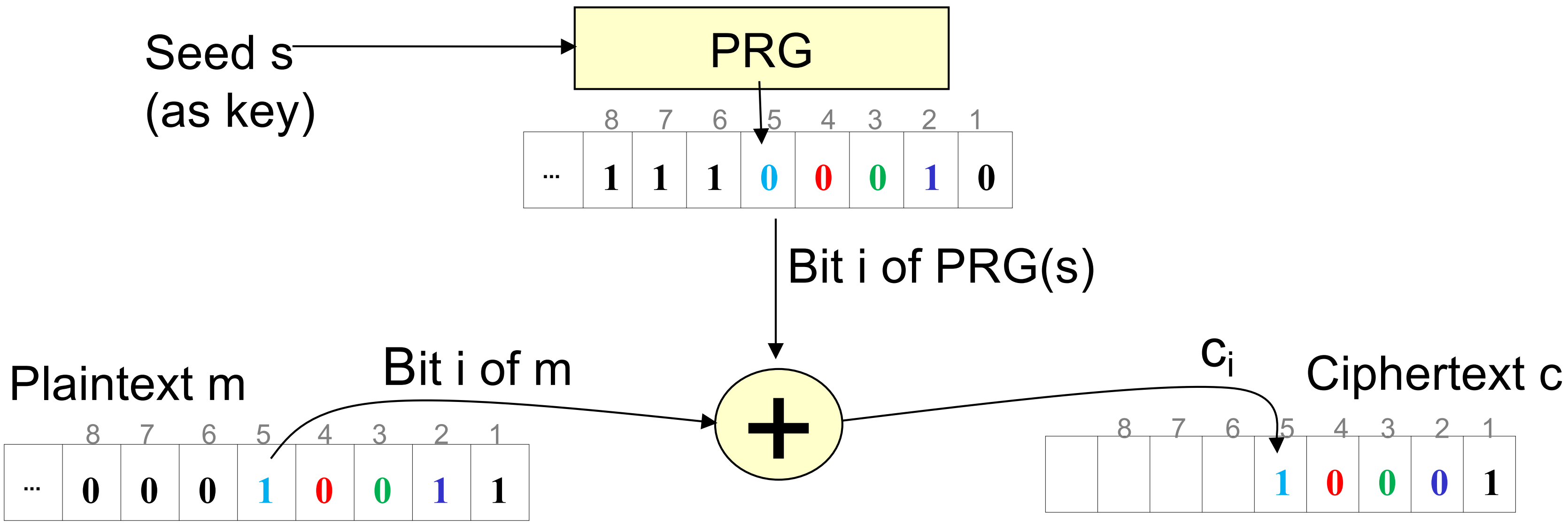
PRG Stream Cipher

- 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 random' bit-stream
  - What is this 'indistinguishability' requirement??
    - This is related to the famous Turing Test!



k
$f_{PRG}(\cdot)$
$pad = f_{PRG}(k)$; $|pad| = |c| = |m| > |k|$.
m
$c = m \oplus f_{PRG}(k)$

PRG Stream Cipher - Example



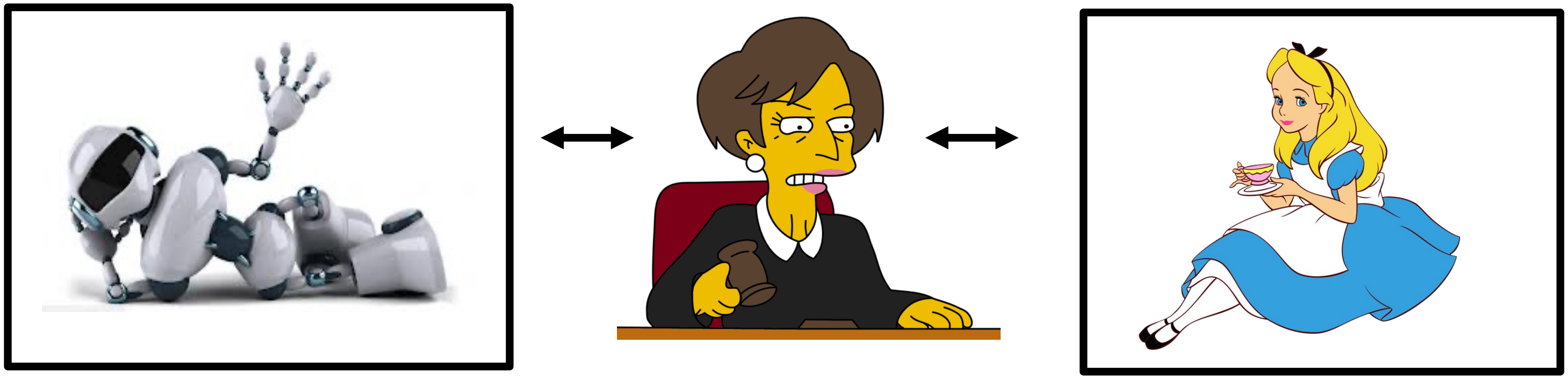
	8	7	6	5	4	3	2	1
...	1	1	1	0	0	0	1	0

	8	7	6	5	4	3	2	1
...	0	0	0	1	0	0	1	1

	8	7	6	5	4	3	2	1
				1	0	0	0	1

The Turing Test [1950]

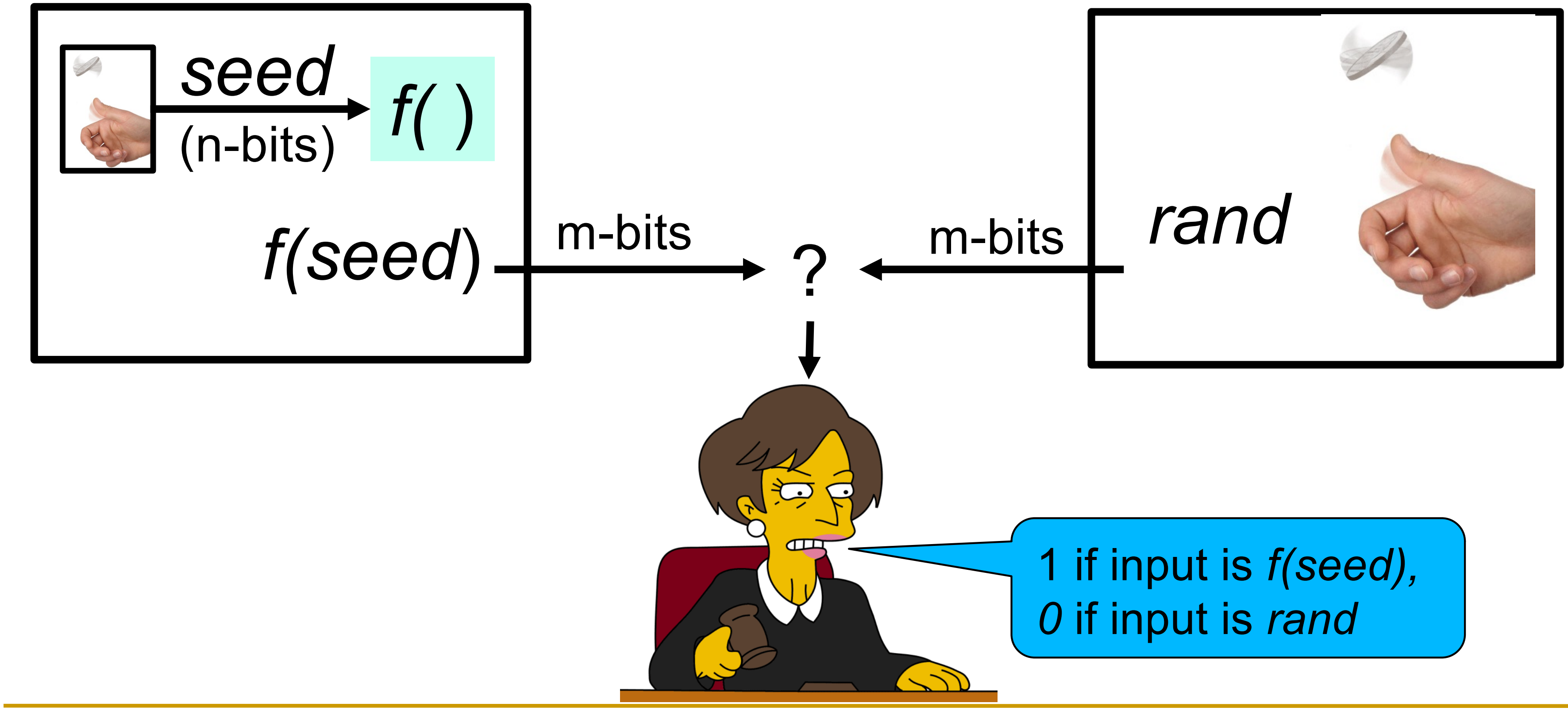
- Defined by Alan Turing
- Machine M is intelligent, if an evaluator cannot ***distinguish*** between M and a human
  - Only textual communication, to avoid `technicalities’



- If M is ‘intelligent’, judge will only be able to make a random guess
  - I.e., probability of distinguishing would be (at most) ½

The PRG Indistinguishabity Test

- Consider function *f* that maps n-bits to m-bits (m>n)
- Let *seed* and *rand* be random strings s.t.: |*seed*|=n, |*rand*|=*m*
- *f* is a PRG if no **efficient** distinguisher *D* can tell which is which.
  - i.e., cannot output 1 for *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?

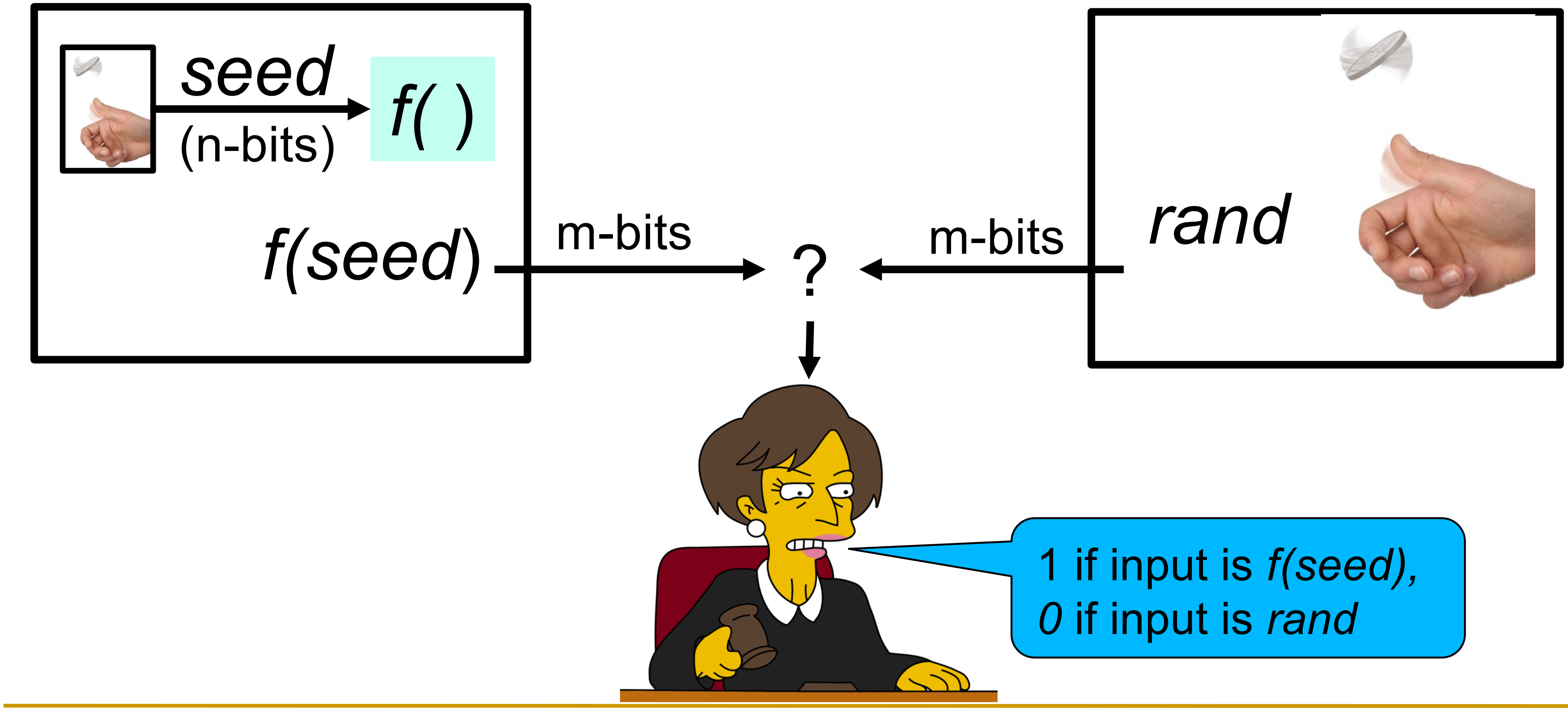
Recall: Negligible Functions

- 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 $p(\mathrm{n})$ and negligible function $\varepsilon(n)$, the function $\varepsilon_4(n) = p(\mathrm{n}).\,\varepsilon(n)$ 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:**

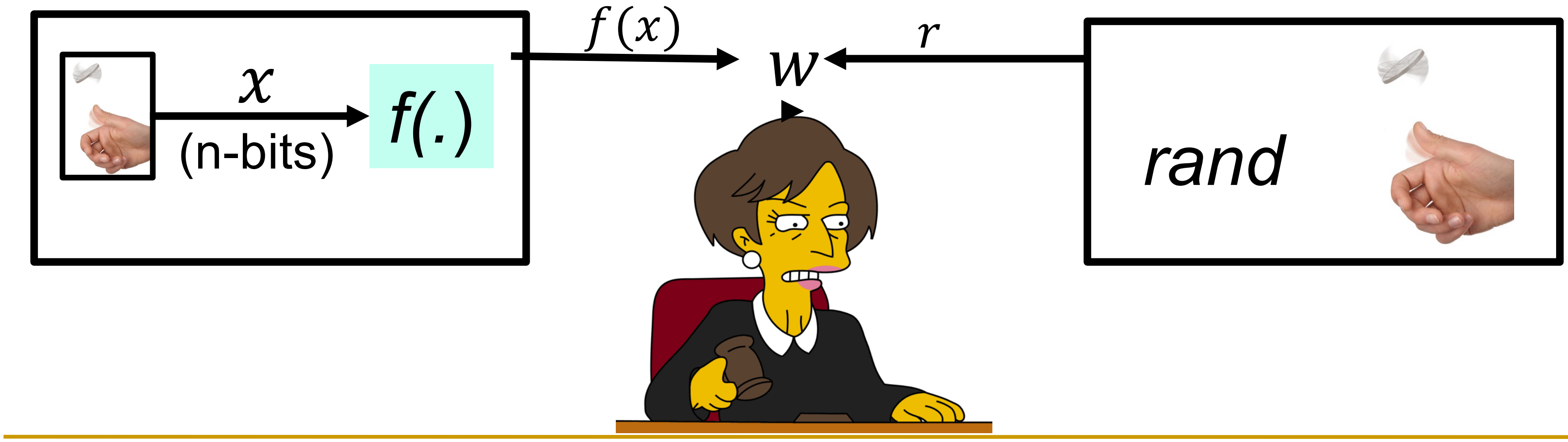
$$\varepsilon_{D,f}^{PRG}(n) \equiv \Pr_{x \xleftarrow{\$} \{0,1\}^n}\left[D\left(f(x)\right) = 1\right] - \Pr_{r \xleftarrow{\$} \{0,1\}^{l_n}}\left[D\left(r\right) = 1\right]$$



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.

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

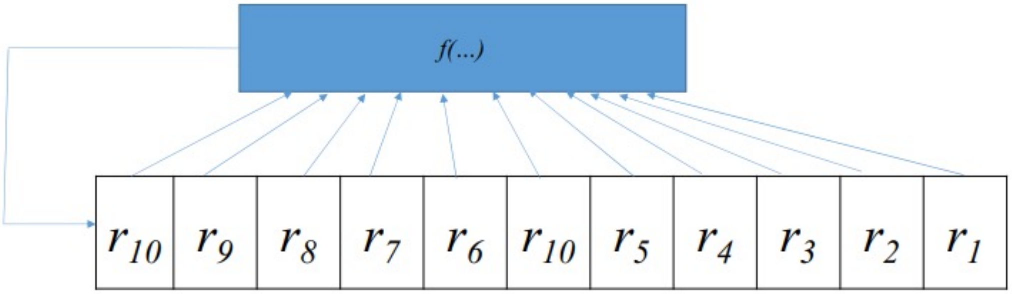


Exercise

- Let f(s) be a PRG, are the following PRGs?
  - g(s) = 1||f(s)
  - q(s) = (parity of s)||f(s)
  - w(s) = ~f(s)
    - ~ is the bitwise complement or negation

Many PRG proposals I

- Often based on Feedback Shift Register(s)
  - Easy construction for efficient hardware implementations.
  - Linear feedback (LFSR), or non-linear feedback function (f(…) 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

- More complex (multi-registers, etc.), e.g. in GSM
  - GSM's original stream-ciphers (A5/1, A5/2): broken
  - RC4; efficient for software implementations, but known attacks on 1st byte ☹
- 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 to encrypt:

PAD = RC4(password)

Save PAD ⊕ Document (bitwise XOR)

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

Attacker gets: c1=PAD xor d1, c2 = PAD xor 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 ➔ 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 ➔ $P \neq NP$
  - The ‘holy grail’ of the theory of complexity
- So don’t expect a ‘real’ provably-secure PRG
- Instead, 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 X ➔ 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 || x) = b || f(x), where $b \in \{0, 1\}$, 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 and how many bits (or bytes) of PRG output have been used so far.
- Can secure encryption be ***stateless***?
  - The answer is…

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: $\{0,1\}^m$
- 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: $\{0,1\}^m$
- 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:

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

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

	f( )
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:

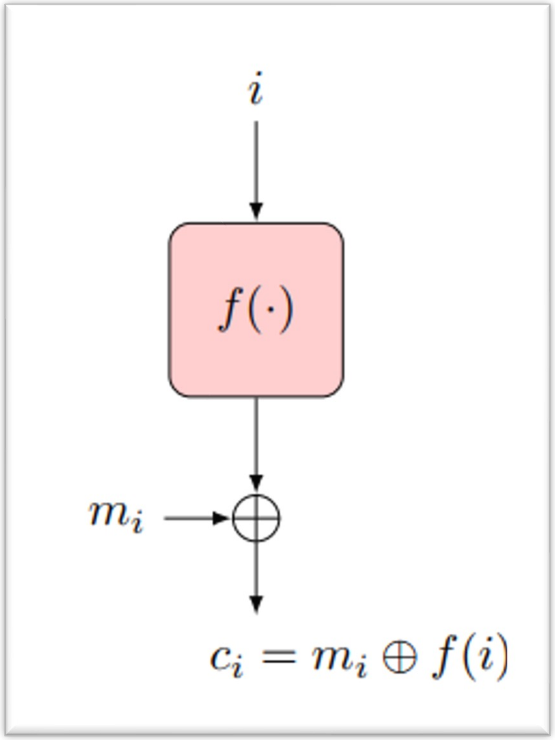
Domain: integers

i	f(i)
1	0
2	1
3	1
4	0
5	0
6	1
…	…

Range: bits {0,1}

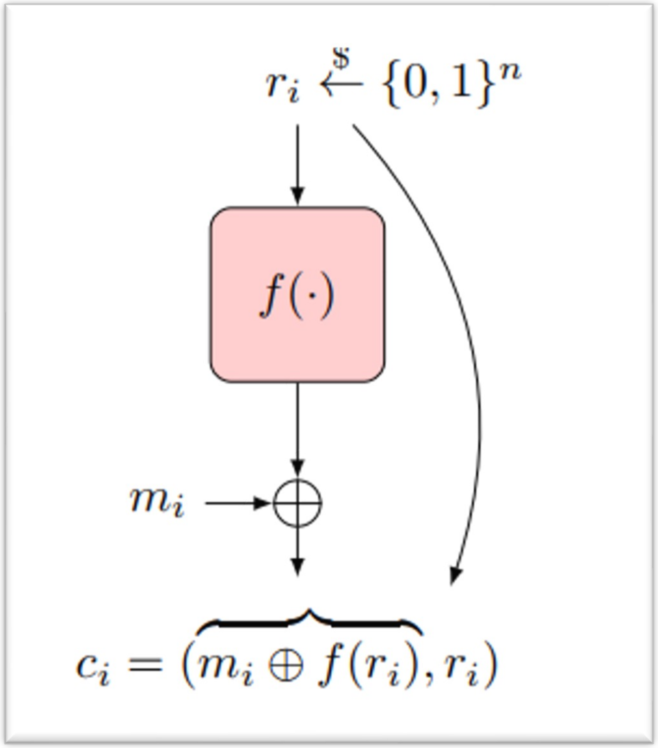
Random-Function-Based Encryption

Stateful (counter) Design



- **Sync-state (counter)**
- No extra random bits required
- |ciphertext|=|plaintext|

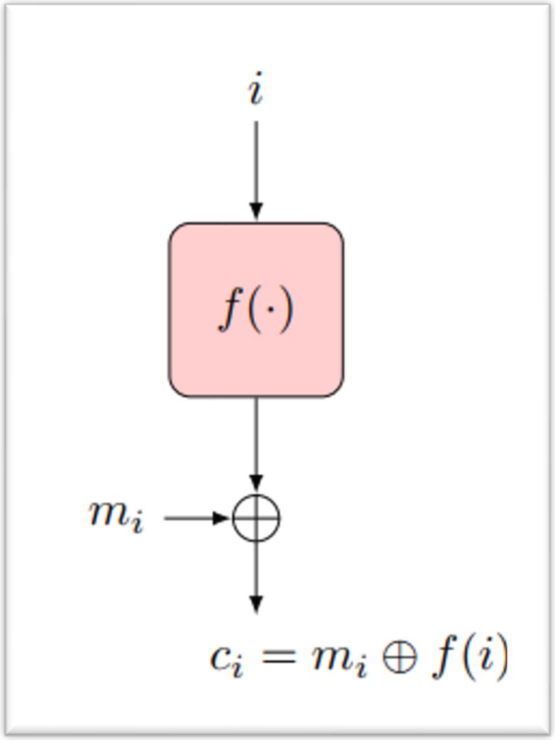
Randomized Design


$r_i \xleftarrow{\$} \{0,1\}^n$
$f(\cdot)$
m_i
$c_i = (\overbrace{m_i \oplus f(r_i), r_i})$

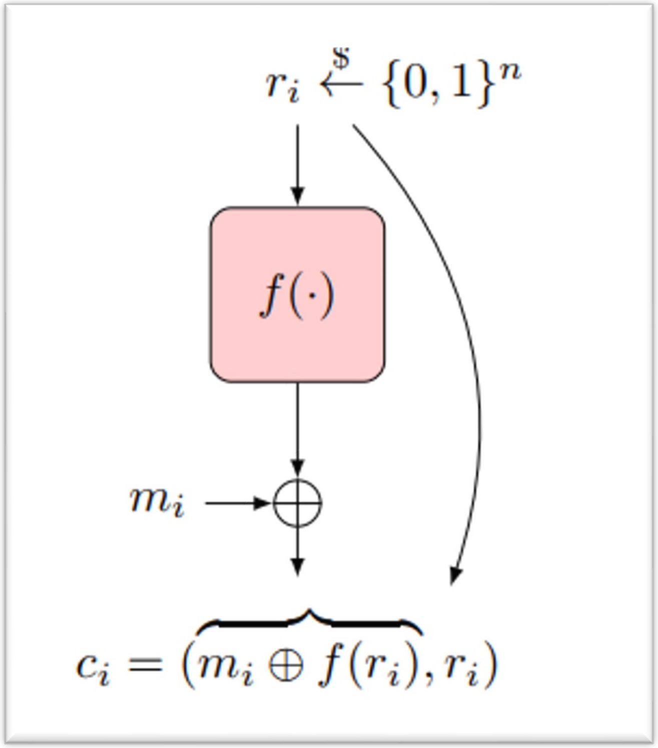
- **Stateless**
- n random bits per plaintext bit
- |ciphertext|=$(n+1)$ ·|plaintext|

Random-Function Bitwise-Encryption

Stateful (counter) Design



Randomized Design

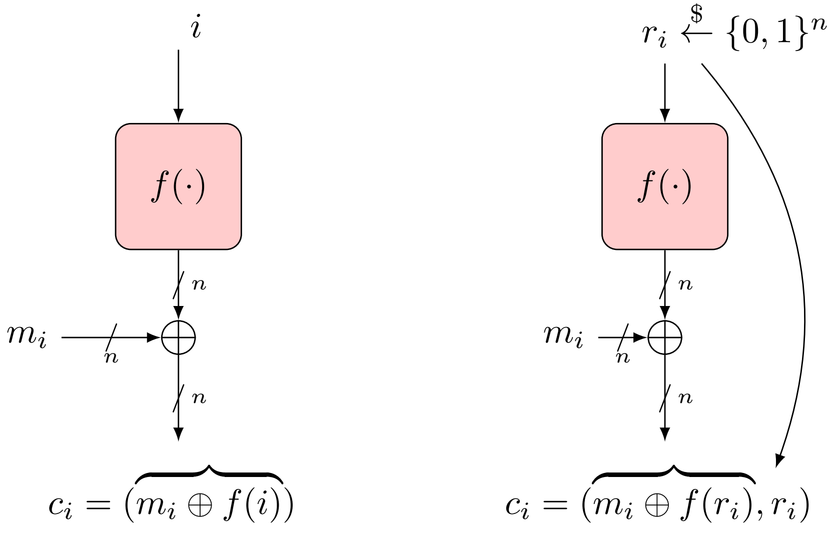

$r_i \xleftarrow{\$} \{0,1\}^n$
$f(\cdot)$
m_i
$c_i = (\overbrace{m_i \oplus f(r_i)}, r_i)$

Drawbacks:

- Require random function (impractical)
- Invoke function once-per-bit (computational overhead)

Reduce Overhead: Block-Encryption

- **Optimization:** operate in blocks (say of n bits)
  - f be random function from n-bits strings (`blocks’) to n-bits strings (`blocks’)
  - p(i) be i-th block of n-bits of plaintext
  - c(i) be i-th block of n-bits of ciphertext



i
$f(\cdot)$
n
m_i
n
n
$c_i = (\overbrace{m_i \oplus f(i)})$
$r_i \overset{\$}{\leftarrow} \{0,1\}^n$
$f(\cdot)$
n
m_i
n
n
$c_i = (\overbrace{m_i \oplus f(r_i)}, r_i)$

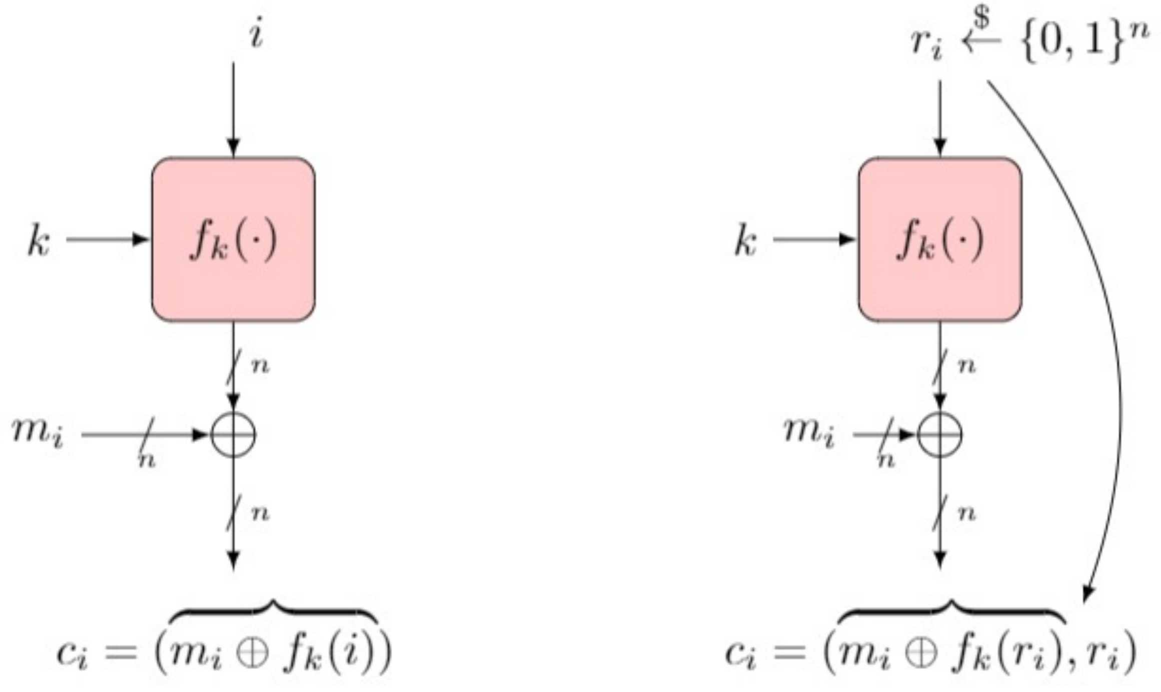
(a) Stateful block encryption with Random Function $f(\cdot)$.

(b) Stateless, randomized block encryption with Random Function $f(\cdot)$.

- **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

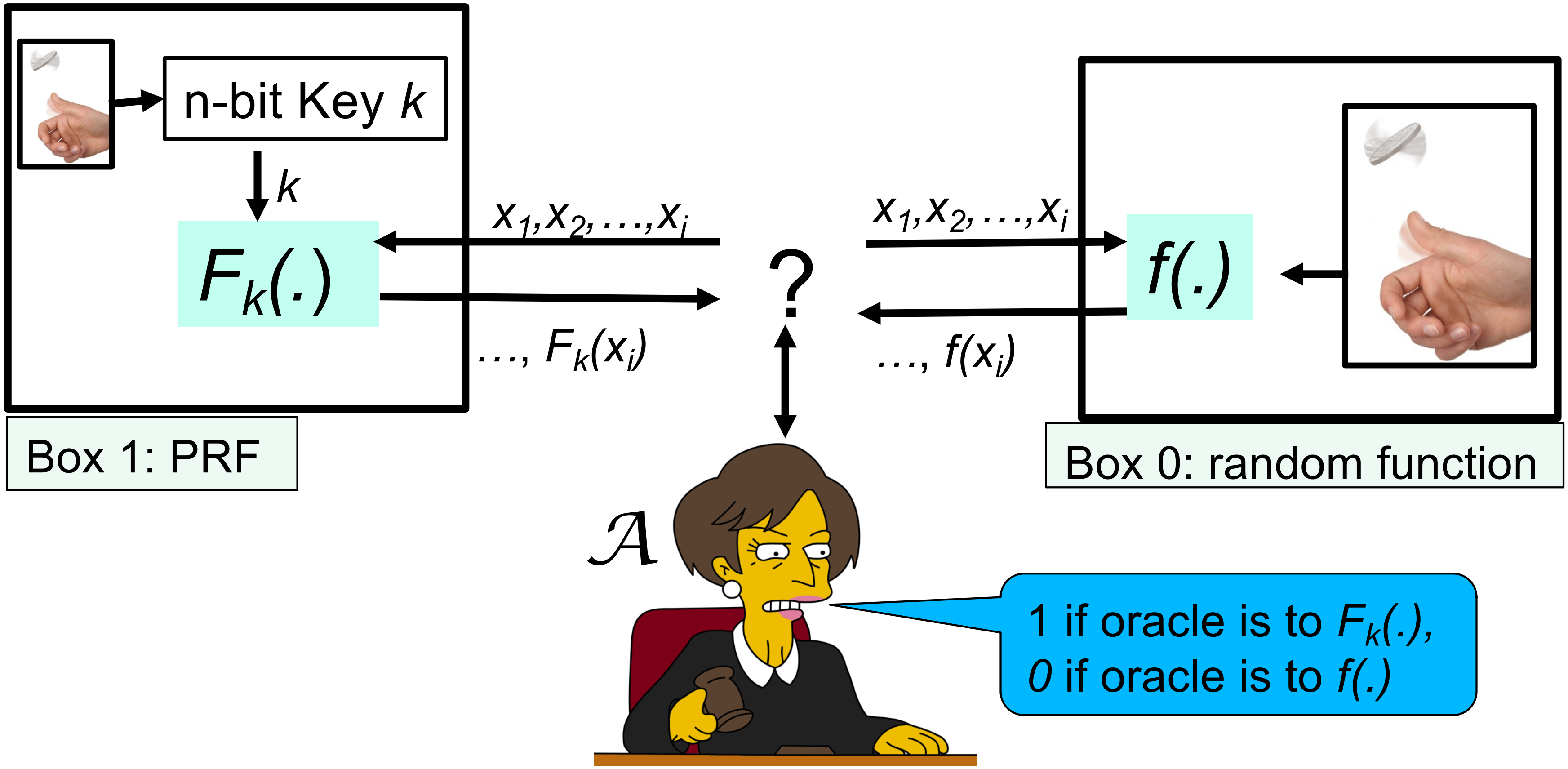
- Operate in blocks (say of n bits)
- Use Pseudo-Random Function (PRF) $f_k(\cdot)$, output n bits
  - Efficient , compact


i
k
$f_k(\cdot)$
n
m_i
n
n
$c_i = (\overbrace{m_i \oplus f_k(i)})$
$r_i \overset{\$}{\leftarrow} \{0,1\}^n$
k
$f_k(\cdot)$
n
m_i
n
n
$c_i = (\overbrace{m_i \oplus f_k(r_i)}, r_i)$

But what's a PRF ?

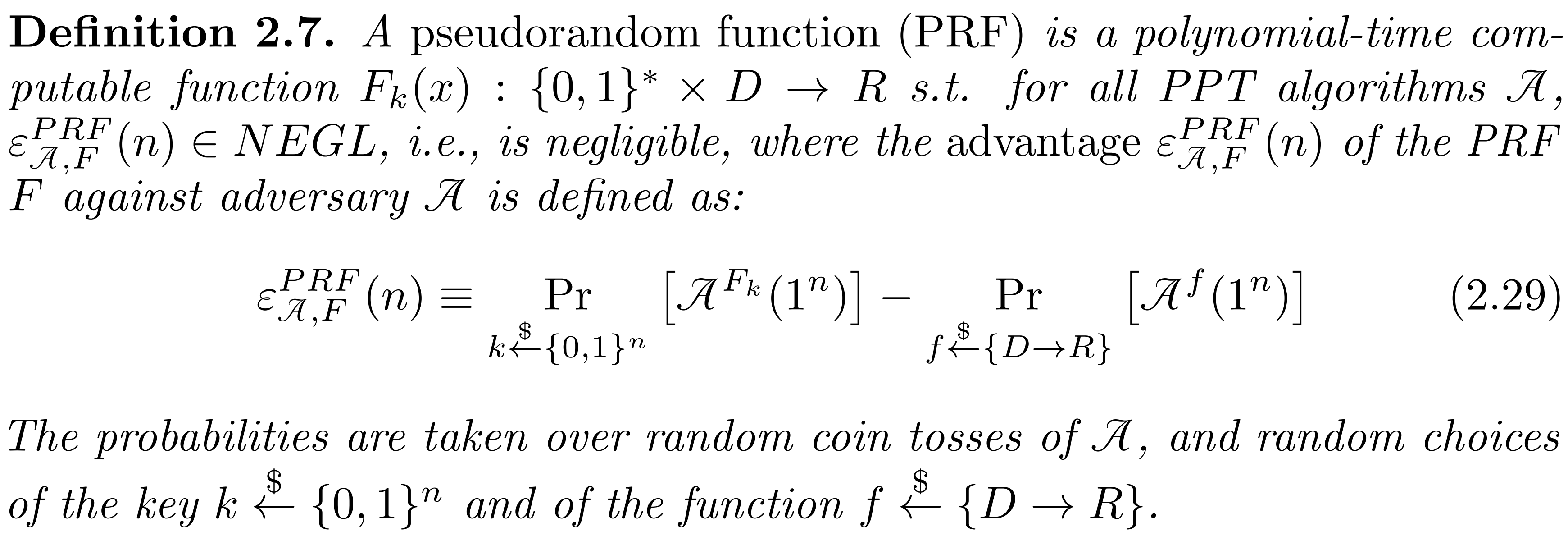
The PRF Indistinguishabity Test

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



Definition 2.7. *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 $\mathcal{A}$, $\varepsilon^{PRF}_{\mathcal{A},F}(n) \in NEGL$, i.e., is negligible, where the* advantage $\varepsilon^{PRF}_{\mathcal{A},F}(n)$ *of the PRF F against adversary $\mathcal{A}$ is defined as:*

$$\varepsilon^{PRF}_{\mathcal{A},F}(n) \equiv \Pr_{k \overset{\$}{\leftarrow} \{0,1\}^n} \left[\mathcal{A}^{F_k}(1^n)\right] - \Pr_{f \overset{\$}{\leftarrow} \{D \rightarrow R\}} \left[\mathcal{A}^{f}(1^n)\right] \tag{2.29}$$

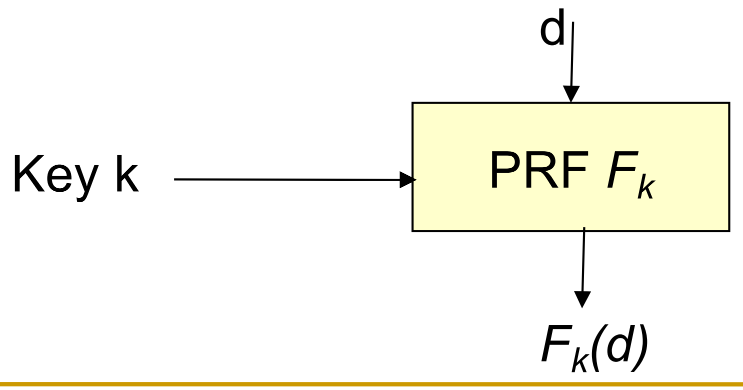
The probabilities are taken over random coin tosses of $\mathcal{A}$, and random choices of the key $k \overset{\$}{\leftarrow} \{0,1\}^n$ and of the function $f \overset{\$}{\leftarrow} \{D \rightarrow R\}$.

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
- Provable secure PRF without assumptions?
  - If exists, would imply that $P \neq NP$. Why?
    - Given the key k , it is trivial to identify the PRF
    - P : problems solvable in polynomial time
    - NP : same, but given also any ‘hint’ (e.g. key k)

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 on the white board

- Let Fk be a PRF, are the following PRFs and why?
  - $F'_k(x) = F_1^n(x) \,||\, F_k(x)$
  - $F''_k(x) = F_k(x) \,||\, lsb(F_k(x))$
    - lsb is the least significant bit

Covered Material From the Textbook

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

Thank You!