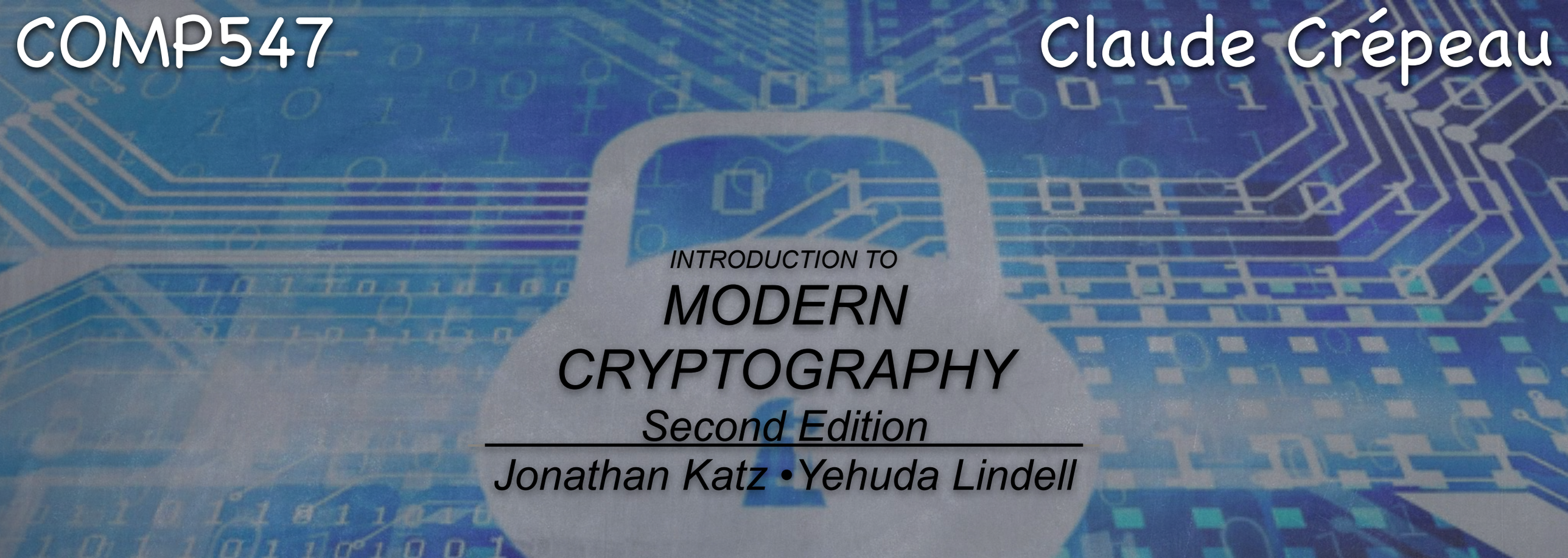
COMP547 Claude Crépeau



Chapter 3 : Private-Key Encryption

Private-Key Encryption

Private-Key Encryption

3.4 Stronger Security Notions
3.4.1 Security for Multiple Encryptions
3.4.2 Chosen-Plaintext Attacks and CPA-Security
3.5 Constructing CPA-Secure Encryption Schemes
3.5.1 Pseudorandom Functions and Block Ciphers
3.5.2 CPA-Secure Encryption from Pseudorandom Functions
3.6 Modes of Operation
3.6.1 Stream-Cipher Modes of Operation
3.6.2 Block-Cipher Modes of Operation
3.7 Chosen-Ciphertext Attacks (CCA)

3.1 Computational Security

- Study the notion of *pseudorandomness*
- Things can "look" completely random even though they are not
- This can be used to achieve secure encryption beating the previous limitations.

Computational Security

- Encryption schemes whereby a short key can be used to securely encrypt many long messages.
- Such schemes are able to bypass the inherent limitations of perfect secrecy
- Achieve the weaker but sufficient notion of computational secrecy.

A Computational Approach to Cryptography

- Modern encryption schemes have the property that they ***can*** be broken given enough time.
- Do not satisfy **Definition 2.3**, but for all practical purposes, the following level of security suffices.
- Under certain assumptions, the amount of computation needed to break these encryption schemes would take more than many lifetimes to carry out even using the fastest available supercomputers.

The Basic Idea of Computational Security

Kerckhoffs actually spelled out six principles, the following of which is very relevant to our discussion here:

> A [cipher] must be practically, if not mathematically, indecipherable.

The Basic Idea of Computational Security

- The computational approach incorporates two relaxations of the notion of perfect security:

1. Security is only preserved against *efficient* adversaries that run in a feasible amount of time
2. Adversaries can potentially succeed with some *very small probability*.

3.1.1 The concrete Approach

- The concrete approach quantifies the security of a given cryptographic scheme by explicitly bounding the maximum success probability of any adversary running for at most some fixed amount of time.
- That is, let t, ε be positive constants with $\varepsilon \leq 1$.
- A concrete definition of security:
  A scheme is **(t,ε)-secure** if every adversary running for time at most t succeeds in breaking the scheme with probability at most ε.

Example 3.1

- Modern private-key encryption schemes are generally assumed to give almost optimal security in the following sense:
- When the key has length n, an adversary running in time t can succeed in breaking the scheme with probability at most $ct/2^n$ for some fixed constant c.

Example 3.1

- Computation on the order of $t = 2^{60}$ is barely within reach today.
- Running on a **4 GHz** computer, 2^{60} CPU cycles require 2^{60} cycles / 4×10^9 cycles/second, or about **9** years.
- Fastest supercomputer : 1 minute.

Example 3.1

- A typical value for the key length might be ***n* = 128**.
- The difference between 2^{60} and 2^{128} is a multiplicative factor of 2^{68} which is a number containing about **21** decimal digits.
- Note that according to physicists' estimates the number of seconds since the big bang is in the order of 2^{58}.

The concrete approach

- When using the concrete security approach, schemes can be **(*t*,ε)-secure** but never just secure.

- For what ranges of ***t*,ε** should we say that a **(*t*,ε)-secure** scheme is "secure"?

- There is no clear answer to this, as a security guarantee that may suffice for the average user may not suffice when encrypting classified government documents.

3.1.2 The asymptotic approach

- This approach, rooted in complexity theory, views the running time of the adversary as well as its success probability as functions of a parameter rather than as concrete numbers.

- A cryptographic scheme will incorporate a security parameter which is an integer ***n***.

- When honest parties generate keys, they choose some value ***n*** for the security parameter; this value is assumed to be known to any adversary attacking the scheme.

The asymptotic approach

- The running time of the adversary (and of the honest parties) as well as the adversary's success probability are all viewed as functions of n.

- We equate the notion of "efficient adversaries" with probabilistic algorithms running in time polynomial in n. This means that for some constants a, c the algorithm runs in time $a \cdot n^c \in O(n^c)$ on security parameter n.

The asymptotic approach

- We require that honest parties run in polynomial time,
- Concerned with achieving security against polynomial-time adversaries.
- Adversarial strategies that require a super-polynomial amount of time are not considered realistic threats (and so are essentially ignored).

The asymptotic approach

- We equate the notion of "small probability of success" with success probabilities smaller than any inverse polynomial in n, meaning that for every constant c the adversary's success probability is smaller than n^{-c} for all large enough values of n.

- A function that grows slower than any inverse polynomial is called negligible. A definition of asymptotic security thus takes the following form:

- A scheme is secure if every ***Probabilistic Polynomial Time*** adversary succeeds in breaking the scheme with only negligible probability.

Example 3.2

- Say we have a scheme that is secure. Then it may be the case that an adversary running for n^3 minutes can succeed in "breaking the scheme" with probability $2^{40} \cdot 2^{-n}$.
- When $n \leq 40$ this means that an adversary running for 40^3 minutes (about **6** weeks) can break the scheme with probability **1**, so such values of n are not going to be very useful.

Example 3.2

- Even for $n = 50$ an adversary running for 50^3 minutes (about **3** months) can break the scheme with probability roughly $1/1000$, which may not be acceptable.

- On the other hand, when $n = 500$ an adversary running for more than **200** years breaks the scheme only with probability roughly 2^{-500}.

Example 3.3

- Let us see the effect that the availability of faster computers might have on security in practice.
- Say we have a cryptographic scheme where honest parties are required to run for $10^6 \cdot n^2$ cycles, and for which an adversary running for $10^8 \cdot n^4$ cycles can succeed in "breaking" the scheme with probability $2^{20} \cdot 2^{-n}$.

Example 3.3

- Say all parties are using a **2 Ghz** computer and $n = 80$.
- Then honest parties run for $10^6 \cdot 6400$ cycles, or **3.2** seconds, and an adversary running for $10^8 \cdot 80^4$ cycles, or roughly **3** week, can break the scheme with probability only 2^{-40}.

Example 3.3

- Say **8 Ghz** computers become available, and all parties upgrade.
- Honest parties can increase ***n*** to **160** (which requires generating a fresh key) and still maintain their running time to **3.2** seconds.
- In contrast, the adversary now has to run for **8 million** seconds, or more than **13** weeks, to achieve success probability $\mathbf{2^{-80}}$.
- The effect of a faster computer has been to make the adversary's job harder!!!

Necessity of the Relaxations

- Assume we have an encryption scheme where the size of the key space $\mathcal{K}$ is much smaller than the size of the message space M.
- Two attacks, lying at opposite extremes, apply regardless of how the encryption scheme is constructed:

Necessity of the Relaxations : **Brute-Force Search**

- Given a ciphertext **c**, an adversary can decrypt **c** using all keys $k \in \mathcal{K}$.
- This gives a list of all possible messages to which **c** can possibly correspond.
- Since this list cannot contain all of $\mathcal{M}$ (because $|\mathcal{K}| < |\mathcal{M}|$), this leaks some information about the message that was encrypted.

Necessity of the Relaxations : Brute-Force Search

- Moreover, say the adversary carries out a *known-plaintext attack* and learns that ciphertexts $c_1,\ldots,c_\ell$ correspond to the messages $m_1,\ldots,m_\ell$ respectively.
- The adversary can again try decrypting each of these ciphertexts with all possible keys until it finds a key k for which $\mathbf{Dec}_k(c_i) = m_i$ for all i.

Necessity of the Relaxations : Brute-Force Search

- This key will be unique with high probability, in which case the adversary has found the key that the honest parties are using.
- Subsequent usage of this key will therefore be insecure.
- The type of attack succeeds with probability essentially 1 in time linear in $|\mathcal{K}|$.

Necessity of the Relaxations : **Random Attack**

- Consider again the case where the adversary learns that $\mathbf{c}_1,\ldots,\mathbf{c}_\ell$ correspond to $\boldsymbol{m}_1,\ldots,\boldsymbol{m}_\ell$.
- The adversary can guess a key $\mathbf{k} \in \mathcal{K}$ at random and check to see whether $\mathbf{Dec}_k(\mathbf{c}_i) = \boldsymbol{m}_i$ for all $\boldsymbol{i}$.
- If so, we again expect that with high probability $\boldsymbol{k}$ is the key that the honest parties are using.
- Here the adversary runs in essentially constant time and succeeds with non-zero (although very small) probability of roughly $1/|\mathcal{K}|$.

Necessity of the Relaxations

- It follows that if we wish to encrypt many messages using a single short key, security can only be achieved if we limit the running time of the adversary[1] and also allow a very small probability of success without considering it a break[2].

[1] so that the adversary does not have time to carry out a brute-force search.
[2] so that the second "random attack" is ruled out.

Efficient Algorithms and Negligible Success

- We define efficient computation as that which can be carried out in *Probabilistic Polynomial Time* (abbreviated **PPT**).

- An algorithm **A** is said to run in polynomial time if there exists a polynomial $p(\cdot)$ such that, for every input $x \in \{0,1\}^*$, the computation of $A(x)$ terminates within at most $p(|x|)$ steps
  (here, $|x|$= length of the string x).

Efficient Algorithms and Negligible Success

- A probabilistic algorithm is one that has the capability of "tossing coins";
- This is a metaphorical way of saying that the algorithm has access to a source of randomness that yields unbiased random bits that are each independently equal to **1** with probability ½ and to **0** with probability ½.

Efficient Algorithms and Negligible Success

DEFINITION 3.4 A function **f** is *negligible* if for every polynomial $p(\cdot)$ there exists an N such that for all integers $n > N$ it holds that

$$f(n) < 1/p(n)$$

Efficient Algorithms and Negligible Success

PROPOSITION 3.6 Let $\mathbf{negl}_1$ and $\mathbf{negl}_2$ be negligible functions of an integer n. Then,

1. The function
$\mathbf{negl}_3(n) = \mathbf{negl}_1(n) + \mathbf{negl}_2(n)$ is also negligible.

2. For any positive polynomial p,
the function $\mathbf{negl}_4(n) = p(n) \cdot \mathbf{negl}_1(n)$ is also negligible.

Efficient Algorithms and Negligible Success

- Events that occur with negligible probability are so unlikely that they can be ignored for all practical purposes.
- Therefore, a break of a cryptographic scheme that occurs with negligible probability is not significant.

Asymptotic Security: A Summary

- The general framework of any security definition will be :

  A scheme is secure if for every **PPT** adversary **A** carrying out an attack of some specified type, the probability that **A** succeeds in this attack (where success is also well-defined) is negligible.

- Such a definition is asymptotic because it is possible that for small values of ***n*** an adversary can succeed with high probability.

Asymptotic Security: A Summary

- In order to see this in more detail, we will use the full definition of "negligible" in the above statement:

  A scheme is secure if for every **PPT** adversary **A** carrying out an attack of some specified type, and for every polynomial $p(\cdot)$, there exists an integer N such that the probability that **A** succeeds is less than $1/p(n)$ for every $n > N$.

- Note that nothing is guaranteed for values $n \leq N$.

3.2 Defining Computationally Secure Encryption

DEFINITION 3.7 A private-key encryption scheme is a tuple of probabilistic polynomial-time algorithms **(Gen, Enc, Dec)** such that:

1/3. The key-generation algorithm **Gen** takes as input the security parameter 1^n and outputs a key k; we write this as $k \leftarrow \mathbf{Gen}(1^n)$ (thus emphasizing the fact that **Gen** is a randomized algorithm).
We will assume without loss of generality that any key $k \leftarrow \mathbf{Gen}(1^n)$ satisfies $|k| \leq n$.

Defining Computationally Secure Encryption

DEFINITION 3.7 A private-key encryption scheme is a tuple of probabilistic polynomial-time algorithms **(Gen, Enc, Dec)** such that:

2/3. The encryption algorithm **Enc** takes as input a key k and a plaintext message $m \in \{0,1\}^*$, and outputs a ciphertext c. Since **Enc** may be randomized, we write $c \leftarrow \mathbf{Enc}_k(m)$.

Defining Computationally Secure Encryption

DEFINITION 3.7 A private-key encryption scheme is a tuple of probabilistic polynomial-time algorithms **(Gen, Enc, Dec)** such that:

3/3. The decryption algorithm **Dec** takes as input a key k and a ciphertext c, and outputs a message m. We assume that **Dec** is deterministic, and so write this as $m := \mathbf{Dec}_k(c)$.

Defining Computationally Secure Encryption

- It is required that for every n, every key k output by $\mathbf{Gen}(1^n)$, and every $m \in \{0,1\}^*$, it holds that $\mathbf{Dec}_k(\mathbf{Enc}_k(m)) = m$.

- If **(Gen, Enc, Dec)** is such that for k output by $\mathbf{Gen}(1^n)$, algorithm $\mathbf{Enc}_k$ is only defined for $m \in \{0,1\}^{\ell(n)}$, then we say that **(Gen, Enc, Dec)** is a fixed-length private-key encryption scheme for messages of length $\ell(n)$.

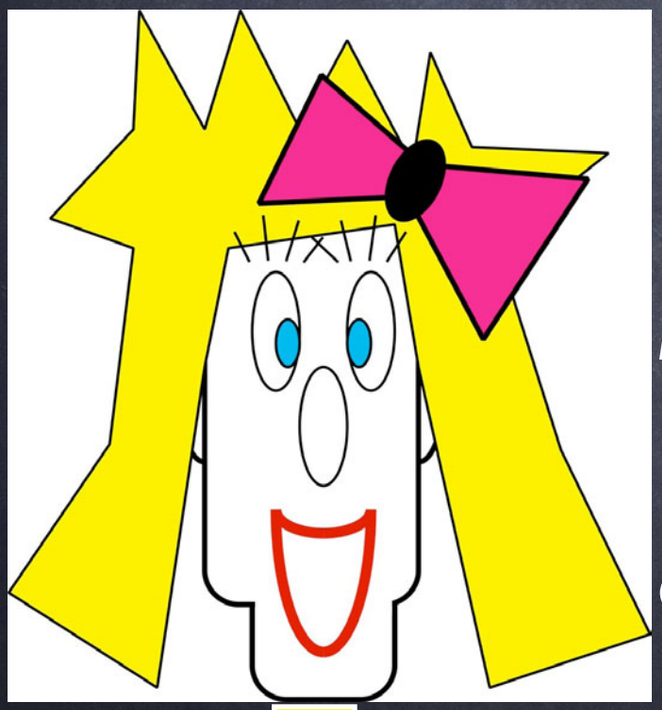
Indistinguishability in the presence of an eavesdropper

An experiment is defined for any private-key encryption scheme Π = (**Gen**, **Enc**, **Dec**), any **PPT** adversary **A** and any value ***n*** for the security parameter.

The eavesdropping indistinguishability experiment

$\mathbf{PrivK}^{eav}_{A,\Pi}(n)$:

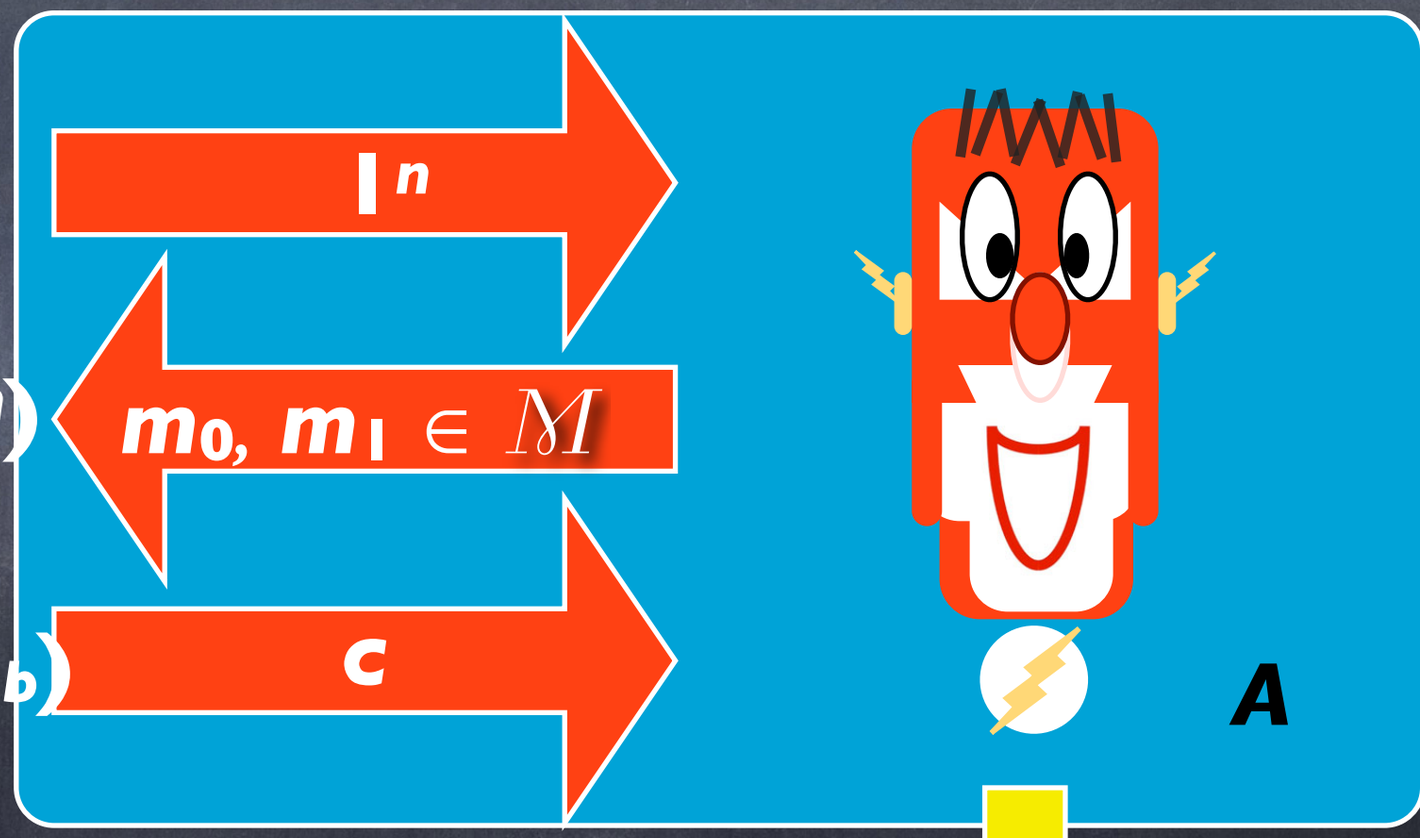
$\mathsf{PrivK}^{\mathsf{eav}}_{A,\Pi}$



$k \leftarrow \mathsf{Gen}(1^n)$

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

$c \leftarrow \mathsf{Enc}_k(m_b)$



b

b'

computationally secret

$\Pr[\, b = b' \,] \leq \frac{1}{2} + \mathsf{negl}(n)$

$\mathbf{PrivK}^{eav}_{A,\Pi}(n)$

1. The adversary **A** is given input 1^n, and outputs a pair of messages m_0 , m_1 of the same length.

2. A key k is generated by running $\mathbf{Gen}(1^n)$, and a random bit $b \leftarrow \{0,1\}$ is chosen. A (challenge) ciphertext $c \leftarrow \mathbf{Enc}_k(m_b)$ is computed and given to **A**.

3. **A** outputs a bit b'.

4. The output of the experiment is defined to be **1** if $b' = b$, and **0** otherwise.

(If $\mathbf{PrivK}^{eav}_{A,\Pi}(n) = 1$, we say that **A** succeeded.)

$\mathbf{PrivK}^{eav}_{A,\Pi}(n)$

- If Π is a fixed-length scheme for messages of length $\ell(n)$, the previous experiment is modified by requiring $m_0, m_1 \in \{0,1\}^{\ell(n)}$.

Defining Computationally-Secure Encryption

DEFINITION 3.8 A private-key encryption scheme **Π = (Gen, Enc, Dec)** has indistinguishable encryptions in the presence of an eavesdropper if for all **PPT** adversaries **A** there exists a negligible function **negl** such that

$$\mathbf{Pr}[\,\mathbf{PrivK}^{\mathbf{eav}}_{\mathbf{A},\Pi}(n) = 1\,] \leq \tfrac{1}{2} + \mathbf{negl}(n),$$

where the probability is taken over the random coins used by **A**, as well as the random coins used in the experiment (for choosing the key, the random bit **b**, and any random coins used in the encryption process).

Defining Computationally-Secure Encryption

DEFINITION 3.9 A private-key encryption scheme **Π = (Gen, Enc, Dec)** has indistinguishable encryptions in the presence of an eavesdropper if for all **PPT** adversaries **A** there exists a negligible function **negl** such that

$$|\Pr[\text{output}(\text{PrivK}^{\text{eav}}_{A,\Pi}(n, b=0)) = 1] - \Pr[\text{output}(\text{PrivK}^{\text{eav}}_{A,\Pi}(n, b=1)) = 1]| \leq \text{negl}(n).$$

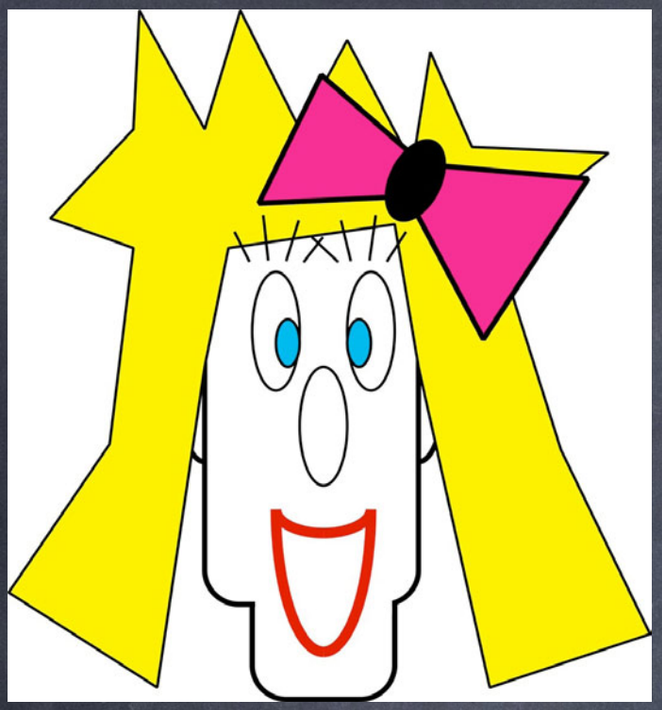
The fact that this definition is equivalent to **Definition 3.8** is left as an exercise.

3.2.2 *Semantic Security

DEFINITION 3.12 A private-key encryption scheme **(Gen, Enc, Dec)** is *semantically* secure in the presence of an eavesdropper if for every **PPT** algorithm **A** there exists a **PPT** algorithm **A′** such that for all efficiently-sampleable distributions $\boldsymbol{X} = (X_1,\ldots)$ and all polynomial-time computable functions **f** and **h**, there exists a negligible function **negl** s.t.

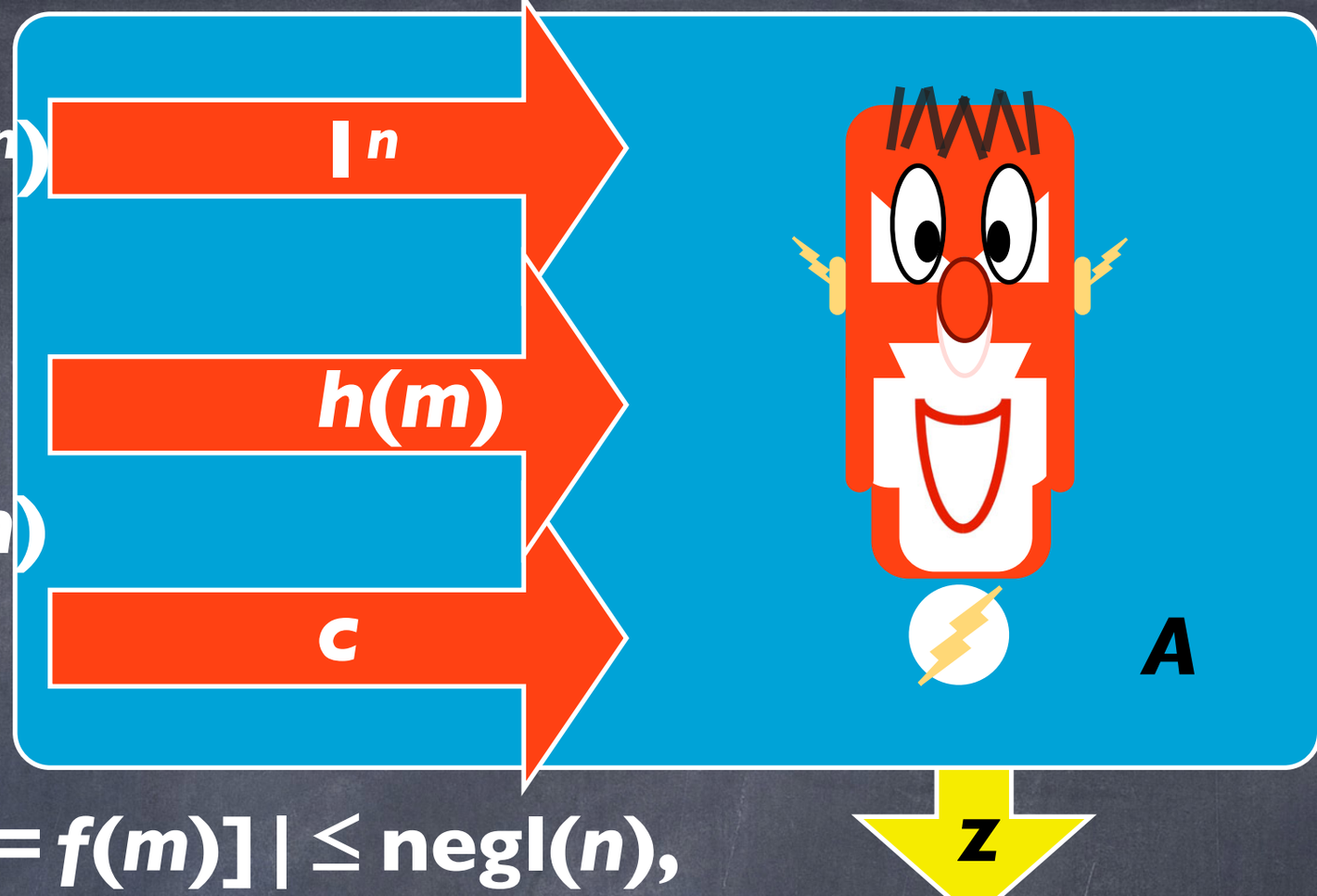
$$|\Pr[A(1^n, \mathrm{Enc}_k(m), h(m)) = f(m)] - \Pr[A'(1^n, |m|, h(m)) = f(m)]| \leq \mathrm{negl}(n),$$

where m is chosen according to distribution X_n , and the probabilities are taken over the choice of m and the key k, and any random coins used by **A, A′**, and the encryption process.

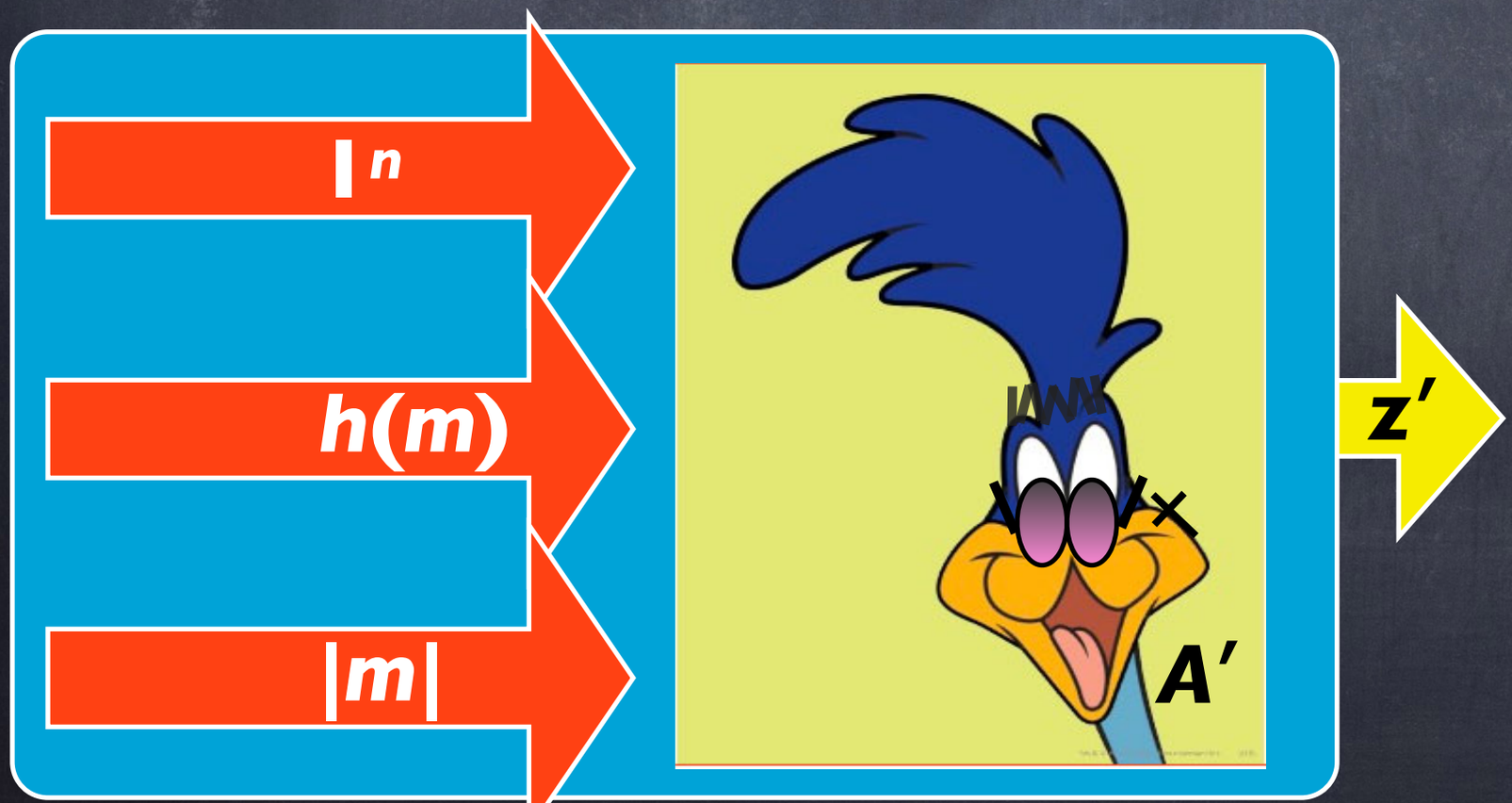


$k \leftarrow \mathrm{Gen}(1^n)$

$c \leftarrow \mathrm{Enc}_k(m)$



$$|\Pr[z = f(m)] - \Pr[z' = f(m)]| \leq \mathrm{negl}(n),$$



Semantic Security

THEOREM 3.13 A private-key encryption scheme has indistinguishable encryptions in the presence of an eavesdropper

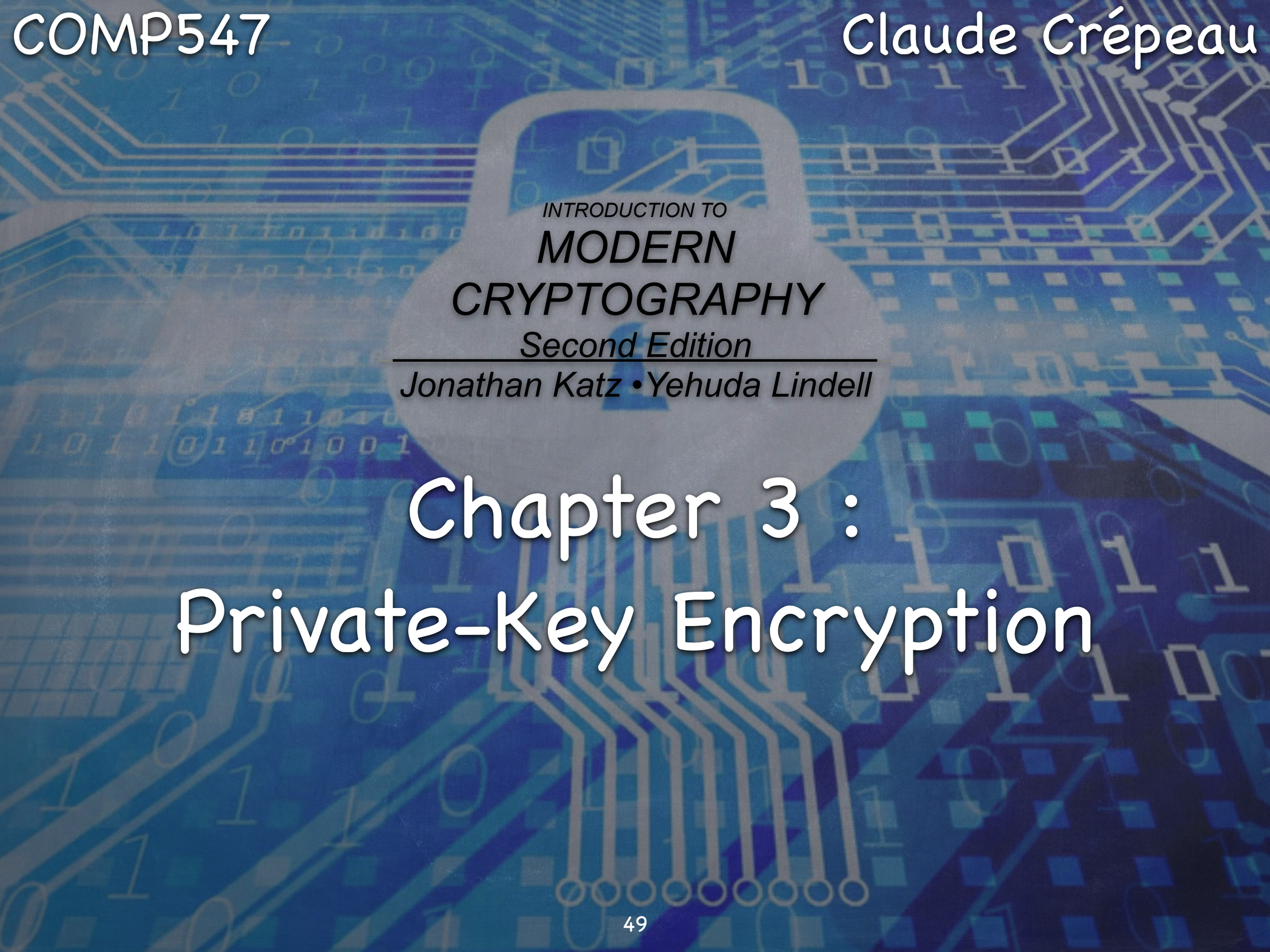
if and only if

it is semantically secure in the presence of an eavesdropper.

Shafi Goldwasser

Silvio Micali

COMP547 Claude Crépeau



Chapter 3 : Private-Key Encryption

3.3 Constructing Secure Encryption Schemes

- Loosely speaking, a pseudorandom string is a string that looks like a uniformly distributed string, as long as the entity that is "looking" runs in polynomial time.
- Just as indistinguishability can be viewed as a computational relaxation of perfect secrecy, pseudorandomness is a computational relaxation of true randomness.

3.3.1 Pseudorandom Generators and Stream Ciphers

- An important conceptual point is that, technically speaking, no fixed string can be said to be "pseudorandom" (in the same way that it does not make much sense to refer to any fixed string as "random").
- Pseudorandomness actually refers to a *distribution* over strings, and when we say that a distribution ***D*** over strings of length ℓ is pseudorandom this means that ***D*** is indistinguishable from the uniform distribution over strings of length ℓ.

Pseudorandomness

- Strictly speaking, since we are in an asymptotic setting we actually need to speak of the pseudorandomness of a *sequence* of distributions $\boldsymbol{D} = \{\boldsymbol{D_n}\}$, where distribution $\boldsymbol{D_n}$ is associated with security parameter $\boldsymbol{n}$.

  We ignore this point in our current discussion.

- More precisely, it is infeasible for any **PPT** algorithm to tell whether it is given a string sampled according to $\boldsymbol{D}$ or an $\boldsymbol{\ell}$-bit string chosen uniformly at random.

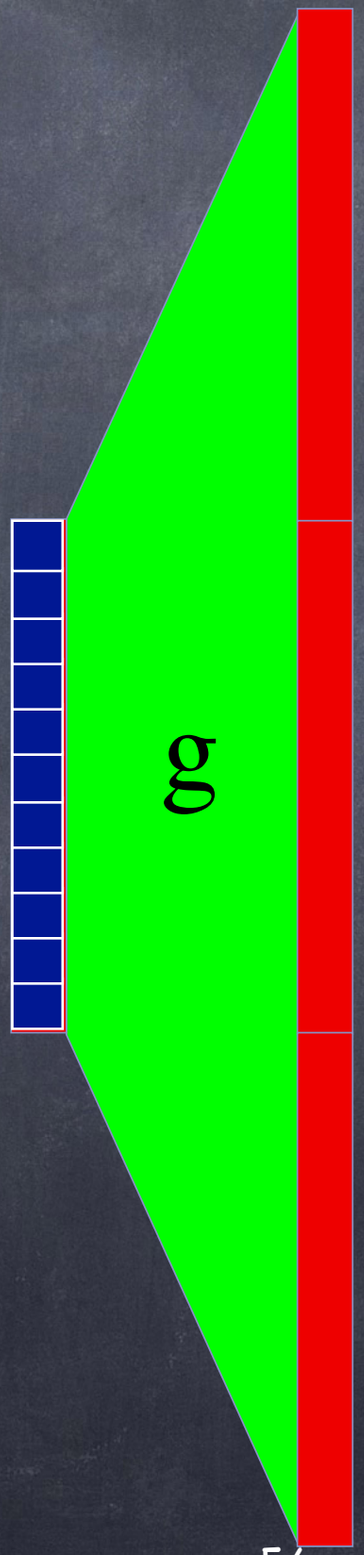
Pseudorandom Generators

- A pseudorandom generator is a *deterministic* algorithm that receives a short truly random seed and stretches it into a long string that is pseudorandom.
- Stated differently, a pseudorandom generator uses a small amount of true randomness in order to generate a large amount of pseudorandomness.

Pseudorandom Generators

RANDOM

x



g(x)

SEEMS RANDOM

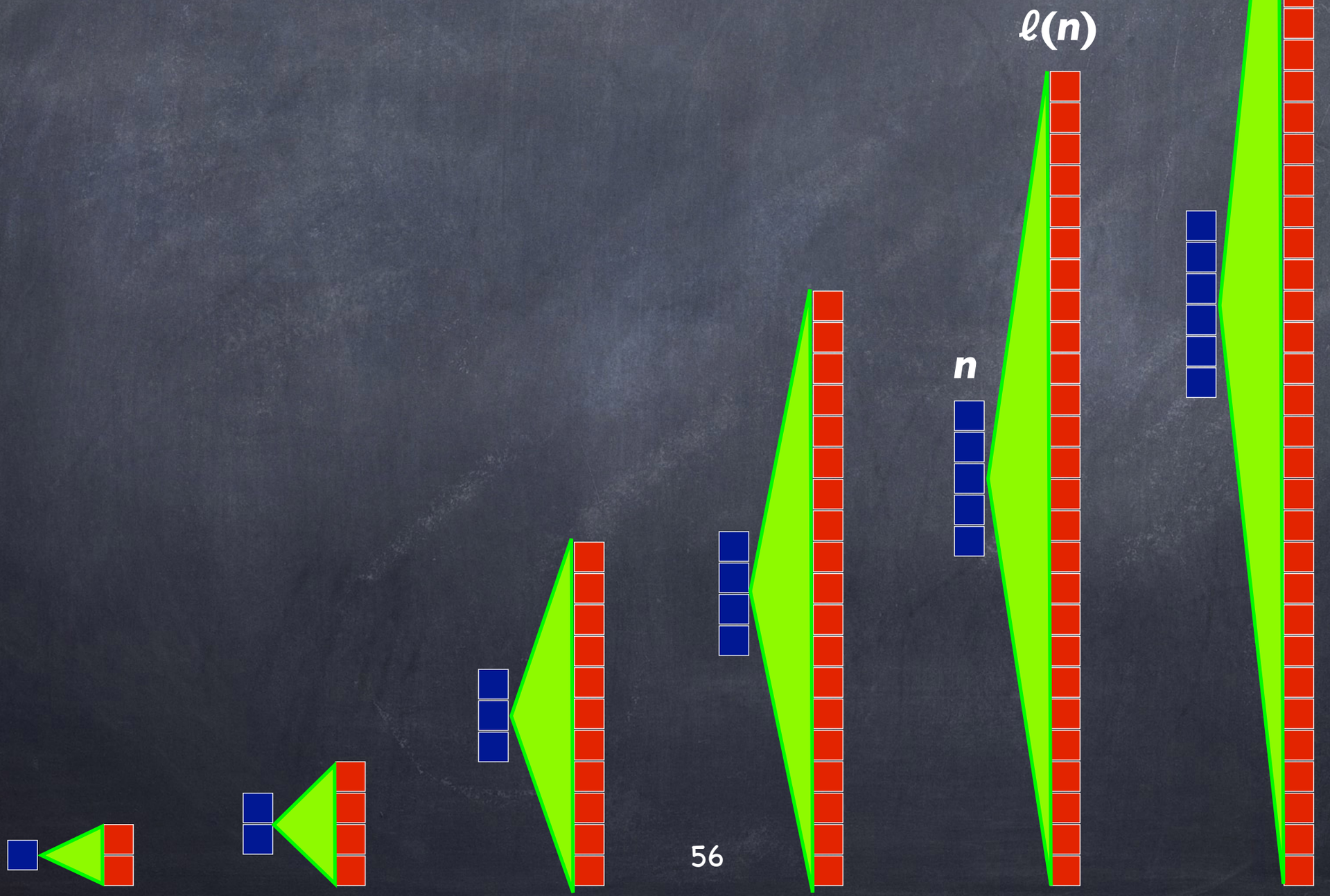
Pseudorandom Generators

In the definition that follows, we set ***n*** to be the length of the seed that is input to the generator and ***ℓ(n)*** to be the output length.

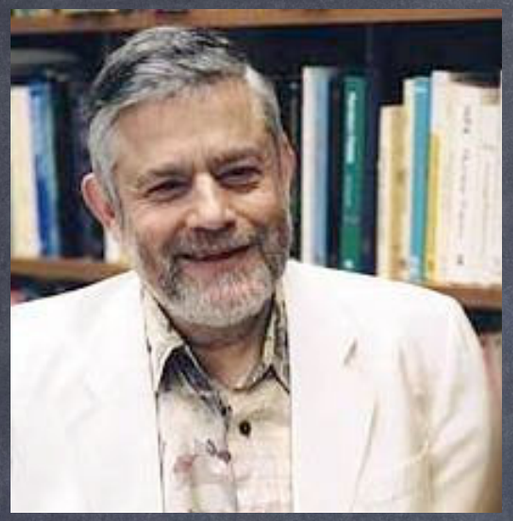
The generator is only interesting if ***ℓ(n) > n***.

Otherwise, it doesn't generate any new (apparent) randomness

Pseudorandom Generators



Pseudorandom Generators



Manuel Blum

Silvio Micali

DEFINITION 3.14 *Let $\ell(\cdot)$ be a polynomial and let* **G** *be a deterministic polynomial-time algorithm such that for any input $s \in \{0,1\}^n$, algorithm* **G** *outputs a string of length $\ell(n)$. (The function ℓ is called the* expansion factor *of* **G***). We say that* **G** *is a* pseudorandom generator *if the following two conditions hold:*

Pseudorandom Generators

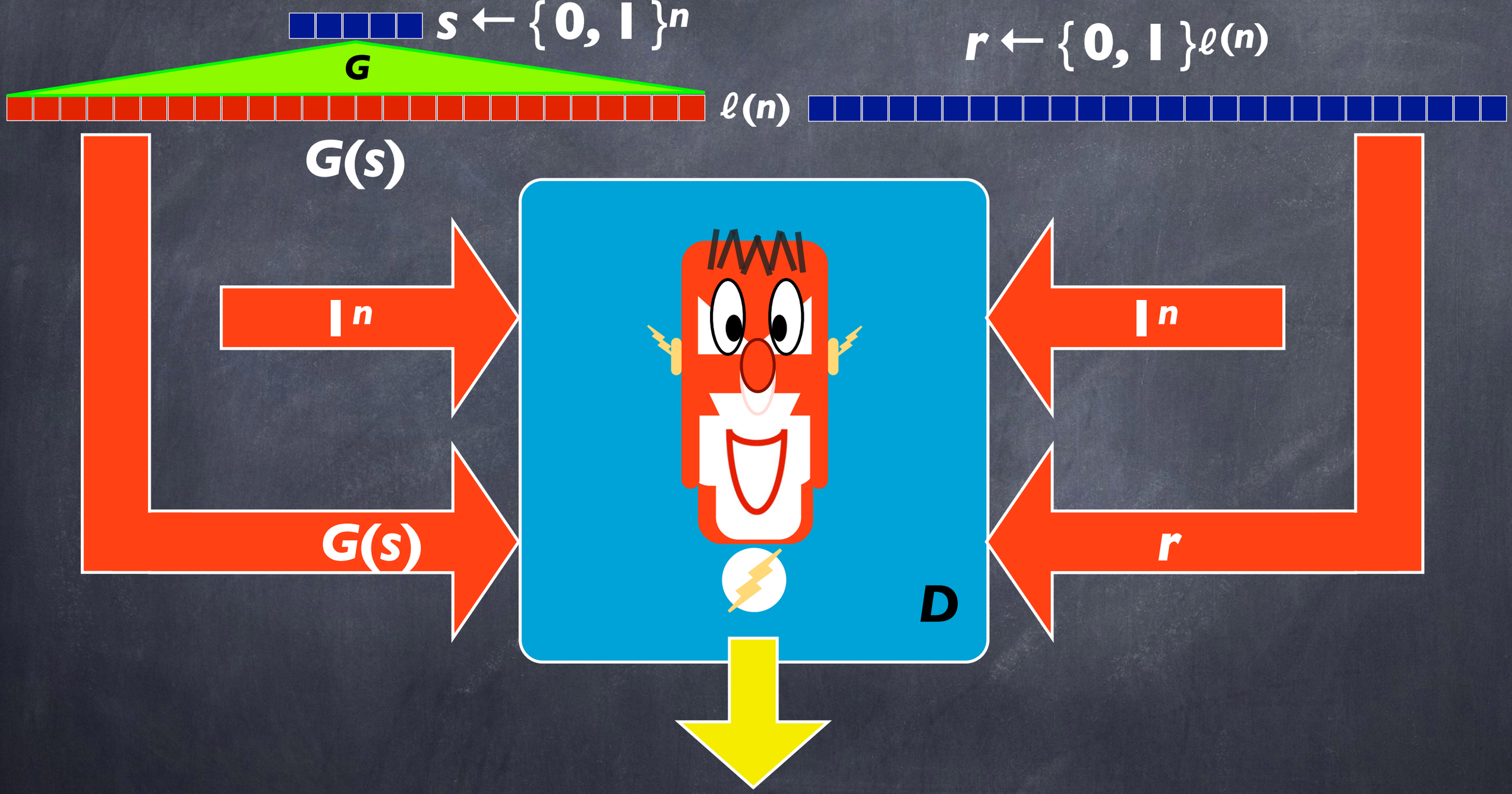
1. (Expansion:) *For every* n *it holds that* $\ell(n) > n$.

2. (Pseudorandomness:) *For all* **PPT** *distinguishers* D, *there exists a negligible function* **negl** *such that:*

$$|\Pr[D(r) = 1] - \Pr[D(G(s)) = 1]| \leq \mathbf{negl}(n),$$

where r *is chosen uniformly at random from* $\{0,1\}^{\ell(n)}$, *the seed* s *is chosen uniformly at random from* $\{0,1\}^n$, *and the probabilities are taken over the random coins used by* D *and the choice of* r *and* s.

Pseudorandom Generators



computational indistinguishability

$$|\Pr[D(G(s)) = 1] - \Pr[D(r) = 1]| \leq \mathrm{negl}(n)$$

Pseudorandom Generators: Discussion.

- It is trivial to distinguish between a random string and a pseudorandom string *given an unlimited amount of time*.

- Upon input some string w, distinguisher D outputs 1 if and only if there exists a string $s \in \{0,1\}^n$ such that $G(s) = w$.

$$|\Pr[D(r) = 1] - \Pr[D(G(s)) = 1]| = 1 - 2^{-n}$$

The seed and its length

- The seed for a pseudorandom generator must be chosen uniformly at random, and be kept entirely secret from the distinguisher.

- Another important point, evident from the above discussion of brute-force attacks, is that ***s*** must be long enough so that no "efficient algorithm" has time to traverse all possible seeds.

- Technically, this is taken care of by the fact that all algorithms are assumed to run in polynomial time and thus cannot search through all 2^n possible seeds when ***n*** is large enough.

Existence of Pseudorandom Generators

- The first question one should ask is whether any entity satisfying **Definition 3.14** exists.

- Unfortunately, we do not know how to unequivocally prove the existence of pseudorandom generators.

Existence of Pseudorandom Generators

- We believe that pseudorandom generators exist, and this belief is based on the fact that they can be constructed under the rather weak assumption that *one-way functions* exist.
- In practice, various constructions believed to act as pseudorandom generators are known.

Stream Ciphers

Formally, we view a stream cipher as a pair of deterministic algorithms (Init, GetBits) where:

- Init takes as input a seed s and an optional initialization vector IV , and outputs an initial state st_0.
- GetBits takes as input state information st_i, and outputs a bit y and updated state st_{i+1}. (In practice, y is a block of several bits; we treat y as a single bit here for generality and simplicity.)

Stream Ciphers

ALGORITHM 3.16
Constructing G_ℓ from (Init, GetBits)

Input: Seed s and optional initialization vector IV
Output: $y_1, \ldots, y_\ell$

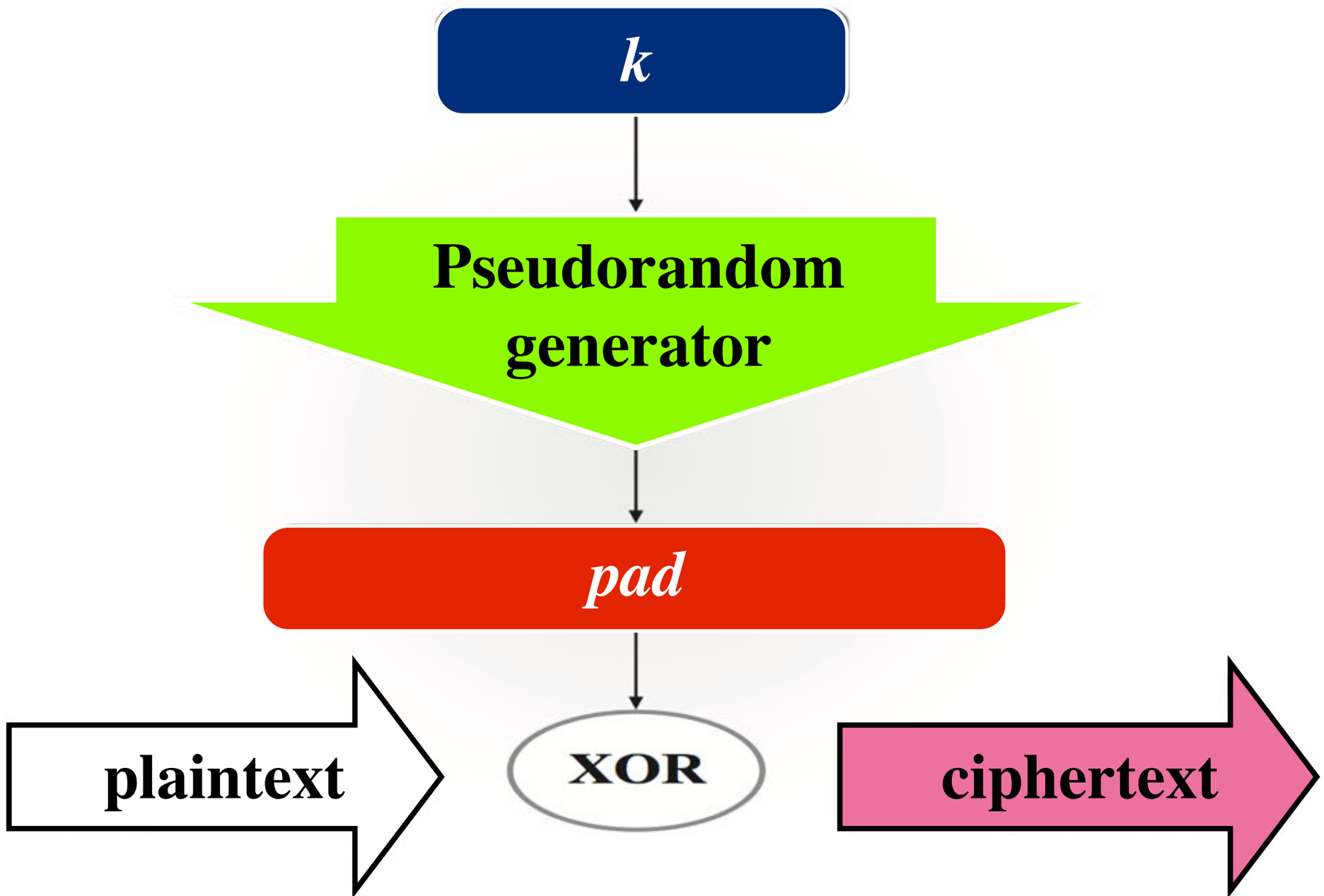
$\mathsf{st}_0 := \mathsf{Init}(s, IV)$
for $i = 1$ to ℓ:
    $(y_i, \mathsf{st}_i) := \mathsf{GetBits}(\mathsf{st}_{i-1})$
return $y_1, \ldots, y_\ell$

Stream Ciphers

A stream cipher is secure:

- In the basic sense if it takes no ***IV*** and for any polynomial ℓ with $\ell(n) > n$, the function G_ℓ is a pseudorandom generator with expansion factor ℓ.
- One possible security notion for stream ciphers that use an ***IV*** is discussed in Section **3.6.1**.

3.3.3 Secure Fixed-Length Encryption Schemes



A Secure Fixed-Length Encryption Scheme

CONSTRUCTION 3.17

Let G be a pseudorandom generator with expansion factor ℓ. Define a private-key encryption scheme for messages of length ℓ as follows:

- Gen: on input 1^n, choose uniform $k \in \{0,1\}^n$ and output it as the key.
- Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^{\ell(n)}$, output the ciphertext
$$c := G(k) \oplus m.$$
- Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c \in \{0,1\}^{\ell(n)}$, output the message
$$m := G(k) \oplus c.$$

A private-key encryption scheme based on any pseudorandom generator.

A Secure Fixed-Length Encryption Scheme

- **THEOREM 3.18** *If* **G** *is a pseudorandom generator, then* **Construction 3.17** *is a fixed-length private-key encryption scheme that has indistinguishable encryptions in the presence of an eavesdropper.*

A Secure Fixed-Length Encryption Scheme

PROOF IDEA

Let Π denote **Construction 3.17**.

We show that if there exists a **PPT** adversary **A** for which **Definition 3.8** does not hold, then we can construct a probabilistic polynomial-time algorithm that distinguishes the output of ***G*** from a truly random string.

A Secure Fixed-Length Encryption Scheme

The intuition behind this claim is that if **Π** used a truly random string in place of the pseudorandom string ***G*(*k*)**, then the resulting scheme would be identical to the one-time pad encryption scheme and ***A*** would be unable to correctly guess which message was encrypted with probability any better than ½.

So, if **Definition 3.8** does not hold then ***A*** must be distinguishing the output of ***G*** from a random string.

A Secure Fixed-Length Encryption Scheme

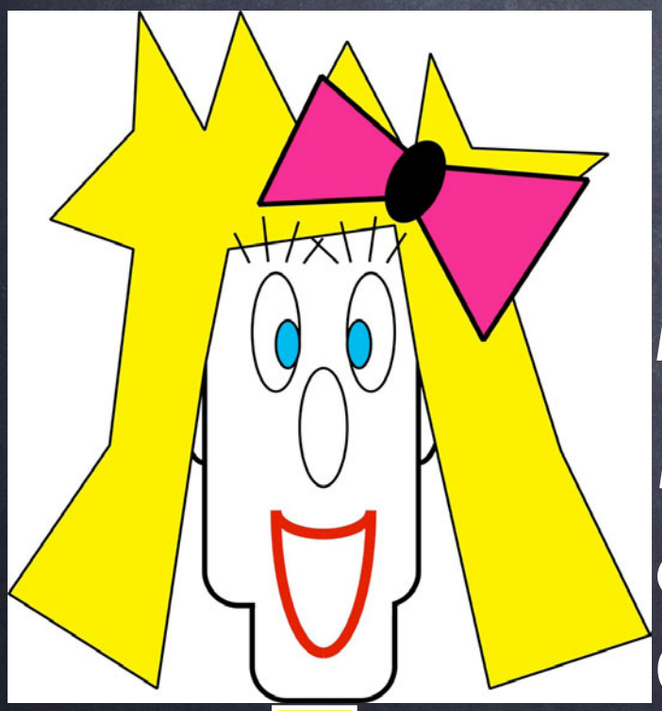
- It is easy to get lost in the details of the proof and wonder whether anything has been gained as compared to the one-time pad; after all, the one-time pad also encrypts an ℓ-bit message by **XOR**ing it with an ℓ-bit string!
- The point of the construction, of course, is that the ℓ-bit string $G(k)$ can be *much longer* than the key k.

3.4 Stronger Security Notions

- Security for Multiple Encryptions
- Security Against (CPA) Chosen-Plaintext Attacks

3.4.1 Security for Multiple Encryptions

$(M_0 := (m_{10},\ldots,m_{t0}),\ M_1 := (m_{11},\ldots,m_{t1})$ *with* $|m_{i0}| = |m_{i1}|$ *for all* $i)$

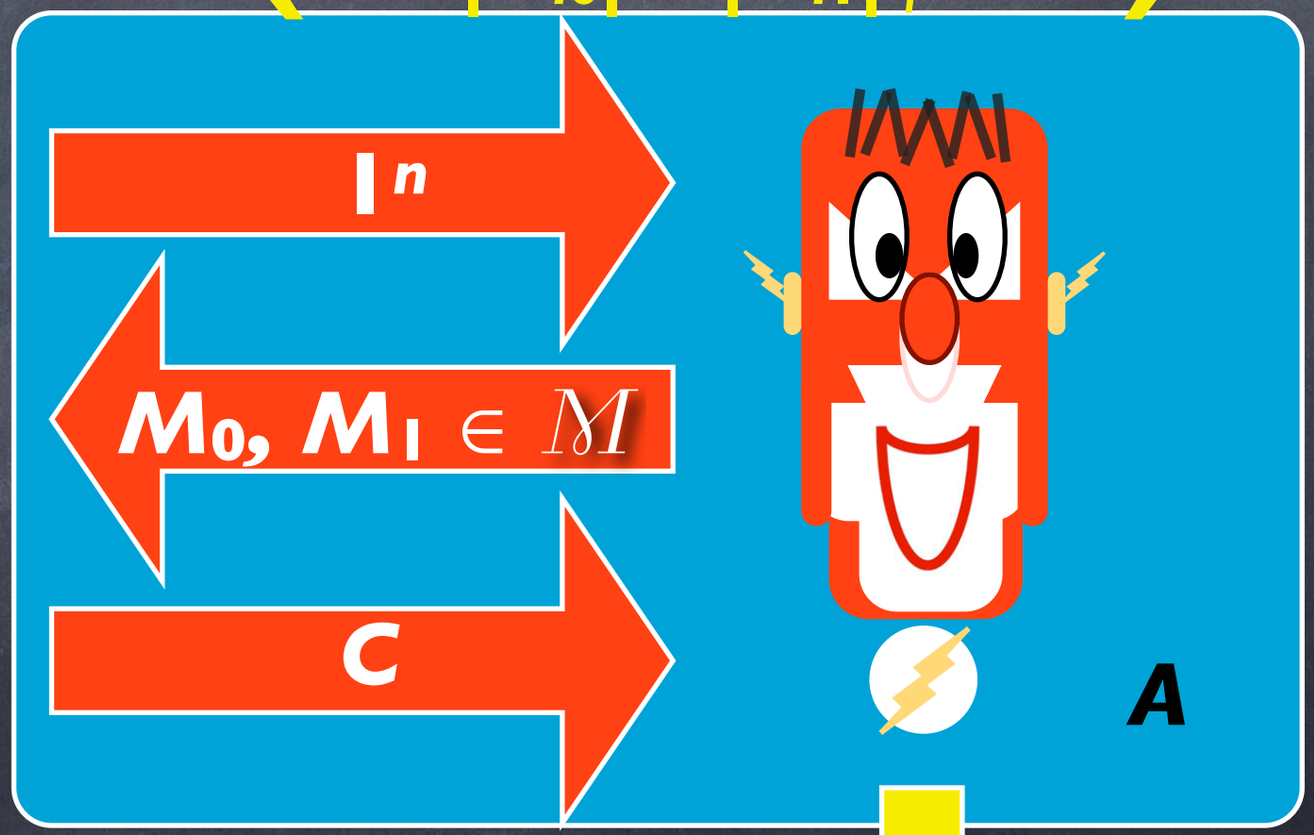


$k \leftarrow \mathrm{Gen}(1^n)$

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

$c_i \leftarrow \mathrm{Enc}_k(m_{ib})$

$C := (c_1, c_2, \ldots, c_t)$



b

b'

computationally secret

$\Pr[\ b = b'\ ] \leq \frac{1}{2} + \mathrm{negl}(n)$

Security for Multiple Encryptions: $\mathbf{PrivK}^{mult}_{A,\Pi}(n)$

1. *The adversary* $\boldsymbol{A}$ *is given input* 1^n*, and outputs a pair of* vectors *of messages* $\boldsymbol{M_0} := (\boldsymbol{m_{10}},\ldots,\boldsymbol{m_{t0}})$ *and* $\boldsymbol{M_1} := (\boldsymbol{m_{11}},\ldots,\boldsymbol{m_{t1}})$ *with* $|\boldsymbol{m_{i0}}| = |\boldsymbol{m_{i1}}|$ *for all* $1 \le \boldsymbol{i} \le \boldsymbol{t}$.

2. *A key* $\boldsymbol{k}$ *is generated by running* $\mathbf{Gen}(1^n)$*, and a random bit* $\boldsymbol{b} \leftarrow \{0, 1\}$ *is chosen. For all* $\boldsymbol{i}$*, the ciphertext* $\boldsymbol{c_i} \leftarrow \mathbf{Enc}_k(\boldsymbol{m_{ib}})$ *is computed and the vector of ciphertexts* $\boldsymbol{C} := (\boldsymbol{c_1},\ldots,\boldsymbol{c_t})$ *is given to* $\boldsymbol{A}$.

3. $\boldsymbol{A}$ *outputs a bit* $\boldsymbol{b'}$.

4. *The output of the experiment is defined to be* $\boldsymbol{1}$ *if* $\boldsymbol{b' = b}$*, and* $\boldsymbol{0}$ *otherwise.*

Security for Multiple Encryptions

DEFINITION 3.19 *A private-key encryption scheme* **Π = (Gen, Enc, Dec)** *has* indistinguishable multiple encryptions in the presence of an eavesdropper *if for all* **PPT** *adversaries* ***A*** *there exists a negligible function* **negl** *s.t.*

$$\mathbf{Pr[PrivK^{mult}_{A,\Pi}(n) = 1] \leq \tfrac{1}{2} + negl(n)},$$

where the probability is taken over the random coins used by ***A****, as well as the random coins used in the experiment (for choosing the key and the random bit* ***b****, as well as for the encryption itself).*

Security for Multiple Encryptions

PROPOSITION 3.20 *There exist private-key encryption schemes that have indistinguishable encryptions in the presence of an eavesdropper but do not have indistinguishable multiple encryptions in the presence of an eavesdropper.*

Necessity of probabilistic encryption

- In the proof of **Proposition 3.20** we show that **Construction 3.17** is not secure for multiple encryptions.

CONSTRUCTION 3.17

Let G be a pseudorandom generator with expansion factor ℓ. Define a private-key encryption scheme for messages of length ℓ as follows:

- Gen: on input 1^n, choose uniform $k \in \{0,1\}^n$ and output it as the key.
- Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^{\ell(n)}$, output the ciphertext
$$c := G(k) \oplus m.$$
- Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c \in \{0,1\}^{\ell(n)}$, output the message
$$m := G(k) \oplus c.$$

- The only feature of that construction used in the proof [is] that encrypting a message always yields the same ciphertext, and so we actually obtain that <u>*any* deterministic scheme must be insecure for multiple encryptions</u>.

Necessity of probabilistic encryption

THEOREM 3.21 *Let* **Π = (Gen, Enc, Dec)** *be an encryption scheme for which* **Enc** *is a deterministic function of the key and the message. Then* **Π** *does not have indistinguishable multiple encryptions in the presence of an eavesdropper.*

3.4.2 Security Against (CPA) Chosen-Plaintext Attacks

- We formally introduce a more powerful type of adversarial attack, called a chosen-plaintext attack (**CPA**).
- The definition of security under **CPA** is the same as in **Definition 3.8**, except that the adversary's attack capabilities are strengthened.

Security Against CPA

The basic idea behind a chosen-plaintext attack is that the adversary ***A*** is allowed to ask for encryptions of multiple messages chosen adaptively.

Security Against CPA

- This is formalized by allowing **A** to interact freely with an encryption oracle, viewed as a "black-box" that encrypts messages of **A**'s choice using the secret key ***k***.
- We denote by $\mathbf{A}^{\mathbf{O}(\cdot)}$ the computation of **A** given access to an oracle **O**.
- We denote the computation of **A** with access to an encryption oracle that uses key ***k*** by $\mathbf{A}^{\mathbf{Enc}_k(\cdot)}$.

Security Against CPA

- When **A** *queries* its oracle by providing it with a plaintext message ***m*** as input, the oracle returns a ciphertext $\mathbf{c} \leftarrow \mathbf{Enc}_k(\boldsymbol{m})$ as the reply.
- When **Enc** is randomized, the oracle uses fresh random coins each time it answers a query.

Security Against CPA

The definition of security requires that **A** should not be able to distinguish the encryption of two arbitrary messages, *even when* **A** is *given access to an encryption oracle.*

CPA Indistinguishability Experiment: $\mathbf{PrivK}^{cpa}_{A,\Pi}(n)$

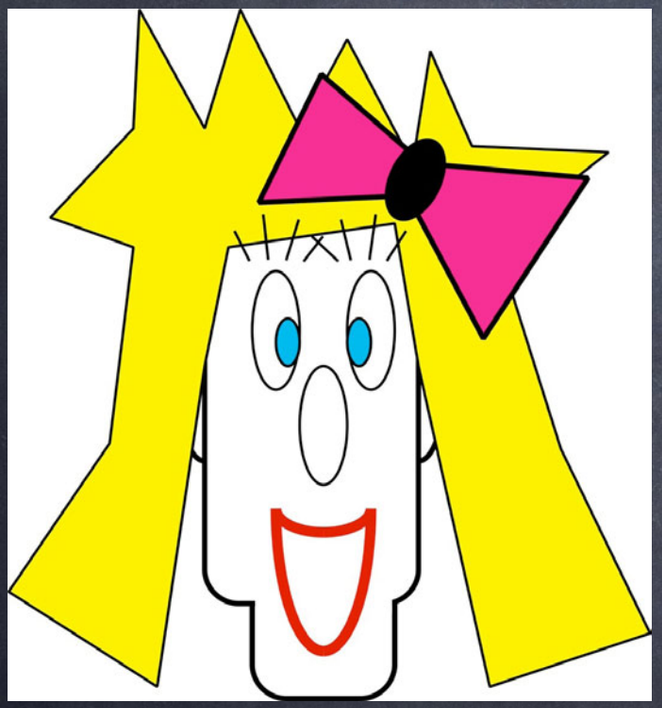
$k \leftarrow \mathbf{Gen}(1^n)$

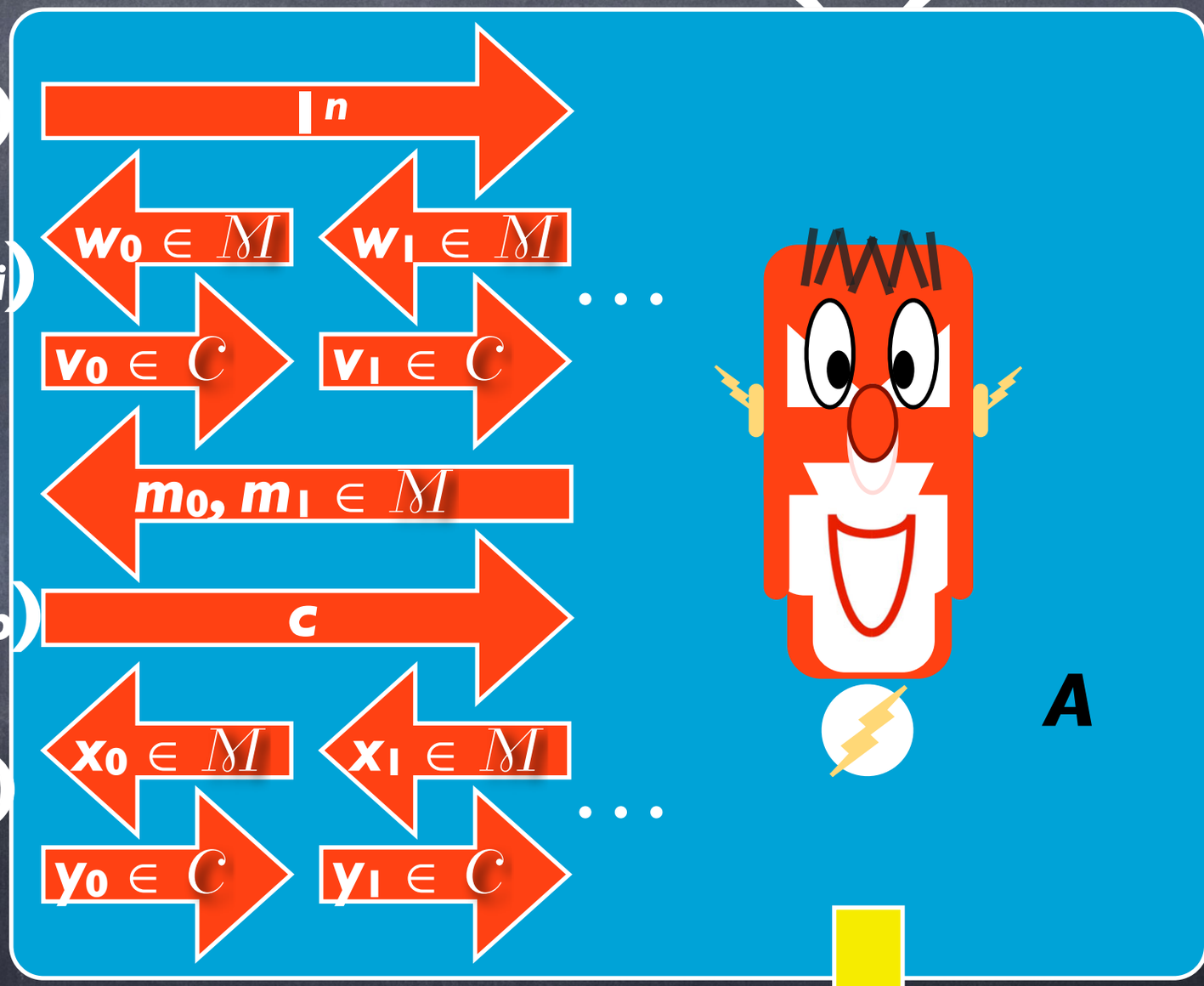
$v_i \leftarrow \mathbf{Enc}_k(w_i)$

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

$c \leftarrow \mathbf{Enc}_k(m_b)$

$y_i \leftarrow \mathbf{Enc}_k(x_i)$





b

b'

computationally secret

$\Pr[b = b'] \leq \frac{1}{2} + \mathrm{negl}(n)$

$\mathbf{PrivK}^{cpa}_{A,\Pi}(n)$

1. *A key* k *is generated by running* $\mathbf{Gen}(1^n)$.

2. *The adversary* A *is given input* 1^n *and oracle access to* $\mathbf{Enc}_k(\cdot)$, *and outputs a pair of messages* m_0, m_1 *of the same length.*

3. *A random bit* $b \leftarrow \{0,1\}$ *is chosen, and then a ciphertext* $c \leftarrow \mathbf{Enc}_k(m_b)$ *is created and given to* A. *We call* c *the* challenge ciphertext.

4. *The adversary* A *continues to have oracle access to* $\mathbf{Enc}_k(\cdot)$, *and outputs a bit* b'.

5. *The output of the experiment is defined to be* 1 *if* $b'=b$, *and* 0 *otherwise.*

(*When* $\mathbf{PrivK}^{cpa}_{A,\Pi}(n) = 1$, *we say that* A succeeded.)

indistinguishable encryptions under CPA

DEFINITION 3.22 *A private-key encryption scheme* **Π = (Gen, Enc, Dec)** *has* indistinguishable encryptions under a chosen-plaintext attack (*or is* **CPA-**secure) *if for all probabilistic polynomial-time adversaries* **A** *there exists a negligible function* **negl** *s.t.*

$$\mathbf{Pr}[\mathbf{PrivK}^{cpa}_{\mathbf{A},\Pi}(n) = 1] \leq \tfrac{1}{2} + \mathbf{negl}(n),$$

where the probability is over the random coins used by **A**, *as well as the random coins used in the experiment.*

indistinguishable encryptions under CPA

- Any scheme that has indistinguishable encryptions under a chosen-plaintext attack clearly also has indistinguishable encryptions in the presence of an eavesdropper.

- This holds because $\mathbf{PrivK}^{\mathbf{eav}}$ is a special case of $\mathbf{PrivK}^{\mathbf{cpa}}$ where the adversary doesn't use its oracle at all.

indistinguishable encryptions under CPA

- It may appear that **Definition 3.22** is impossible to achieve.
- Consider an adversary that outputs (m_0, m_1) and then receives the ciphertext $c \leftarrow \mathbf{Enc}_k(m_b)$.
- Since the adversary $\mathbf{A}$ has oracle access to $\mathbf{Enc}_k$, it can request that this oracle encrypts the messages m_0 and m_1 and thus obtain $c_i \leftarrow \mathbf{Enc}_k(m_i)$.

indistinguishable encryptions under CPA

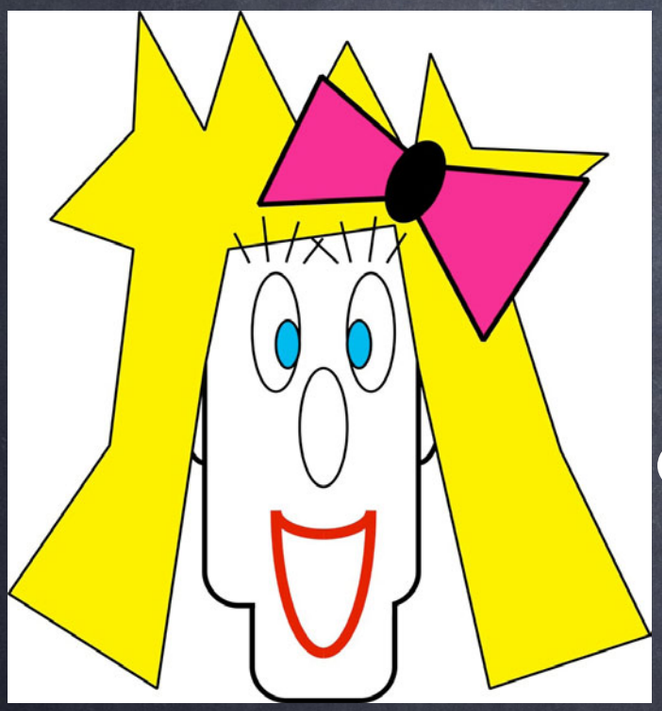
- Adversary **A** can then compare c_0 and c_1 to c: if $c = c_0$ then, seemingly, **A** knows that $b = 0$, and if $c = c_1$ then it knows that $b = 1$.
- Why doesn't this strategy allow **A** to determine b with probability one?

indistinguishable encryptions under CPA

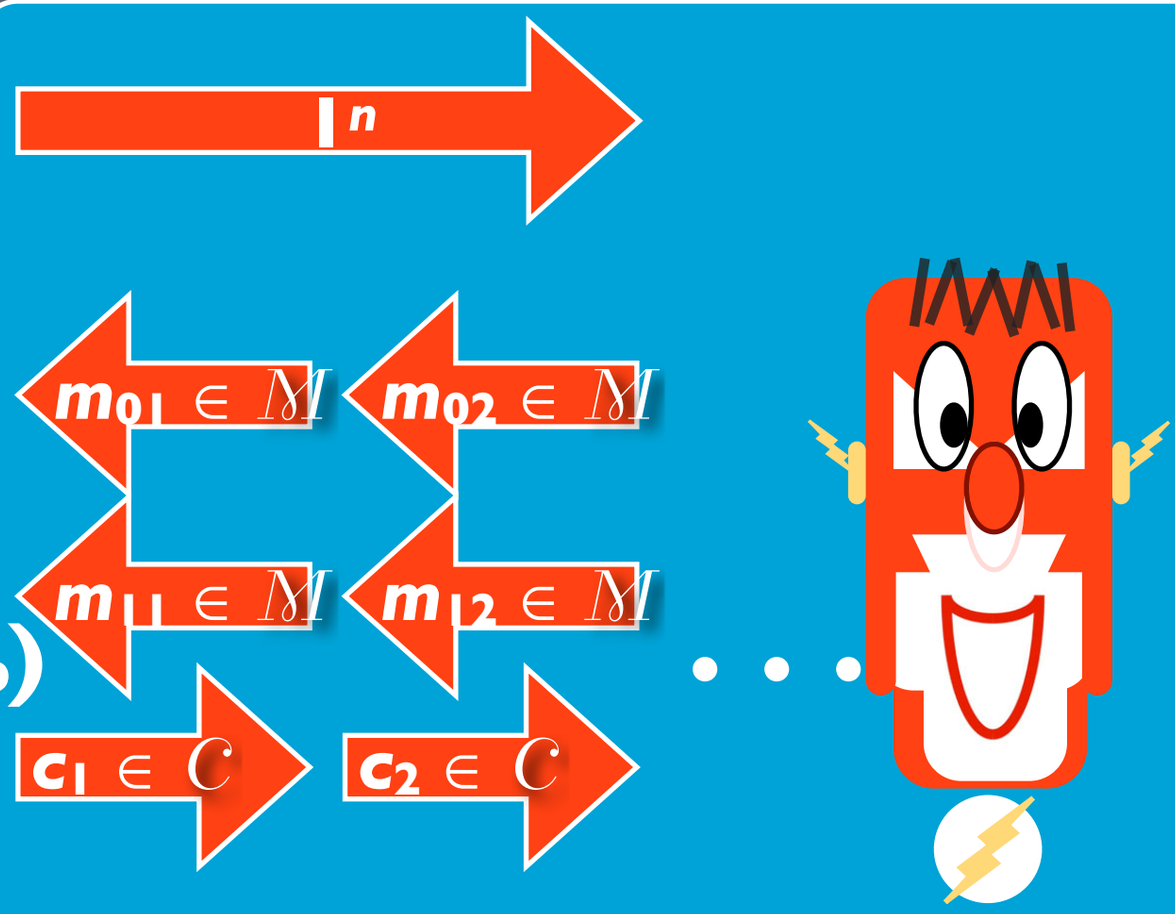
- The answer is that such an attack would indeed work if the encryption scheme was *deterministic*.
- As with security under multiple encryptions, no *deterministic* encryption scheme can be secure against chosen-plaintext attacks.
- Any **CPA**-secure encryption scheme *must* be probabilistic.

CPA Indistinguishability Experiment: $\mathbf{PrivK}^{\mathrm{cpa}}_{A,\Pi}(n)$

$k \leftarrow \mathbf{Gen}(1^n)$
$b \leftarrow \{0, 1\}$



$c_i \leftarrow \mathbf{Enc}_k(m_{ib})$



b

b′

computationally secret

$\Pr[\, b = b' \,] \leq \frac{1}{2} + \mathrm{negl}(n)$

$\mathbf{PrivK}^{\mathbf{LR-cpa}}_{A,\Pi}(n)$

1. *A key* k *is generated by running* $\mathbf{Gen}(1^n)$.

2. *A random bit* $b \leftarrow \{0,1\}$ *is chosen.*

3. *The adversary* **A** *is given input* 1^n *and oracle access to* $\mathrm{LR}_{k,b}$ **such that** $\mathrm{LR}_{k,b}(m_0,m_1) := \mathbf{Enc}_k(m_b)$.

4. *The adversary* **A** *outputs a bit* b'.

5. *The output of the experiment is defined to be* **1** *if* $b'=b$, *and* **0** *otherwise.*

(*When* $\mathbf{PrivK}^{\mathbf{LR-cpa}}_{A,\Pi}(n) = 1$, *we say that* **A** succeeded.)

CPA security for multiple encryptions

DEFINITION 3.23 *A private-key encryption scheme* **Π = (Gen, Enc, Dec)** *has* indistinguishable multiple encryptions under a chosen-plaintext attack (*or is* **CPA-**secure) *if for all probabilistic polynomial-time adversaries* **A** *there exists a negligible function* **negl** *s.t.*

$$\Pr[\mathbf{PrivK}^{\mathrm{R}}_{\mathbf{A},\Pi}(n) = 1] \leq \tfrac{1}{2} + \mathbf{negl}(n),$$

where the probability is over the random coins used by **A**, *as well as the random coins used in the experiment.*

CPA security for multiple encryptions

PROPOSITION 3.24 *Any private-key encryption scheme that has indistinguishable encryptions under a chosen-plaintext attack also has indistinguishable* multiple *encryptions under a chosen-plaintext attack.*

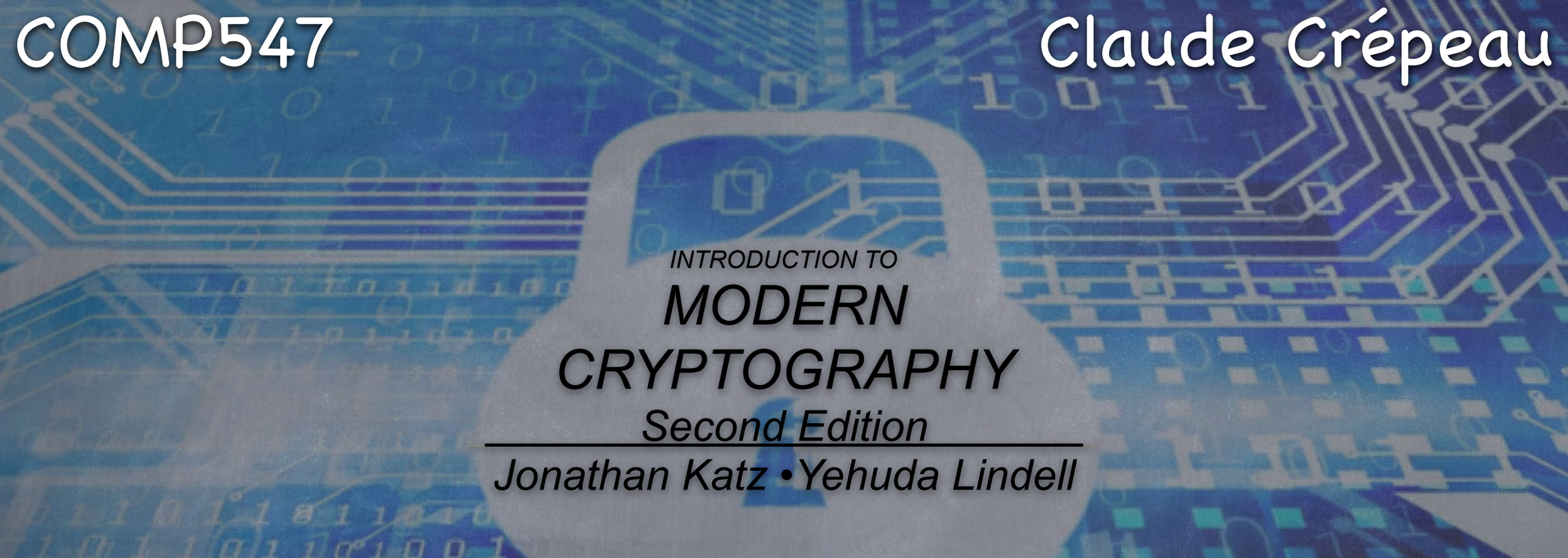
Fixed-length vs. arbitrary-length

Given any **CPA**-secure *fixed-length* encryption scheme $\Pi = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec})$, it is possible to construct a **CPA**-secure encryption scheme

$$\Pi' = (\mathbf{Gen}', \mathbf{Enc}', \mathbf{Dec}')$$

for *arbitrary-length* messages quite easily:

$$\mathbf{Enc}'_k(m) := \mathbf{Enc}_k(m_1), \ldots, \mathbf{Enc}_k(m_\ell)$$



Chapter 3 : Private-Key Encryption

3.5 Constructing CPA-Secure Encryption Schemes

- We will construct encryption schemes that are secure against chosen-plaintext attacks.
- We begin by introducing the important notion of *Pseudorandom Functions*.

3.5.1 Pseudorandom Functions

- Instead of considering Pseudorandom *Strings*, we consider Pseudorandom *Functions*.
- We will specifically be interested in Pseudorandom Functions mapping n-bit strings to n-bit strings.
- It does not make much sense to say that any *fixed* Function $f : \{0,1\}^n \rightarrow \{0,1\}^n$ is Pseudorandom.
- We must technically refer to the Pseudorandomness of a *distribution* over functions.

Pseudorandom Functions

- A keyed function F is a two-input function $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$, where the first input is called *the key* and denoted k, and the second input is just called *the input*.
- In general the key k will be chosen and then *fixed*, and we will then be interested in the single-input function $F_k : \{0,1\}^* \to \{0,1\}^*$ defined by

$$F_k(x) \stackrel{\text{def}}{=} F(k,x).$$

Pseudorandom Functions

- We assume that the function F is only defined when the key k and the input x have the same length, in which case $|F_k(x)| = |x| = |k|$.
- By fixing a key $k \in \{0,1\}^n$ we obtain a function $F_k(\cdot)$ mapping n-bit strings to n-bit strings.

Pseudorandom Functions

- Intuitively, we call ***F*** *pseudorandom* if the function F_k (for a randomly-chosen key ***k***) is indistinguishable from a function chosen uniformly at random from the set of all functions having the same domain and range.
- That is, if no polynomial-time adversary can distinguish whether it is interacting with F_k (for randomly-chosen key ***k***) or ***f*** (where ***f*** is chosen at random from the set of all functions mapping ***n***-bit strings to ***n***-bit strings).

Pseudorandom Functions

- A function ***f*** is fully specified by giving its value on each point in its domain.
- In fact, we can view any function (over a finite domain) as a large look-up table that stores ***f*(x)** in the row of the table labeled by **x**.

Pseudorandom Functions

- For $f : \{0, 1\}^n \to \{0, 1\}^n$ the look-up table for f has 2^n rows and each row contains an n-bit string.
- Any such table can thus be represented using exactly $n \cdot 2^n$ bits.

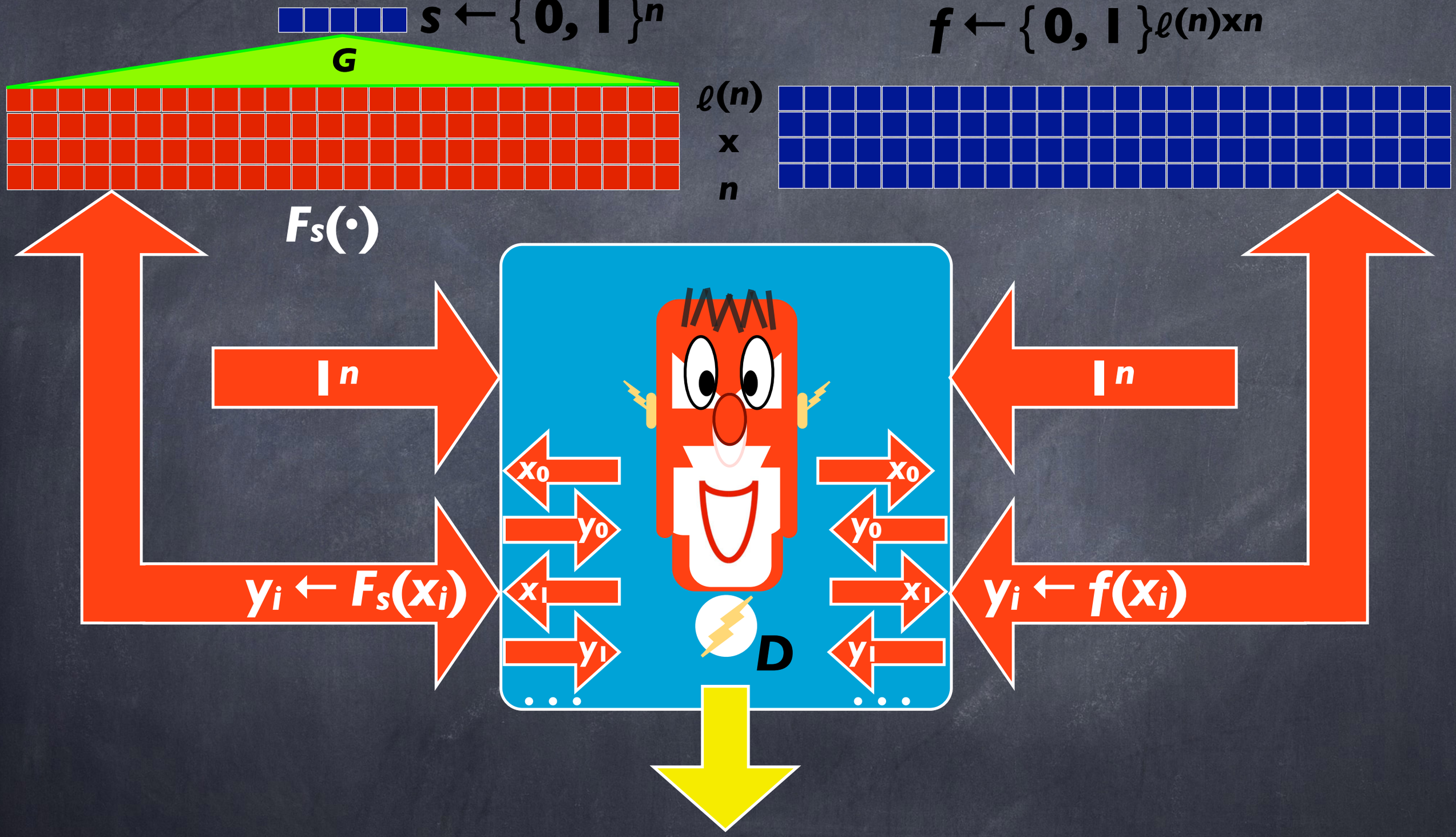
Pseudorandom Functions

DEFINITION 3.25 *Let* $F:\{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *be an efficient, length-preserving, keyed function. We say that* F *is a* Pseudorandom Function *if for all* **PPT** *distinguishers* D*, there exists a negligible function* **negl** *such that:*

$$|\Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1]| \leq \mathbf{negl}(n)$$

where $k \leftarrow \{0,1\}^n$ *is chosen uniformly at random and* f *is chosen uniformly at random from the set of functions mapping* n*-bit strings to* n*-bit string.*

Pseudorandom Function Generators



<u>computational indistinguishability</u>

$$|\Pr[D^{F_s(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1]| \leq \text{negl}(n)$$

On the Existence of Pseudorandom Functions

- As with Pseudorandom Generators, it is important to ask whether Pseudorandom Functions exist and, if so, under what assumptions.
- In practice, very efficient primitives called *block ciphers* are used and are widely believed to act as Pseudorandom Functions.
- This is discussed further in **Chapter 6**.

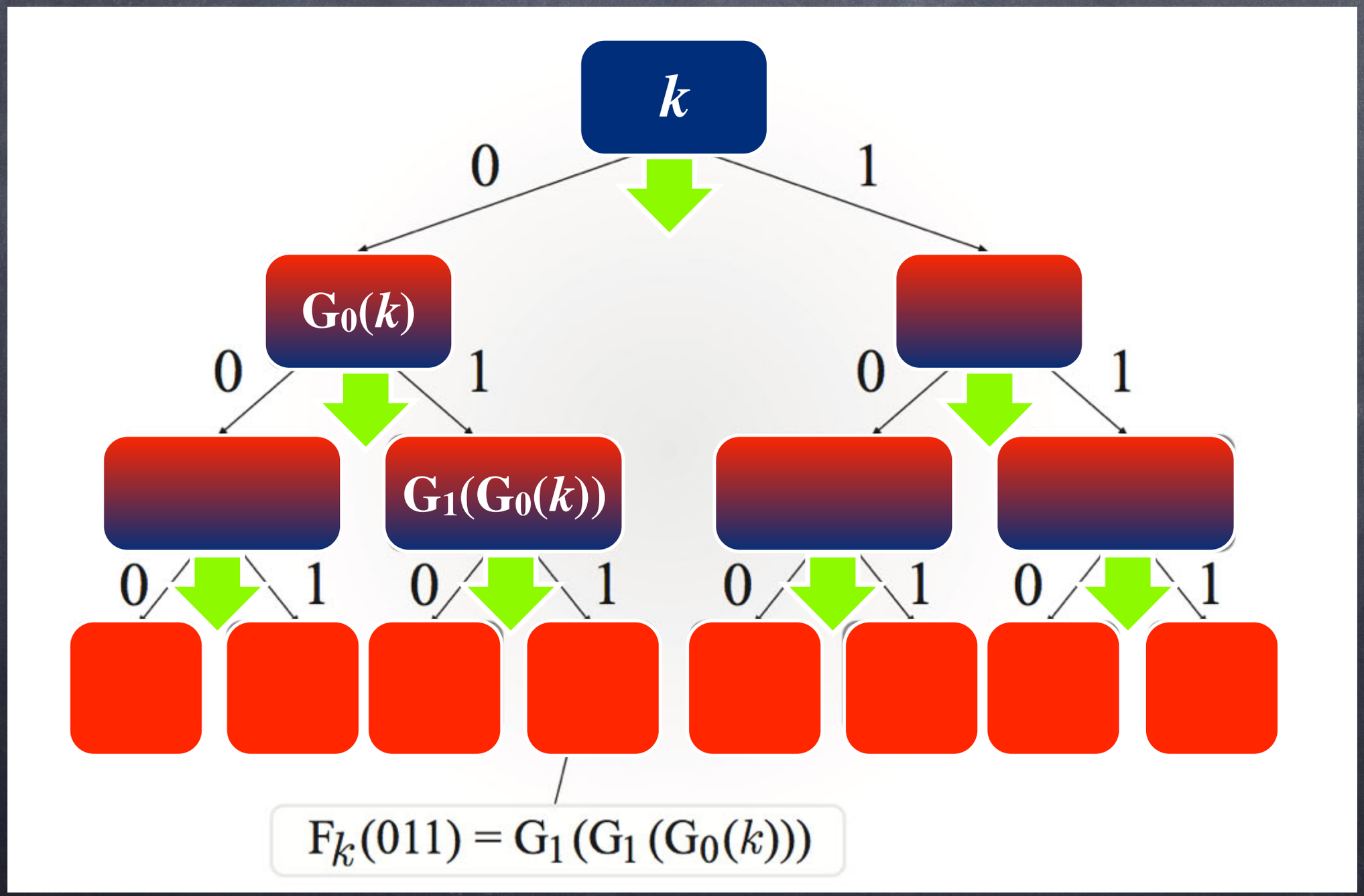
On the Existence of Pseudorandom Functions

- From a theoretical point of view, it is known that pseudorandom functions exist if and only if pseudorandom generators exist.

- Pseudorandom functions can be constructed based on any of the hard problems that allow the construction of pseudorandom generators. (This is discussed at length in **Chapter 7** if you are curious).

- The existence of pseudorandom functions based on these hard problems represents one of the surprising and amazing contributions of modern cryptography.

On the Existence of Pseudorandom Functions



On the Existence of Pseudorandom Functions

CONSTRUCTION 7.21

Let G be a pseudorandom generator with expansion factor $\ell(n) = 2n$, and define G_0, G_1 as in the text. For $k \in \{0,1\}^n$, define the function $F_k : \{0,1\}^n \to \{0,1\}^n$ as:

$$F_k(x_1 x_2 \cdots x_n) = G_{x_n}\left(\cdots\left(G_{x_2}\left(G_{x_1}(k)\right)\right)\cdots\right).$$

A pseudorandom function from a pseudorandom generator.

On the Existence of Pseudorandom Functions

THEOREM 7.22 If **G** is a Pseudorandom Generator with expansion factor $\ell(n) = 2n$, then **Construction 7.21** is a Pseudorandom Function.

Oded Goldreich

Shafi Goldwasser

Silvio Micali

Pseudorandom Permutations and Block Ciphers

- Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient, length-preserving, keyed function.
- We call F a keyed permutation if for every k, the function $F_k(\cdot)$ is *a* bijection.
- We say that a keyed permutation is efficient if there is a **PPT** algorithm computing $F_k(x)$ given k and x, *as well as* a **PPT** algorithm computing $F_k^{-1}(x)$ given k and x.

Pseudorandom Permutations and Block Ciphers

- We define what it means for an efficient keyed permutation ***F*** to be a *pseudorandom* permutation in a manner exactly analogous to **Definition 3.25**.
- The only change is that we now require that F_k (for a randomly-chosen ***k***) be indistinguishable from a randomly-chosen *permutation* rather than a randomly-chosen function.

Pseudorandom Permutations and Block Ciphers

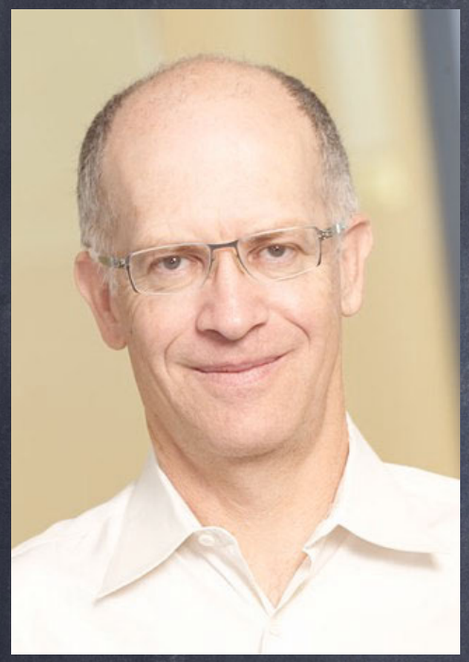
- Actually, this is merely an aesthetic decision since random permutations and (length-preserving) random functions are anyway indistinguishable using polynomially-many queries.

- Intuitively this is due to the fact that a random function ***f*** looks identical to a random permutation unless a distinct pair of values ***x*** and ***y*** are found for which ***f(x)*** = ***f(y)*** (since in such a case the function cannot be a permutation). However, the probability of finding such points ***x, y*** using a polynomial number of queries is negligible.

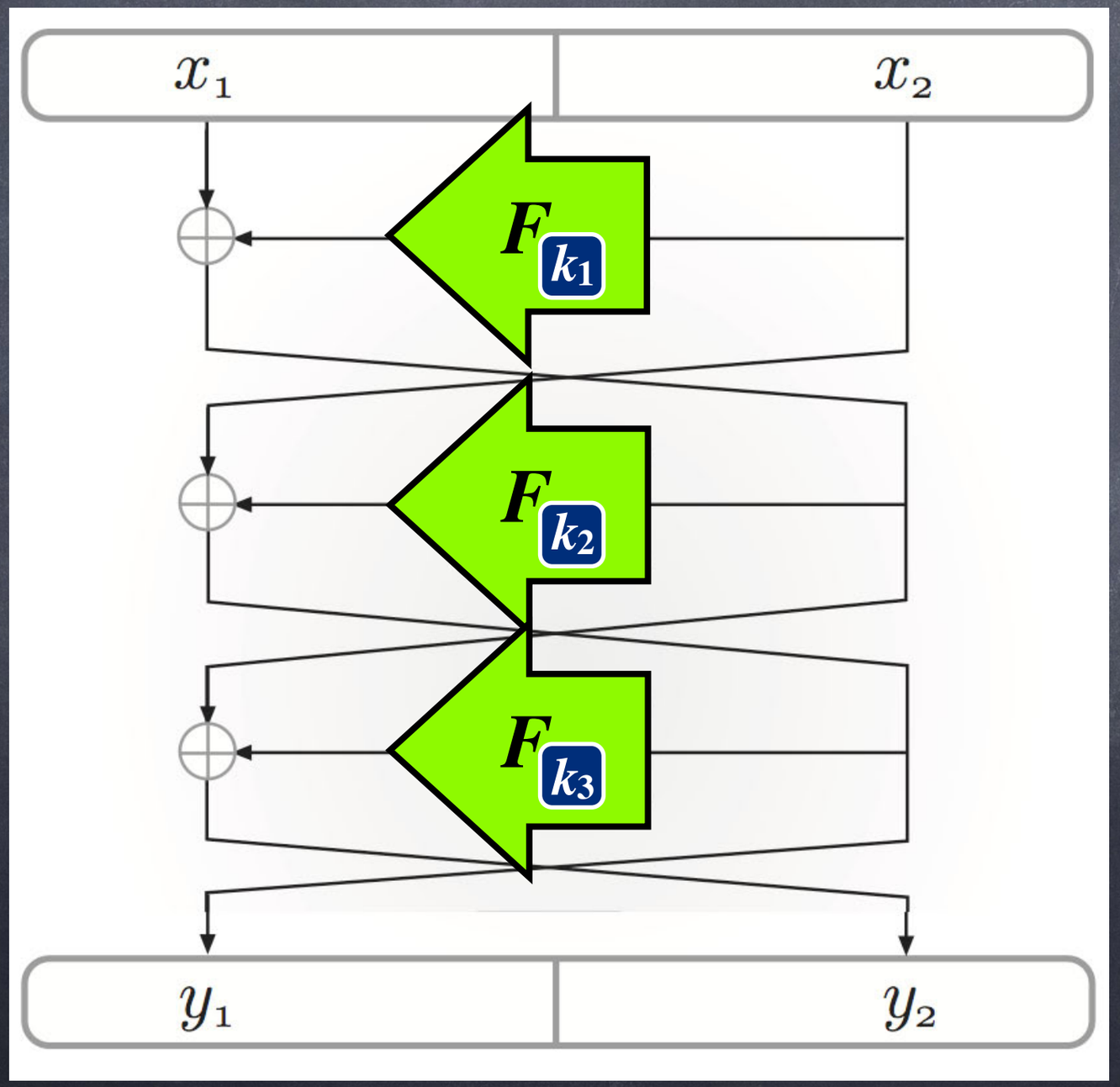
Pseudorandom Permutations and Block Ciphers

- ***PROPOSITION 3.27*** *If* **F** *is a Pseudorandom Permutation then it is also a Pseudorandom Function.*

On the existence of Pseudorandom Permutations



Michael Luby



Charles Rackoff

On the existence of Pseudorandom Permutations

THEOREM 7.23

If F is a length-preserving Pseudorandom Function, then $F^{(3)}$ is a Pseudorandom Permutation that maps $2n$-bit strings to $2n$-bit strings (and uses a key of length $3n$).

Pseudorandom Permutations and Block Ciphers

- If F is an efficient pseudorandom permutation then cryptographic schemes based on F might require honest parties to compute the inverse F_k^{-1} in addition to the permutation F_k itself.
- This potentially introduces new security concerns that are *not* covered by the fact that F is pseudorandom.

Pseudorandom Permutations and Block Ciphers

- We may need to impose the stronger requirement that F_k be indistinguishable from a random permutation *even if the distinguisher is* additionally *given oracle access to the inverse of the permutation.*
- If ***F*** has this property, we call it a

  Strong Pseudorandom Permutation.

Pseudorandom Permutations and Block Ciphers

DEFINITION 3.28 *Let* $F: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *be an efficient, keyed permutation. We say that* F *is a strong pseudorandom permutation if for all probabilistic polynomial-time distinguishers* D*, there exists a negligible function* **negl** *such that:*

$$\left| \Pr[D^{F_k(\cdot), F_k^{-1}(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot), f^{-1}(\cdot)}(1^n) = 1] \right| \leq \mathbf{negl}(n),$$

where $k \leftarrow \{0,1\}^n$ *is chosen uniformly at random and* f *is chosen uniformly at random from the set of permutations on* n*-bit strings.*

Pseudorandom Permutations and Block Ciphers

- Unfortunately, it is often not stated in the literature that a block cipher is actually assumed to be a strong pseudorandom permutation.

- Explicitly modeling block ciphers in this way enables a formal analysis of many practical constructions that rely on block ciphers.

- These constructions include encryption schemes (as studied here), message authentication codes (to be studied in **Chapter 4**), authentication protocols, and more.

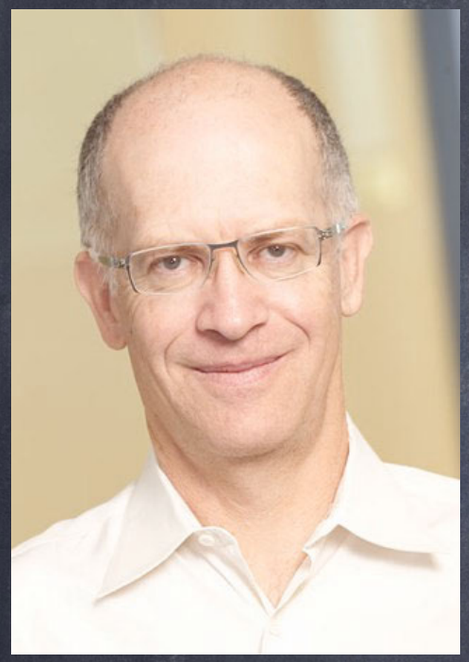
Pseudorandom Permutations and Block Ciphers

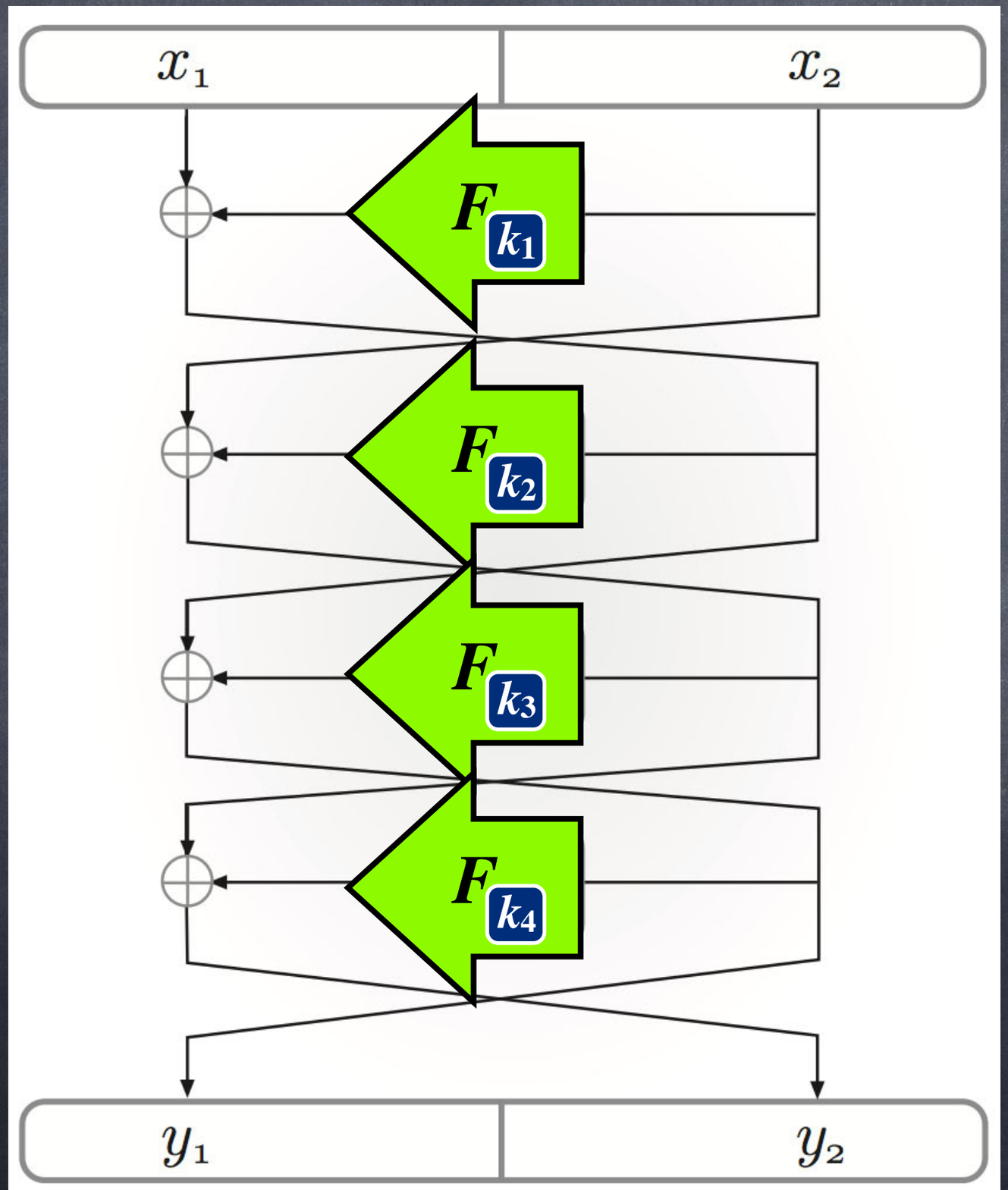
- As with stream ciphers, block ciphers themselves are *not* secure encryption schemes.
- Rather, they are building blocks that can be used to construct secure encryption schemes.
- For example, using a block cipher in **Construction 3.30** yields a **CPA**-secure private-key encryption scheme.

Pseudorandom Permutations and Block Ciphers

- In contrast, an encryption scheme that works by just computing $c := F_k(m)$, where F_k is a strong pseudorandom permutation, is *not* **CPA** secure.
- This distinction between block ciphers as building blocks and encryption schemes that use block ciphers is of great importance and one that is too often missed.

On the existence of Strong Pseudorandom Permutations



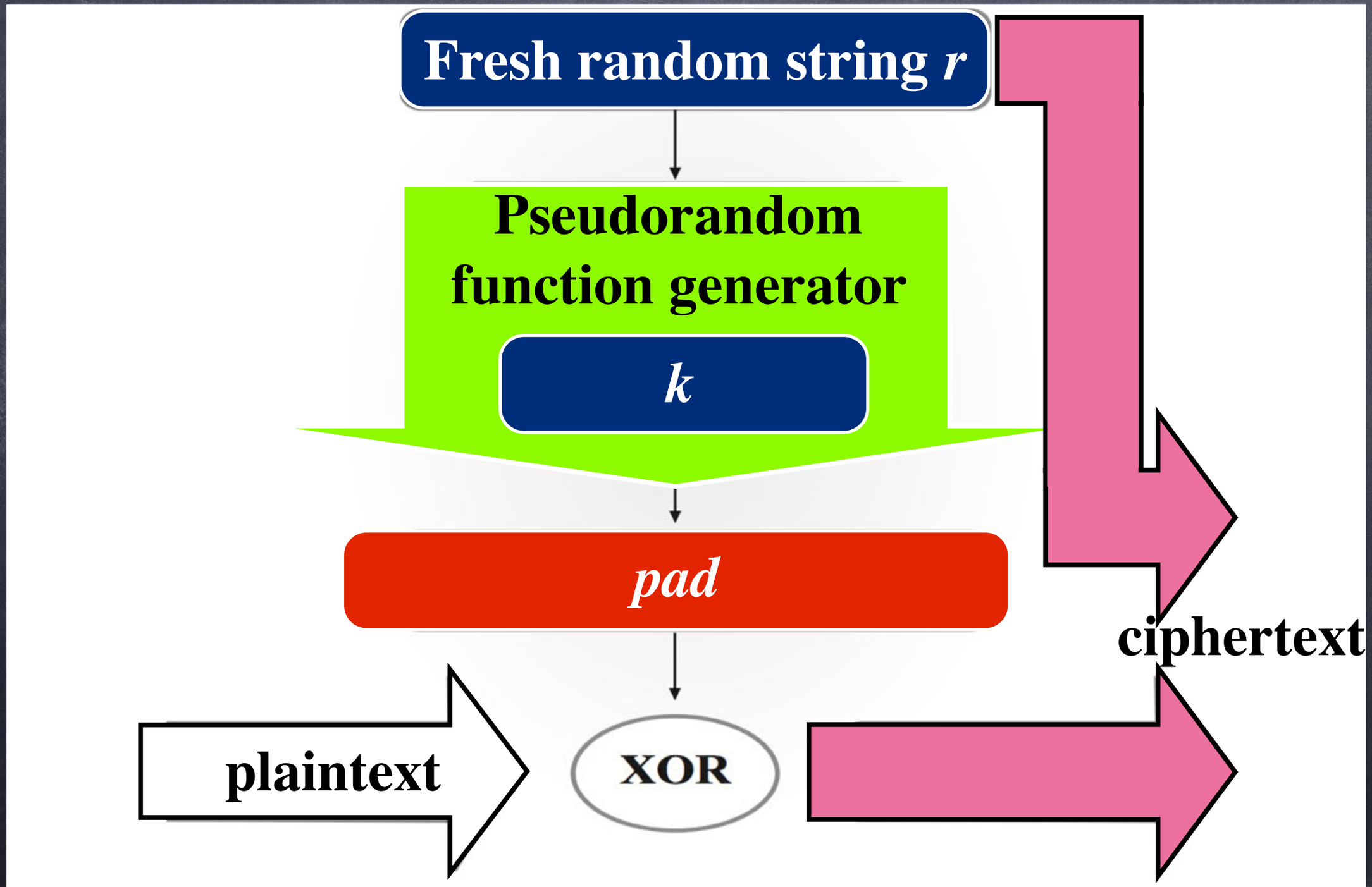


On the existence of Strong Pseudorandom Permutations

THEOREM 7.25

If F is a length-preserving Pseudorandom Function, then $F^{(4)}$ is a *Strong Pseudorandom Permutation* that maps $2n$-bit strings to $2n$-bit strings (and uses a key of length $4n$).

3.5.2 CPA-Secure Encryption from Pseudorandom Functions



Using a Pseudorandom Function in Cryptography

- Pseudorandom functions turn out to be a very useful building block for a number of different cryptographic constructions.
- We use them below to obtain **CPA**-secure encryption and in **Chapter 4** to construct message authentication codes.

Using a Pseudorandom Function in Cryptography

- Given a scheme that is based on a pseudorandom function, a general way of analyzing the scheme is to first prove its security under the assumption that a truly random function is used instead.
- Next, the security of the original scheme is derived by proving that if an adversary can break the scheme when a pseudorandom function is used, then it must implicitly be distinguishing the function from random.

CPA-Secure Encryption from Pseudorandom Functions

CONSTRUCTION 3.30

Let F be a pseudorandom function. Define a private-key encryption scheme for messages of length n as follows:

- Gen: on input 1^n, choose uniform $k \in \{0,1\}^n$ and output it.
- Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, choose uniform $r \in \{0,1\}^n$ and output the ciphertext

$$c := \langle r,\ F_k(r) \oplus m \rangle.$$

- Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c = \langle r, s \rangle$, output the plaintext message

$$m := F_k(r) \oplus s.$$

CPA-Secure Encryption from Pseudorandom Functions

- Intuitively, security holds because $F_k(r)$ looks completely random to an adversary who observes a ciphertext $\langle r,s \rangle$ *as long as the value* r *was not used in some previous encryption*.
- Moreover, this "bad event" (namely, a repeating value of r) occurs with only negligible probability.

CPA-Secure Encryption from Pseudorandom Functions

THEOREM 3.31 *If* ***F*** *is a pseudorandom function, then* ***Construction 3.30*** *is a fixed-length private-key encryption scheme for messages of length* ***n*** *that has indistinguishable encryptions under* **CPA**.

Efficiency of Construction 3.30

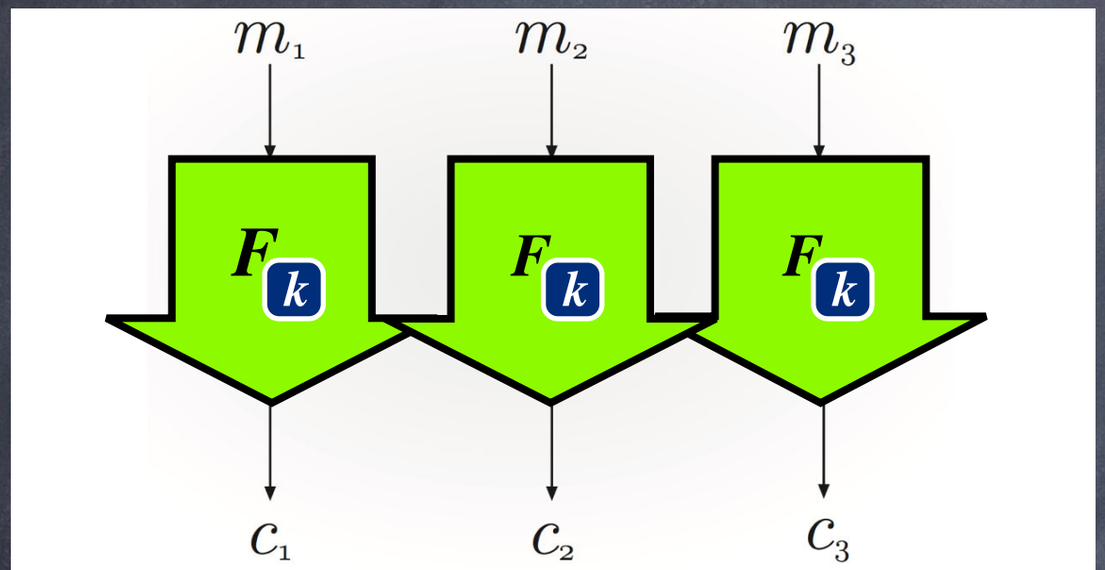
- **Construction 3.30** has the drawback that the length of the ciphertext is (at least) *double* the length of the plaintext.
- This is because each block of size ***n*** is encrypted using an ***n***-bit random string which must be included as part of the ciphertext.
- In **Section 3.6.2** we will show how the ciphertext length can be significantly reduced.

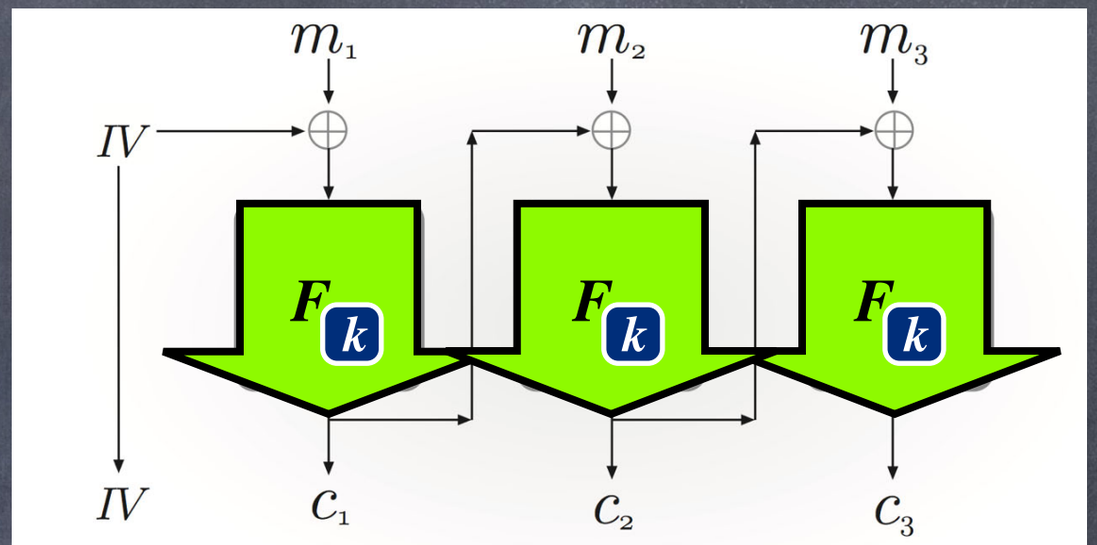
COMP547 Claude Crépeau

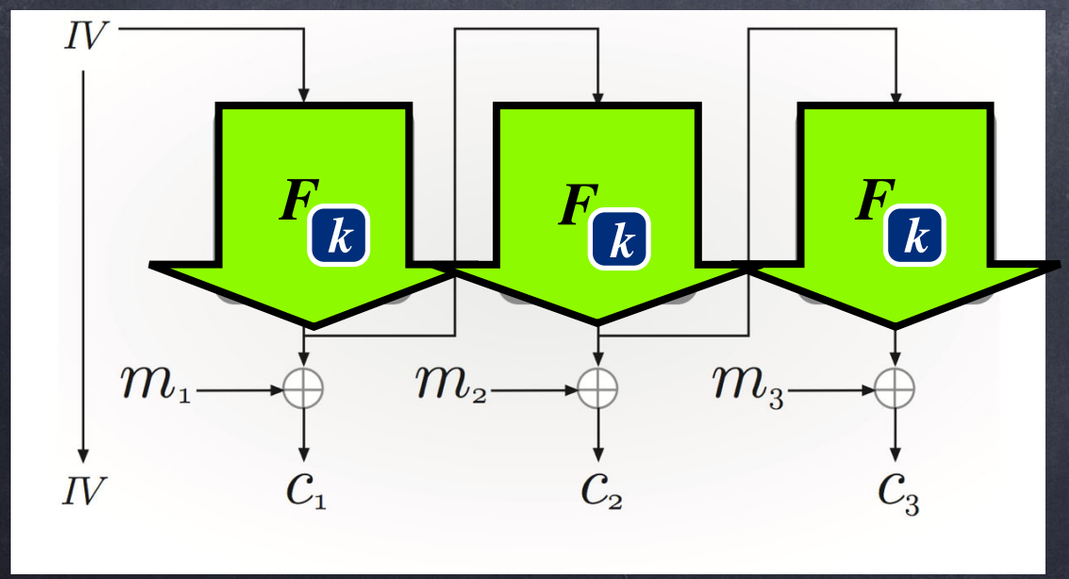
INTRODUCTION TO
MODERN
CRYPTOGRAPHY
Second Edition
Jonathan Katz • Yehuda Lindell

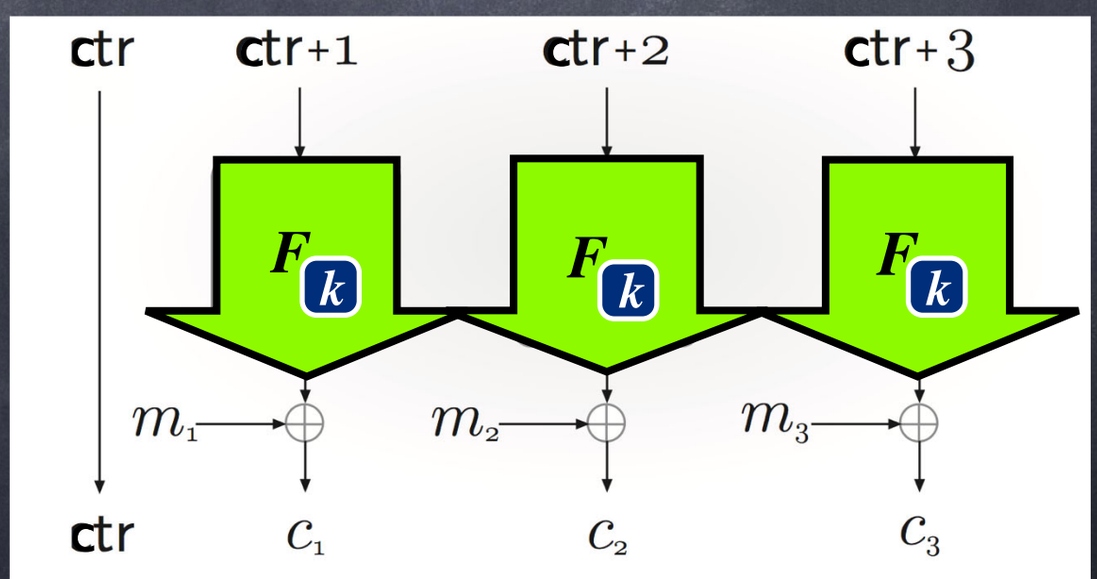
Chapter 3 : Private-Key Encryption

3.6 Modes of Operation





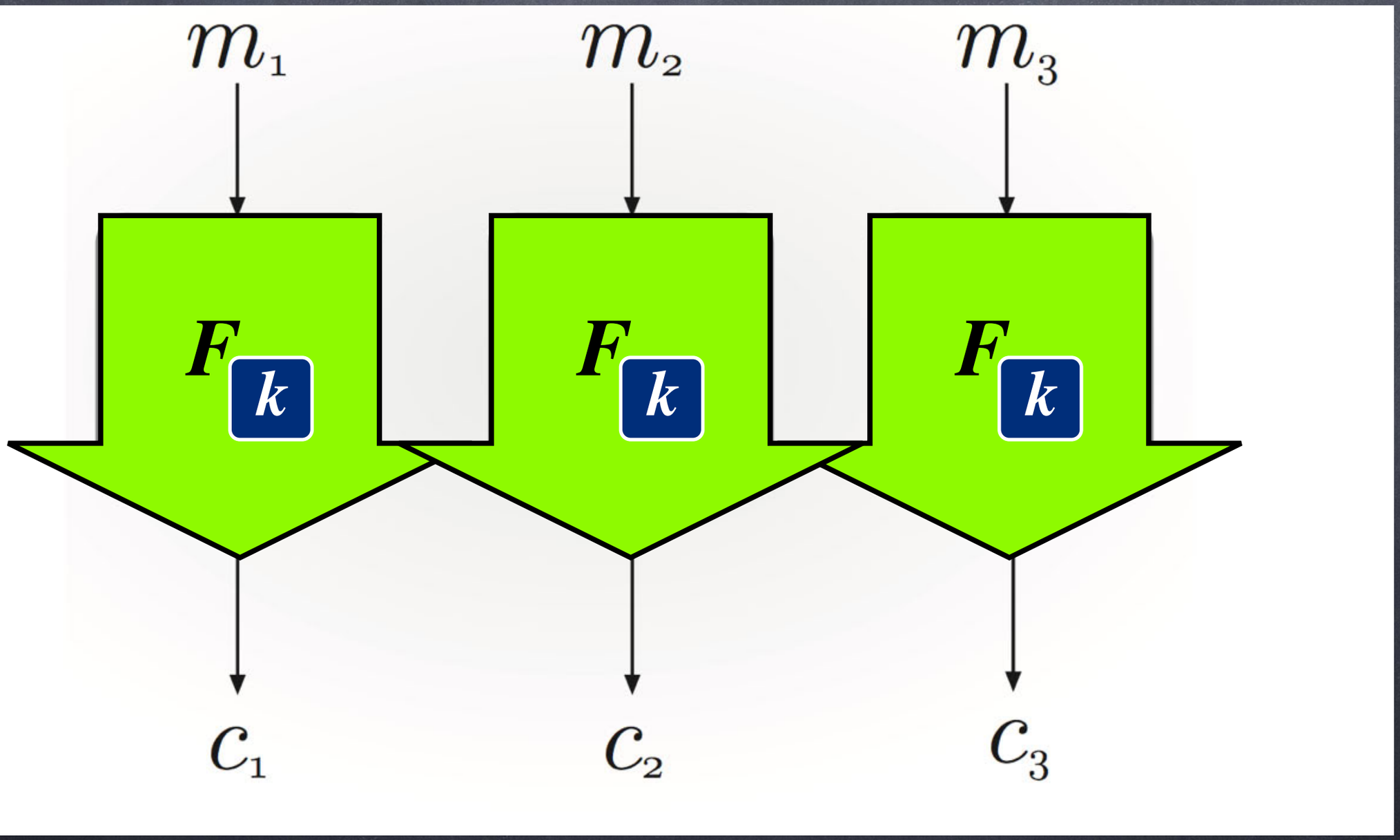




3.6.2 Block Cipher Modes of Operation

- Note that arbitrary-length messages can be unambiguously padded to a total length that is a multiple of any desired block size by appending a **1** followed by sufficiently-many **0**s.
- We will therefore just assume that the length of the plaintext message is an exact multiple of the block size.
- Throughout this section, we will refer to a pseudorandom permutation/block cipher F with block length n, and will consider the encryption of messages consisting of ℓ blocks each of length n.

Mode 1 — (ECB) Electronic Code Book mode



Electronic Code Book (ECB) mode

- This is the most naive mode of operation possible.
- Given a plaintext message $m := m_1, m_2, \ldots, m_\ell$, the ciphertext is obtained by "encrypting" each block separately.

Electronic Code Book (ECB) mode

- "Encryption" here means a direct application of the pseudorandom permutation to the plaintext block:

$$c := F_k(m_1), F_k(m_2), \ldots, F_k(m_\ell)$$

- Decryption is carried in the obvious way, using the fact that F_k^{-1} is efficiently computable.
- The encryption process here is *deterministic* and therefore this mode of operation cannot be **CPA**-secure.

Electronic Code Book (ECB) mode

- Even worse, **ECB**-*mode encryption does not have indistinguishable encryptions in the presence of an eavesdropper.*
  (This is due to the fact that if the same block is repeated twice in the plaintext, this can be detected as a repeating block in the ciphertext.)
- It is easy to distinguish an encryption of a plaintext that consists of two identical blocks from an encryption of a plaintext that consists of two different blocks.

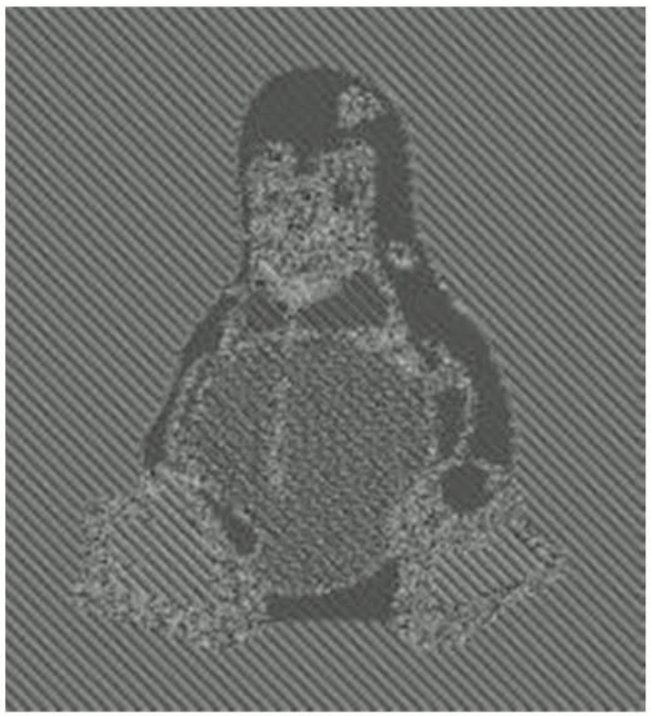
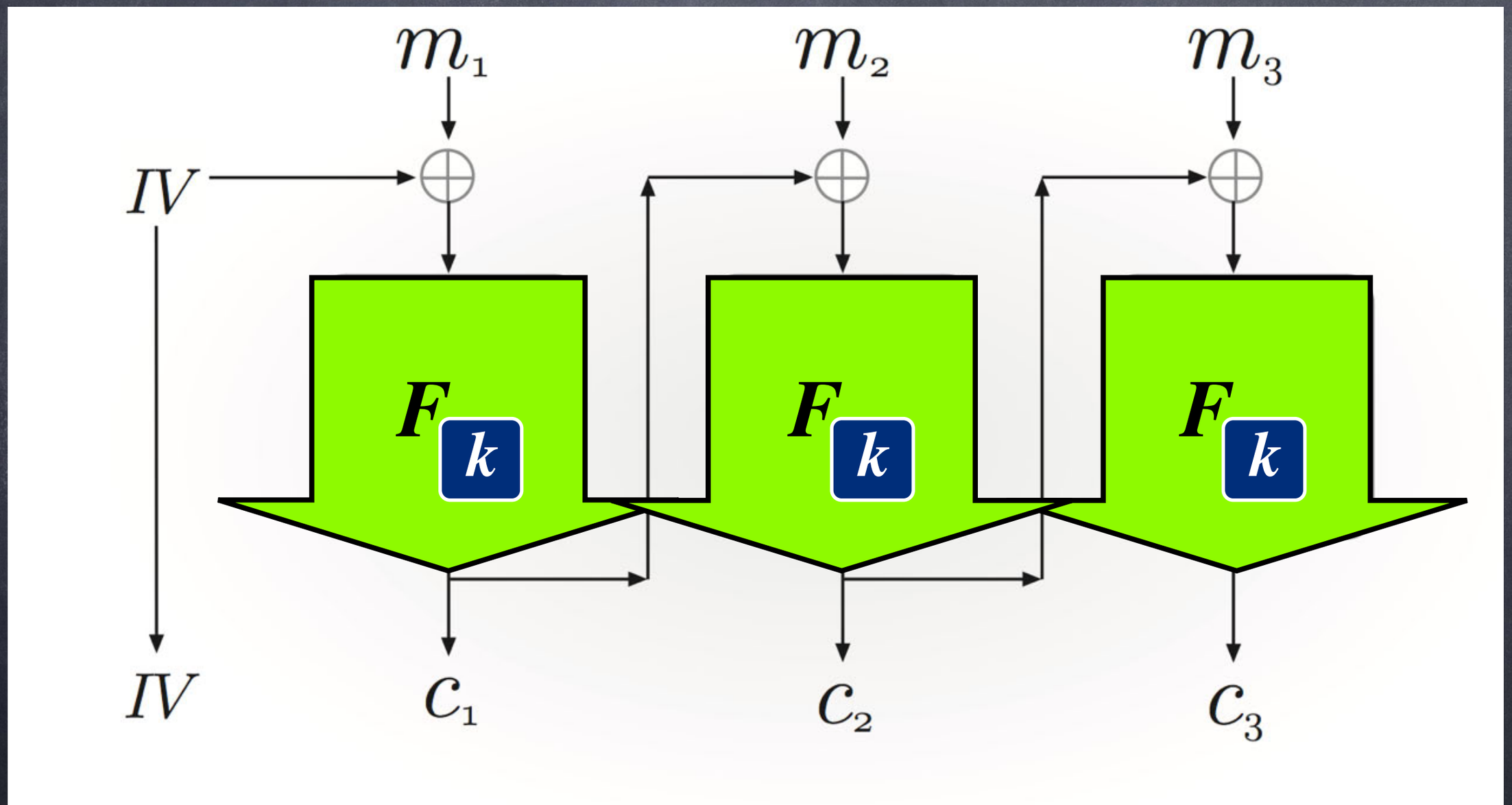


FIGURE 3.6: An illustration of the dangers of using ECB mode. The middle figure is an encryption of the image on the left using ECB mode; the figure on the right is an encryption of the same image using a secure mode. (Taken from `http://en.wikipedia.org` and derived from images created by Larry Ewing (`lewing@isc.tamu.edu`) using The GIMP.)

Mode 2 — (CBC) Cipher Block Chaining mode



Cipher Block Chaining (CBC) mode

- In this mode, a random initial vector (***IV***) of length ***n*** is first chosen.
- Each of the remaining ciphertext blocks is generated by applying the pseudorandom permutation to the **XOR** of the current plaintext block and the previous ciphertext block.

Cipher Block Chaining (CBC) mode

- Set $c_0 := IV$ and then,
  for $i := 1$ **to** ℓ,
  set $c_i := F_k(c_{i-1} \oplus m_i)$.

- The final ciphertext is $c_0, c_1, \ldots, c_\ell$.

- We stress that the ***IV*** is sent in the clear as part of the ciphertext; this is crucial so that decryption can be carried out.

Cipher Block Chaining (CBC) mode

- Importantly, encryption in **CBC** mode is probabilistic and it has been proven that if ***F*** is a pseudorandom permutation then **CBC**-mode encryption is **CPA**-secure.

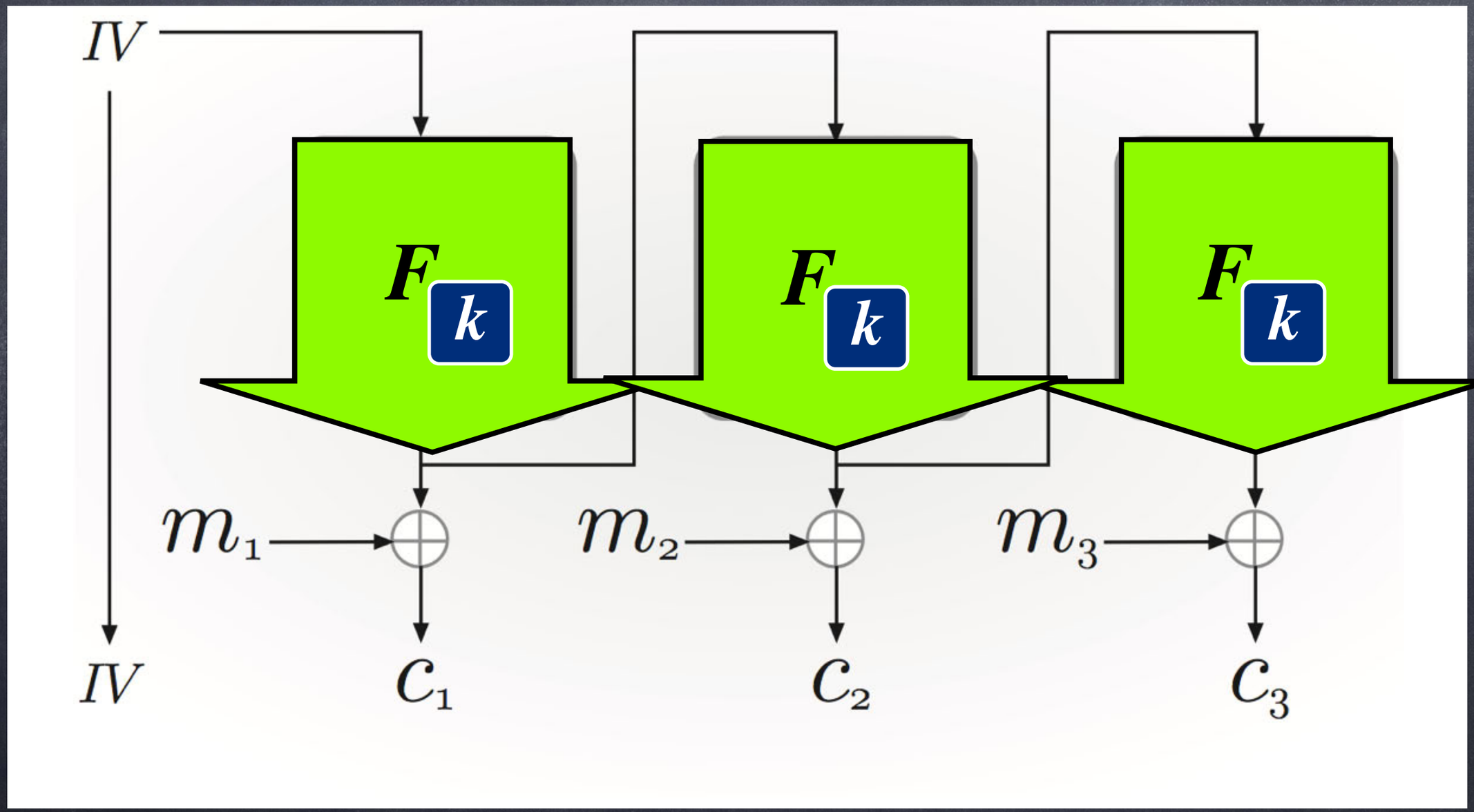
Mihir Bellare

Phil Rogaway

Cipher Block Chaining (CBC) mode

- The main drawback of this mode is that encryption must be carried out sequentially because the ciphertext block c_{i-1} is needed in order to encrypt the plaintext block m_i.
- If parallel processing is available, **CBC**-mode encryption may not be the most efficient choice.

Mode 3 — (OFB) Output Feedback mode



Output Feedback (OFB) mode

- The third mode we present here is called **OFB**.
- Essentially, this mode is a way of using a block cipher to generate a pseudorandom stream that is then **XOR**ed with the message.

Output Feedback (OFB) mode

- First, a random $IV \leftarrow \{0,1\}^n$ is chosen and a stream is generated from IV (independently of the plaintext message) in the following way:
- Define $r_0 := IV$, and set the i^{th} block r_i of the stream to $r_i := F_k(r_{i-1})$.
- Then, each block of the plaintext is encrypted by **XOR**ing it with the appropriate block of the stream; that is, $c := m \oplus r$.

Output Feedback (OFB) mode

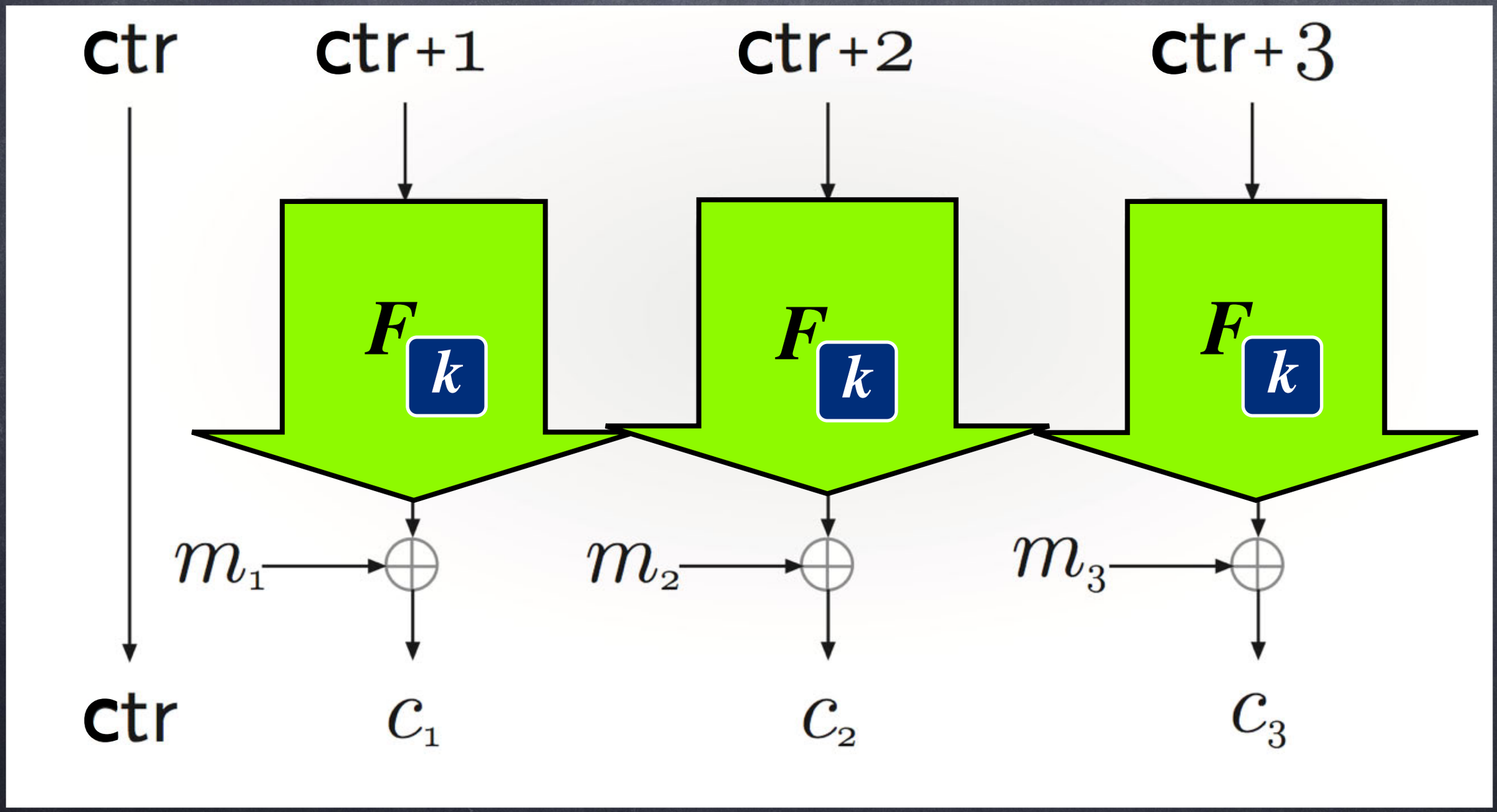
- This mode is also probabilistic, and it can be shown that it too is a **CPA**-secure encryption scheme if ***F*** is a pseudorandom function.

- Encryption and decryption are Sequential.

Output Feedback (OFB) mode

- This mode has the advantage that the bulk of the computation can be done independently of the actual message to be encrypted. Using pre-processing, encryption of the plaintext (once it is known) is incredibly fast.
- In contrast to **CBC** mode, here it is not required that ***F*** be invertible (in fact, it need not even be a permutation)

Mode 4 — (CTR) Counter mode



Counter (CTR) mode

- There are different variants of **CTR**-mode encryption; we describe the *randomized counter mode* here.
- As with **OFB**, counter mode can be viewed as a way of generating a pseudorandom stream from a block cipher.

Counter (CTR) mode

- First, a random $\mathbf{ctr} \leftarrow \{0,1\}^n$ is chosen.
- A stream is generated as $r_i := F_k(\mathbf{ctr} + i)$ (where **ctr** and i are viewed as integers and addition is performed modulo 2^n).
- Finally, the i^{th} block is computed as $c_i := r_i \oplus m_i$, and the **ctr** is again sent as part of the ciphertext.
- Note once again that decryption does not require F to be invertible, or even a permutation.

Counter (CTR) mode

- First and foremost, randomized counter mode is **CPA**-secure.

- Second, both encryption and decryption can be fully parallelized and, as with **OFB** mode, it is possible to generate the pseudorandom stream ahead of time, independently of the message.
- Finally, it is possible to (en- &) de-crypt the i^{th} block of the ciphertext without (en- &) decrypting anything else; this property is called *random access*.

Block length and security

- Most of the above modes use a random ***IV***.
- The ***IV*** has the effect of randomizing the encryption process, and ensures that (with high probability) the block cipher is always evaluated on a *new* input that was never used before.

Block length and security

- This is important because, if an input to the block cipher is used more than once then security can be violated. (E.g., in the case of counter mode, the same pseudorandom string will be **XOR**ed with two different plaintext blocks.)

Block length and security

- Interestingly, this shows that it is not only the *key length* of a block cipher that is important in evaluating its security, but also its *block length*. For example, say we use a block cipher with a **64**-bit block length.
- In randomized counter mode, even if a completely random function with this block length is used (i.e., even if the block cipher is "perfect"), an adversary can achieve success probability roughly $\frac{1}{2} + q^2/2^{63}$ in a chosen-plaintext attack when it makes q queries to its encryption oracle, each q blocks long.

Block length and security

- Although this is asymptotically negligible (when the block length grows as a function of the security parameter n), security no longer holds in any practical sense (for this particular block length) when $q \approx 2^{30}$.
- Depending on the application, one may want to switch to a block cipher having a larger block length (2^{30} is only one gigabyte, which is not much considering today's storage needs).

3.7 Security Against Chosen-Ciphertext Attacks (CCA)

- We need the tools of **Chapter 4** (Message Authentication Codes) to address this issue.
- We will return to it in due time...

TASKS / SECURITY	Encryption	Authentication	Identification	Quantum
Symmetric Informational	Miller-Vernam One-Time PAD	Wegman-Carter Universal Hash	Simple Solutions	Quantum Key Distribution
Symmetric Computational	from PRBG from PRFG DES, AES, etc	from PRBG from PRFG DES, AES, etc	from PRBG from PRFG DES, AES, etc	Quantum Attacks, Q-Safety
Asymmetric Computational	RSA, ElGammal, Blum-Goldwasser	RSA, DSA, etc	Guilloux-Quisquater, Schnor, etc	Quantum Attacks, Q-Safety

DONE	IN PROGRESS	TO DO	GIVE UP

COMP547 Claude Crépeau

INTRODUCTION TO
MODERN
CRYPTOGRAPHY
Second Edition
Jonathan Katz • Yehuda Lindell

Chapter 3 : Private-Key Encryption