

<https://brown-csci1660.github.io>

CS1660: Intro to Computer Systems Security Spring 2025

Lecture 4: Cryptography III

Co-Instructor: **Nikos Triandopoulos**

February 4, 2025



BROWN

CS1660: Announcements

- ◆ Override requests
 - ◆ Status update
- ◆ Course updates
 - ◆ Homework 1, Project 1 to get new submission dates
 - ◆ To provide more time & better preparation
 - ◆ To avoid possible confusion
 - ◆ Due to the specific order/pace with which topics are covered in class
 - ◆ Ed Discussion, Top Hat (code: 084705), Gradescope (to become available soon)

Today

- ◆ Cryptography
 - ◆ Ciphers in practice
 - ◆ Stream & Block ciphers
 - ◆ Modes of operations for encryption
 - ◆ DES, AES

4.0 Symmetric encryption in practice

Big picture

Secret communication

- ◆ We learned what it means for a cipher to be perfectly secure
- ◆ We learned that the simple OTP cipher achieves this property
 - ◆ XOR (**mask**) message (**once**) with the secret key (**random pad**)
 - ◆ ...but it cannot be used in practice!
- ◆ We learned how we can fix this problem
 - ◆ just use OTP with a freshly-generated “**random looking**” pads
 - ◆ **mask** each message **once** with a **pseudorandom pad**

Big picture (cont.)

Secret communication

- ◆ But there is no free lunch...
 - ◆ if we **mask** each message **once** with a **pseudorandom pad**, we must lose **perfect** secrecy!
 - ◆ because “**random looking**” pads are not **random**...
- ◆ But not perfect won't be imperfect – it will be close to perfect
 - ◆ for all practical purposes
 - ◆ “**random looking**” pads will be as random as **truly random** ones
 - ◆ **OTP + pseudo-randomness** will be as secure as (standard) **OTP**

4.1 Computational security

The big picture: OTP is perfect but impractical!

We formally defined and constructed the perfectly secure OTP cipher

- ◆ This scheme has some major drawbacks
 - ◆ it employs a very large key which can be used only once!
- ◆ Such limitations are unavoidable and make OTP not practical
 - ◆ why?



Now, what?

Our approach: Relax perfectness for cipher security

Initial model

- ◆ **Perfect secrecy** (or security) guarantees that
 - ◆ the ciphertext leaks (absolutely) **no extra information** about the plaintext
 - ◆ (unconditionally) to adversaries of **unlimited computational power**

Refined model

- ◆ **Computational security** guarantees a relaxed notion of security, namely that
 - ◆ the ciphertext leaks **a tiny amount of extra information** about the plaintext
 - ◆ to adversaries with **bounded computational power**

Computational security

General concept in Cryptography

Computational security of a cryptographic scheme guarantees that

- ◆ (1) the scheme can be broken only with **a tiny likelihood**
- ◆ (2) by adversaries with **bounded computational power**

In contrast to **perfect** or **information-theoretic** or **unconditional security**

- ◆ which is typically harder, more costly or, often impossible, to achieve

Computational security (cont.)

General concept in Cryptography

- ◆ *de facto* model for security in most settings
 - ◆ based on an underlying hardness (computational) assumption
 - ◆ integral part of modern cryptography
 - ◆ still allowing for rigorous mathematical proof of security
- ◆ **Asymptotic** description of results

“A scheme is **computationally secure** if any **efficient** attacker succeeds in breaking it with at most **negligible** probability”

Computational security (cont.)

General concept in Cryptography

- ◆ entails two relaxations
 - ◆ security is guaranteed against **efficient** adversaries
 - ◆ if an attacker invests in **sufficiently large resources**, it may break security
 - ◆ goal: make required resources larger than those available to any realistic attacker!
 - ◆ security is guaranteed in a **probabilistic** manner
 - ◆ with some **small probability**, an attacker may break security
 - ◆ goal: make attack probability sufficiently small so that it can be practically ignored!

Security relaxation for encryption

Perfect security: $|k| = 128$ bits, M , $\text{Enc}_k(M)$ are independent, **unconditionally**

- ◆ no extra information is leaked to any attacker

Computational security: M , $\text{Enc}_k(M)$ are independent, **for all practical purposes**

- ◆ no extra information is leaked **but a tiny amount**
 - ◆ e.g., with prob. 2^{-128} (or much less than the likelihood of being hit by lightning)
- ◆ to **computationally bounded** attackers
 - ◆ e.g., who cannot count to 2^{128} (or invest work of more than one century)
- ◆ attacker's best strategy remains **ineffective**
 - ◆ **random guess** a secret key or **exhaustive search** over key space (brute-force attack)

Towards a rigorous definition of computational security

Concrete approach

- ◆ “A scheme is (t, ϵ) -secure if any attacker A , running for time at most t , succeeds in breaking the scheme with probability at most ϵ ”

Asymptotic approach

- ◆ “A scheme is secure if any efficient attacker A succeeds in breaking the scheme with at most negligible probability”

Examples

- ◆ almost optimal security guarantees
 - ◆ if key length n , the number of possible keys is 2^n
 - ◆ attacker running for time t succeeds w/ prob. at most $\sim t/2^n$ (brute-force attack)
- ◆ if $n = 60$, security is enough for attackers running a desktop computer
 - ◆ 4 GHz (4×10^9 cycles/sec), checking all 2^{60} keys require about 9 years
 - ◆ if $n = 80$, a supercomputer would still need ~ 2 years
- ◆ today's recommended security parameter is at least $n = 128$
 - ◆ large difference between 2^{80} and 2^{128} ; e.g., #seconds since Big Bang is $\sim 2^{58}$
 - ◆ a once-in-100-years event corresponds to probability 2^{-30} of happening at a particular sec
 - ◆ if within 1 year of computation attack is successful w/ prob. $1/2^{60}$ then it is more likely that Alice and Bob are hit by lightning

Examples: Big Numbers in the real world

- ◆ Odds for all 5 numbers + Powerball
 - ◆ $292 \times 10^6 \Rightarrow 2^{38}$
- ◆ The Age of the Universe in Seconds
 - ◆ $4.3 \times 10^{17} \Rightarrow 2^{58}$
- ◆ # of cycles in a century of a 4 GHz CPU $\Rightarrow 2^{64}$
- ◆ # of arrangements of a Rubik's cube $4.3 \times 10^{19} \Rightarrow 2^{65}$
- ◆ Atoms in the Earth $1.33 \times 10^{50} \Rightarrow 2^{166}$
- ◆ Electrons in the universe $10^{80} \Rightarrow 2^{266}$

4.2 Introduction to modern cryptography

Cryptography / cryptology

- ◆ Etymology

- ◆ two parts: “crypto” + “graphy” / “logy”
- ◆ original meaning: κρυπτός + γράφω / λόγος (in Greek)
- ◆ English translation: secret + write / speech, logic
- ◆ meaning: secret writing / the study of secrets

- ◆ Historically developed/studied for secrecy in communications

- ◆ i.e., message encryption in the symmetric-key setting
- ◆ main application area: use by military and governments

Classical cryptography Vs. modern cryptography

antiquity – ~70s

- ◆ “*the art or writing and solving codes*”
- ◆ approach
 - ◆ ad-hoc design
 - ◆ trial & error methods
 - ◆ empirically evaluated

~80s – today

- ◆ “*the study of **mathematical techniques** for **securing** digital information, systems, and distributed computations against **adversarial attacks**”*”
- ◆ approach
 - ◆ systematic development & analysis
 - ◆ formal notions of security / adversary
 - ◆ rigorous proofs of security (or insecurity)

Example: Classical Vs. modern cryptography for encryption

antiquity – ~70s

“the **art** of **writing** and **solving codes**”

◆ **ad-hoc study**

- ◆ vulnerabilities/insecurity of
 - ◆ Caesar's cipher
 - ◆ shift cipher
 - ◆ mono-alphabetic substitution cipher

~80s – today

“the study of **mathematical techniques** for **securing** information, systems, and distributed computations against **adversarial attacks**”

◆ **rigorous study**

- ◆ **problem statement:** secret communication over insecure channel
- ◆ **abstract solution concept:** symmetric encryption, Kerckhoff's principle, perfect secrecy
- ◆ **concrete solution & analysis:** OTP cipher, proof of security

Example: Differences of specific ciphers

Caesar's/shift/mono-alphabetic cipher

- ◆ substitution ciphers
 - ◆ Caesar's cipher
 - ◆ **shift is always 3**
 - ◆ shift cipher
 - ◆ **shift is unknown but the same for all characters**
 - ◆ mono-alphabetic substitution/Vigènere cipher
 - ◆ **shift is unknown but the same for all/many character occurrences**

The one-time pad

- ◆ also, a substitution cipher
 - ◆ **shift is unknown and independent for each character occurrence**

Approach in modern cryptography

Formal treatment

- ◆ **fundamental notions** underlying the **design & evaluation** of crypto primitives

Systematic process

- ◆ A) **formal definitions** (what it means for a crypto primitive to be “secure”?)
- ◆ B) **precise assumptions** (which forms of attacks are allowed – and which aren’t?)
- ◆ C) **provable security** (why a candidate instantiation is indeed secure – or not?)

Recall: Secure against what?

- ◆ “Security” has no meaning per se...
- ◆ The security of a system, application, or protocol is always relative to
 - ◆ A set of desired properties
 - ◆ An adversary with specific capabilities
- ◆ Recall: Difficult to define general rules for security
 - ◆ Adapt best practices, heuristics based on the system we are considering!

Example: Physical safes



TL-15 (\$3,000)
15 minutes with
common tools



TL-30 (\$4,500)
30 minutes with
common tools



TRTL-30 (\$10,000)
30 minutes with
common tools and a
cutting torch



TXTL-60 (>\$50,000)
60 minutes with
common tools, a
cutting torch, and up
to 4 oz of explosives

The 3 pillars in Cryptography

- ◆ We have already been familiar with:
 - ◆ A) formal definitions
 - ◆ B) precise assumptions
 - ◆ C) provable security

- ◆ Let's remind ourselves...

The 3 pillars in Cryptography

- ◆ We have already been familiar with:
 - ◆ **A) formal definitions**
 - ◆ B) precise assumptions
 - ◆ C) provable security

- ◆ Let's remind ourselves...

A) Formal definitions

abstract but rigorous description of security problem

- ◆ **computing setting** (to be considered)
 - ◆ involved parties, communication model, core functionality
- ◆ **underlying cryptographic scheme** (to be designed)
 - ◆ e.g., symmetric-key encryption scheme
- ◆ **desired properties** (to be achieved)
 - ◆ security related
 - ◆ non-security related
 - ◆ e.g., correctness, efficiency, etc.

Why formal definitions are important?

- ◆ **successful project management**
 - ◆ good design requires clear/specific security goals
 - ◆ helps to avoid critical omissions or over engineering
- ◆ **provable security**
 - ◆ rigorous evaluation requires a security definition
 - ◆ helps to separate secure from insecure solutions
- ◆ **qualitative analysis/modular design**
 - ◆ thorough comparison requires an exact reference
 - ◆ helps to secure complex computing systems

Example: Problem at hand

abstract but rigorous description of **security problem** (to be solved)



secret communication

Insecure channel



Example: Formal definitions (1)

- ◆ computing setting (to be considered)
 - ◆ e.g., **involved parties**, **communication model**, core functionality



Alice, Bob, Eve



Alice wants to send a message m to Bob; Eve can eavesdrop sent messages



Alice/Bob may transform the transmitted/received message and share info



Example: Formal definitions (2)

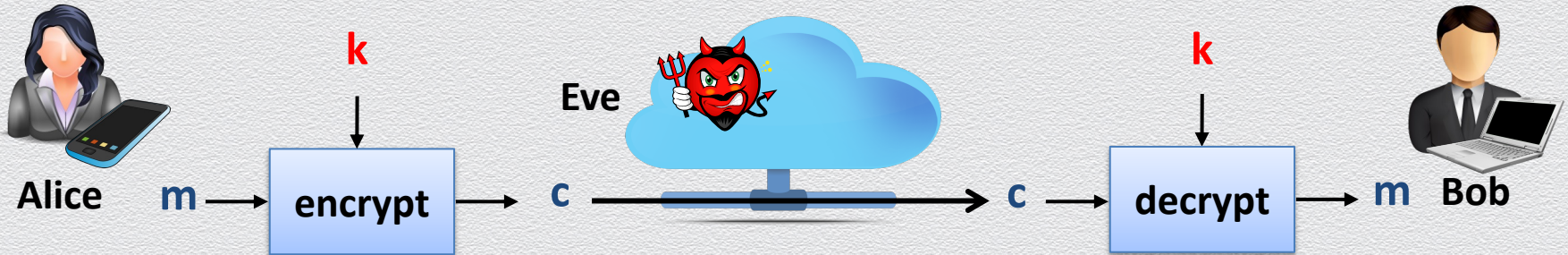
- ◆ **underlying cryptographic scheme**

(to be designed)



symmetric-key encryption scheme

- ◆ Alice and Bob share and use a key k
- ◆ Alice encrypts plaintext m to ciphertext c and sends c instead of m
- ◆ Bob decrypts received c to get a message m'



Example: Formal definitions (3)

- ◆ **desired properties**

(to be achieved)

- ◆ **security (informal)**



Eve “cannot learn” m (from c)

- ◆ **correctness (informal)**



If Alice encrypts m to c , then Bob decrypts c to (the original message) m



Example: Probabilistic view of symmetric encryption

A symmetric-key encryption scheme is defined by

- ◆ a **message space** \mathcal{M} , $|\mathcal{M}| > 1$, and a triple (**Gen**, **Enc**, **Dec**)
- ◆ **Gen**: probabilistic key-generation algorithm, defines **key space** \mathcal{K}
 - ◆ $\text{Gen}(1^n) \rightarrow k \in \mathcal{K}$ (security parameter n)
- ◆ **Enc**: probabilistic encryption algorithm, defines **ciphertext space** \mathcal{C}
 - ◆ $\text{Enc}: \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C}$, $\text{Enc}(k, m) = \text{Enc}_k(m) \rightarrow c \in \mathcal{C}$
- ◆ **Dec**: deterministic encryption algorithm
 - ◆ $\text{Dec}: \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{M}$, $\text{Dec}(k, c) = \text{Dec}_k(c) := m \in \mathcal{M}$ or \perp

Example: Formal definitions (4)

Perfect correctness

- ◆ for any $k \in \mathcal{K}$, $m \in \mathcal{M}$ and any ciphertext c output of $\text{Enc}_k(m)$, it holds that

$$\Pr[\text{Dec}_k (c) = m] = 1$$

Perfect security (or information-theoretic security)

- ◆ the adversary should be able to learn no additional information on m

random
experiment

$$\mathcal{D}_{\mathcal{M}} \rightarrow m$$
$$\mathcal{D}_{\mathcal{K}} \rightarrow k$$
$$\text{Enc}_k(m) \rightarrow c$$


Eve's view
remains
the same!



Example: Equivalent definitions of perfect security

1) a posteriori = a priori

For every $\mathcal{D}_{\mathcal{M}}$, $m \in \mathcal{M}$ and $c \in \mathcal{C}$, for which $\Pr [C = c] > 0$, it holds that

$$\Pr[M = m \mid C = c] = \Pr[M = m]$$

2) C is independent of M

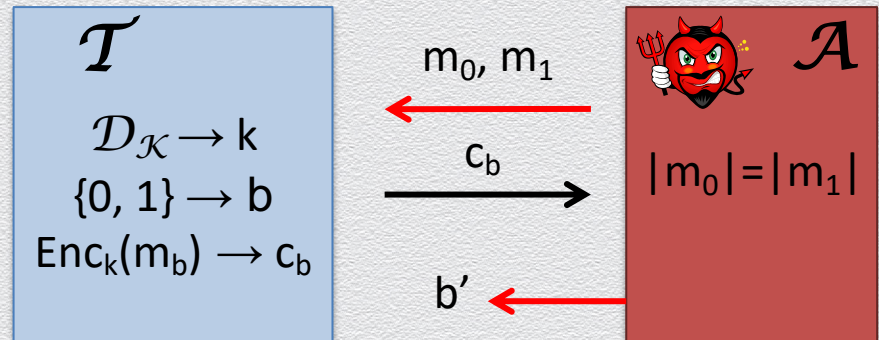
For every $m, m' \in \mathcal{M}$ and $c \in \mathcal{C}$, it holds that

$$\Pr[\text{Enc}_K(m) = c] = \Pr[\text{Enc}_K(m') = c]$$

3) indistinguishability

For every \mathcal{A} , it holds that

$$\Pr[b' = b] = 1/2$$



From perfect to computational EAV-security

- ◆ **perfect** security: $M, \text{Enc}_K(M)$ are independent
 - ◆ absolutely **no information is leaked** about the plaintext
 - ◆ to adversaries that **unlimited computational power**
- ◆ **computational** security: for all **practical** purposes, $M, \text{Enc}_K(M)$ are independent
 - ◆ **a tiny amount of information is leaked** about the plaintext (e.g., w/ prob. 2^{-60})
 - ◆ to adversaries with **bounded computational power** (e.g., attacker invests 200ys)
- ◆ attacker's **best strategy** remains **ineffective**
 - ◆ **random guess** on secret key; or
 - ◆ **exhaustive search** over key space (**brute force attack**)

Relaxing indistinguishability

Relax the definition of perfect secrecy – that is based on indistinguishability

- ◆ require that m_0, m_1 are chosen by a **PPT adversary**
- ◆ require that no **PPT adversary** can distinguish $\text{Enc}_k(m_0)$ from $\text{Enc}_k(m_1)$

non-negligibly better than guessing

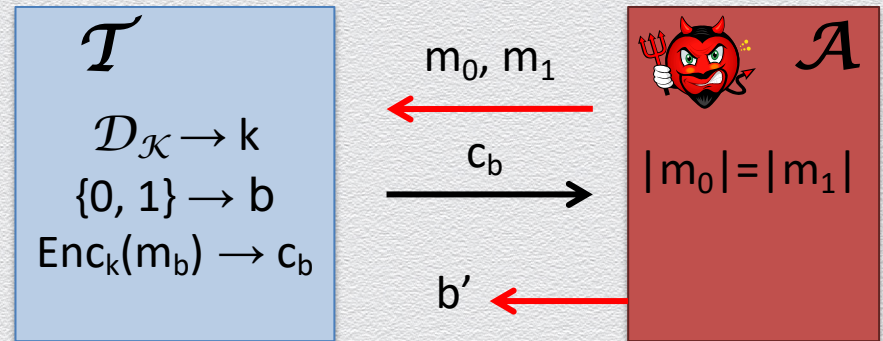
3) indistinguishability

For every \mathcal{A} , it holds that

$$\Pr[b' = b] = 1/2 + \text{negl}$$

PPT

negl



The 3 pillars in Cryptography

- ◆ We have already been familiar with:
 - ◆ A) formal definitions
 - ◆ **B) precise assumptions**
 - ◆ C) provable security


- ◆ Let's remind ourselves...

B) Why precise assumptions are important?

- ◆ **basis** for proofs of security
 - ◆ security holds under specific assumptions
- ◆ **comparison** among possible solutions
 - ◆ relations among different assumptions
 - ◆ stronger/weaker (i.e., less/more plausible to hold), “A implies B” or “A and B are equivalent”
 - ◆ refutable Vs. non-refutable
- ◆ **flexibility** (in design & analysis)
 - ◆ **validation** – to gain confidence or refute
 - ◆ **modularity** – to choose among concrete schemes that satisfy the same assumptions
 - ◆ **characterization** – to identify simplest/minimal/necessary assumptions

Example: Precise assumptions (1)

- ◆ **adversary**

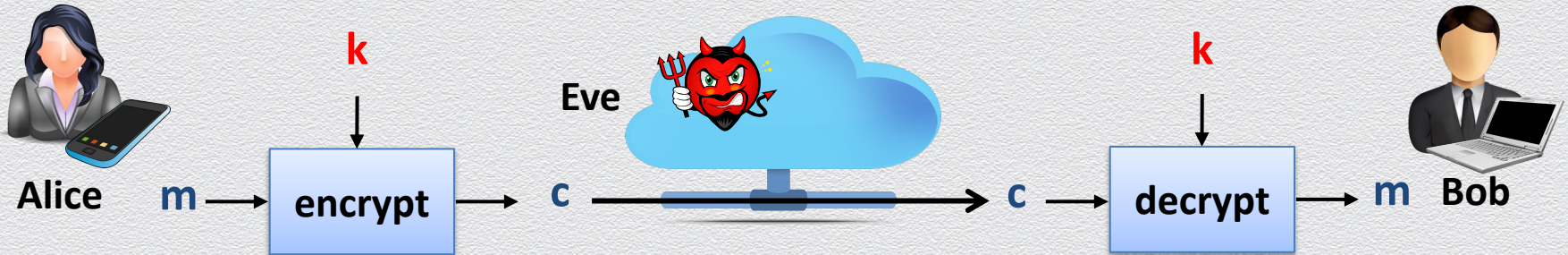
- ◆ type of attacks – a.k.a. **threat model**  **eavesdropping**
- ◆ **capabilities** (e.g., a priori knowledge, access to information, party corruptions)
- ◆ **limitations** (e.g., bounded memory, passive Vs. active)



Eve may know the a priori distribution of messages sent by Alice



Eve doesn't know/learn the secret k (shared by Alice and Bob)

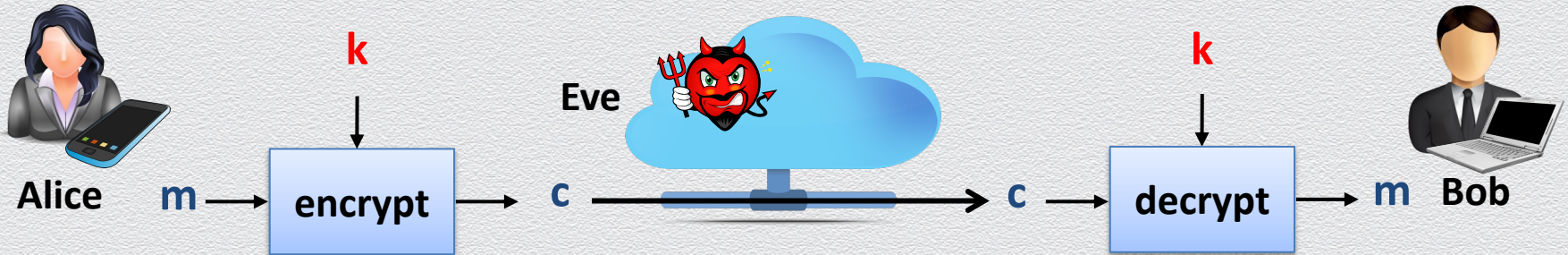


Example: Precise assumptions (2)

- ◆ **computational assumptions** (about hardness of certain tasks)
 - ◆ e.g., factoring of large composite numbers is hard





no computational assumptions
– a.k.a. perfect secrecy (or information-theoretic security)





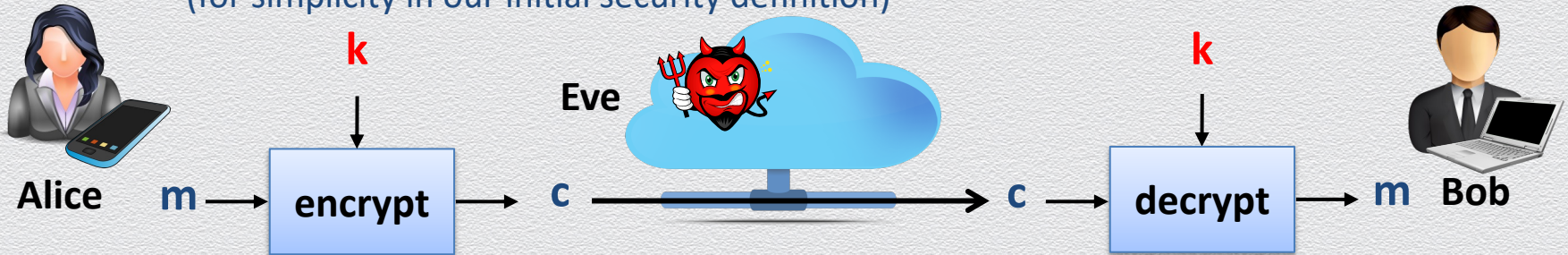
Example: Precise assumptions (3)

- ◆ **computing setting**

- ◆ **system set up**, initial state, **key distribution**, **randomness**...  key k is generated randomly using the uniform distribution
- ◆ means of **communication** (e.g., channels, rounds, messages...)
- ◆ timing assumptions (e.g., synchronicity, epochs, ...)

 key k is securely distributed to and securely stored at Alice and Bob

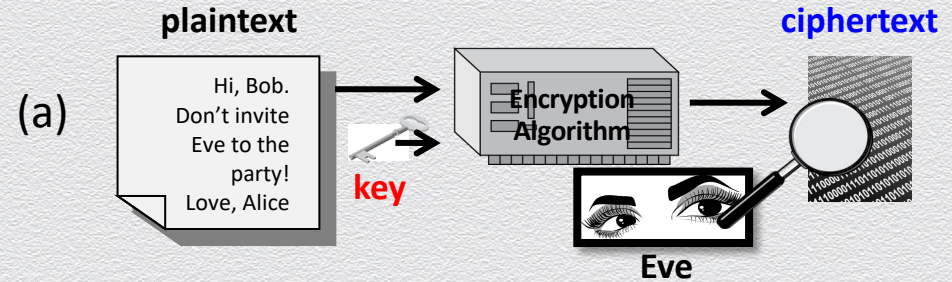
 one message m is only communicated (for simplicity in our initial security definition)  k, m are chosen independently



Possible eavesdropping attacks (I)

An attacker may possess a

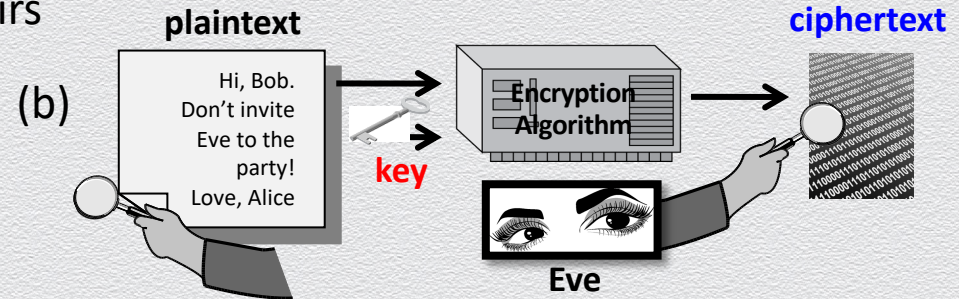
- ◆ (a) collection of ciphertexts
 - ◆ **ciphertext only attack**
 - ◆ this will be the **default attack type** when we will next define the concept of perfect security



Possible eavesdropping attacks (II)

An attacker may possess a

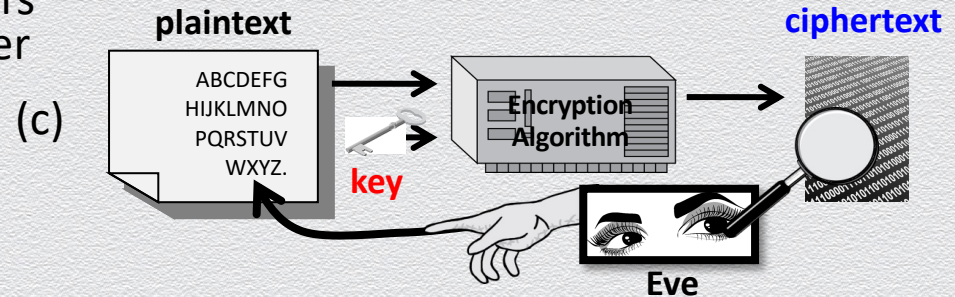
- ◆ (a) collection of ciphertexts
 - ◆ ciphertext only attack
- ◆ (b) collection of plaintext/ciphertext pairs
 - ◆ known plaintext attack



Possible eavesdropping attacks (III)

An attacker may possess a

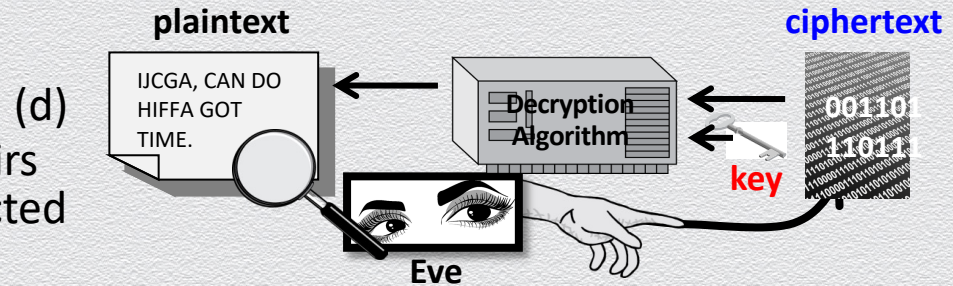
- ◆ (a) collection of ciphertexts
 - ◆ **ciphertext only attack**
- ◆ (b) collection of plaintext/ciphertext pairs
 - ◆ **known plaintext attack**
- ◆ (c) collection of plaintext/ciphertext pairs for plaintexts selected by the attacker
 - ◆ **chosen plaintext attack**



Possible eavesdropping attacks (IV)

An attacker may possess a

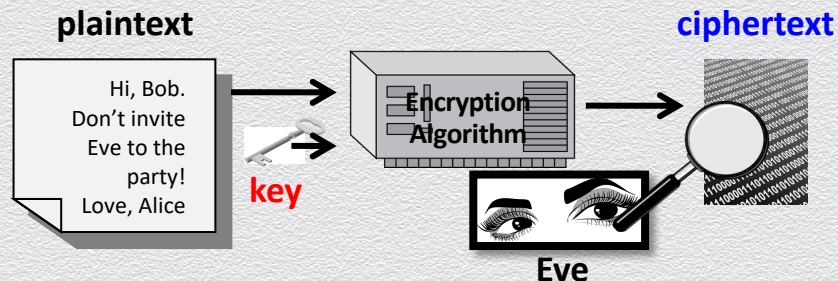
- ◆ (a) collection of ciphertexts
 - ◆ **ciphertext only attack**
- ◆ (b) collection of plaintext/ciphertext pairs
 - ◆ **known plaintext attack**
- ◆ (c) collection of plaintext/ciphertext pairs for plaintexts selected by the attacker
 - ◆ **chosen plaintext attack**
- ◆ (d) collection of plaintext/ciphertext pairs for (plaintexts and) ciphertexts selected by the attacker
 - ◆ **chosen ciphertext attack**



Main security properties against eavesdropping

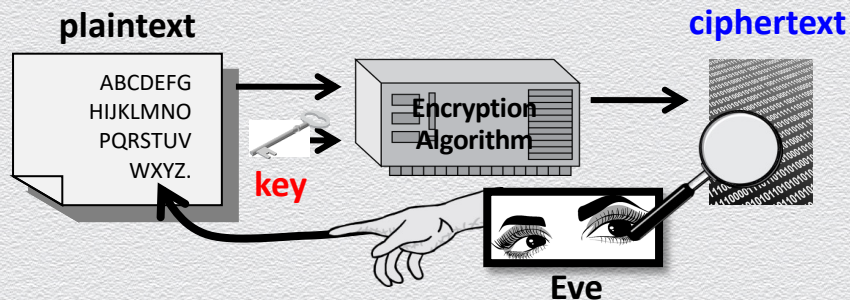
“plain” security

- ◆ protects against ciphertext-only attacks
 - ◆ **EAV-attack**



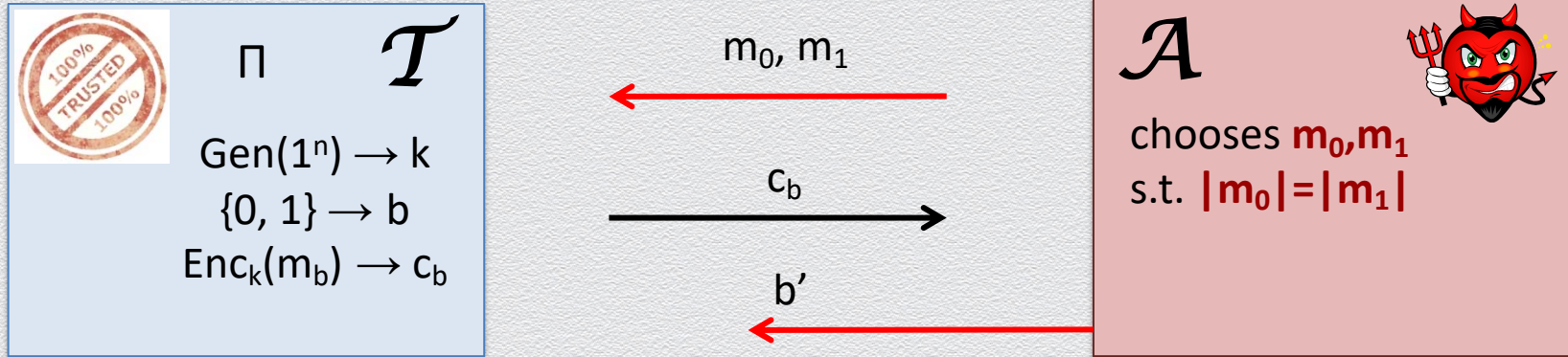
“advanced” security

- ◆ protects against chosen plaintext attacks
 - ◆ **CPA-attack**



Game-based computational EAV-security

encryption scheme $\Pi = \{\mathcal{M}, (\text{Gen}, \text{Enc}, \text{Dec})\}$

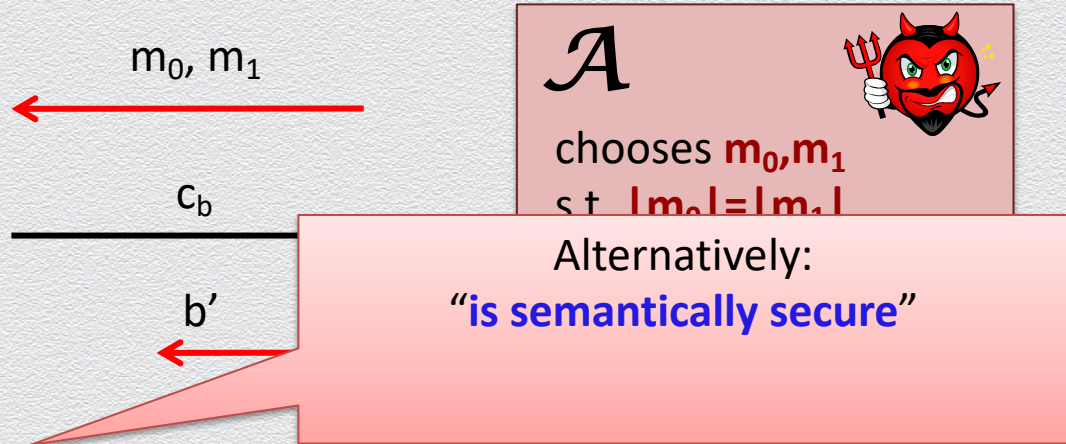
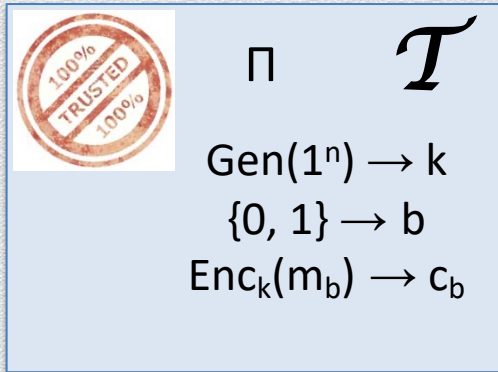


We say that (Enc, Dec) is **EAV-secure** if any PPT adversary \mathcal{A} guesses b correctly with probability at most $0.5 + \epsilon(n)$, where ϵ is a negligible function

I.e., no PPT \mathcal{A} computes b correctly non-negligibly better than randomly guessing

Game-based computational EAV-security

encryption scheme $\Pi = \{\mathcal{M}, (\text{Gen}, \text{Enc}, \text{Dec})\}$

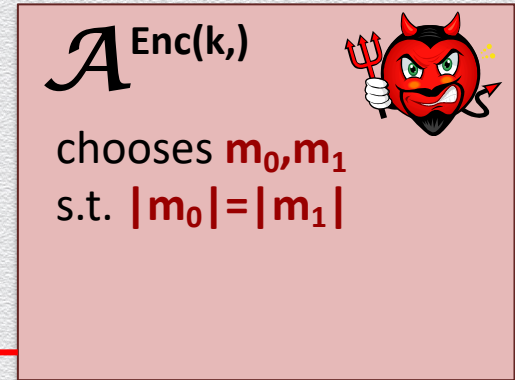
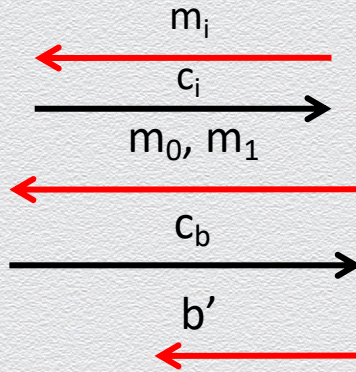
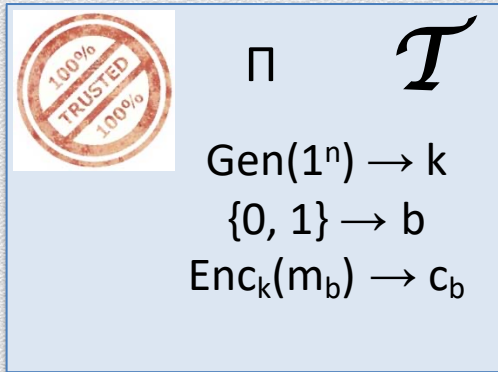


We say that (Enc, Dec) is **EAV-secure** if any PPT adversary \mathcal{A} guesses b correctly with probability at most $0.5 + \epsilon(n)$, where ϵ is a negligible function

i.e., no PPT \mathcal{A} computes b correctly non-negligibly better than randomly guessing

Game-based computational CPA-security

encryption scheme $\Pi = \{\mathcal{M}, (\text{Gen}, \text{Enc}, \text{Dec})\}$



We say that (Enc, Dec) is **CPA-secure** if any PPT adversary \mathcal{A} guesses b correctly with probability at most $0.5 + \epsilon(n)$, where ϵ is a negligible function

I.e., no PPT \mathcal{A} computes b correctly non-negligibly better than randomly guessing, **even when it learns the encryptions of messages of its choice**

On CPA security

Facts

- ◆ Any encryption scheme that is CPA-secure is also CPA-secure for multiple encryptions
- ◆ **CPA security implies probabilistic encryption – can you see why?**
- ◆ EAV-security for multiple messages implies probabilistic encryption

The 3 pillars in Cryptography

- ◆ We have already been familiar with:
 - ◆ A) formal definitions
 - ◆ B) precise assumptions
 - ◆ **C) provable security**

- ◆ Let's remind ourselves...

C) Provably security

Security

- ◆ subject to certain **assumptions**, a scheme is proved to be **secure** according to a specific **definition**, against a specific **adversary**
 - ◆ in practice the scheme may break if
 - ◆ some assumptions do not hold or the attacker is more powerful

Insecurity

- ◆ a scheme is proved to be **insecure** with respect to a specific **definition**
 - ◆ it suffices to find a **counterexample attack**

Why provable security is important?

Typical performance

- ◆ in some areas of computer science
formal proofs may not be essential
- ◆ simulate hard-to-analyze algorithm to experimentally study its performance on “typical” inputs
- ◆ in practice, **typical/average case** occurs

Worst case performance

- ◆ in cryptography and secure protocol design
formal proofs are essential
- ◆ “experimental” security analysis is not possible
- ◆ the notion of a “typical” adversary makes little sense and is unrealistic
- ◆ in practice, **worst case attacks will occur**
 - ◆ an adversary will use any means in its power to break a scheme

4.3 (Using OTP with) Pseudo-randomness

Perfect secrecy & randomness

Role of randomness in encryption is **integral**

- ◆ in a perfectly secret cipher, the ciphertext **doesn't depend** on the message
 - ◆ the ciphertext appears to be **truly random**
 - ◆ the uniform key-selection distribution **is imposed also onto** produced ciphertexts
 - ◆ e.g., $c = k \text{ XOR } m$ (for uniform k and any distribution over m)

When security is computational, randomness is **relaxed** to “pseudorandomness”

- ◆ the ciphertext appears to be “**pseudorandom**”
 - ◆ it **cannot be efficiently distinguished** from truly random

Symmetric encryption as “OPT with pseudorandomness”

Stream cipher

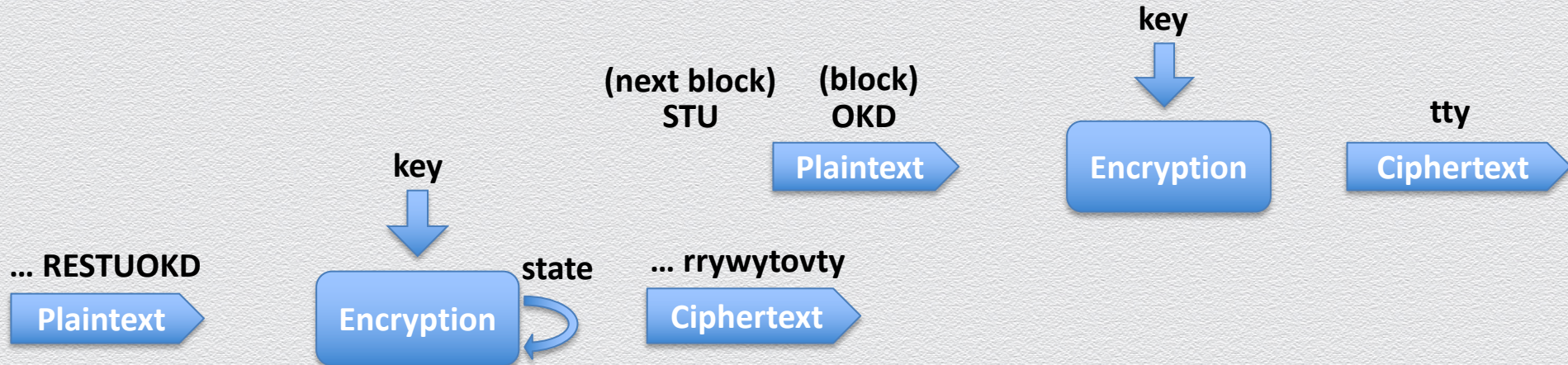
Uses a **short** key to encrypt **long** symbol **streams** into a **pseudorandom** ciphertext

- ◆ based on abstract crypto primitive of **pseudorandom generator (PRG)**

Block cipher

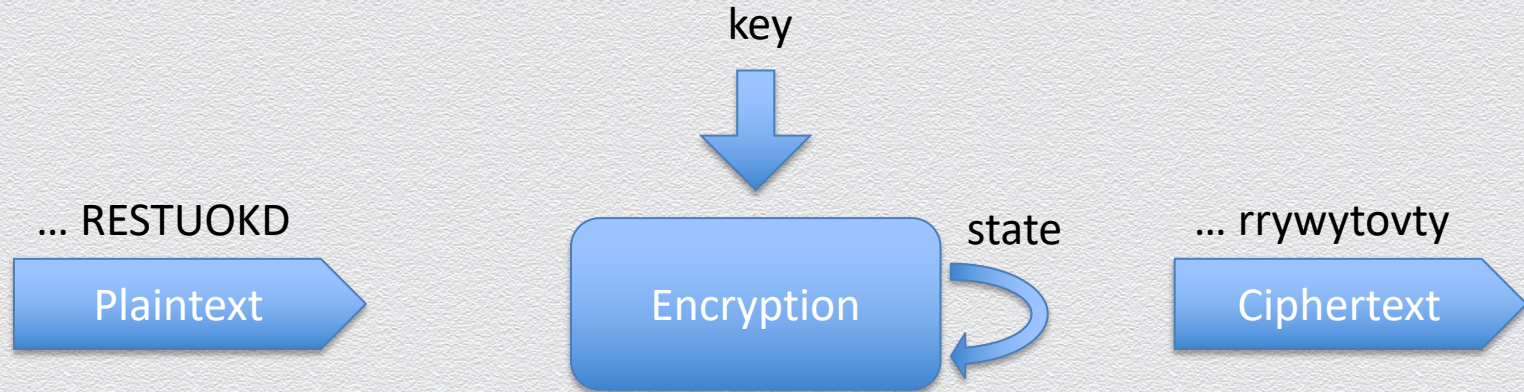
Uses a **short** key to encrypt **blocks** of symbols into **pseudorandom** ciphertext blocks

- ◆ based on abstract crypto primitive of **pseudorandom function (PRF)**



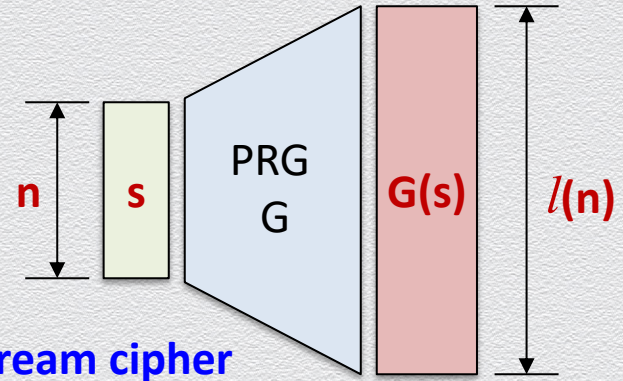
4.3.1 Pseudorandom generators

Stream ciphers



Pseudorandom generators (PRGs)

Deterministic algorithm G that on input a **seed** $s \in \{0,1\}^t$, outputs $G(s) \in \{0,1\}^{l(t)}$

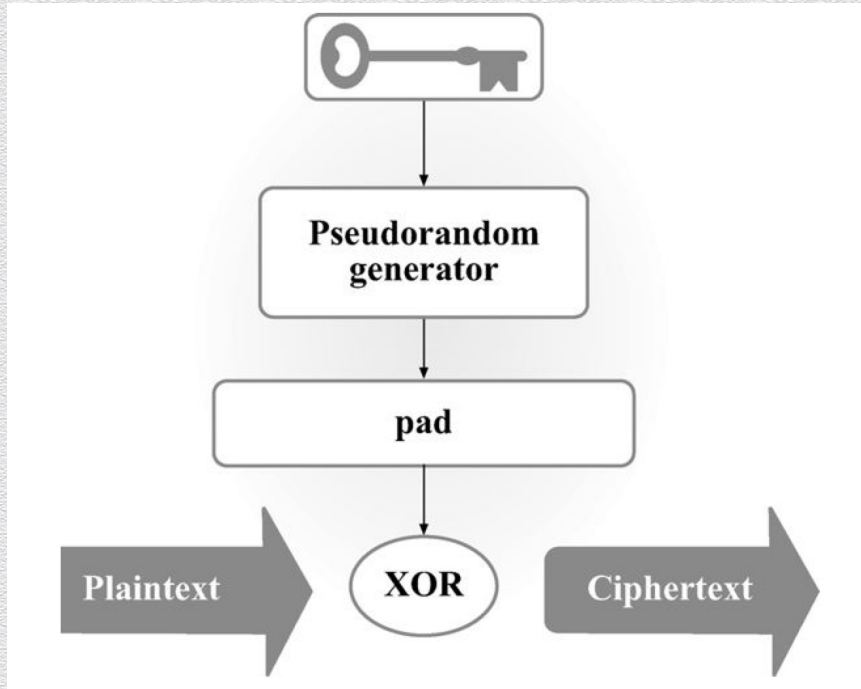


G is a PRG if:

- ◆ **expansion**
 - ◆ for polynomial l , it holds that for any n , $l(n) > n$
 - ◆ models the process of **extracting** randomness from a short random string
- ◆ **pseudorandomness**
 - ◆ no efficient statistical test can tell apart $G(s)$ from a truly random string

Generic PRG-based symmetric encryption

- ◆ **Fixed-length** message encryption



encryption scheme is plain-secure
as long as the underlying PRG is secure

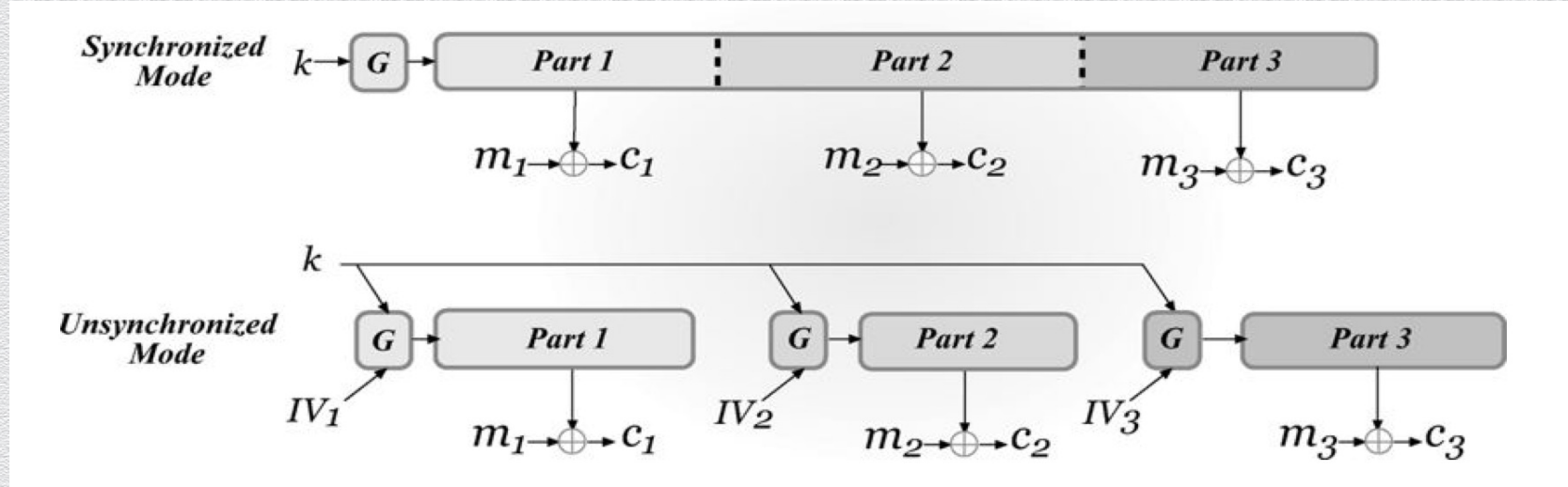
Generic PRG-based symmetric encryption (cont.)

- ◆ **Bounded- or arbitrary-length** message encryption
 - ◆ specified by a mode of operation for using an underlying stateful stream cipher, repeatedly, to encrypt/decrypt a stream of symbols

Stream ciphers: Modes of operations

- ◆ **Bounded- or arbitrary-length** message encryption

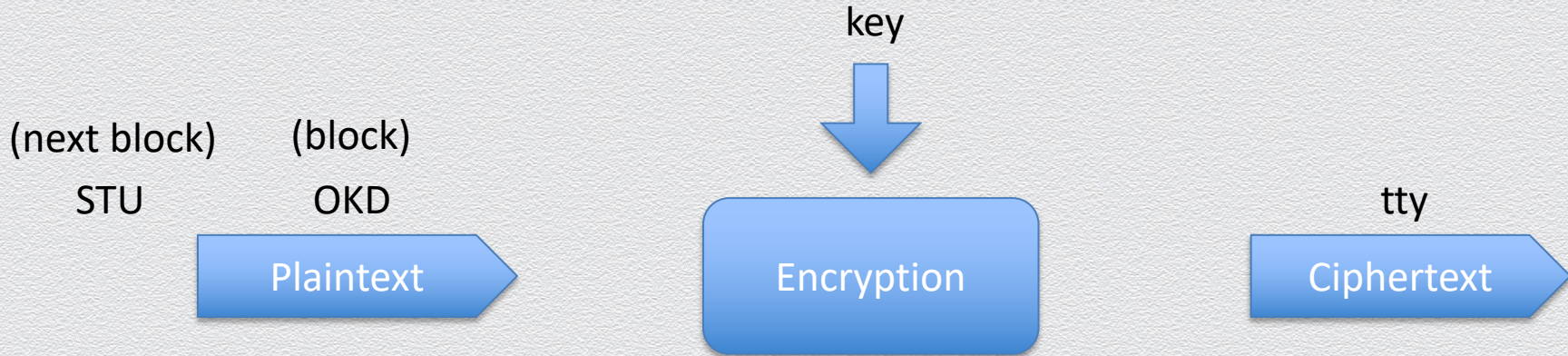
on-the-fly computation of new pseudorandom bits, no IV needed, plain-secure



random IV used for every new message is sent along with ciphertext, advanced-secure

4.3.2 Pseudorandom functions

Block ciphers



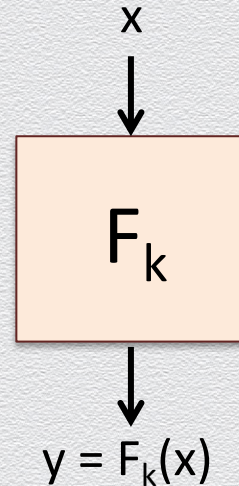
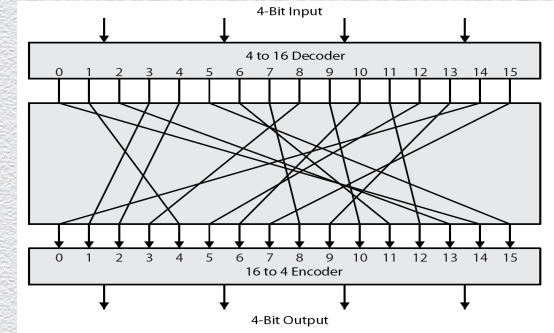
Realizing ideal block ciphers in practice

We want a **random** mapping of n-bit inputs to n-bit outputs

- ◆ there are $\sim 2^{(n2^n)}$ possible such mappings
- ◆ none of the above can be implemented in practice

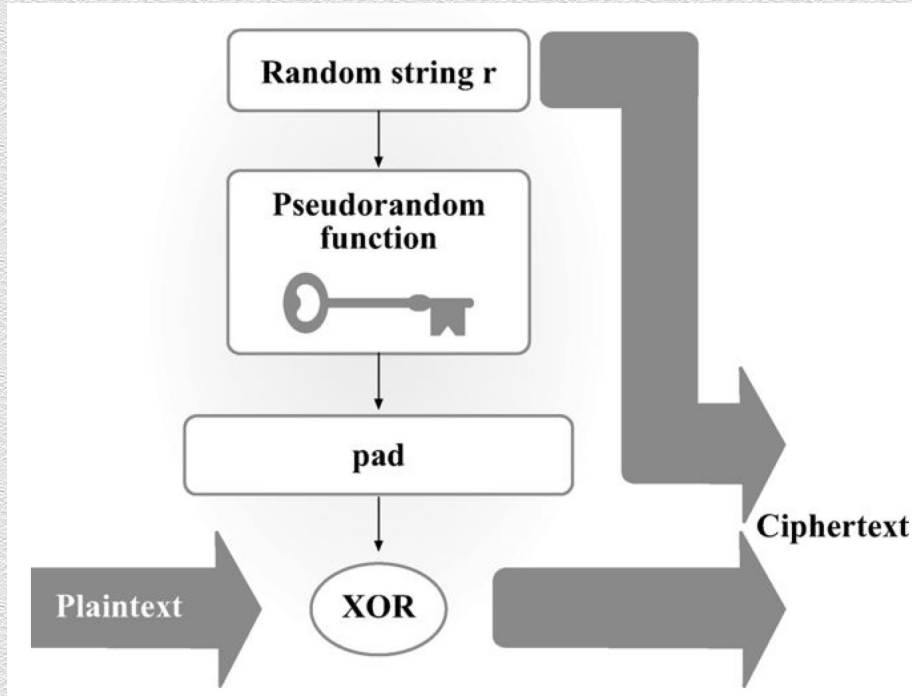
Instead, we use a keyed function $F_k : \{0,1\}^n \rightarrow \{0,1\}^n$

- ◆ indexed by a t-bit key k
- ◆ there are only 2^t such keyed functions
- ◆ a random key selects a “random-enough” mapping or a **pseudorandom function**



Generic PRF-based symmetric encryption

- ◆ **Fixed-length** message encryption



encryption scheme is advanced-secure
as long as the underlying PRF is secure

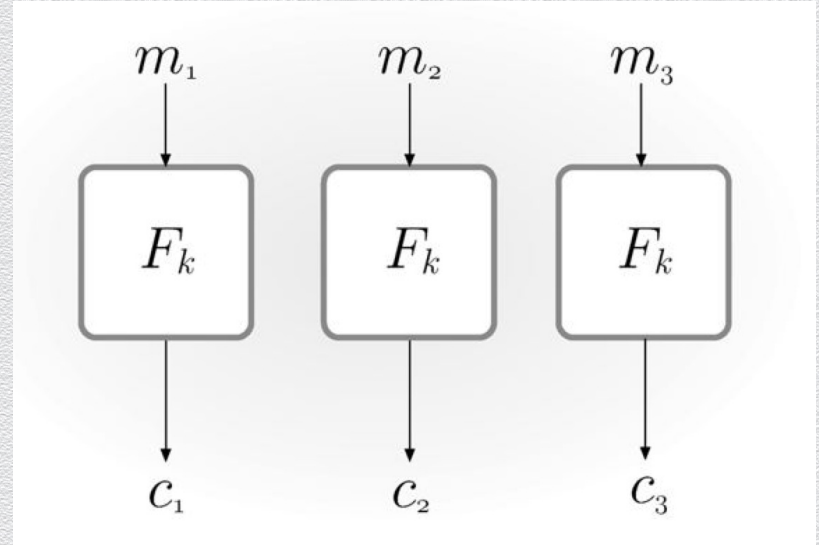
Generic PRF-based symmetric encryption (cont.)

- ◆ **Arbitrary-length** message encryption
 - ◆ specified by a mode of operation for using an underlying stateless block cipher, repeatedly, to encrypt/decrypt a sequence of message blocks

4.4 Modes of operations

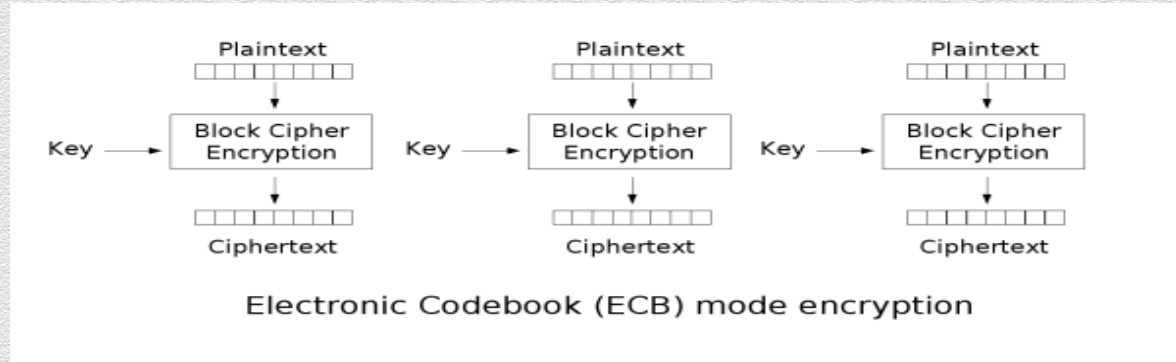
Block ciphers: Modes of operations (I)

- ◆ ECB - electronic code book
 - ◆ insecure, of only historic value
 - ◆ deterministic, thus not CPA-secure
 - ◆ actually, not even EAV-secure



Electronic Code Book (ECB)

- ◆ The simplest mode of operation
 - ◆ block $P[i]$ encrypted into ciphertext block $C[i] = \text{Enc}_k(P[i])$
 - ◆ block $C[i]$ decrypted into plaintext block $M[i] = \text{Dec}_k(C[i])$



Strengths & weaknesses of ECB

Strengths

- ◆ very simple
- ◆ allows for parallel encryptions of the blocks of a plaintext
- ◆ can tolerate the loss or damage of a block

Weaknesses

- ◆ poor security
- ◆ produces the same ciphertext on the same plaintext (under the same key)
- ◆ documents and images are not suitable for ECB encryption, since patterns in the plaintext are repeated in the ciphertext
- ◆ e.g.,

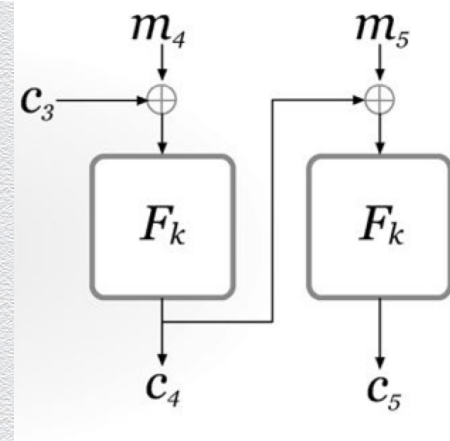
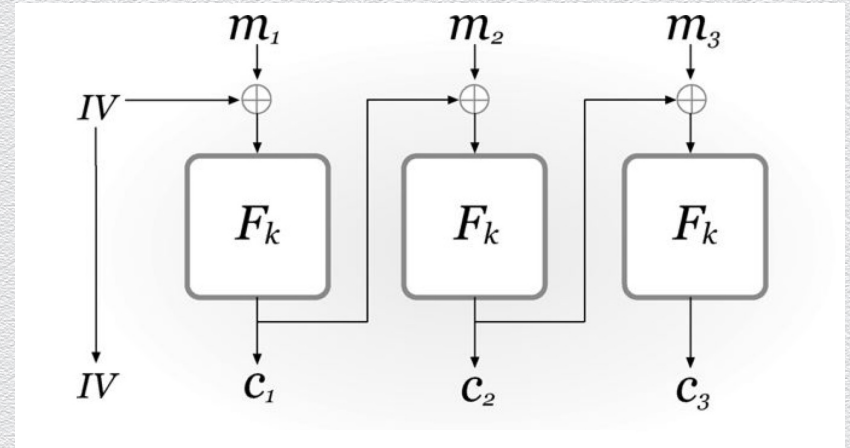


ECB



Block ciphers: Modes of operations (II)

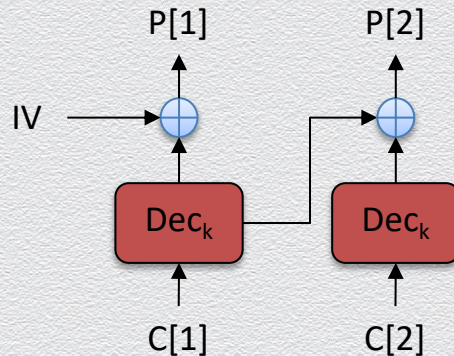
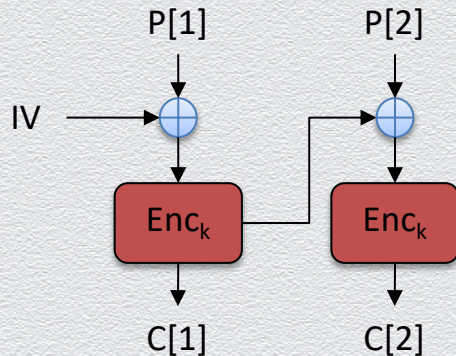
- ◆ CBC – cipher block chaining
 - ◆ CPA-secure if F_k a permutation
 - ◆ uniform IV
 - ◆ otherwise security breaks
- ◆ Chained CBC
 - ◆ use last block ciphertext of current message as IV of next message
 - ◆ saves bandwidth but not CPA-secure



Cipher Block Chaining (CBC) [or chaining]

Alternatively, the previous-block ciphertext is “mixed” with the current-block plaintext

- ◆ e.g., using XOR
 - ◆ each block is encrypted as $C[i] = \text{Enc}_k (C[i - 1] \oplus P[i])$,
 - ◆ each ciphertext is decrypted as $P[i] = C[i - 1] \oplus \text{Dec}_k (C[i])$
 - ◆ here, $C[0] = \text{IV}$ is a uniformly random initialization vector that is transmitted separately

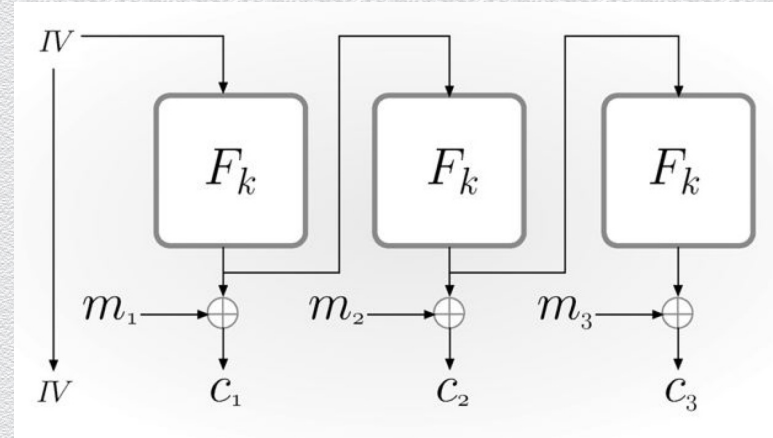


CBC



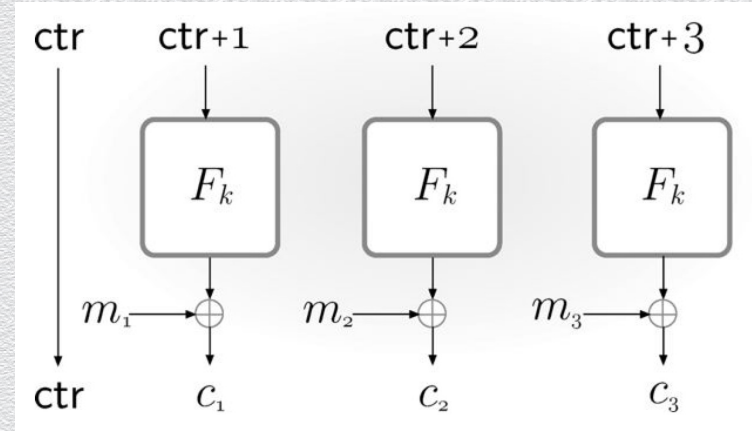
Block ciphers: Modes of operations (III)

- ◆ OFB – output feedback
 - ◆ uniform IV
 - ◆ no need message length to be multiple of n
 - ◆ resembles synchronized stream-cipher mode
 - ◆ CPA-secure if F_k is PRF



Block ciphers: Modes of operations (IV)

- ◆ CTR – counter mode
 - ◆ uniform ctr
 - ◆ no need message length to be multiple of n
 - ◆ resembles synchronized stream-cipher mode
 - ◆ CPA-secure if F_k is PRF
 - ◆ no need for F_k to be invertible
 - ◆ parallelizable



Notes on modes of operation

- ◆ block length matters
 - ◆ if small, IV or ctr can be “recycled”
- ◆ IV are often misused
 - ◆ e.g., reused or not selected uniformly at random
 - ◆ in this case, CBC is a better option than OFB/CTR

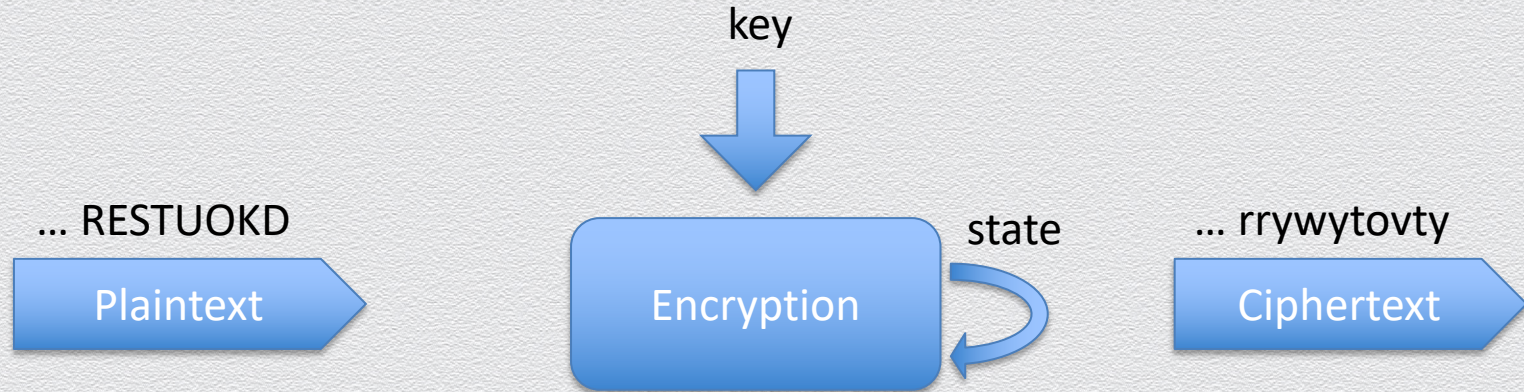
Brute-force attacks against stream/block ciphers

Brute-force attack amounts to checking all possible 2^t seeds/keys

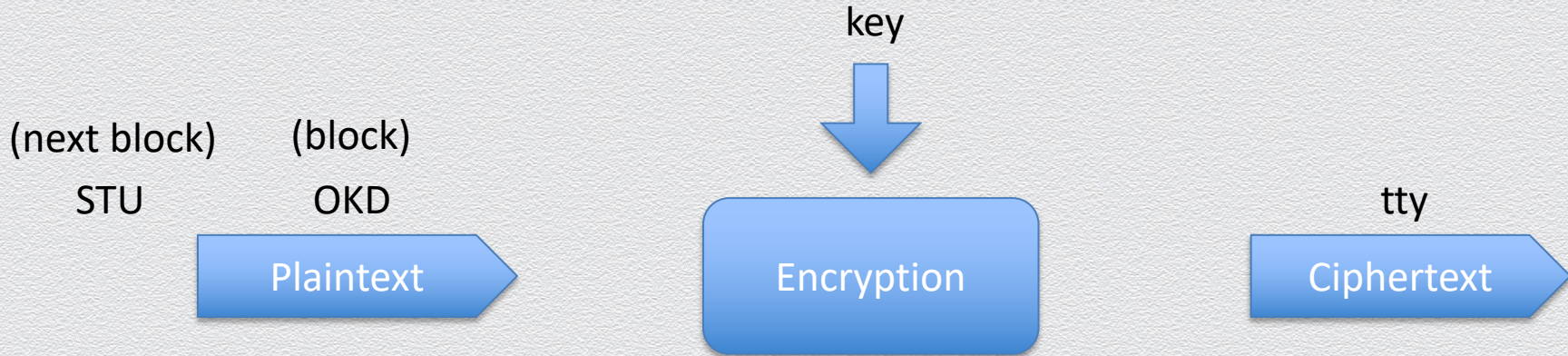
- ◆ for block ciphers, by construction (due to confusion & diffusion, as we will see), the key cannot be extracted even if a valid plaintext/ciphertext pair is captured
- ◆ thus, as expected, **the longer the key size the stronger the security**

4.5 Block ciphers in practice: DES & AES

Recall: Stream ciphers



Recall: Block ciphers



Stream Vs. Block ciphers

	Stream	Block
Advantages	<ul style="list-style-type: none">• Speed of transformation• Low error propagation	<ul style="list-style-type: none">• High diffusion• Immunity to insertion of symbol
Disadvantages	<ul style="list-style-type: none">• Low diffusion• Susceptibility to malicious insertions and modifications	<ul style="list-style-type: none">• Slowness of encryption• Padding• Error propagation

Techniques used in practice for symmetric encryption

- ◆ Substitution
 - ◆ exchanging one set of bits for another set
- ◆ Transposition
 - ◆ rearranging the order of the ciphertext bits
 - ◆ to break any regularities in the underlying plaintext
- ◆ Confusion
 - ◆ enforcing complex functional relationship between the plaintext/key pair & the ciphertext
 - ◆ e.g., flipping a bit in plaintext or key causes unpredictable changes to new ciphertext
- ◆ Diffusion
 - ◆ distributes information from single plaintext characters over entire ciphertext output
 - ◆ e.g., even small changes to plaintext result in broad changes to ciphertext

Substitution boxes

- ◆ substitution can also be done on binary numbers
- ◆ such substitutions are usually described by substitution boxes, or S-boxes

	00	01	10	11		0	1	2	3
00	0011	0100	1111	0001	0	3	8	15	1
01	1010	0110	0101	1011	1	10	6	5	11
10	1110	1101	0100	0010	2	14	13	4	2
11	0111	0000	1001	1100	3	7	0	9	12
		(a)					(b)		

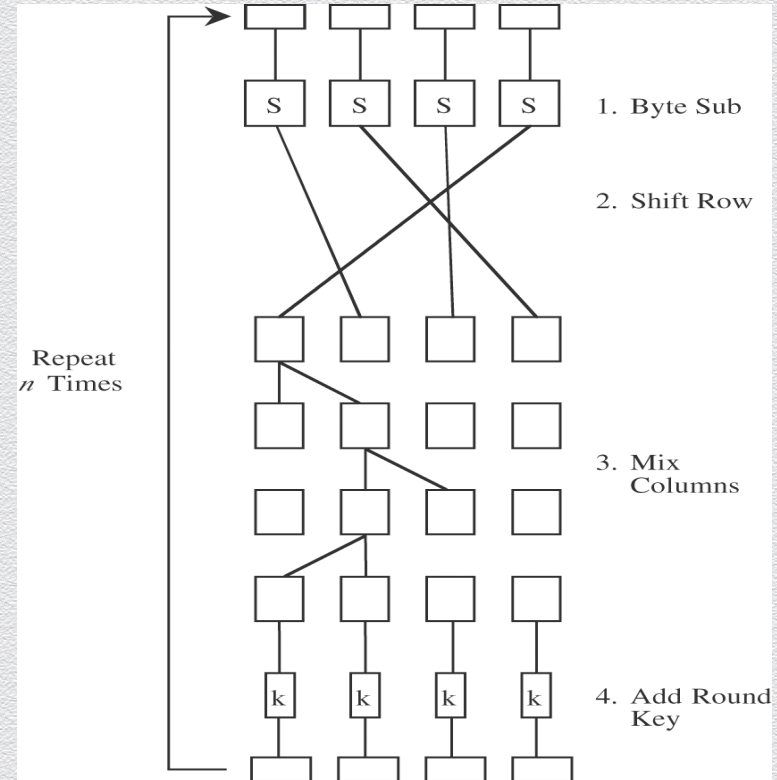
Figure 8.3: A 4-bit S-box (a) An S-box in binary. (b) The same S-box in decimal.

DES vs. AES

	DES	AES
Date designed	1976	1999
Block size	64 bits	128 bits
Key length	56 bits (effective length); up to 112 bits with multiple keys	128, 192, 256 (and possibly more) bits
Operations	16 rounds	10, 12, 14 (depending on key length); can be increased
Encryption primitives	Substitution, permutation	Substitution, shift, bit mixing
Cryptographic primitives	Confusion, diffusion	Confusion, diffusion
Design	Open	Open
Design rationale	Closed	Open
Selection process	Secret	Secret, but open public comments and criticisms invited
Source	IBM, enhanced by NSA	Independent Dutch cryptographers

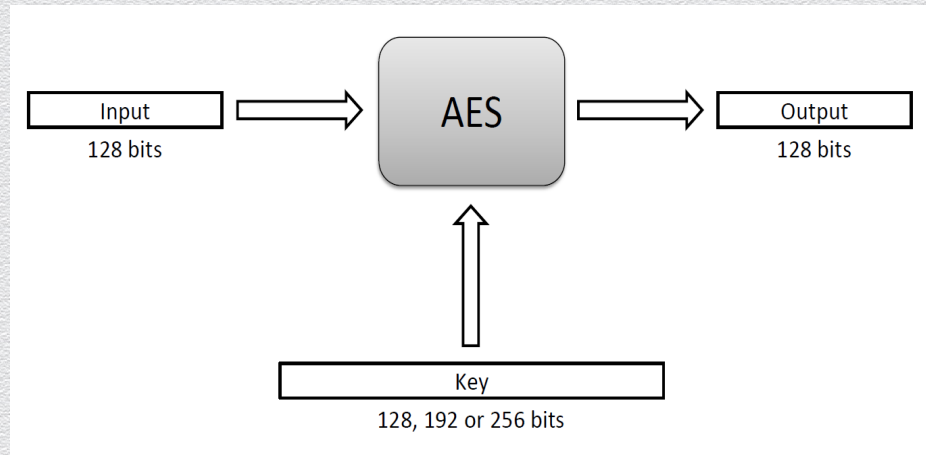
AES: Advanced Encryption System

- ◆ symmetric block cipher, a.k.a. Rijndael
- ◆ developed in 1999 by independent Dutch cryptographers in response to the 1997 NIST's public call for a replacement to DES
- ◆ still in common use
 - ◆ on the longevity of AES
 - ◆ larger key sizes possible to use
 - ◆ not known serious practical attacks

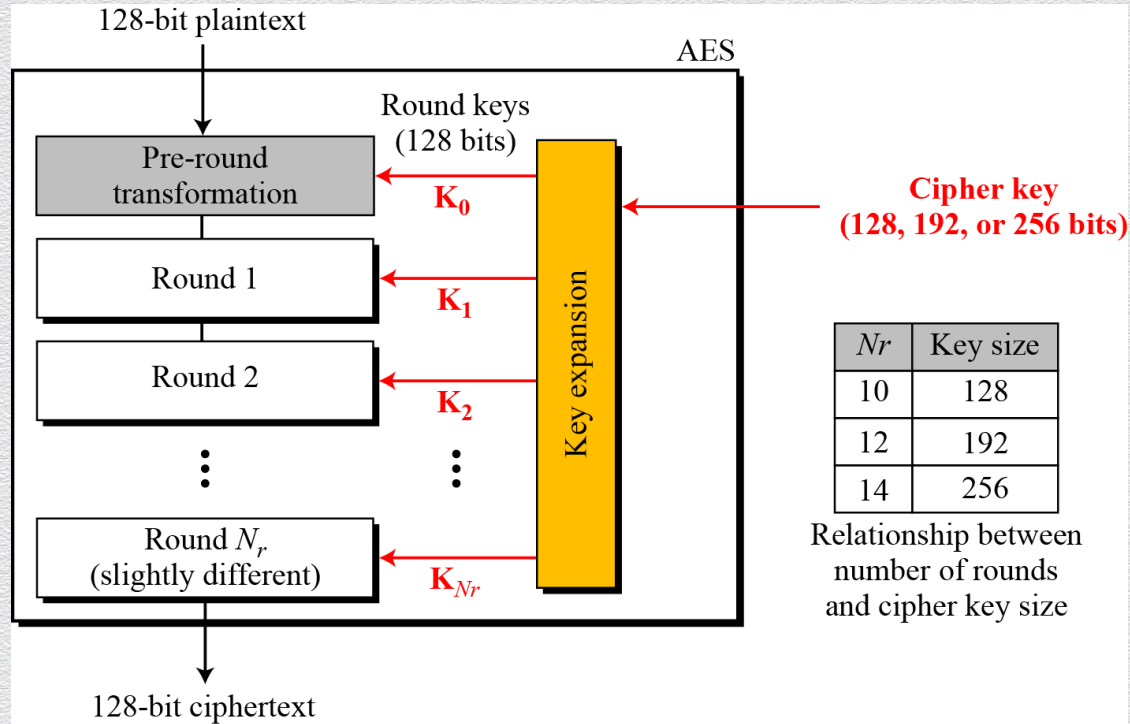


AES: Key design features

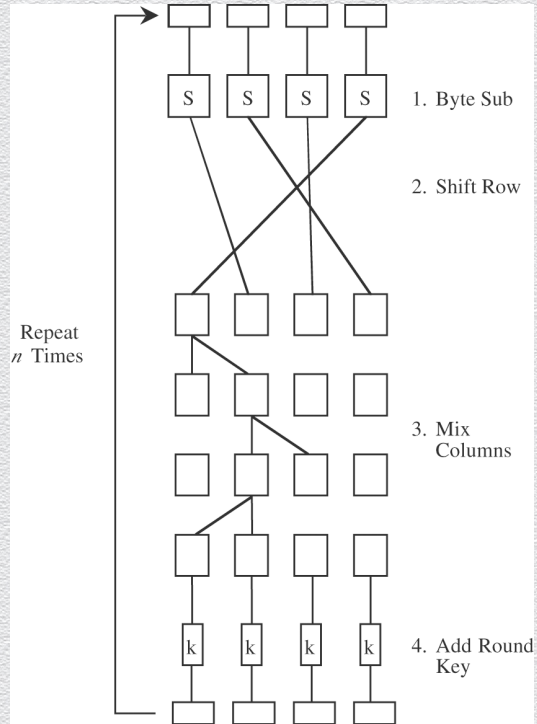
- ◆ use of substitution, confusion & diffusion
- ◆ block size is 128 bits
- ◆ variable-length keys: key size is 128, 192 or 256 bits
 - ◆ variable number of rounds: 10, 12 or 14 rounds for keys of resp. 128, 192 or 256 bits
 - ◆ depending on key size, yields ciphers known as AES-128, AES-192, and AES-256



AES: Basic structure



AES: Basic structure (cont.)



DES: The Data Encryption Standard

- ◆ Symmetric block cipher
- ◆ Developed in 1976 by IBM for the US National Institute of Standards and Technology (NIST)
- ◆ Employs substitution & transposition, on top of each other, for 16 rounds
 - ◆ block size = 64 bits, key size = 56 bits
- ◆ Strengthening (since 56-bit security is not considered adequately strong)
 - ◆ double DES: $E(k_2, E(k_1, m))$, not effective!
 - ◆ triple DES: $E(k_3, E(k_2, E(k_1, m)))$, more effective
 - ◆ two keys, i.e., $k_1=k_3$, with E-D-E pattern, 80-bit security
 - ◆ three keys with E-E-E pattern, 112-bit security

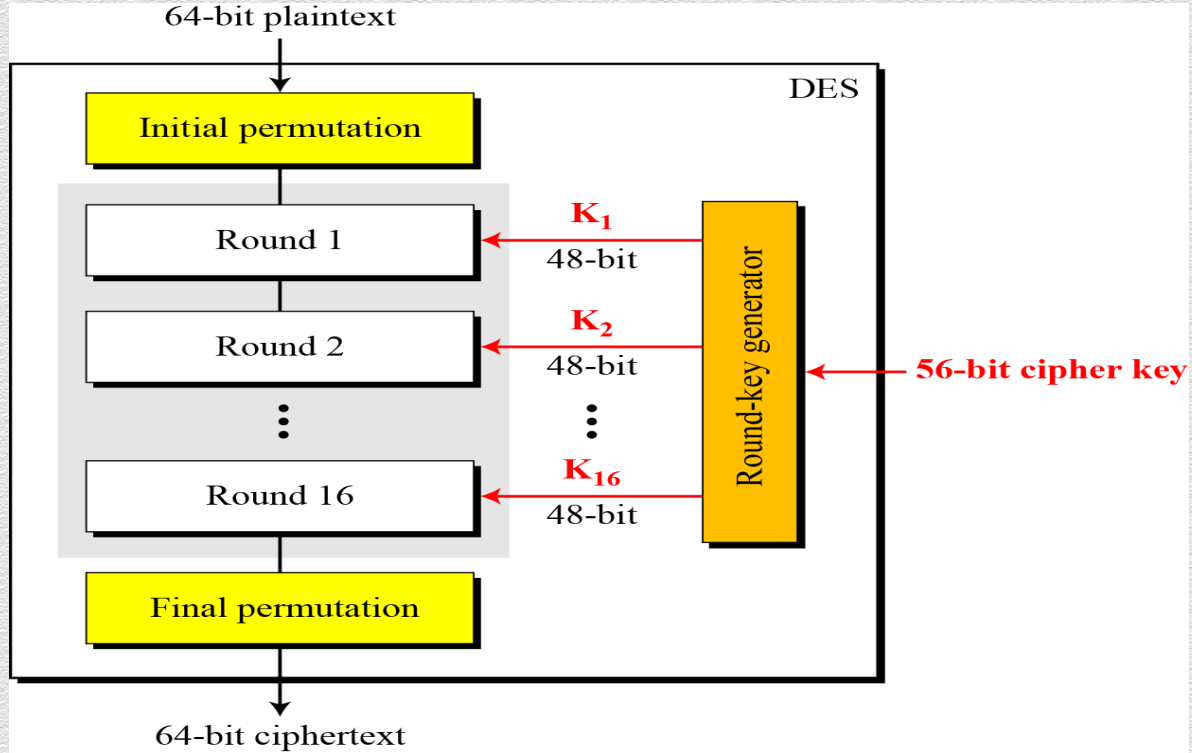
DES: Security strength

Form	Operation	Properties	Strength
DES	Encrypt with one key	56-bit key	Inadequate for high-security applications by today's computing capabilities
Double DES	Encrypt with first key; then encrypt result with second key	Two 56-bit keys	Only doubles strength of 56-bit key version
Two-key triple DES	Encrypt with first key, then encrypt (or decrypt) result with second key, then encrypt result with first key (E-D-E)	Two 56-bit keys	Gives strength equivalent to about 80-bit key (about 16 million times as strong as 56-bit version)
Three-key triple DES	Encrypt with first key, then encrypt or decrypt result with second key, then encrypt result with third key (E-E-E)	Three 56-bit keys	Gives strength equivalent to about 112-bit key about 72 quintillion ($72 \cdot 10^{15}$) times as strong as 56-bit version

DES: High-level view

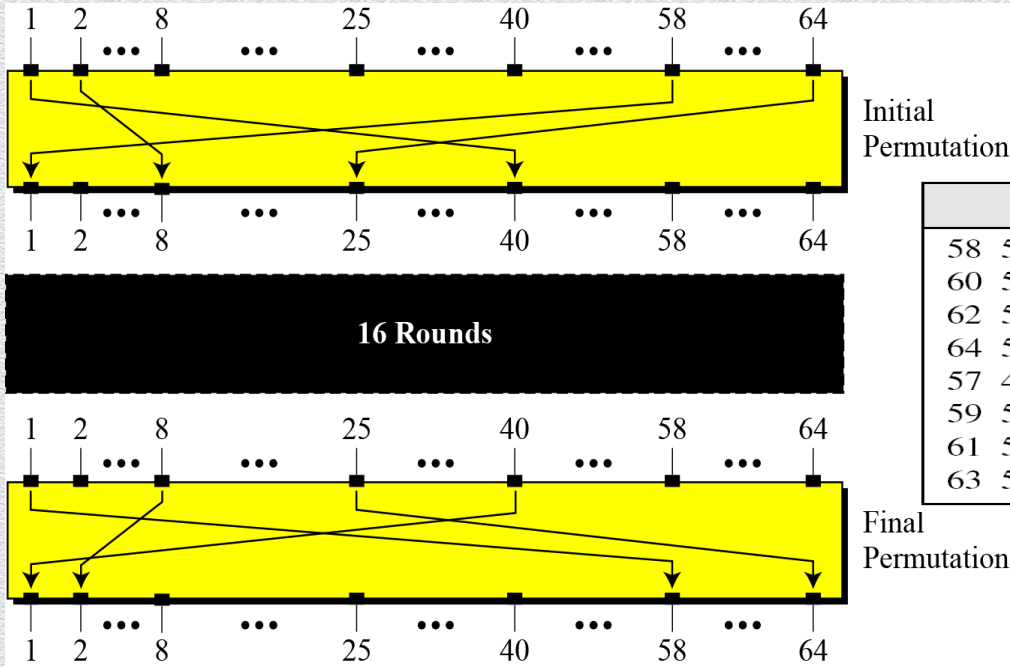


DES: Basic structure



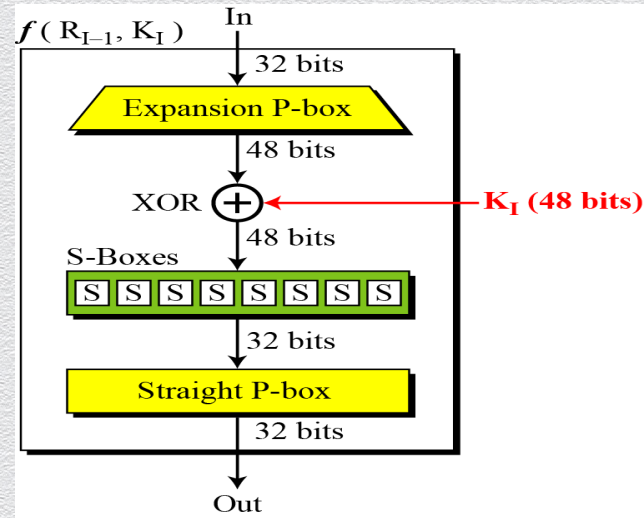
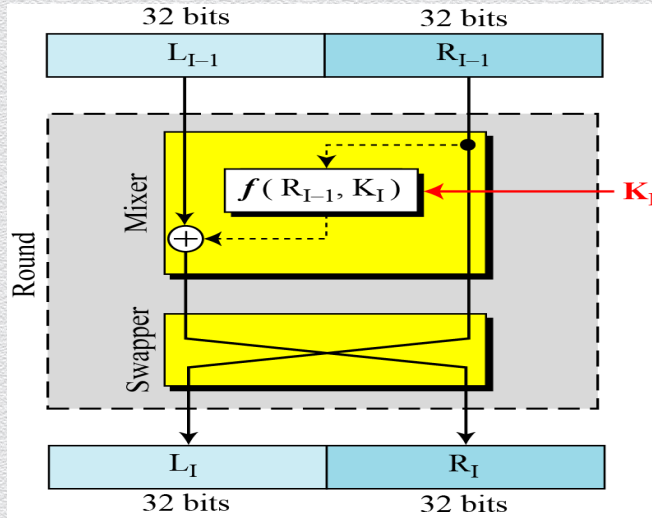
DES: Initial and final permutations

- ◆ Straight P-boxes that are inverses of each other w/out crypto significance



<i>Initial Permutation</i>								<i>Final Permutation</i>							
58	50	42	34	26	18	10	02	40	08	48	16	56	24	64	32
60	52	44	36	28	20	12	04	39	07	47	15	55	23	63	31
62	54	46	38	30	22	14	06	38	06	46	14	54	22	62	30
64	56	48	40	32	24	16	08	37	05	45	13	53	21	61	29
57	49	41	33	25	17	09	01	36	04	44	12	52	20	60	28
59	51	43	35	27	19	11	03	35	03	43	11	51	19	59	27
61	53	45	37	29	21	13	05	34	02	42	10	50	18	58	26
63	55	47	39	31	23	15	07	33	01	41	09	49	17	57	25

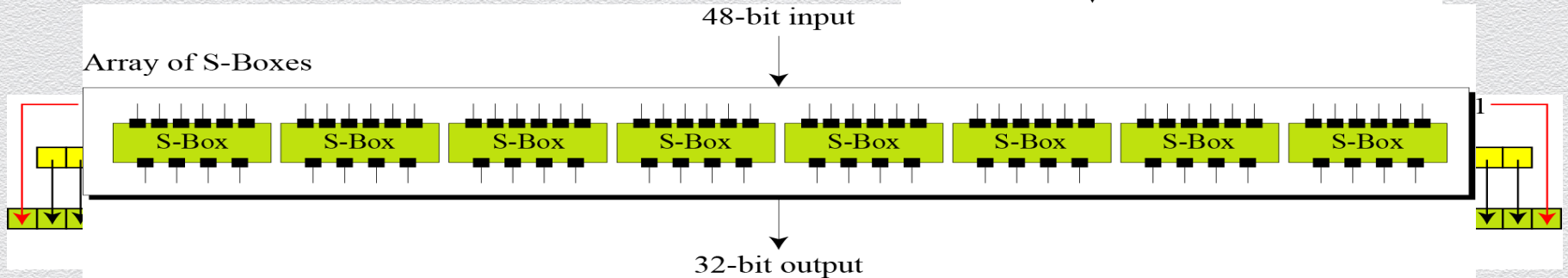
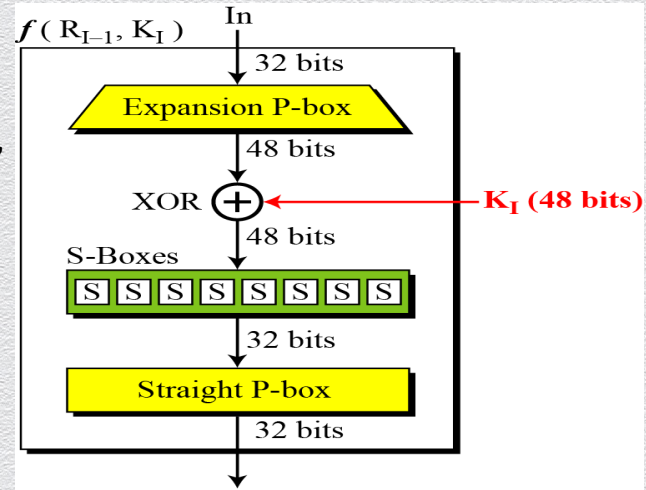
DES: Round via Feistel network



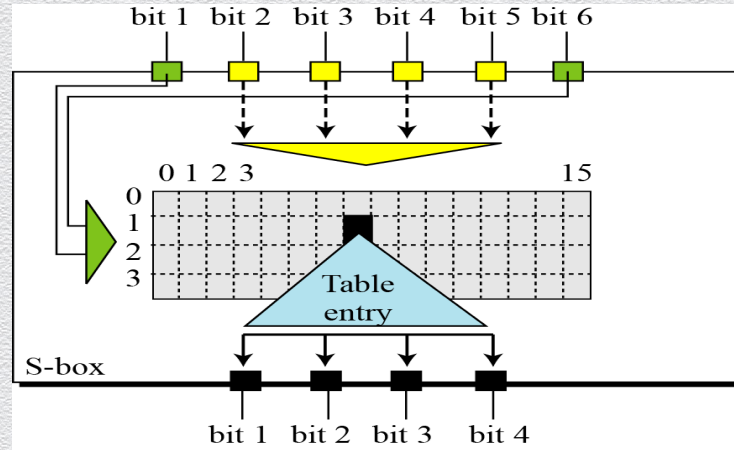
- ◆ DES uses 16 rounds, each applying a Feistel cipher
 - ◆ $L(i) = R(i-1)$
 - ◆ $R(i) = L(i-1) \text{ XOR } f(K(i), R(i-1))$, where f applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output

DES: Low-level view

- ◆ Expansion box
 - ◆ since R_{I-1} is a 32-bit input & K_I is a 48-bit key, we first need to expand R_{I-1} to 48 bits
- ◆ S-box
 - ◆ where real mixing (confusion) occurs
 - ◆ DES uses 8 6-to-4 bits S-boxes



DES: S-box in detail



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13