

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

CS1660: Intro to Computer Systems Security

Spring 2026

Lecture 5: Integrity I

Instructor: **Nikos Triandopoulos**

February 5, 2026



BROWN

Last class

- ◆ Cryptography
 - ◆ Symmetric-key encryption in practice
 - ◆ Computational security, pseudo-randomness
 - ◆ Stream & block ciphers, modes of operations for encryption, DES & AES
 - ◆ Introduction to modern cryptography

Today

- ◆ Cryptography
 - ◆ Symmetric-key encryption in practice
 - ◆ Computational security, pseudo-randomness
 - ◆ Stream & block ciphers, modes of operations for encryption, DES & AES
 - ◆ Introduction to modern cryptography
 - ◆ Integrity & reliable communication
 - ◆ Message authentication codes (MACs)

5.0 Introduction to modern cryptography

Recall: Approach in modern cryptography

Formal treatment

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

Systematic process

- ◆ A) **formal definitions**
- ◆ B) **precise assumptions**
- ◆ C) **provable security**

A) Formal definitions

abstract but rigorous description of security problem

- ◆ **computing setting**
 - ◆ involved parties, communication model, core functionality
- ◆ **underlying cryptographic scheme**
 - ◆ e.g., symmetric-key encryption scheme
- ◆ **desired properties**
 - ◆ 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

B) Precise assumptions

abstract but rigorous description of security problem

- ◆ **computing setting**
 - ◆ system set up, initial state, randomness, communication, timing
- ◆ **adversary**
 - ◆ threat model, capabilities, limitations
- ◆ **rules of the game**
 - ◆ key management, security of used tools, hardness of computational problems

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

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

The 3 pillars in Cryptography

- ◆ We have already been familiar with all three!
 - ◆ **A) formal definitions**
 - ◆ **B) precise assumptions**
 - ◆ **C) provable security**
- ◆ Let's remind ourselves...

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

Equivalent definitions of perfect security

1) a posteriori = a priori

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

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

2) C is independent of M

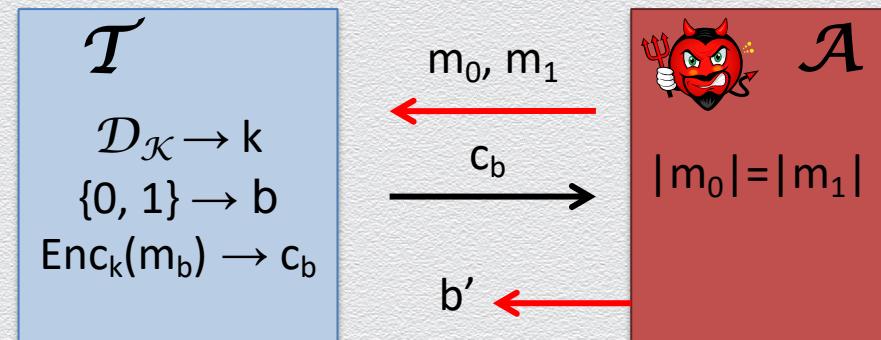
For every $m, m' \in \mathcal{M}$ and $c \in C$, it holds that

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

3) indistinguishability

For every \mathcal{A} , it holds that

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



OTP is perfectly secure (using Definition 2)

For all n -bit long messages m_1 and m_2 and ciphertexts c , it holds that

$$\Pr[E_K(m_1) = c] = \Pr[E_K(m_2) = c],$$

where probabilities are measured over the possible keys chosen by Gen.

Proof

- ◆ events “ $\text{Enc}_K(m_1) = c$ ”, “ $m_1 \oplus K = c$ ” and “ $K = m_1 \oplus c$ ” are equal-probable
- ◆ K is chosen at random, irrespectively of m_1 and m_2 , with probability 2^{-n}
- ◆ thus, the ciphertext does not reveal anything about the plaintext

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^{-128})
 - ◆ 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

PPT

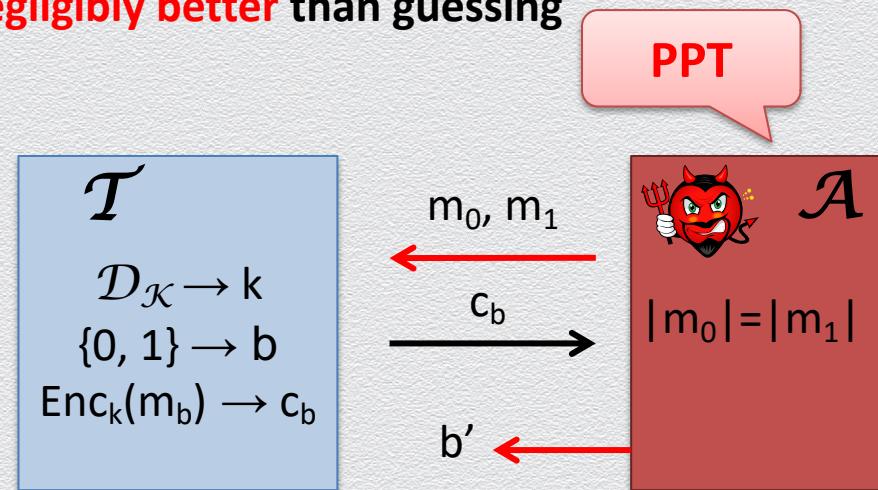
3) indistinguishability

For every \mathcal{A} , it holds that

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

PPT

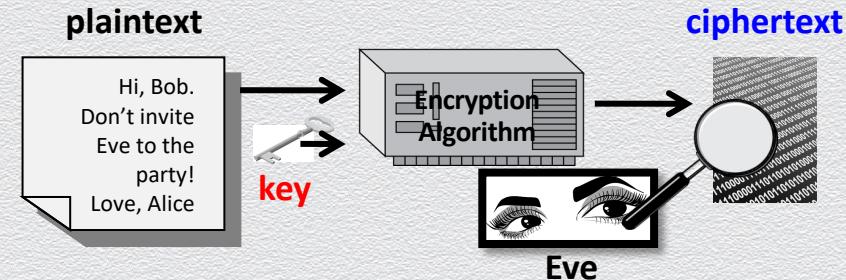
negl



Main security properties against eavesdropping

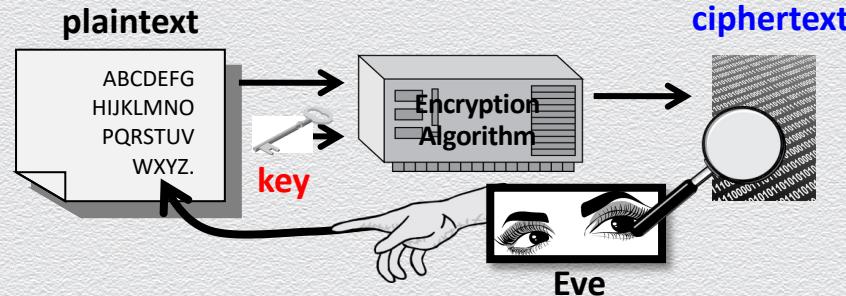
“plain” security

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



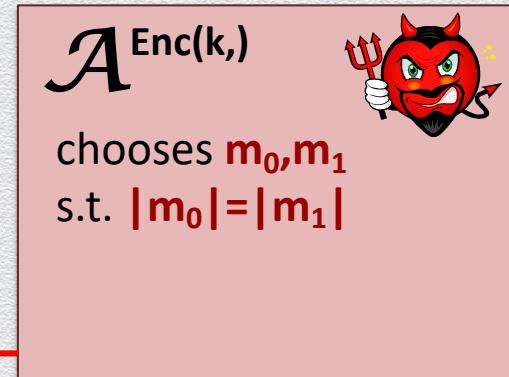
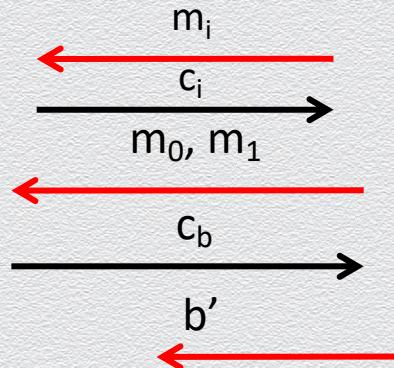
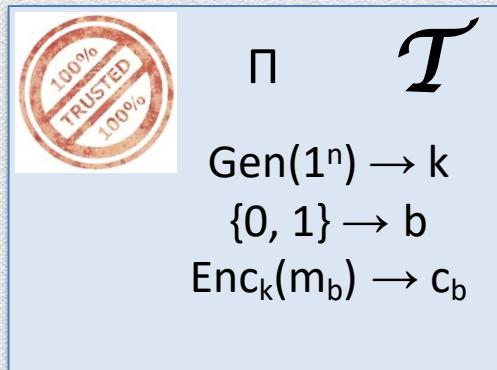
“advanced” security

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



Game-based computational 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 $0.5 + \varepsilon(n)$, 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 randomized encryption – can you see why?**
- ◆ EAV-security for multiple messages implies probabilistic encryption

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”

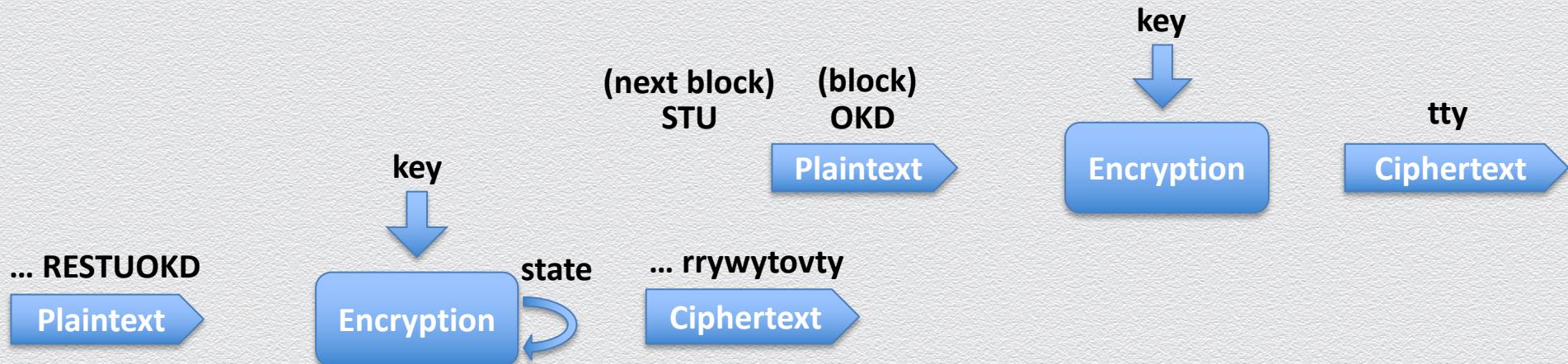
- ◆ the ciphertext appears to be “**pseudorandom**”
 - ◆ **it cannot be efficiently distinguished** from truly random

Tools for “OPT with pseudorandomness”

Stream cipher

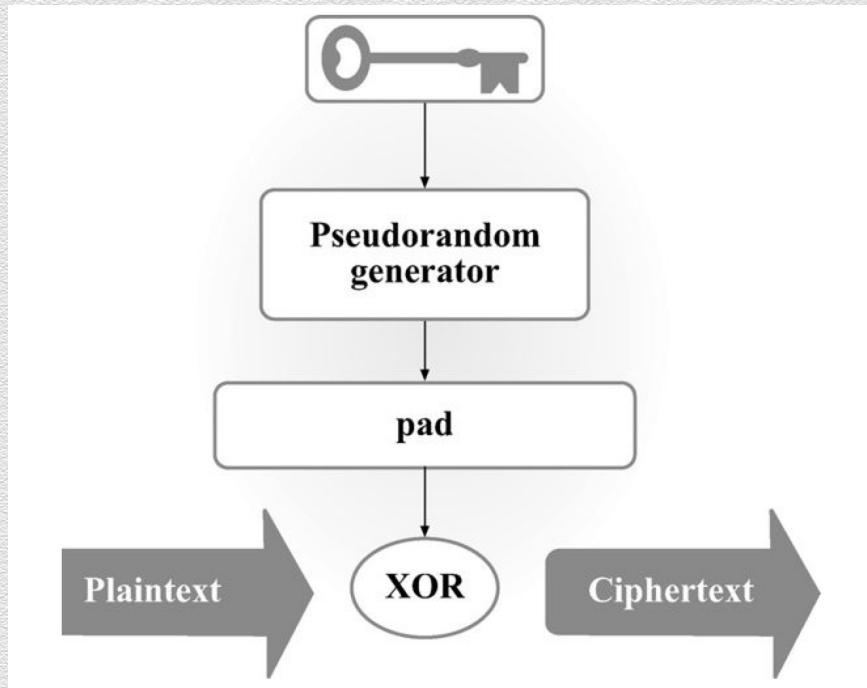
Uses a **short** key to encrypt **long** symbol streams into a **pseudorandom** ciphertext

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



Generic PRG-based symmetric encryption

- ◆ **Fixed-length** message encryption

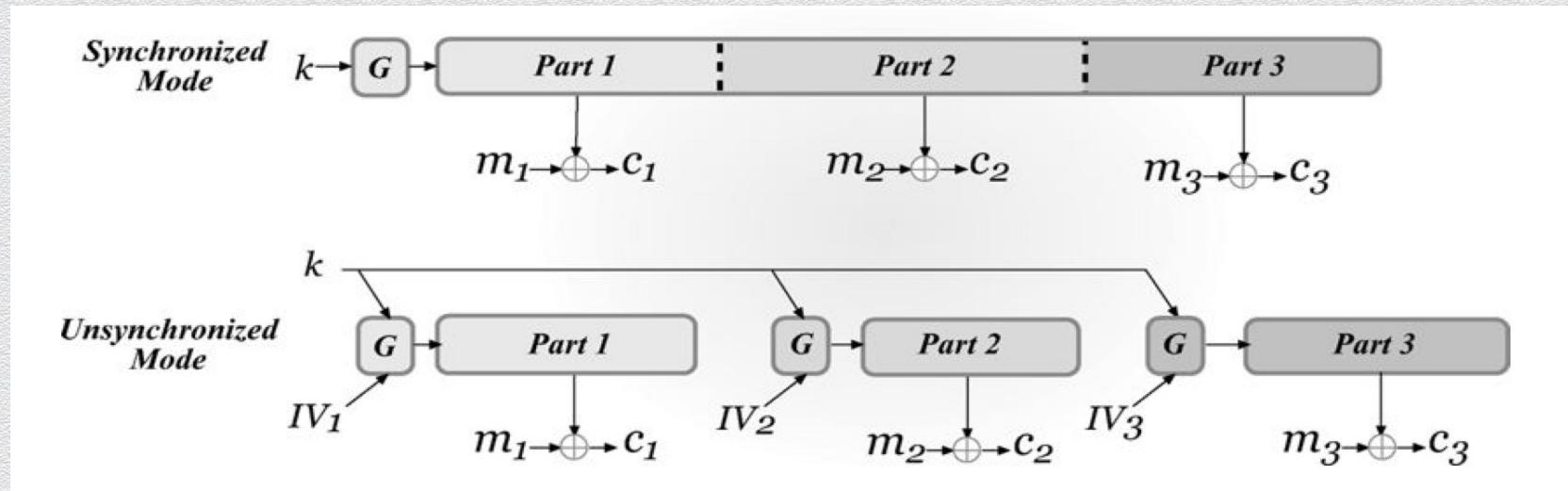


encryption scheme is plain-secure as long as the underlying PRG is secure

Stream ciphers: Modes of operations

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

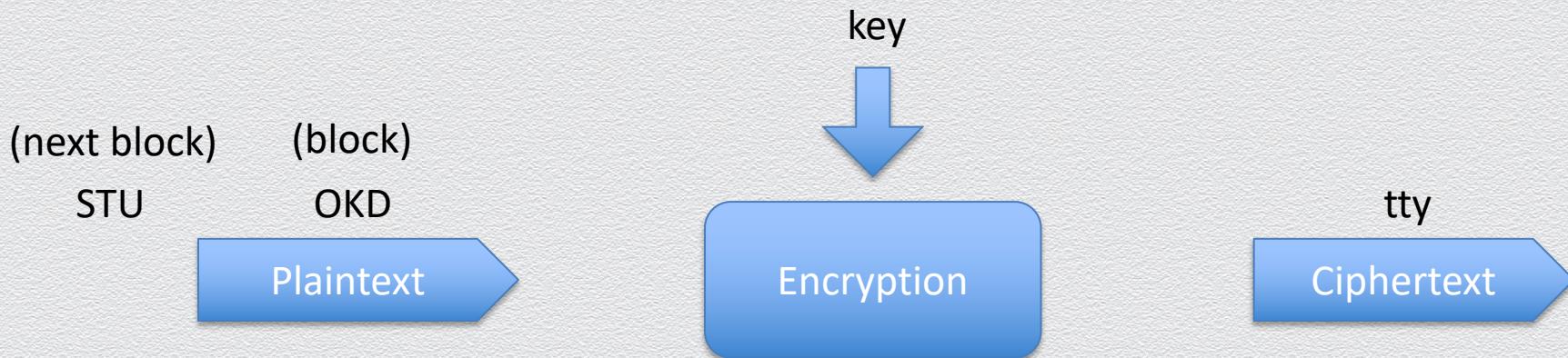
on-the-fly computation of new pseudorandom bits, no IV needed, plain-secure



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

5.1 Pseudorandom functions (or block ciphers)

Block ciphers



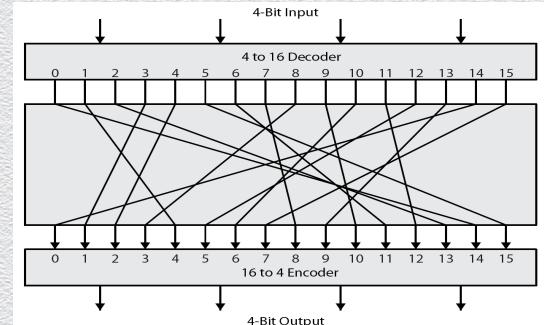
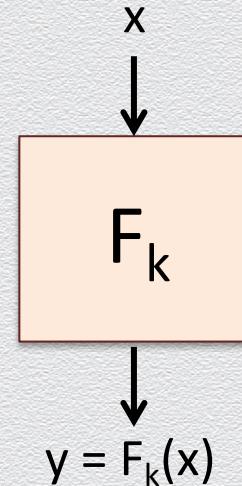
Realizing ideal block ciphers in practice

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

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

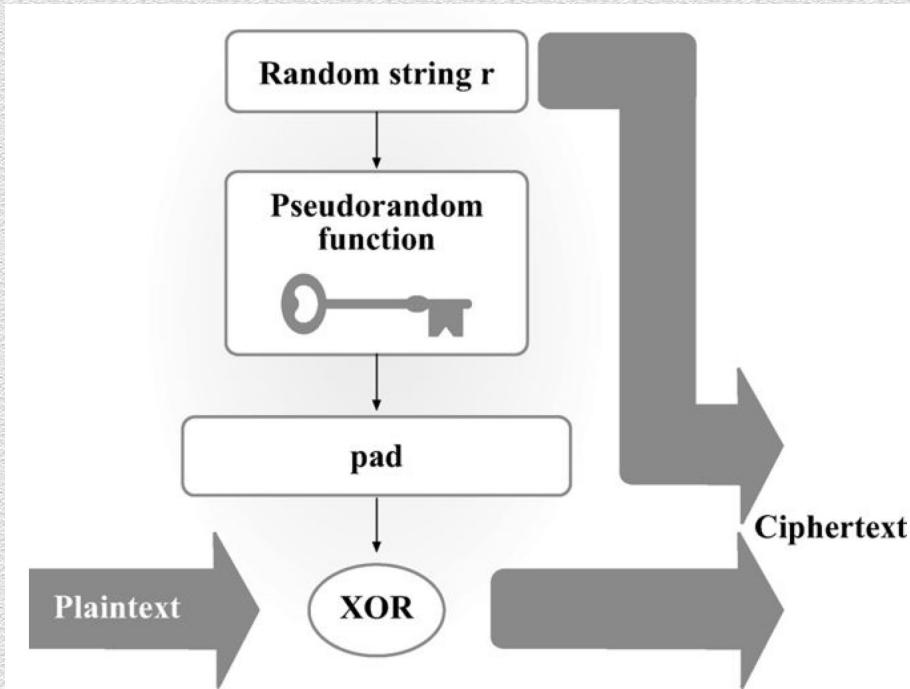
Instead, we use a keyed function $F_k : \{0,1\}^n \rightarrow \{0,1\}^n$

- ◆ 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

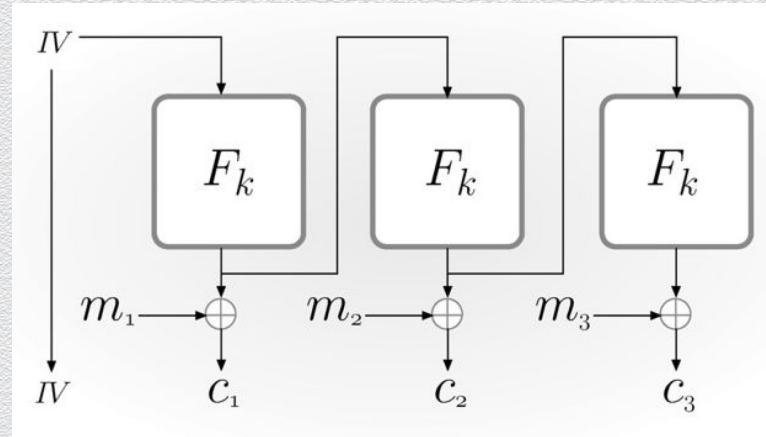
Generic PRF-based symmetric encryption (cont.)

- ◆ **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

5.2 Modes of operations (of block ciphers)

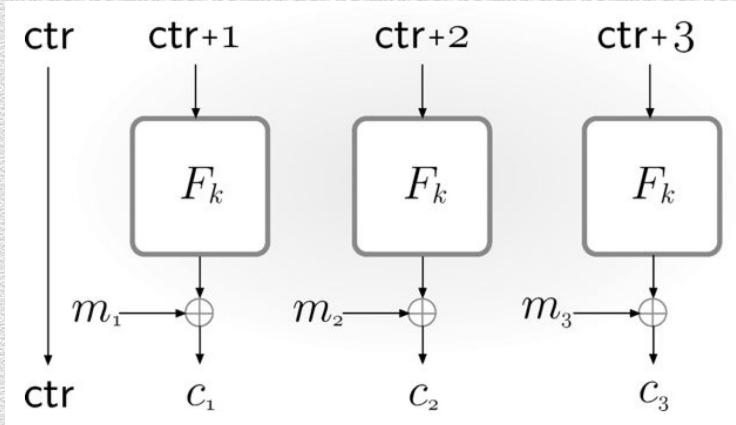
Block ciphers: Modes of operations (I)

- ◆ 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



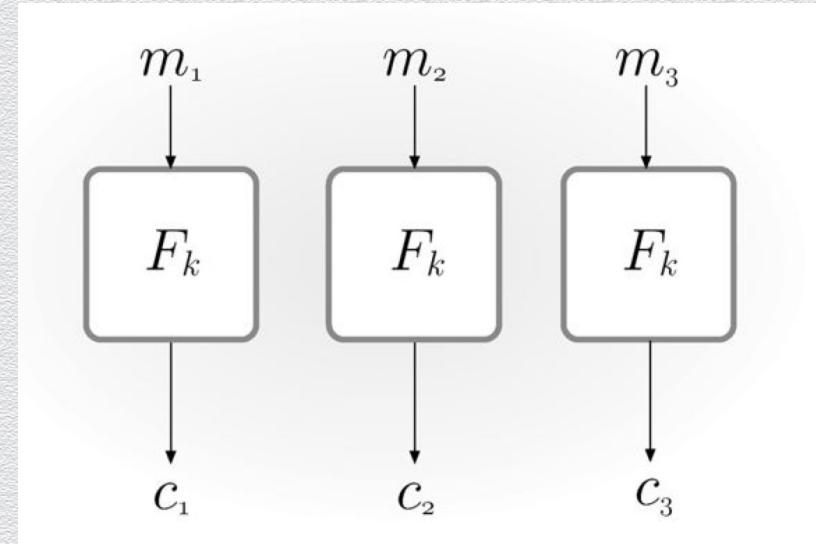
Block ciphers: Modes of operations (II)

- ◆ 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



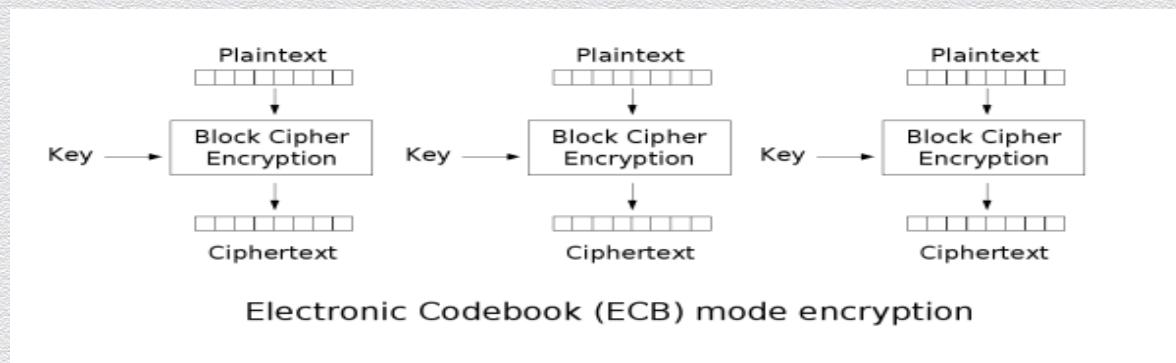
Block ciphers: Modes of operations (III)

- ◆ 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 $M[i] = \text{Dec}_k(C[i])$



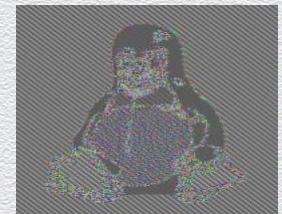
Strengths & weaknesses of ECB

Strengths

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

Weaknesses

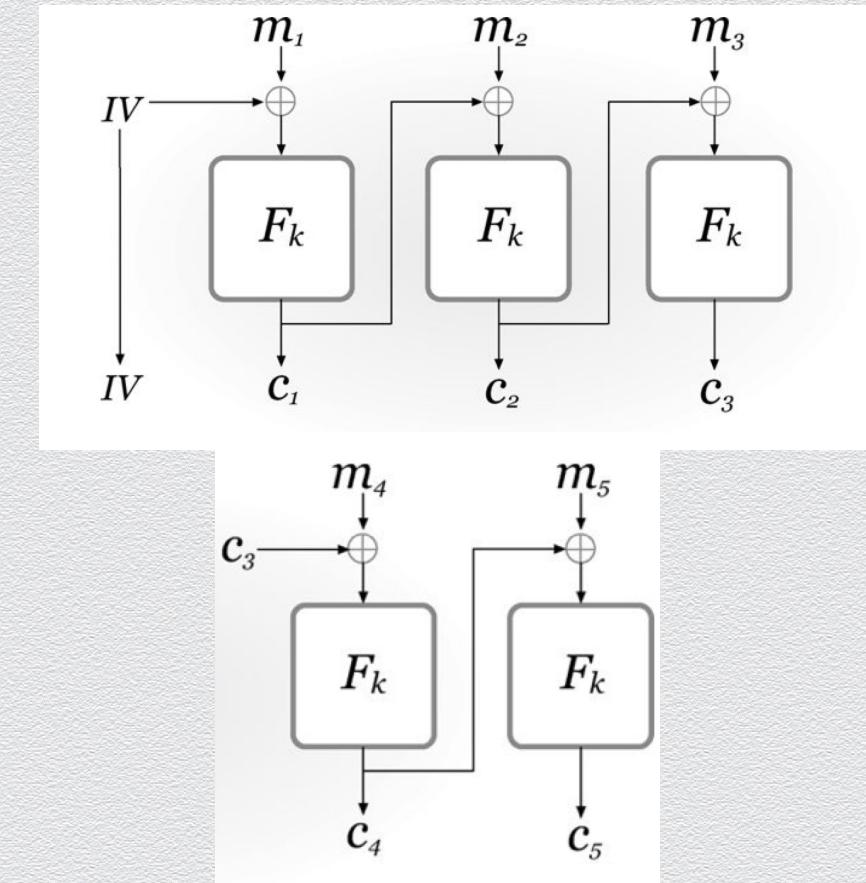
- ◆ poor security
- ◆ produces the same ciphertext on the same plaintext (under the same key)
- ◆ documents and images are not suitable for ECB encryption, since patterns in the plaintext are repeated in the ciphertext
- ◆ e.g.,



ECB

Block ciphers: Modes of operations (IV)

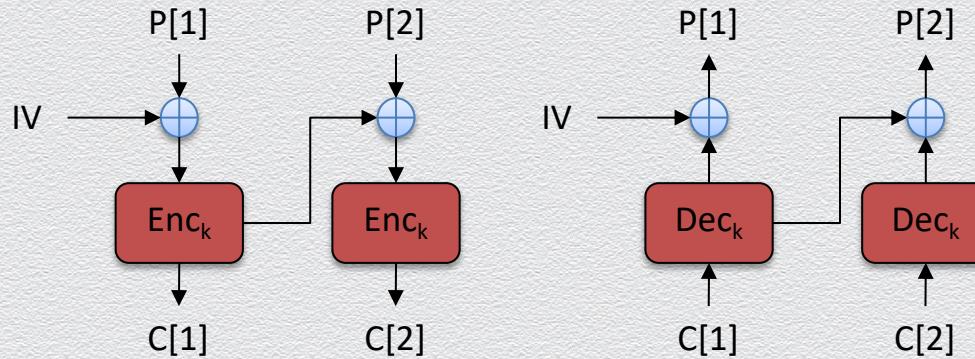
- ◆ CBC – cipher block chaining
 - ◆ CPA-secure if F_k a permutation
 - ◆ uniform IV
 - ◆ otherwise security breaks
- ◆ Chained CBC
 - ◆ use last block ciphertext of current message as IV of next message
 - ◆ saves bandwidth but not CPA-secure



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

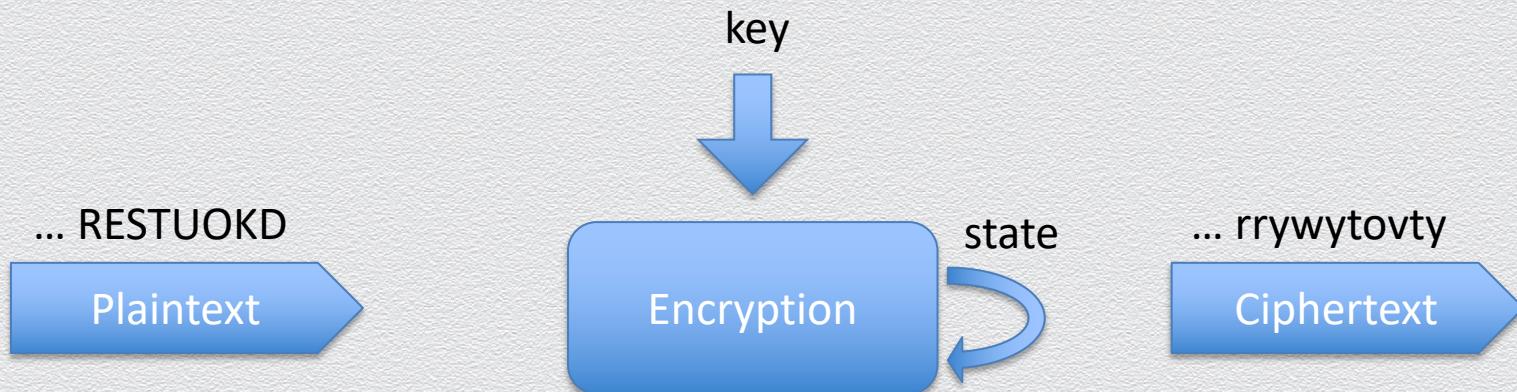


Notes on modes of operation

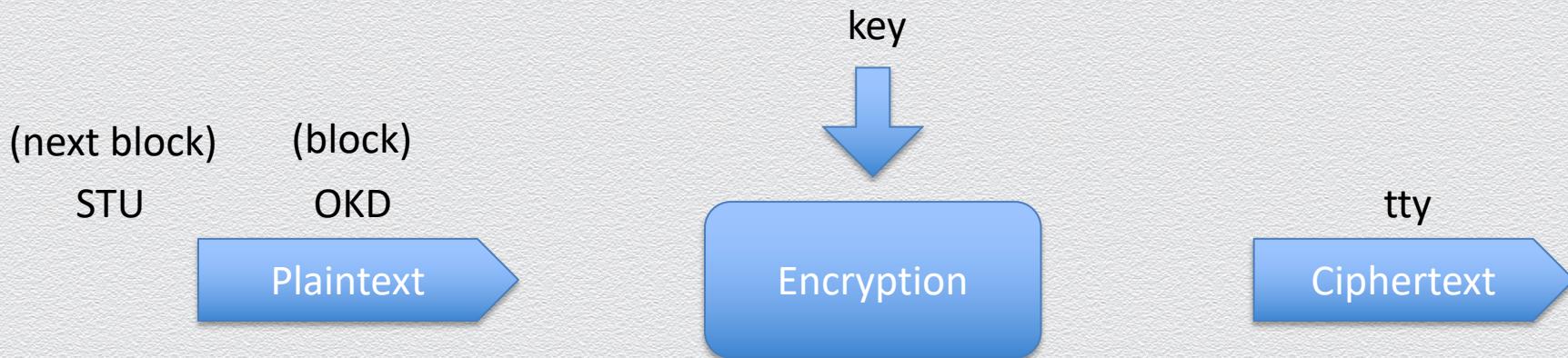
- ◆ block length matters
 - ◆ if small, IV or ctr can be “recycled”
- ◆ IV are often misused
 - ◆ e.g., reused or not selected uniformly at random
 - ◆ in this case, CBC is a better option than OFB/CTR

5.3 (Stream & block) Ciphers in practice

Recall: Stream ciphers



Recall: Block ciphers



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
00	0011	0100	1111	0001
01	1010	0110	0101	1011
10	1110	1101	0100	0010
11	0111	0000	1001	1100

(a)

	0	1	2	3
0	3	8	15	1
1	10	6	5	11
2	14	13	4	2
3	7	0	9	12

(b)

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

Brute-force attacks against stream/block ciphers

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

- ◆ **Due to confusion & diffusion**, for stream/block ciphers, by construction 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**

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

5.4 Block ciphers in practice: DES & AES

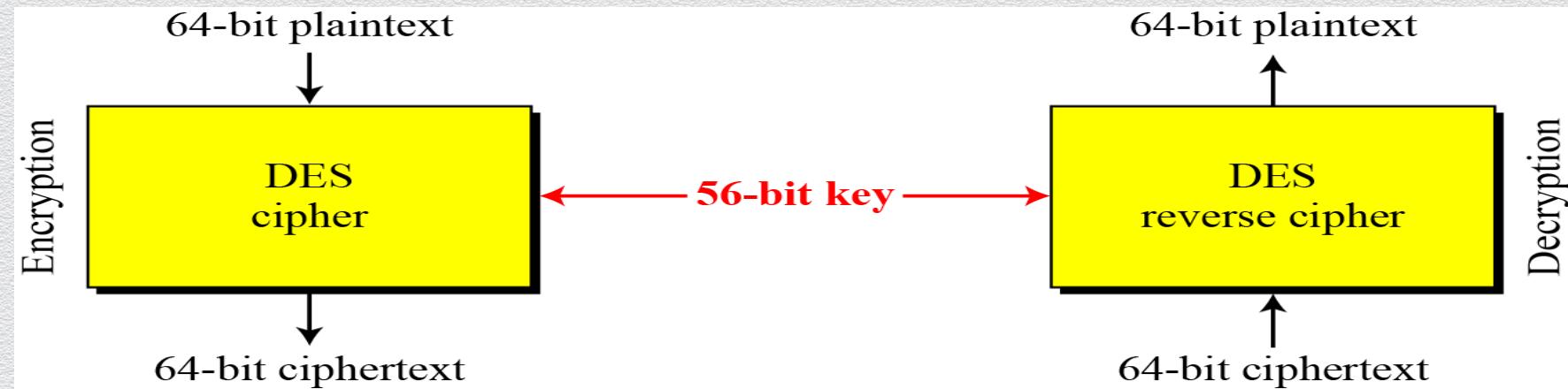
DES: The Data Encryption Standard

- ◆ 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

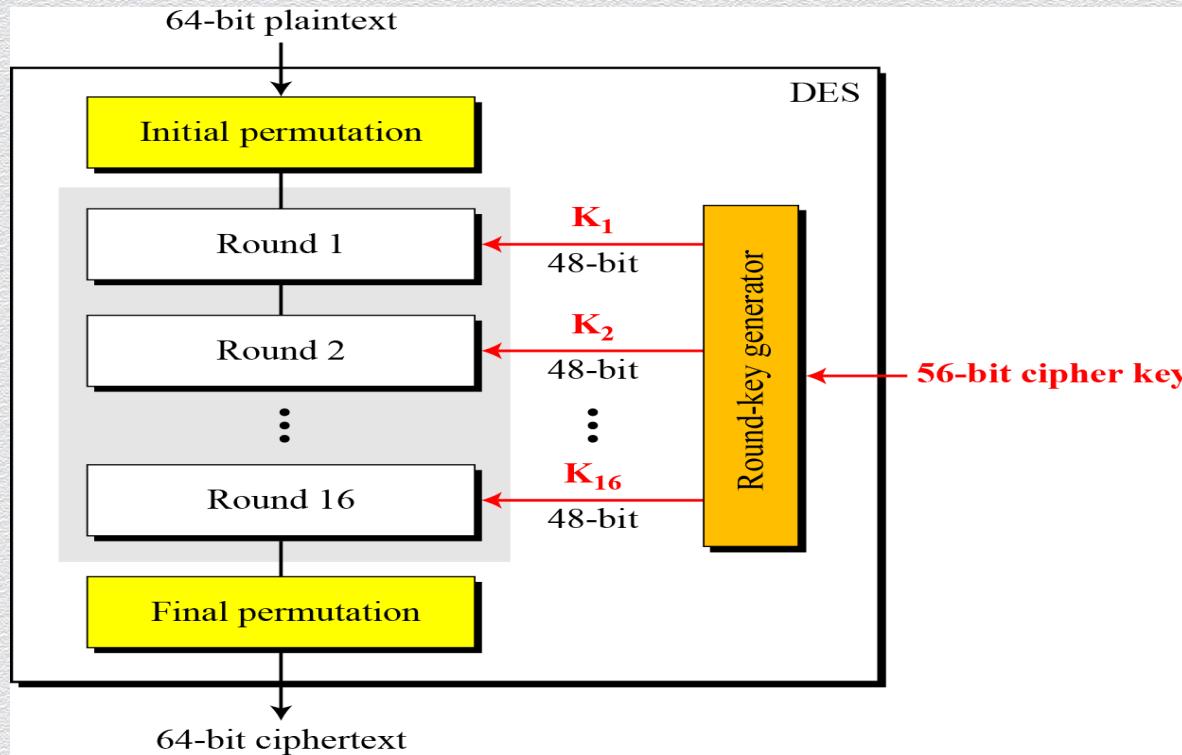
DES: Security strength

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 (72×10^{15}) times as strong as 56-bit version

DES: High-level view

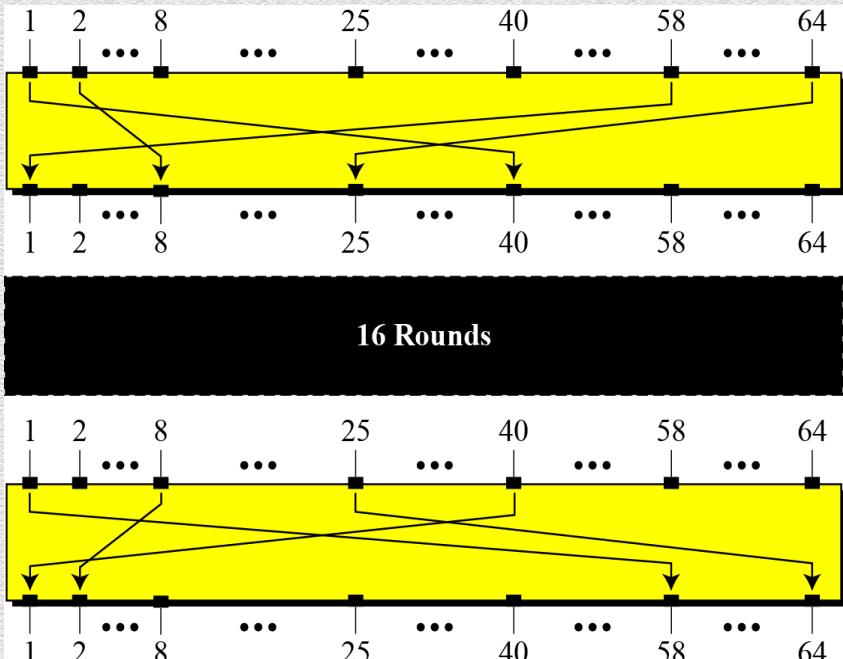


DES: Basic structure



DES: Initial and final permutations

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

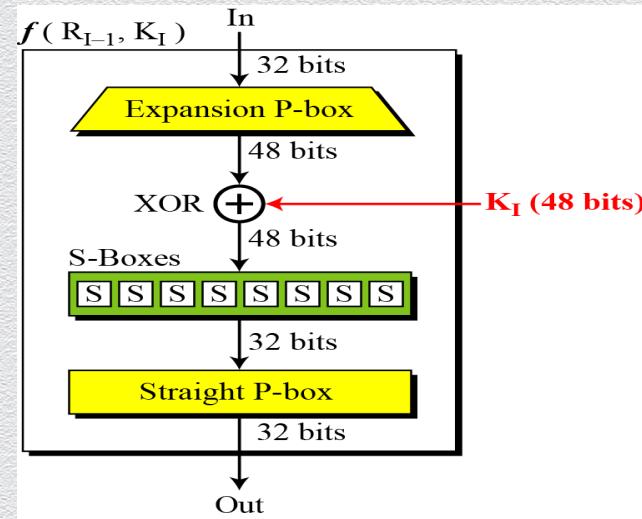
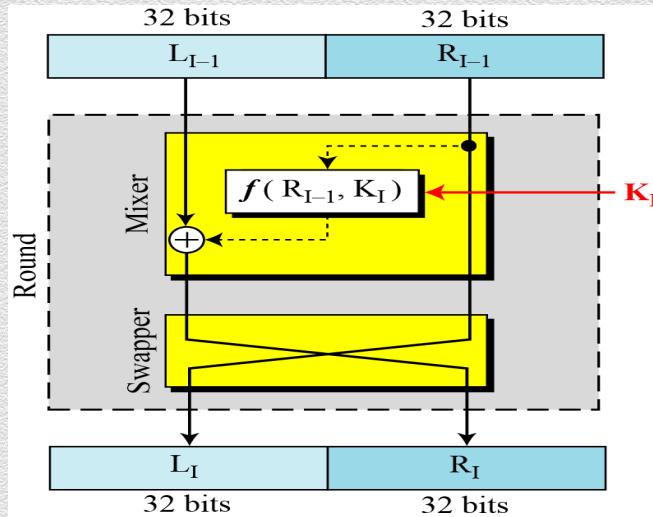


Initial
Permutation

Final
Permutation

<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

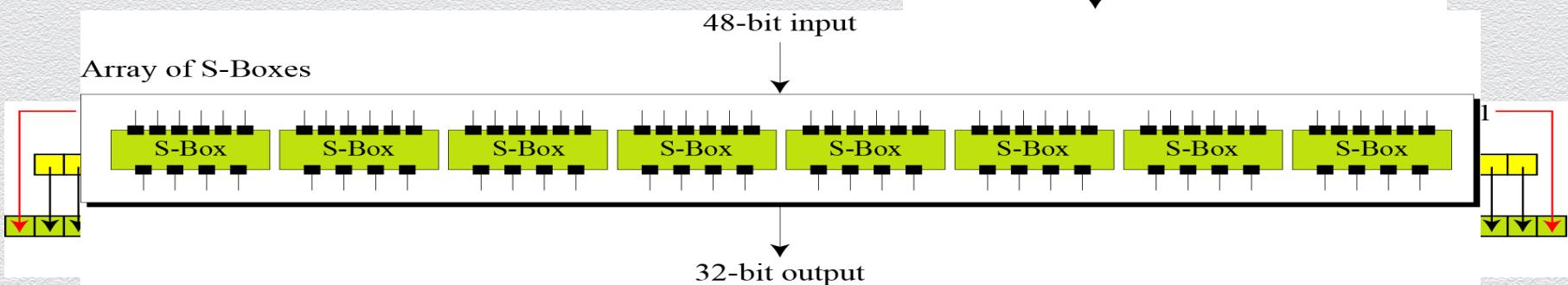
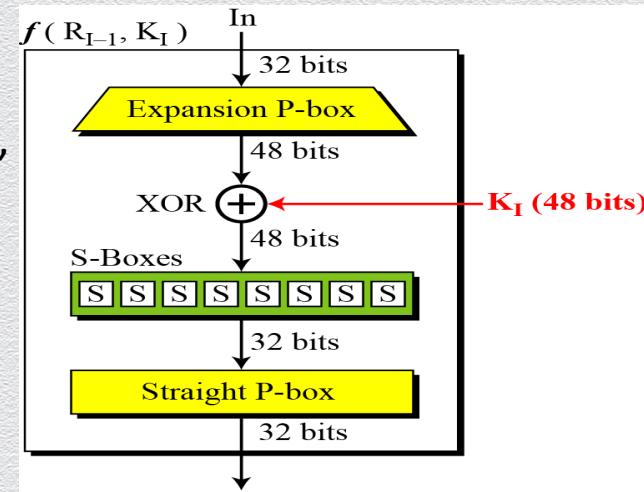
DES: Round via Feistel network



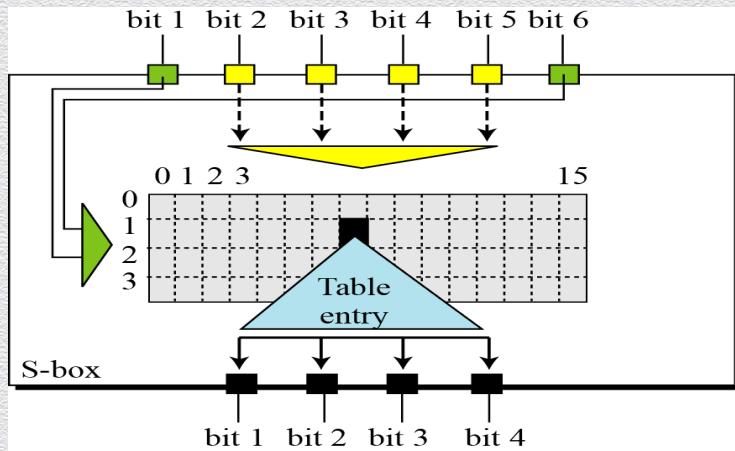
- ◆ DES uses 16 rounds, each applying a Feistel cipher
 - ◆ $L(i) = R(i-1)$
 - ◆ $R(i) = L(i-1) \text{ XOR } f(K(i), R(i-1))$,
where f applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output

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



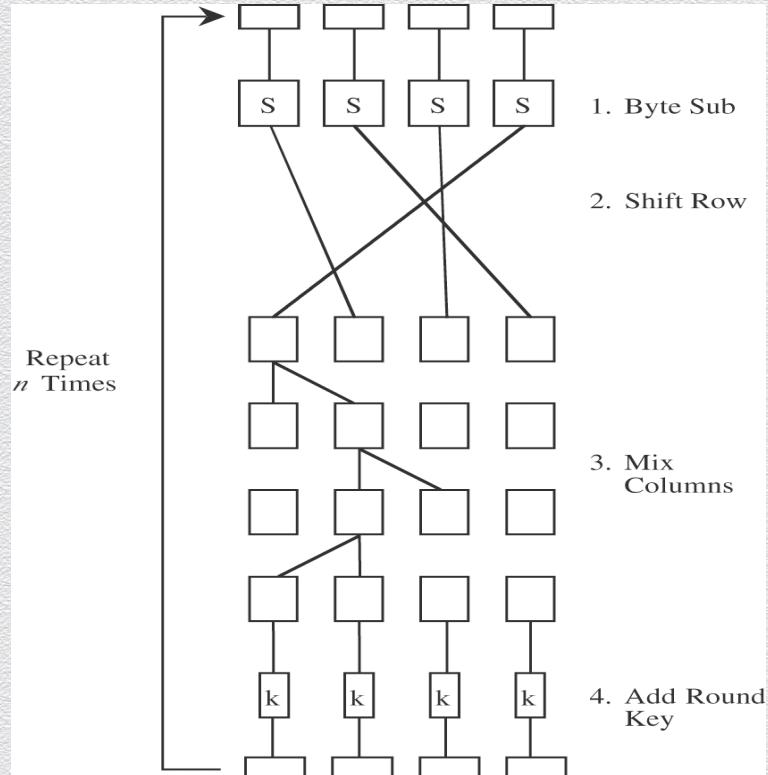
DES: S-box in detail



	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

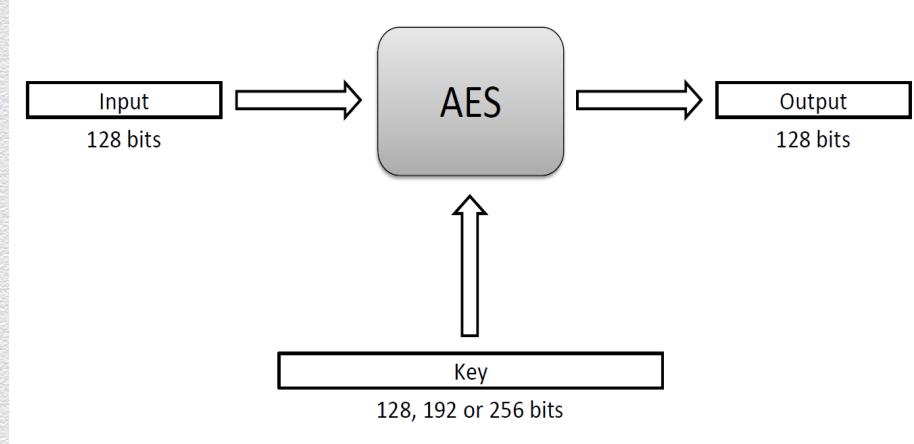
AES: Advanced Encryption System

- ◆ 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

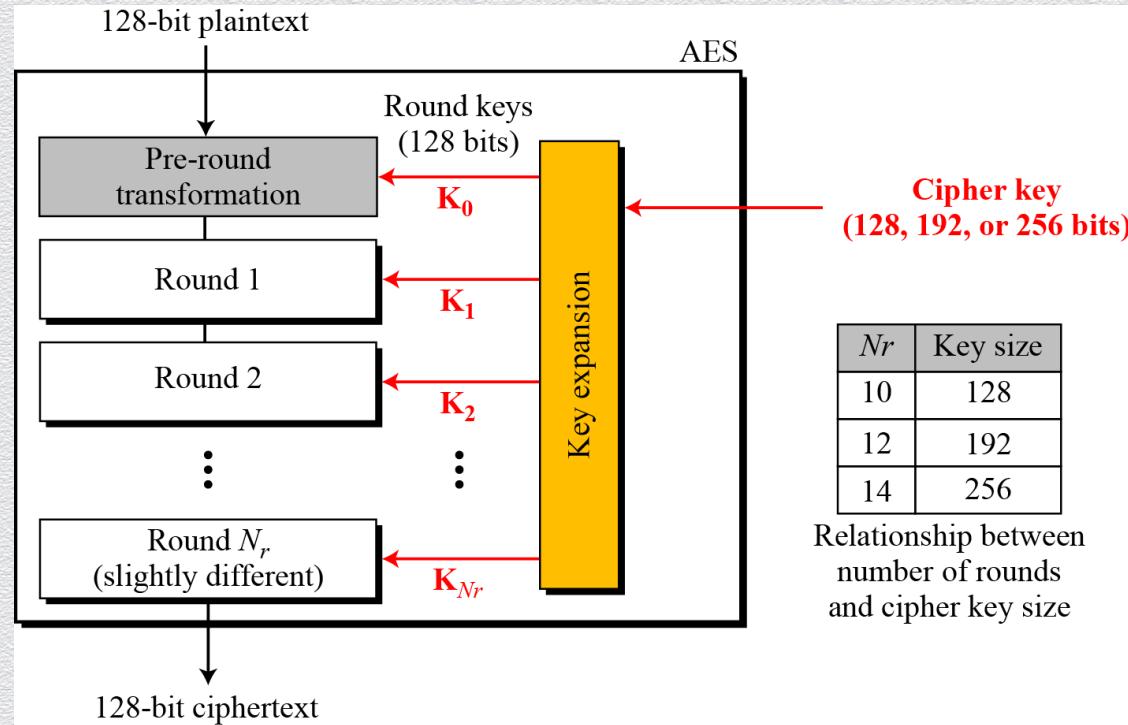


AES: Key design features

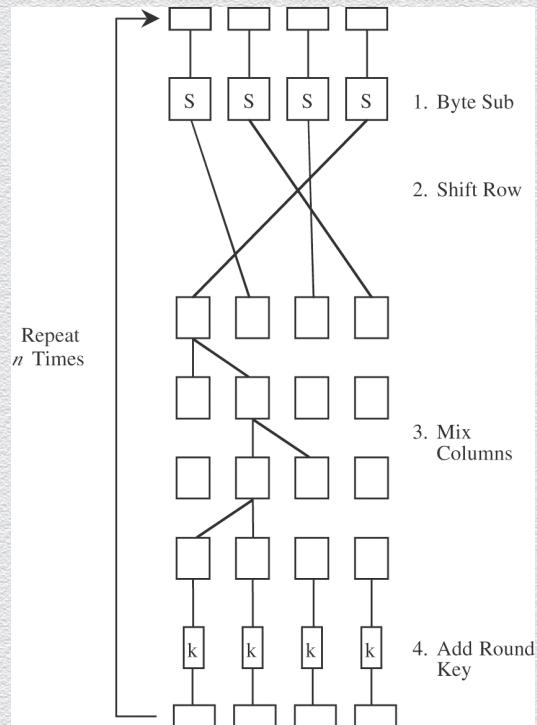
- ◆ use of substitution, confusion & diffusion
- ◆ block size is 128 bits
- ◆ variable-length keys: key size is 128, 192 or 256 bits
 - ◆ variable number of rounds: 10, 12 or 14 rounds for keys of resp. 128, 192 or 256 bits
 - ◆ depending on key size, yields ciphers known as AES-128, AES-192, and AES-256



AES: Basic structure



AES: Basic structure (cont.)



DES 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

5.5 Message authentication

Recall: Integrity

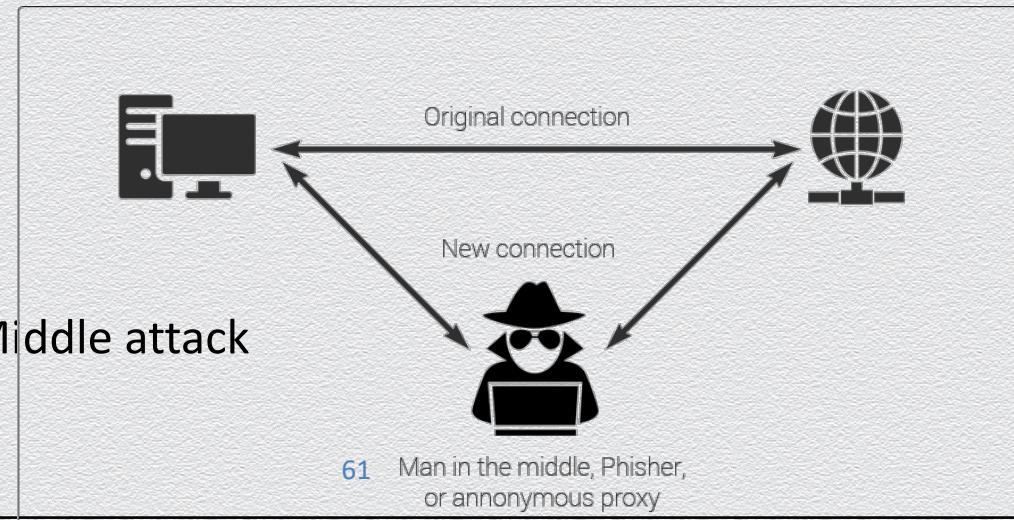
Fundamental security property

- ◆ **an asset is modified only by authorized parties**
- ◆ “I” in the CIA triad

*“computer security seeks to prevent **unauthorized** viewing (confidentiality) or **modification (integrity)** of **data** while preserving access (availability)”*

Alteration

- ◆ main threat against integrity of **in-transit** data
- ◆ e.g., Attacker-In-The-Middle attack



Security problems studied by modern cryptography

- ◆ Classical cryptography: **message encryption**
 - ◆ early crypto schemes tried to provide **secrecy / confidentiality**
- ◆ Modern cryptography: **wide variety** of security problems
 - ◆ today we need to study a large set of **security properties** beyond secrecy
- ◆ The sibling of message encryption: **message authentication**
 - ◆ another cornerstone of any secure system aiming to provide **authenticity & integrity**

Message authentication: Motivation

Information has **value**, but only when it is **correct**

- ◆ random, incorrect, inaccurate or maliciously altered data is **useless** or **harmful**
 - ◆ **message authentication = message integrity + authenticity**
 - ◆ while in transit (or at rest), no message should be **modified** by an outsider
 - ◆ no outsider can **impersonate** the stated message sender (or owner)
- ◆ it is often necessary / worth to protect critical / valuable data
 - ◆ **message encryption**
 - ◆ while in transit (or at rest), no message should be **leaked** to an outsider

Example 1

Secure electronic banking

- ◆ a bank receives an electronic request to transfer \$1,000 from Alice to Bob

Concerns

- ◆ who ordered the transfer, Alice or an attacker (e.g., Bob)?
- ◆ is the amount the intended one or was maliciously modified while in transit?
 - ◆ adversarial Vs. random message-transmission errors
 - ◆ standard error-correction is not sufficient to address this concern

Example 2

Web browser cookies

- ◆ a user is performing an online purchase at Amazon
- ◆ a “cookie” contains session-related info, as client-server HTTP traffic is stateless
 - ◆ stored at the client, included in messages sent to server
 - ◆ contains client-specific info that affects the transaction
 - ◆ e.g., the user’s shopping cart along with a discount due to a coupon

Concern

- ◆ was such state maliciously altered by the client (possibly harming the server)?

Integrity of communications / computations

Highly important

- ◆ any unprotected system cannot be assumed to be trustworthy w.r.t.
 - ◆ origin/source of information (due to impersonation attacks, phishing, etc.)
 - ◆ contents of information (due to man-in-the-middle attacks, email spam, etc.)
 - ◆ overall system functionality

Prevention Vs. detection

- ◆ unless system is “closed,” adversarial tampering with its integrity **cannot be avoided!**
- ◆ goal: identify system components that are not trustworthy
 - ◆ **detect tampering or prevent undetected tampering**
 - ◆ e.g., avoid “consuming” falsified information

Encryption does not imply authentication

A common misconception

“since ciphertext c hides message m , Mallory cannot meaningfully modify m via c ”

Why is this incorrect?

- ◆ all encryption schemes (seen so far) are based on one-time pad, i.e., masking via XOR
- ◆ consider flipping a single bit of ciphertext c ; what happens to plaintext m ?
 - ◆ such property of one-time pad does not contradict the secrecy definitions

Generally, secrecy and integrity are distinct properties

- ◆ encrypted traffic generally provides **no integrity** guarantees

5.6 Message authentication codes (MACs)

Problem setting: Reliable communication

Two parties wish to communicate over a channel

- ◆ Alice (sender/source) wants to send a message m to Bob (recipient/destination)

Underlying channel is unprotected

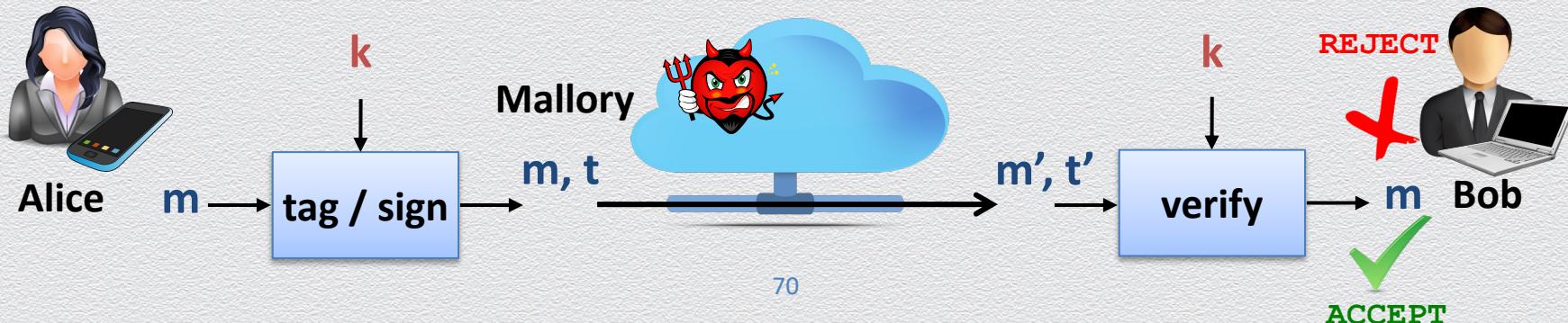
- ◆ Mallory (attacker/adversary) can manipulate any sent messages
- ◆ e.g., message transmission via a compromised router



Solution concept: Symmetric-key message authentication

Main idea

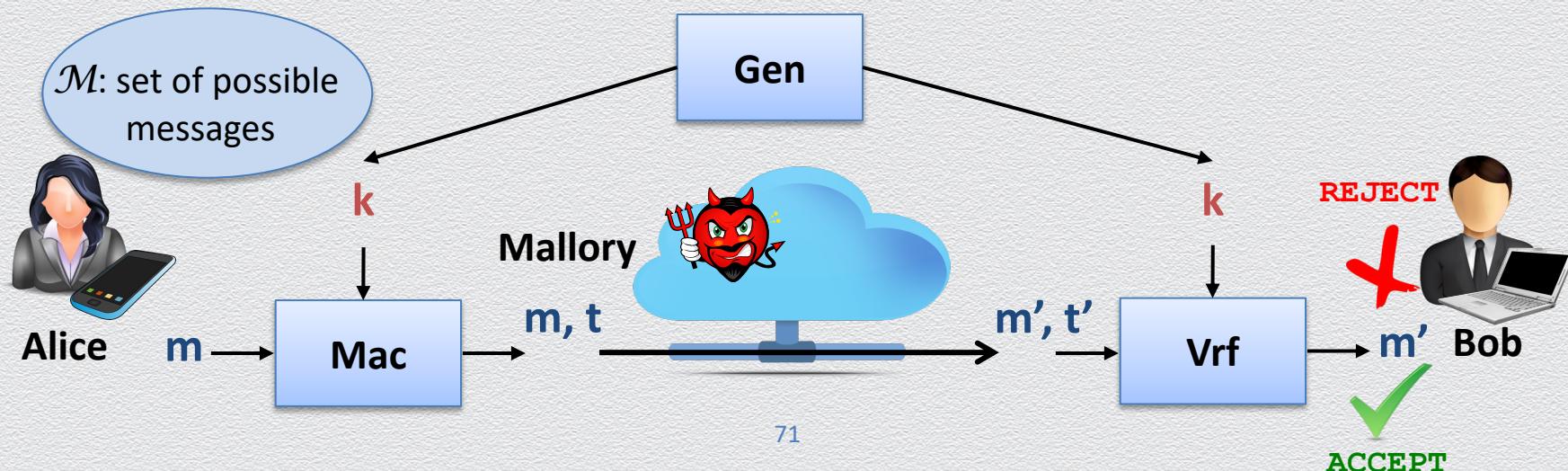
- ◆ secretly annotate or “sign” message so that it is **unforgeable** while in transit
 - ◆ Alice **tags** her message m with **tag t** , which is sent **along** with **plaintext m**
 - ◆ Bob **verifies** authenticity of received message using tag t
 - ◆ Mallory can manipulate m, t but “**cannot forge**” a fake verifiable pair m', t'
 - ◆ Alice and Bob share a **secret key k** that is used for both operations



Security tool: Message Authentication Code

Abstract cryptographic primitive, a.k.a. **MAC**, defined by

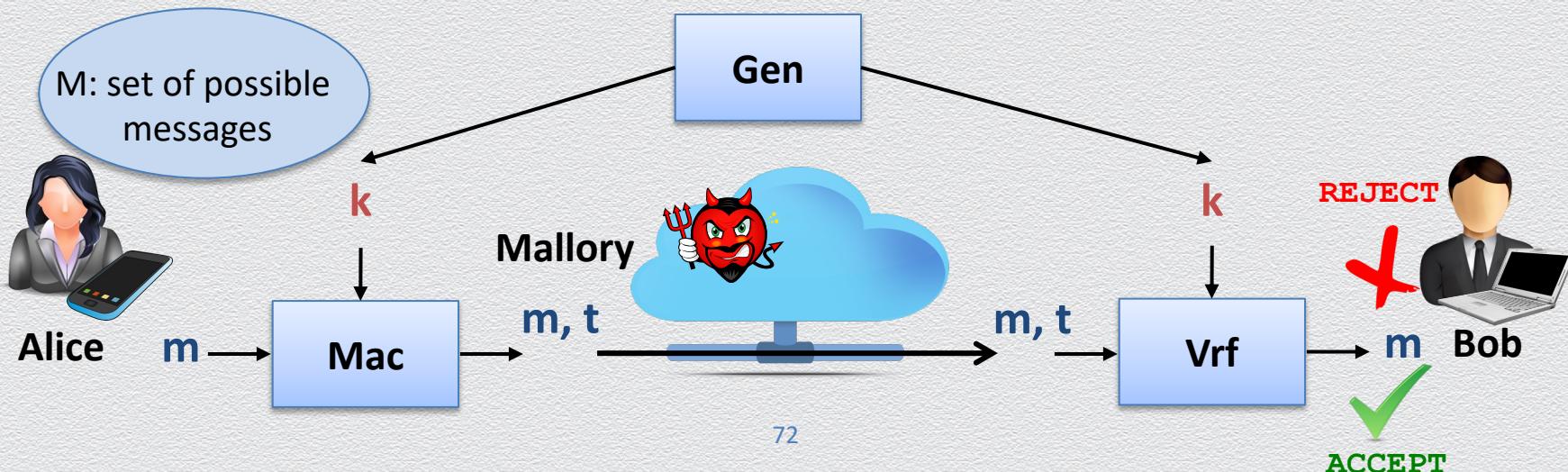
- ◆ a **message space \mathcal{M}** ; and
- ◆ a triplet of algorithms **(Gen, Mac, Vrf)**
 - ◆ Gen, Mac are probabilistic algorithms, whereas Vrf is deterministic
 - ◆ Gen outputs a uniformly random key k (from some key space \mathcal{K})



Desired properties for MACs

By design, any MAC should satisfy the following

- ◆ **efficiency:** key generation & message transformations “are fast”
- ◆ **correctness:** for all m and k , it holds that $Vrf_k(m, Mac_k(m)) = \text{ACCEPT}$
- ◆ **security:** one “cannot forge” a fake verifiable pair m', t'



Main application areas

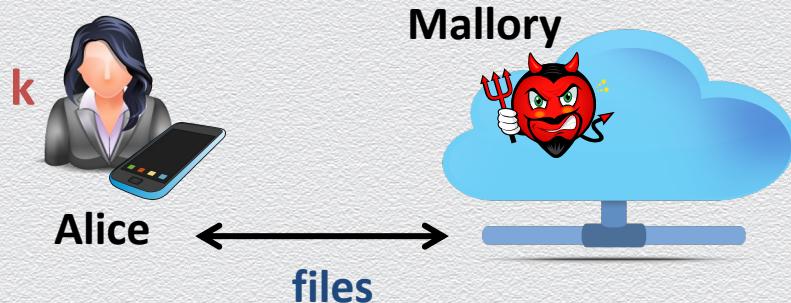
Secure communication

- ◆ **verify authenticity of messages** sent among parties
- ◆ assumption
 - ◆ Alice and Bob **securely generate, distribute and store shared key k**
 - ◆ attacker does not learn key k



Secure storage

- ◆ **verify authenticity of files** outsourced to the cloud
- ◆ assumption
 - ◆ Alice **securely generates and stores key k**
 - ◆ attacker does not learn key k



Conventions

Random key selection

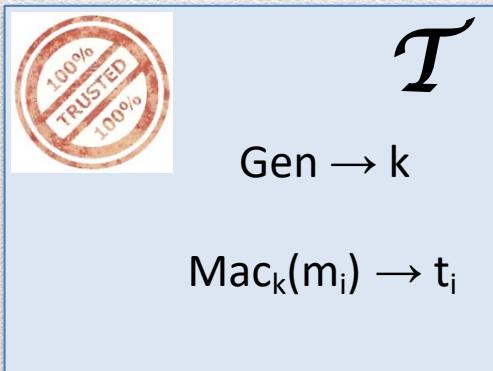
- ◆ typically, Gen selects key k **uniformly at random** from the key space \mathcal{K}

Canonical verification

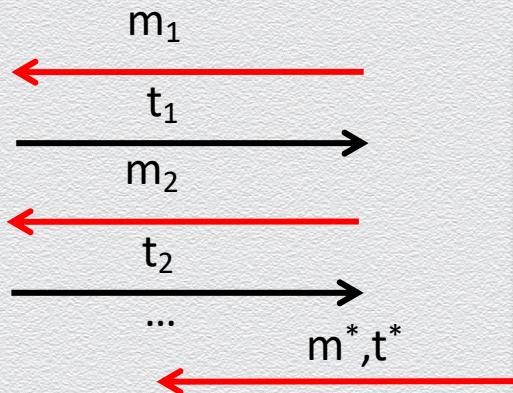
- ◆ when Mac is deterministic, Vrf typically amounts to re-computing the tag t
 - ◆ $Vrf_k(m, t)$: 1. $t' := Mac_k(m)$ 2. if $t = t'$, output ACCEPT else output REJECT
- ◆ but conceptually the following operations are distinct
 - ◆ authenticating m (i.e., running Mac) Vs. verifying authenticity of m (i.e., running Vrf)

MAC security

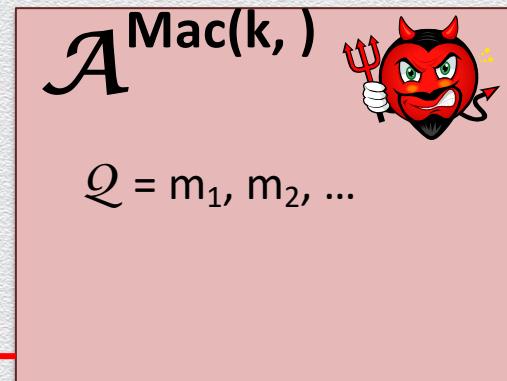
MAC scheme
(Gen, Mac, Vrf)



Attacker **wins** the game if



1. $Vrf_k(m^*, t^*) = \text{ACCEPT}$ &
2. m^* not in \mathcal{Q}



The MAC scheme is **secure** if any PPT \mathcal{A} wins the game only negligibly often.