https://brown-csci1660.github.io

CS1660: Intro to Computer Systems Security
Spring 2026

Lecture 5: Integrity

Instructor: **Nikos Triandopoulos**

February 5, 2026

BROWN

Last class

- Cryptography
  - 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
  - Reliable communication
    - Message authentication codes (MACs)
    - Authenticated encryption
  - Public-key encryption and digital signatures (introduction)

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

$$\textbf{Pr[M = m | C = c] = Pr[M = m]}$$

2) C is independent of M

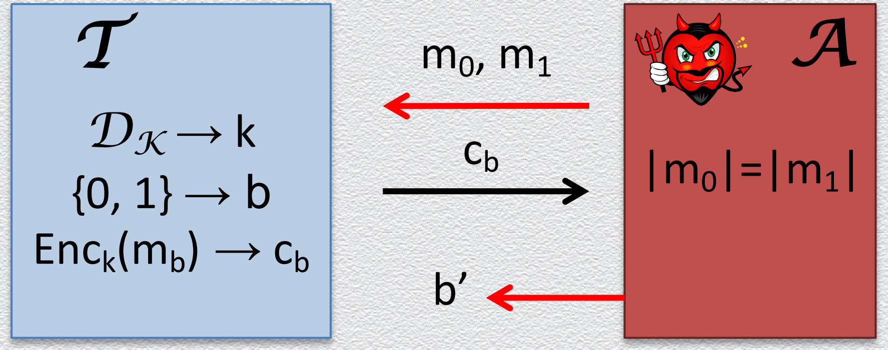
For every m, m' ∈ $\mathcal{M}$ and c ∈ $\mathcal{C}$, it holds that

$$\textbf{Pr[} Enc_K(m) = c \textbf{] = Pr[} Enc_K(m') = c \textbf{]}$$

3) indistinguishability

For every $\mathcal{A}$, it holds that

$$\textbf{Pr[b' = b] = 1/2}$$



OTP is perfectly secure (using Definition 2)

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

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

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

Proof

- events "$Enc_K(m_1) = c$", "$m_1 \oplus K = c$" 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, $Enc_K(M)$ are independent
  - absolutely **no information is leaked** about the plaintext
  - to adversaries that **unlimited computational power**
- **computational** security: for all **practical** purposes, M, $Enc_K(M)$ 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 $\mathbf{m_0, m_1}$ are chosen by a **PPT adversary**
- require that no **PPT adversary** can distinguish $\mathbf{Enc_k(m_0)}$ from $\mathbf{Enc_k(m_1)}$

non-negligibly better than guessing

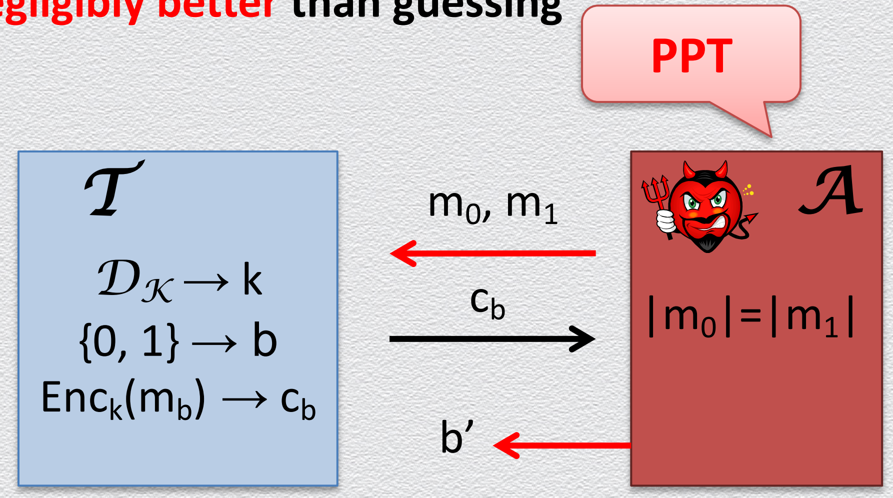
3) indistinguishability

For every $\mathcal{A}$, it holds that

PPT

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

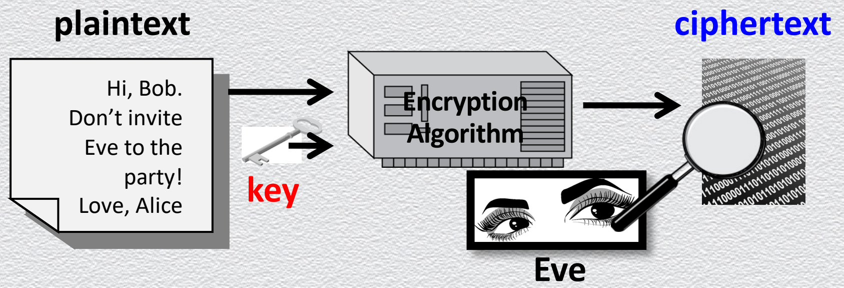
negl



Main security properties against eavesdropping

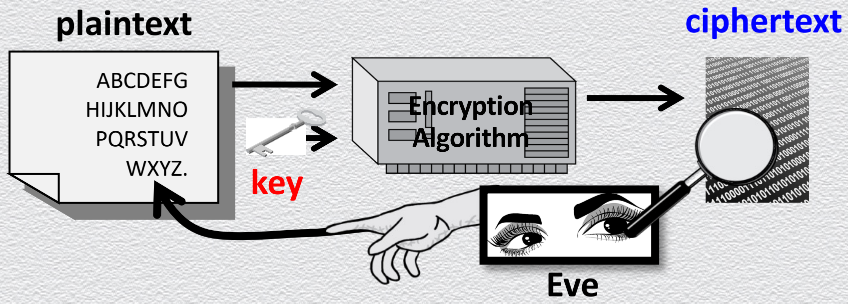
"plain" security

- protects against ciphertext-only attacks
  - EAV-attack



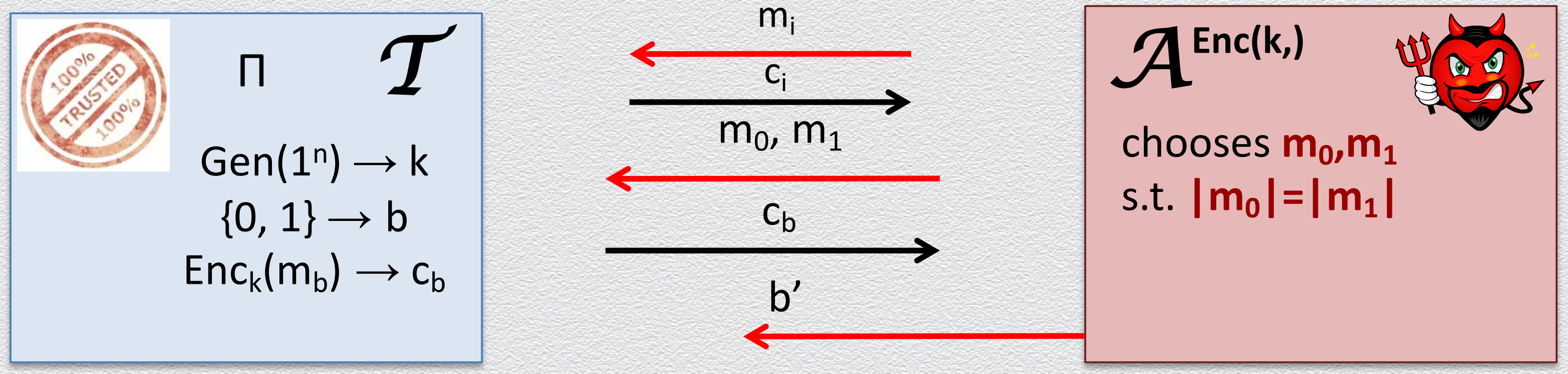
"advanced" security

- protects against chosen plaintext attacks
  - CPA-attack



Game-based computational CPA-security

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



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

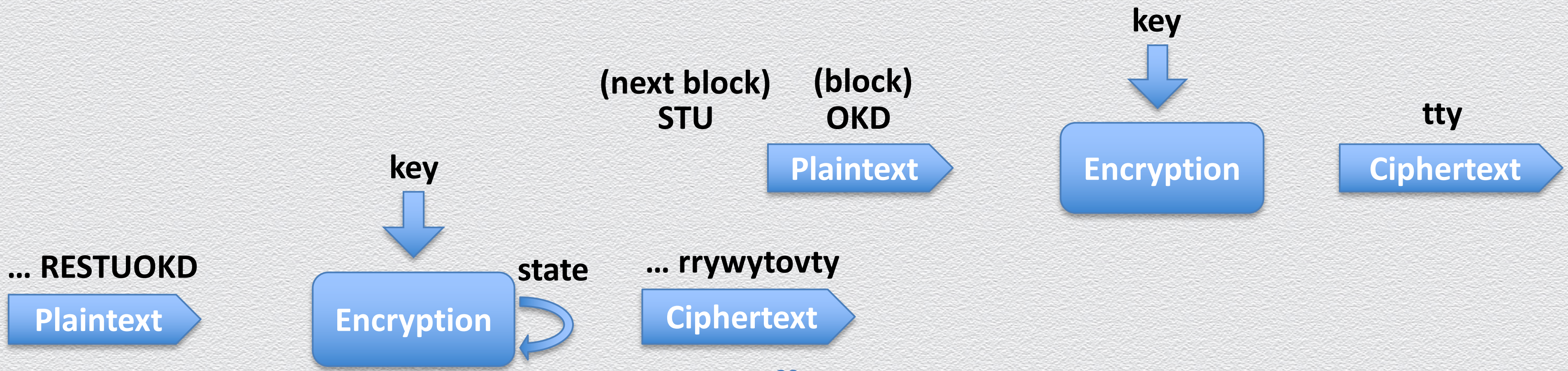
Uses a **short** key to encrypt **long** symbol **streams** into a **pseudorandom** ciphertext

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

Block cipher

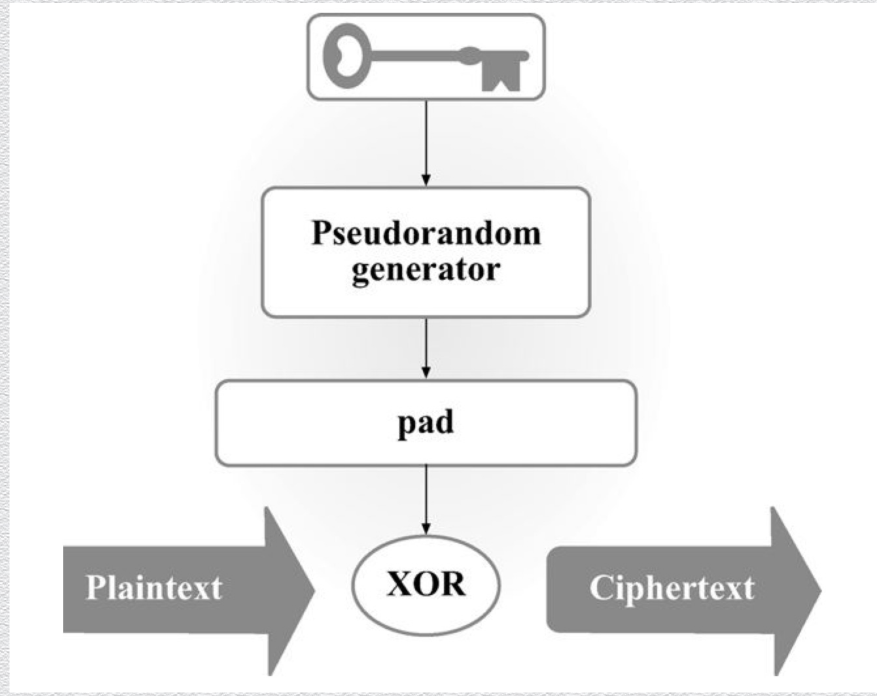
Uses a **short** key to encrypt **blocks** of symbols into **pseudorandom** ciphertext blocks

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



Generic PRG-based symmetric encryption

- **Fixed-length** message encryption

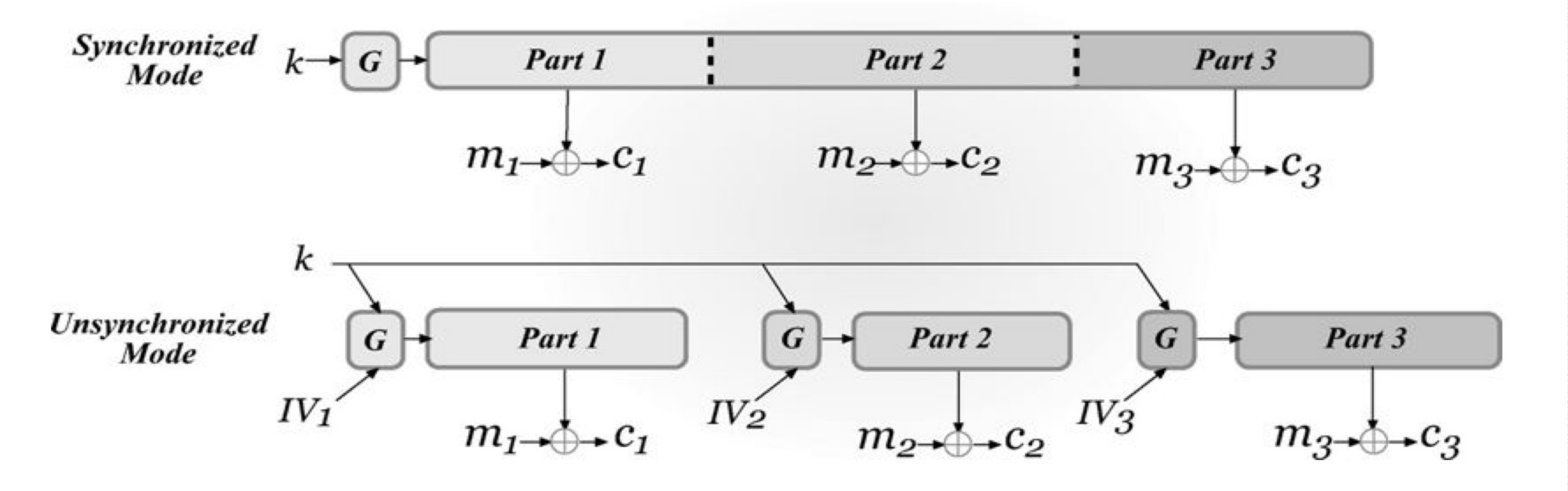


encryption scheme is plain-secure
as long as the underlying PRG is secure

Stream ciphers: Modes of operations

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

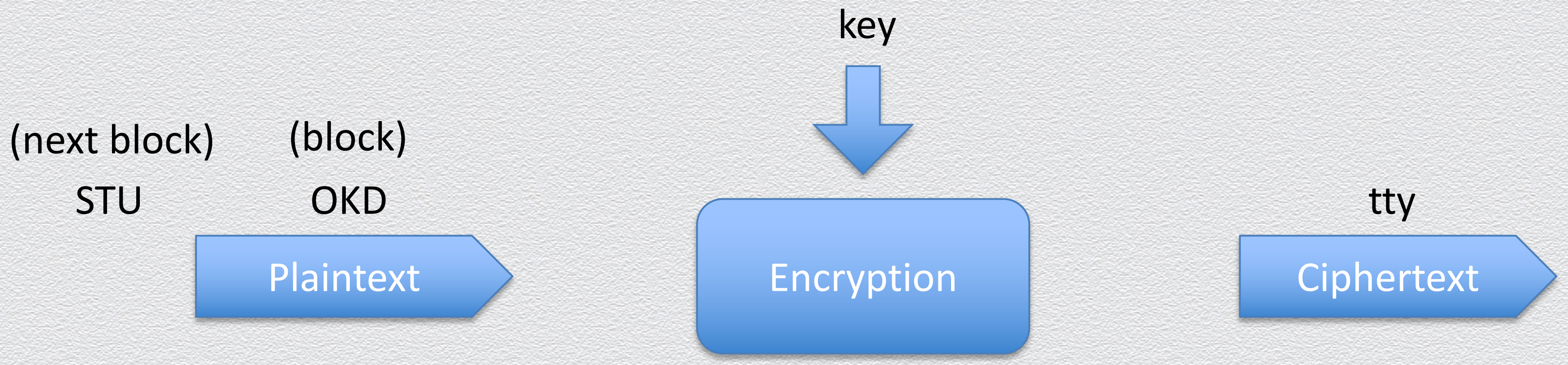
on-the-fly computation of new pseudorandom bits, no IV needed, plain-secure



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

5.1 Pseudorandom functions (or block ciphers)

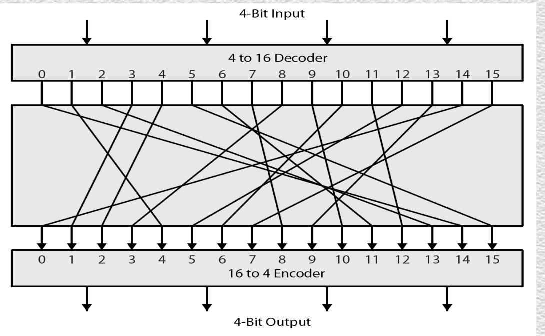
Block ciphers



Realizing ideal block ciphers in practice

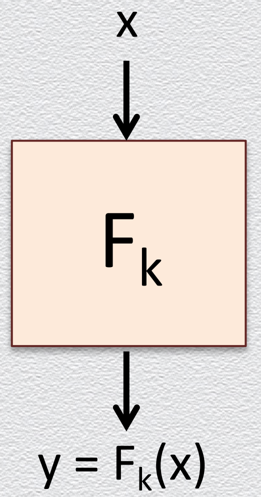
We want a **random** mapping of n-bit inputs to n-bit outputs

- there are ~2^($n2^n$) possible such mappings
- none of the above can be implemented in practice



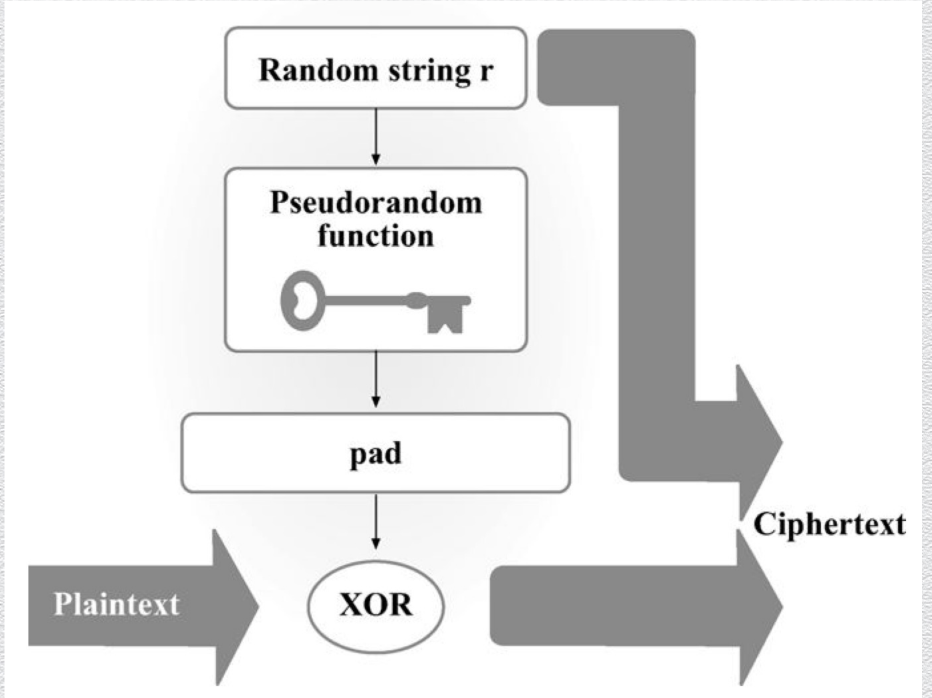
Instead, we use a keyed function $F_k : \{0,1\}^n \rightarrow \{0,1\}^n$

- 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

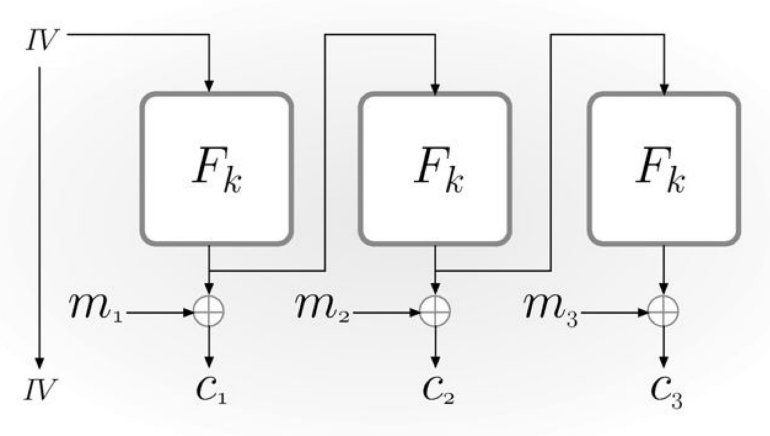
Generic PRF-based symmetric encryption (cont.)

- **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

5.2 Modes of operations (of block ciphers)

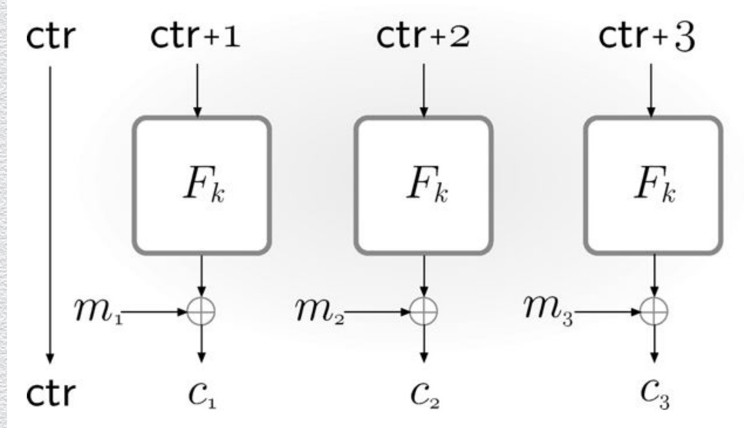
Block ciphers: Modes of operations (I)

- 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



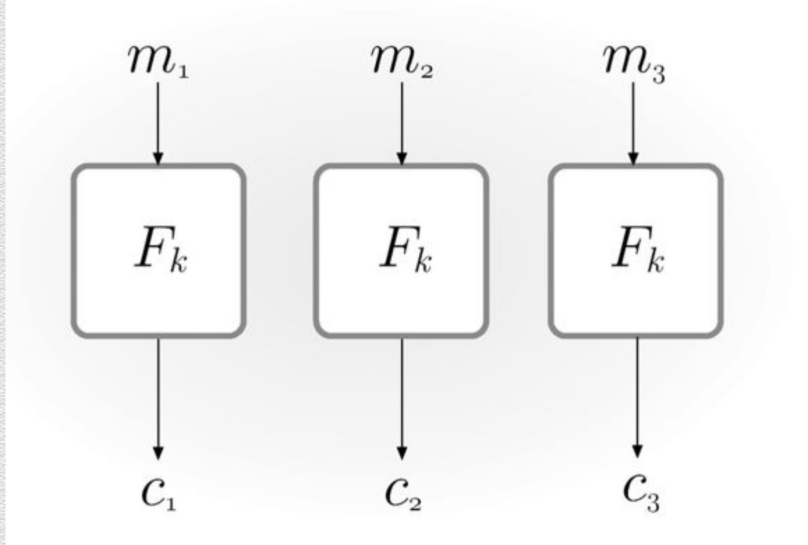
Block ciphers: Modes of operations (II)

- 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



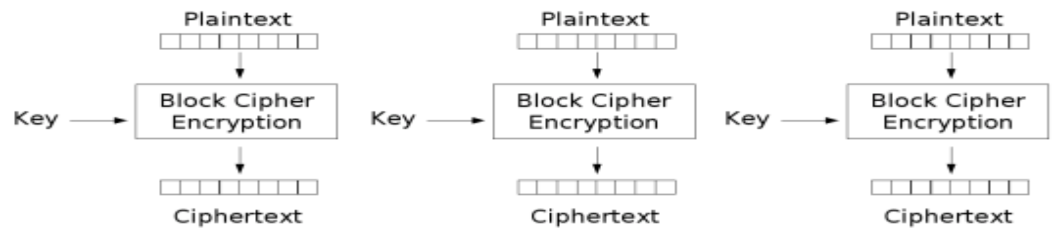
Block ciphers: Modes of operations (III)

- 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] = Enc_k(P[i])
  - block C[i] decrypted into plaintext block M[i] = Dec_k(C[i])



Electronic Codebook (ECB) mode encryption

Strengths & weaknesses of ECB

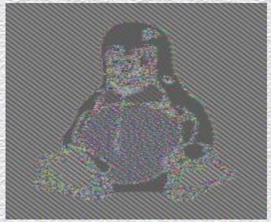
Strengths

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

Weaknesses

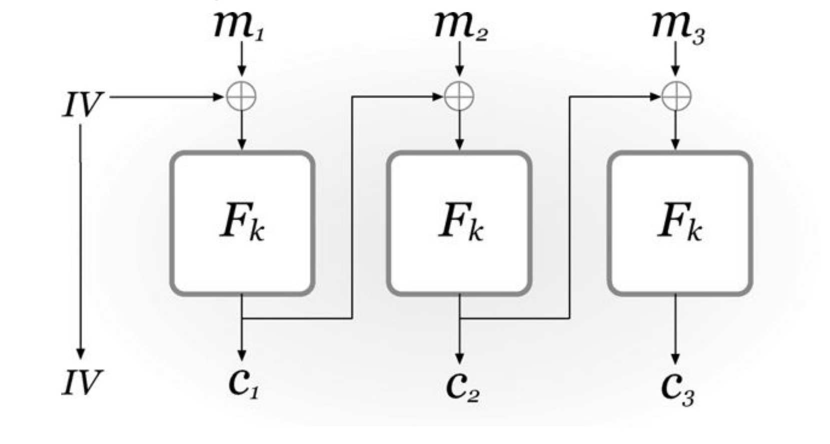
- poor security
- produces the same ciphertext on the same plaintext (under the same key)
- documents and images are not suitable for ECB encryption, since patterns in the plaintext are repeated in the ciphertext
- e.g.,

ECB

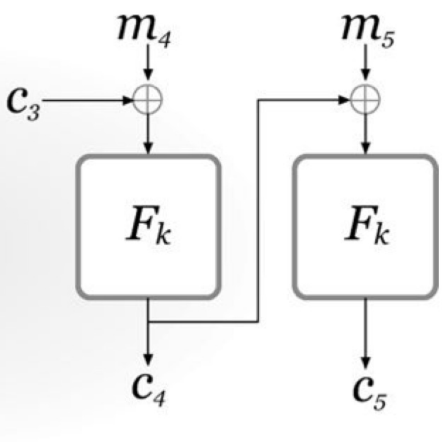


Block ciphers: Modes of operations (IV)

- CBC – cipher block chaining
  - CPA-secure if F_k a permutation
  - uniform IV
    - otherwise security breaks



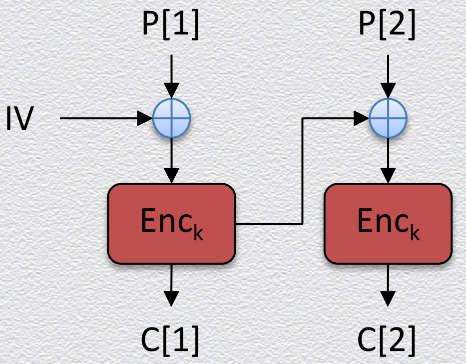
- Chained CBC
  - use last block ciphertext of current message as IV of next message
  - saves bandwidth but not CPA-secure

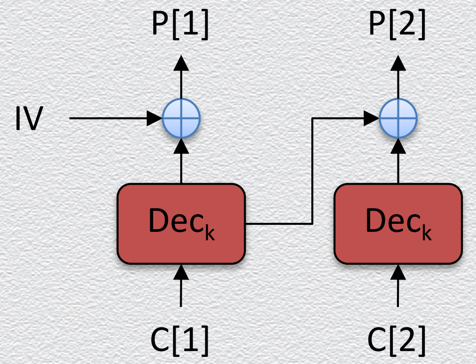


Cipher Block Chaining (CBC) [or chaining]

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

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





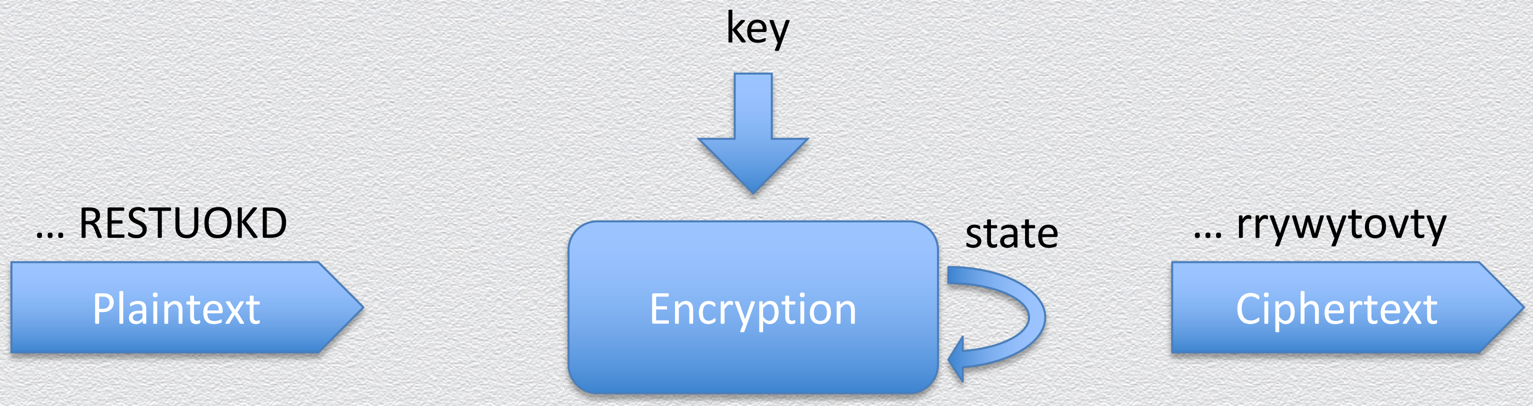
CBC

Notes on modes of operation

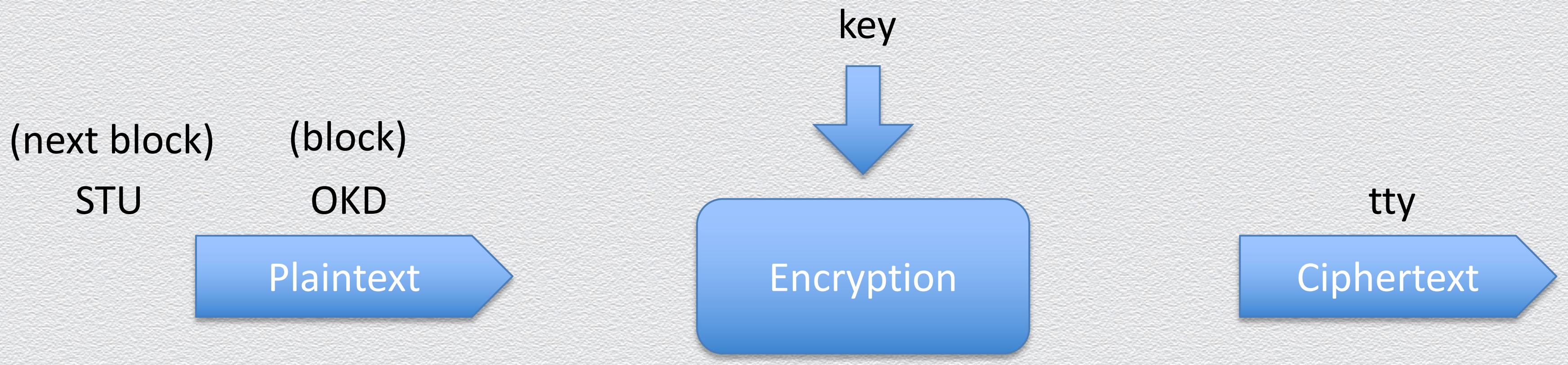
- block length matters
  - if small, IV or ctr can be “recycled”
- IV are often misused
  - e.g., reused or not selected uniformly at random
  - in this case, CBC is a better option than OFB/CTR

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	• Speed of transformation • Low error propagation	• High diffusion • Immunity to insertion of symbol
Disadvantages	• Low diffusion • Susceptibility to malicious insertions and modifications	• Slowness of encryption • Padding • Error propagation

5.4 Block ciphers in practice: DES & AES

DES: The Data Encryption Standard

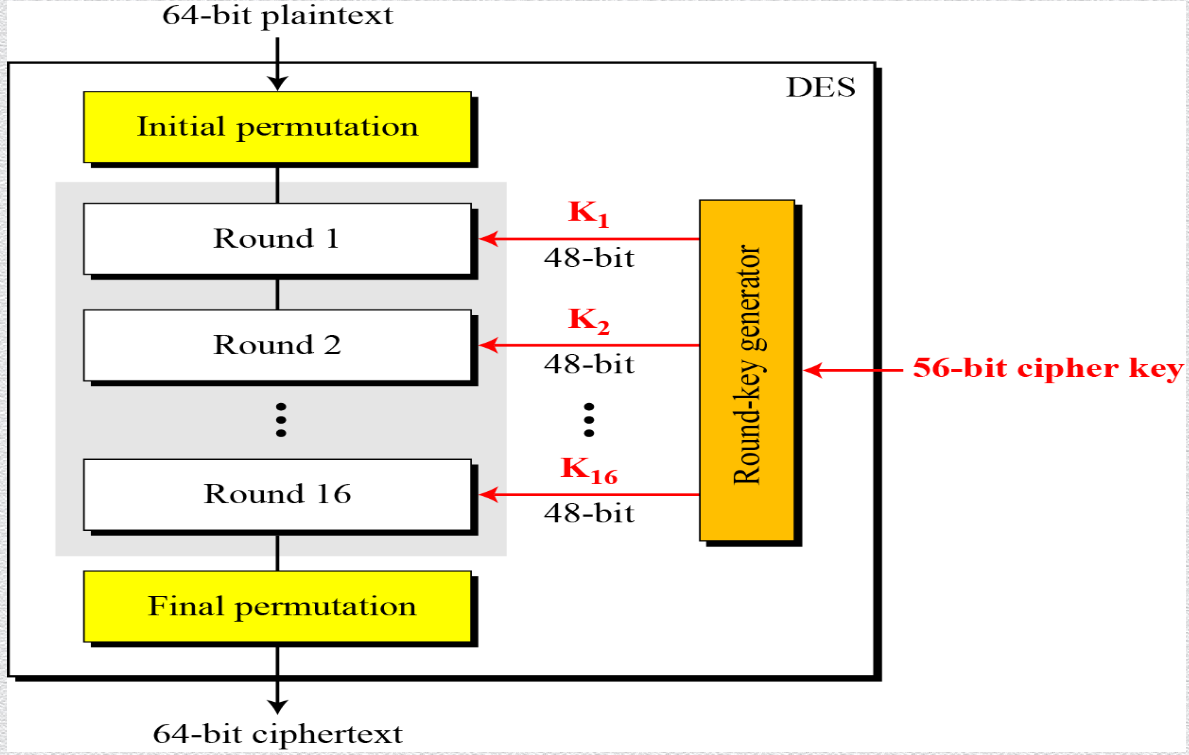
- Symmetric block cipher
- Developed in 1976 by IBM for the US National Institute of Standards and Technology (NIST)
- Employs substitution & transposition, on top of each other, for 16 rounds
  - block size = 64 bits, key size = 56 bits
- Strengthening (since 56-bit security is not considered adequately strong)
  - double DES: $E(k_2, E(k_1, m))$, not effective!
  - triple DES: $E(k_3, \mathbf{E}(k_2, E(k_1, m)))$, 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

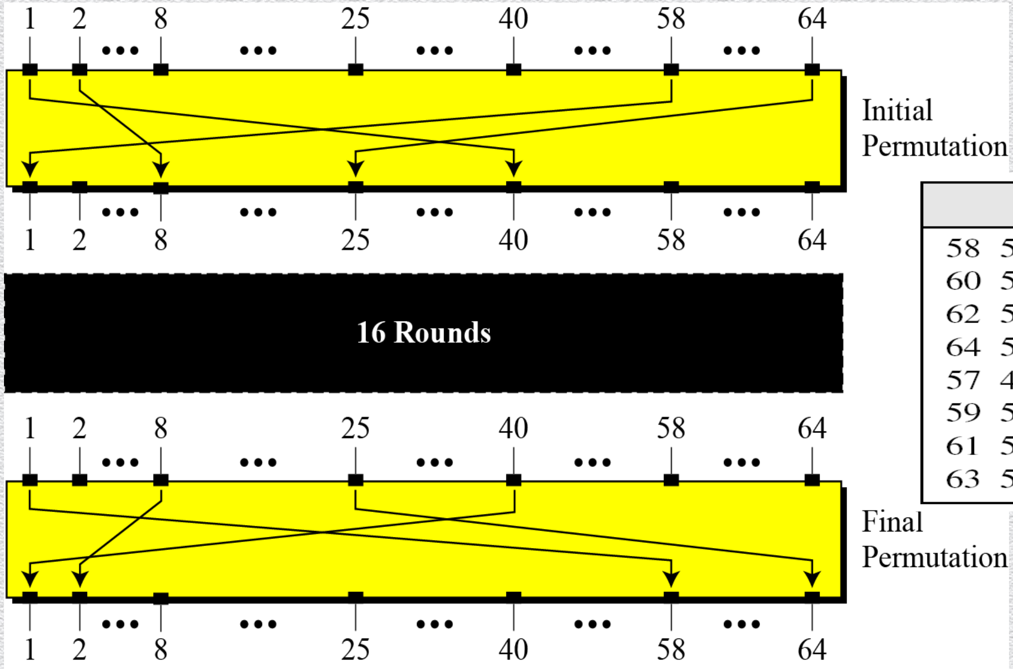
DES: High-level view

DES: Basic structure



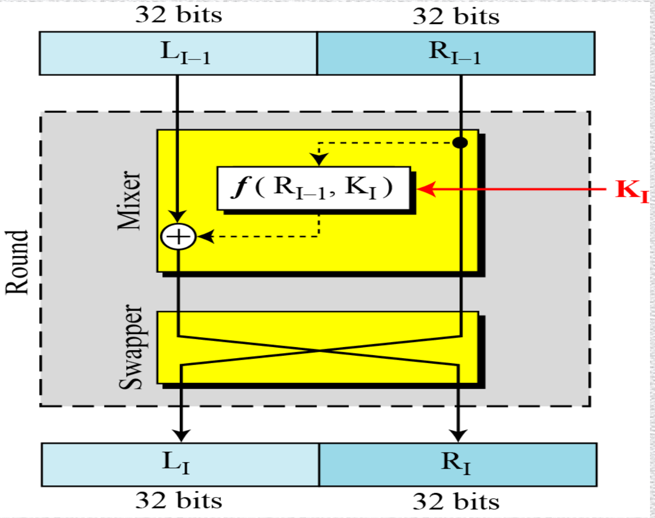
DES: Initial and final permutations

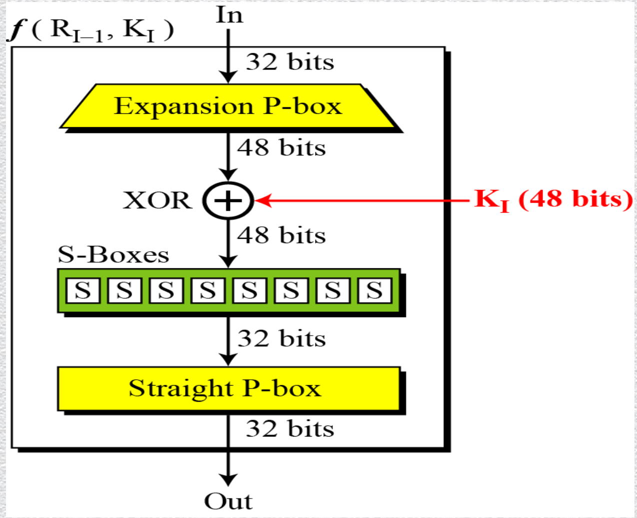
- Straight P-boxes that are inverses of each other w/out crypto significance



Initial Permutation								*Final Permutation*							
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

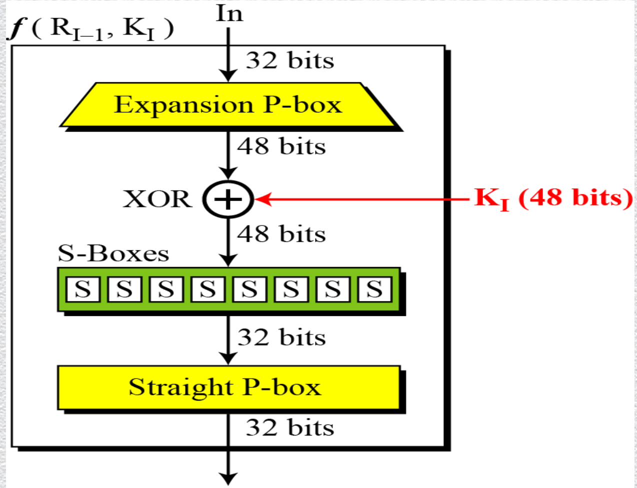


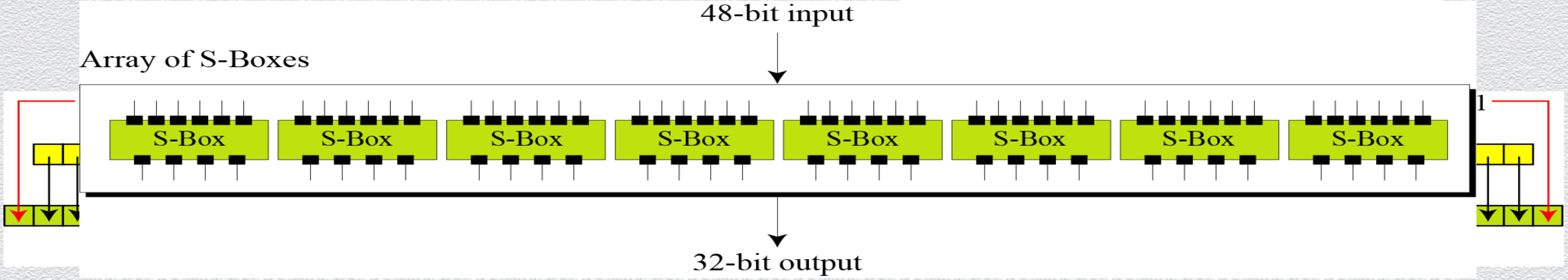


- DES uses 16 rounds, each applying a Feistel cipher
  - L(i) = R(i-1)
  - R(i) = L(i-1) XOR *f* (K(i),R(i-1)),
    where *f* applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output

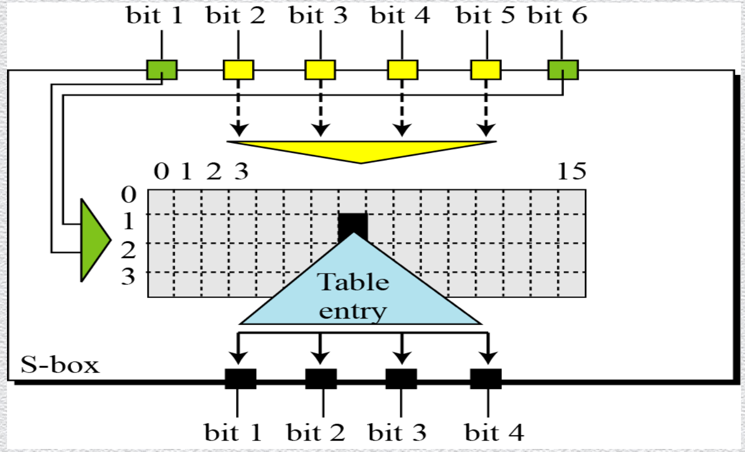
DES: Low-level view

- 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





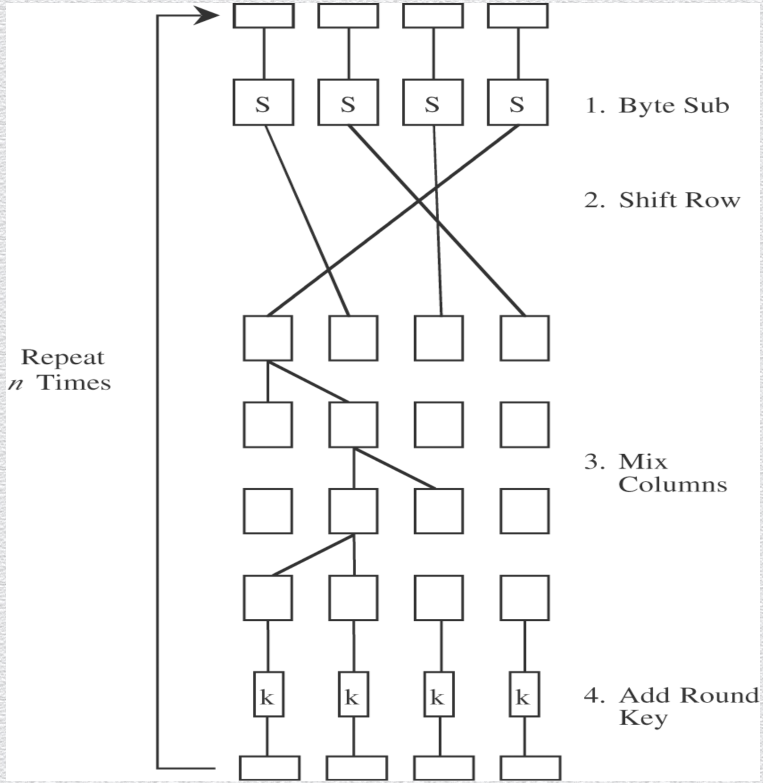
DES: S-box in detail



	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

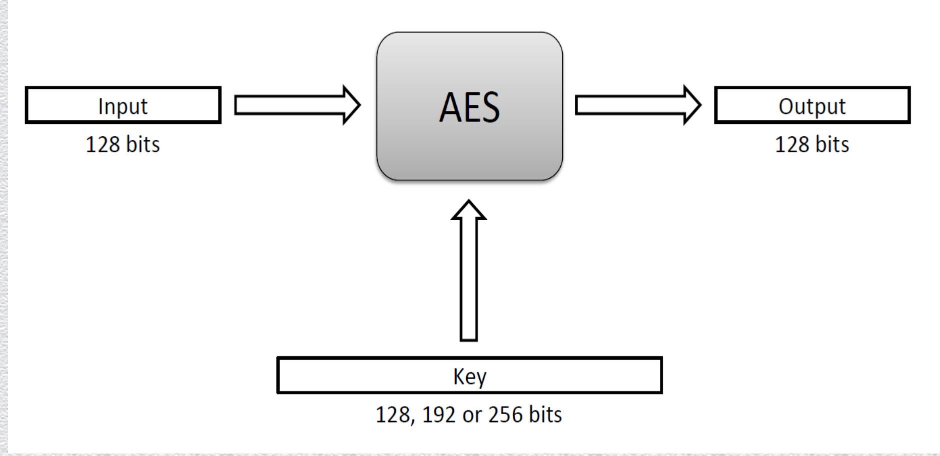
AES: Advanced Encryption System

- 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

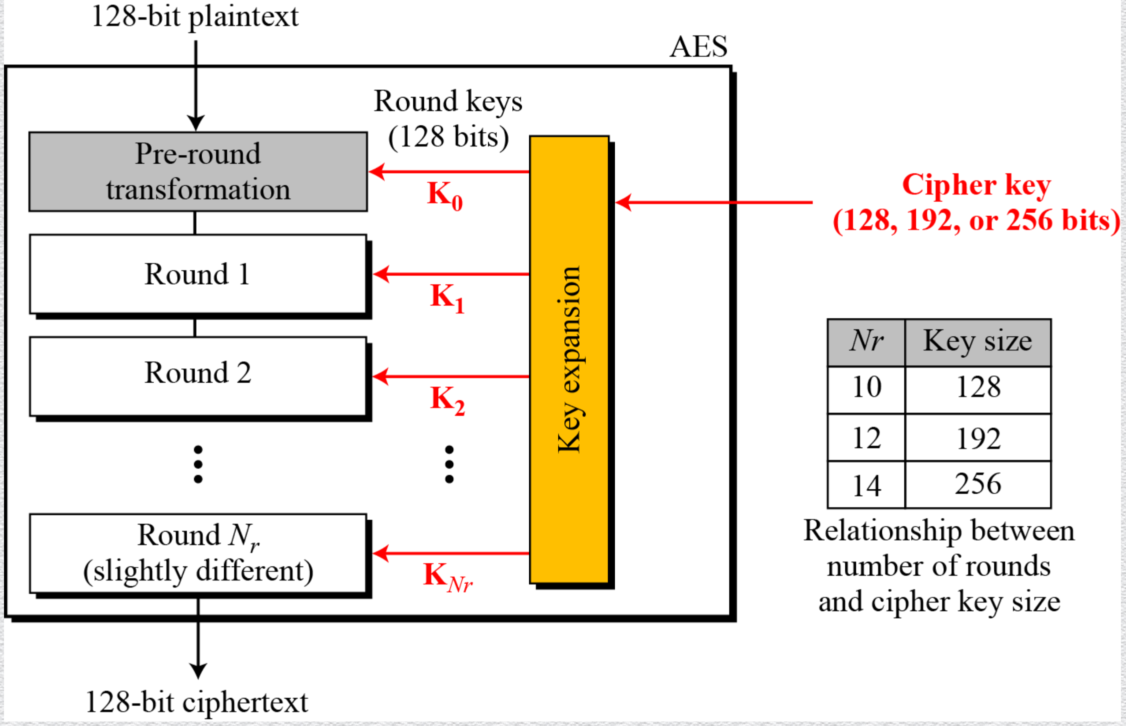


AES: Key design features

- 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



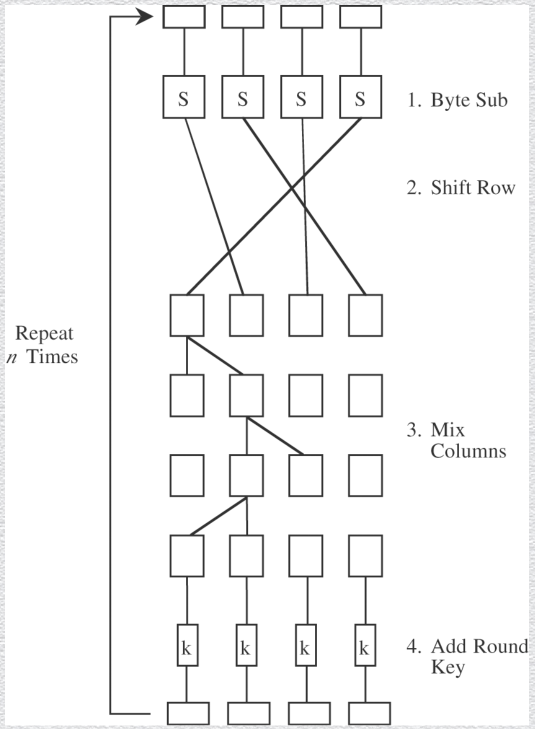
AES: Basic structure



Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

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

5.1 Message authentication

Recall: Integrity

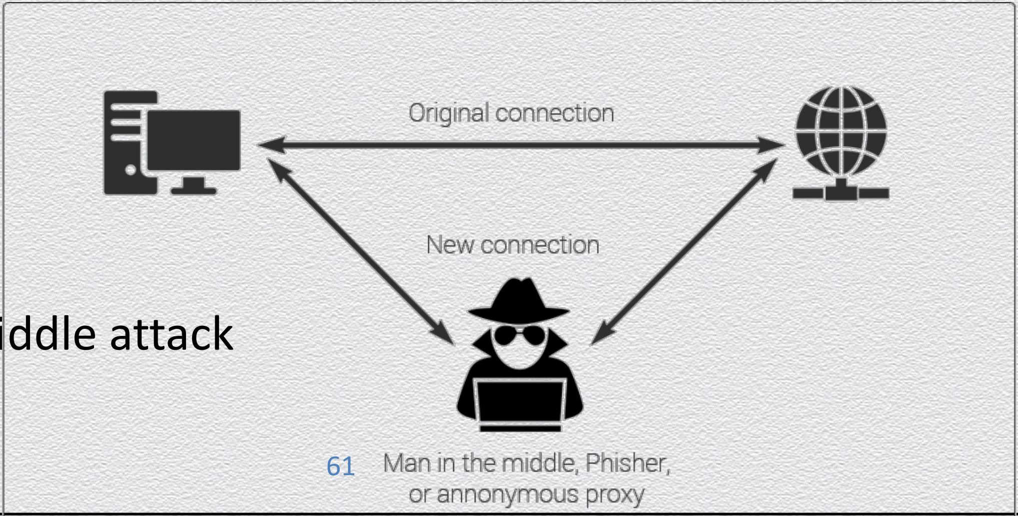
Fundamental security property

- **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

5.2 Message authentication codes (MACs)

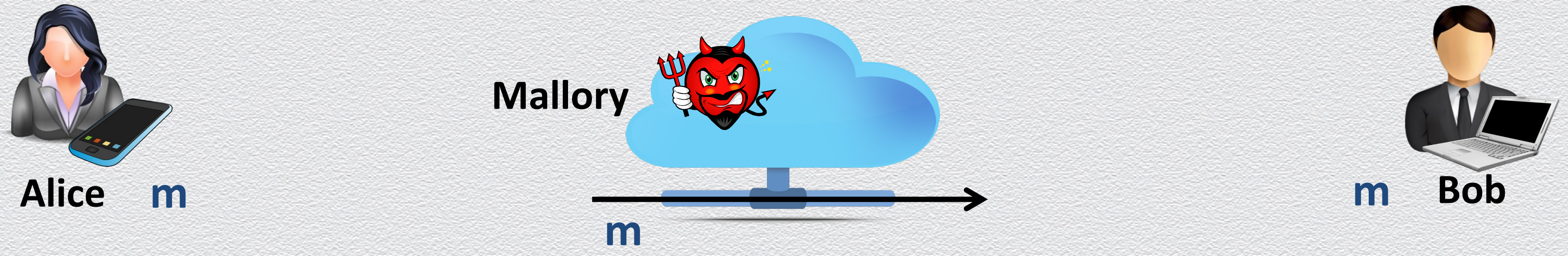
Problem setting: Reliable communication

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

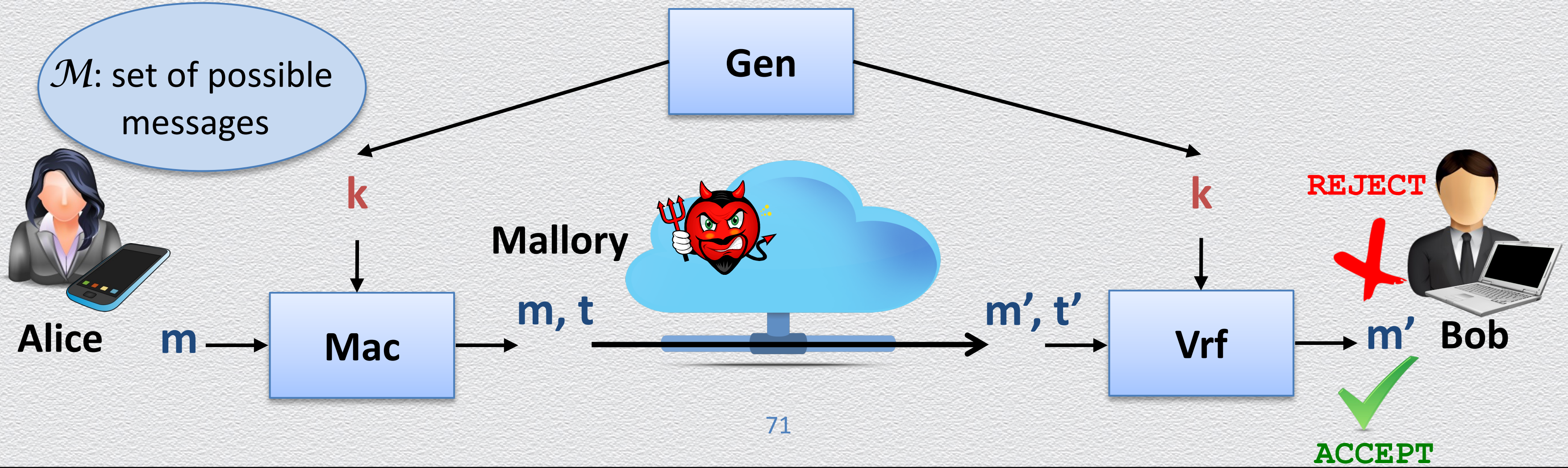
Main idea

- 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

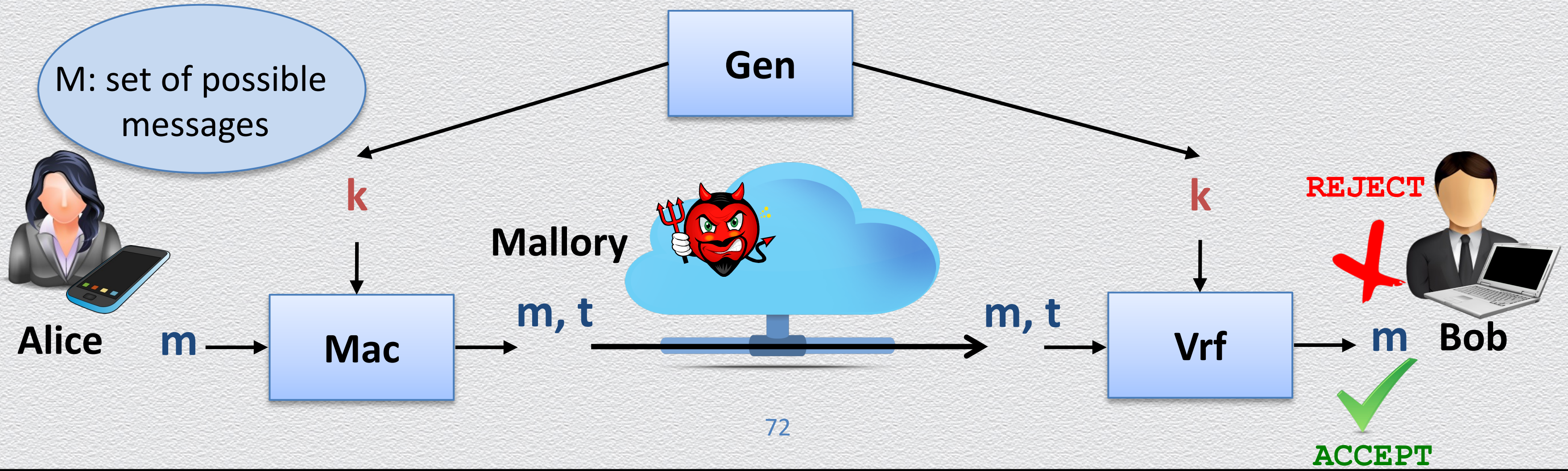
- a **message space $\mathcal{M}$**; and
- a triplet of algorithms **(Gen, Mac, Vrf)**
  - Gen, Mac are probabilistic algorithms, whereas Vrf is deterministic
  - Gen outputs a uniformly random key k (from some key space $\mathcal{K}$)



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 $Vrf_k(m, Mac_k(m))$ = ACCEPT
- **security**: one "cannot forge" a fake verifiable pair m', t'



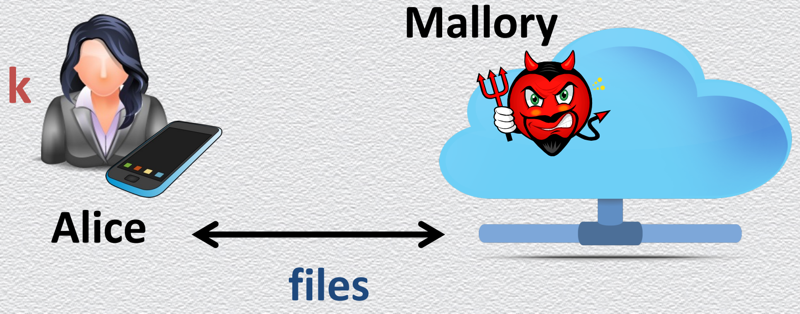
Main application areas

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

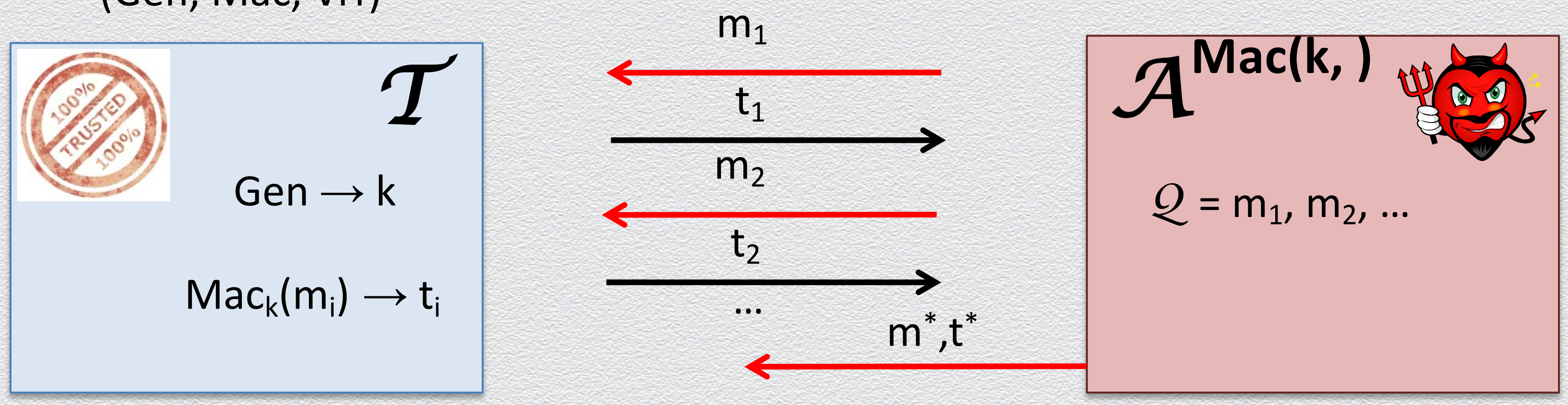
- when Mac is deterministic, Vrf typically amounts to re-computing the tag t
  - $Vrf_k(m, t)$: 1. $t' := Mac_k(m)$ 2. if $t = t'$, output `ACCEPT` else output `REJECT`
- but conceptually the following operations are distinct
  - authenticating m (i.e., running Mac) Vs. verifying authenticity of m (i.e.,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

MAC security

Attacker **wins** the game if

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

MAC scheme
(Gen, Mac, Vrf)



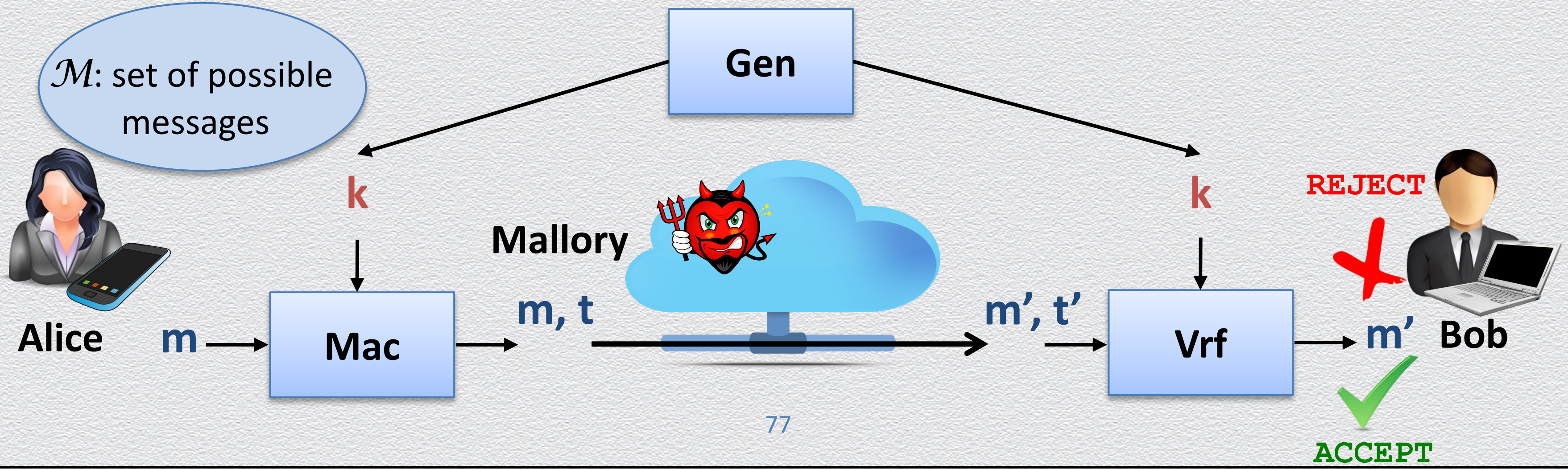
The MAC scheme is **secure** if any PPT $\mathcal{A}$ wins the game only negligibly often.

5.2.1 Replay attacks

Recall: MAC

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

- a **message space** $\mathcal{M}$; and
- a triplet of algorithms **(Gen, Mac, Vrf)**

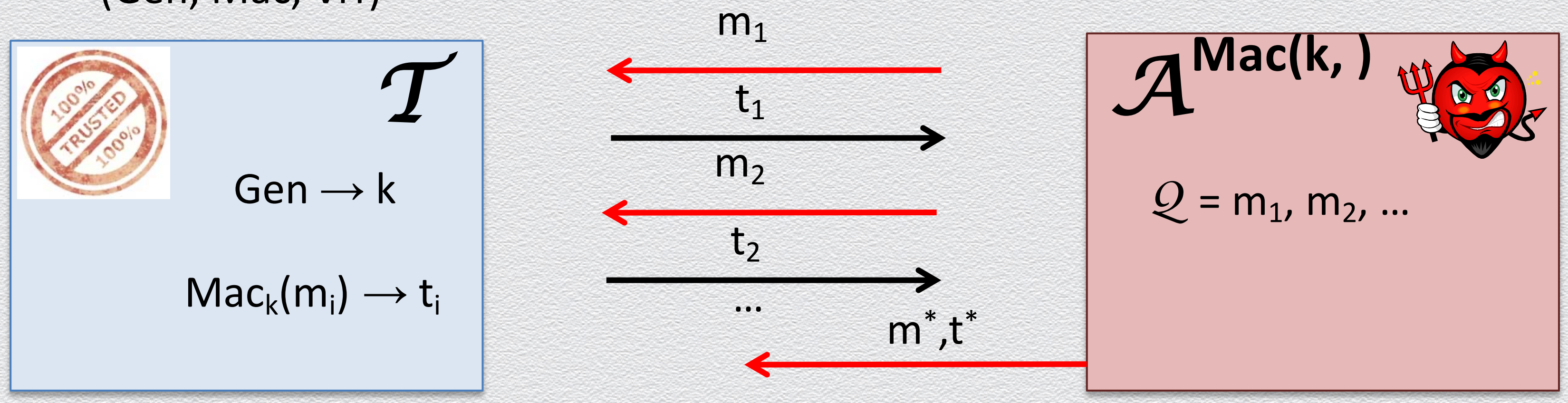


Recall: MAC security

Attacker **wins** the game if

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

MAC scheme
(Gen, Mac, Vrf)



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

Real-life attacker

In practice, an attacker may

- observe a traffic of authenticated (and successfully verified) messages
- manipulate (or often also partially influences) traffic
  - aims at inserting an invalid but verifiable message m^*, t^* into the traffic
    - interesting case: forged message is a <u>new</u> (unseen) one
    - trivial case: forged message is a <u>previously observed</u> one, a.k.a. a **replay attack**
- launch a **brute-force attack** (given that $Mac_k(m) \rightarrow t$ is publicly known)
  - given any observed pair m, t, exhaustively search key space to find the used key k

Threat model

In the security game, Mallory is an adversary $\mathcal{A}$ who is

- "active" (on the wire)
  - we allow $\mathcal{A}$ to **observe** and **manipulate** sent messages
- "well-informed"
  - we allow $\mathcal{A}$ to **request MAC tags** of messages of **its choice**
- "replay-attack safe"
  - we restrict $\mathcal{A}$ to **forge only new** messages
- "PPT"
  - we restrict $\mathcal{A}$ to be **computationally bounded**
  - new messages may be forged undetectably only **negligibly** often

Notes on security definition

Is it a rather strong security definition?

- we allow $\mathcal{A}$ to **query MAC tags for any message**
  - but real-world senders will authenticate only "meaningful" messages
- we allow $\mathcal{A}$ to break the scheme by **forging any new message**
  - but real-world attackers will forge only "meaningful" messages

Yes, it is the right approach…

- message "**meaningfulness" depends on higher-level application**
  - text messaging apps require authentication of English-text messages
  - other apps may require authentication of binary files
  - security definition should better be **agnostic** of the specific higher application

Notes on security definition (II)

Are replay attacks important in practice?

- absolutely yes: a **very realistic & serious threat!**
  - e.g., what if a money transfer order is "replayed"?

Yet, a "replay-attack safe" security definition is preferable

- again, whether replayed messages are valid depends on higher-lever app
- better to delegate to this app the specification of such details
  - e.g., semantics on traffic or validity checks on messages before they're "consumed"

Eliminating replay attacks

- use of counters (i.e., common shared state) between sender & receiver
- use of timestamps along with a (relaxed) authentication window for validation

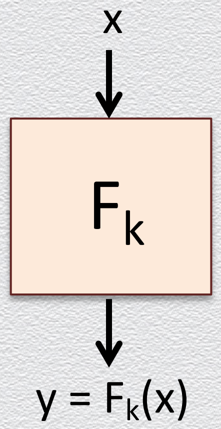
5.2.2 MAC constructions

Three generic MAC constructions

- fixed-length MAC
  - direct application of a PRF for tagging
  - limited applicability
- domain extension for MACs
  - straightforward secure extension of fix-length MAC
  - inefficient
- CBC-MAC
  - resembles CBC-mode encryption
  - efficient

1. Fixed-length MAC

- based on use of a PRF
  - employ a PRF F_k in the obvious way
    to compute and canonically verify tags
  - set tag t to be the pseudorandom string
    derived by evaluating F_k on message m
- secure, provided that F_k is a secure PRF



<u>MAC scheme Π</u>

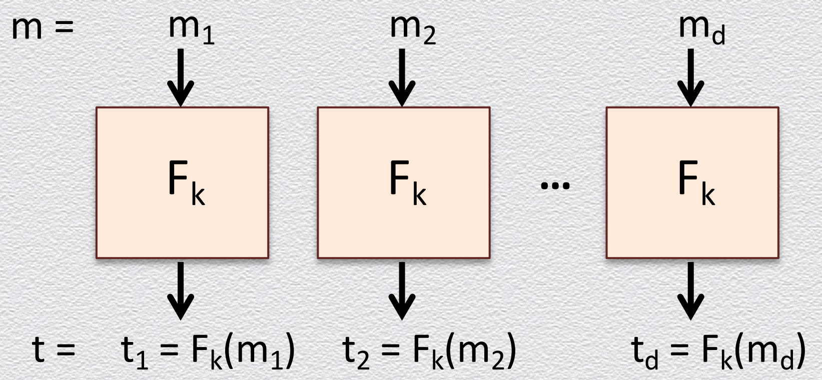
Gen(1^n): $\{0,1\}^n \rightarrow k$

$Mac_k(m)$: set $t = F_k(m)$

$Vrfy_k(m,t)$: return 1 iff $t = F_k(m)$

2. Domain extension for MACs (I)

- suppose we have the previous fix-length MAC scheme
- how can we authenticate a message m of arbitrary length?
- naïve approach
  - pad m and view it as d blocks $m_1, m_2, \ldots, m_d$
  - separately apply MAC to block m_i



- security issues
  - reordering attack; verify block index, $t = F_k(m_i || i)$
  - truncation attack; verify message length $\delta = |m|$, $t = F_k(m_i || i || \delta)$
  - mix-and-match attack; randomize tags (using message-specific fresh nonce)

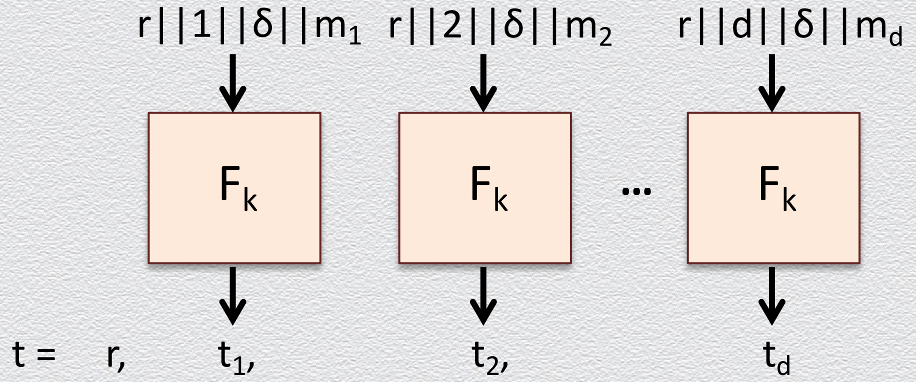
2. Domain extension for MACs (II)

Final scheme

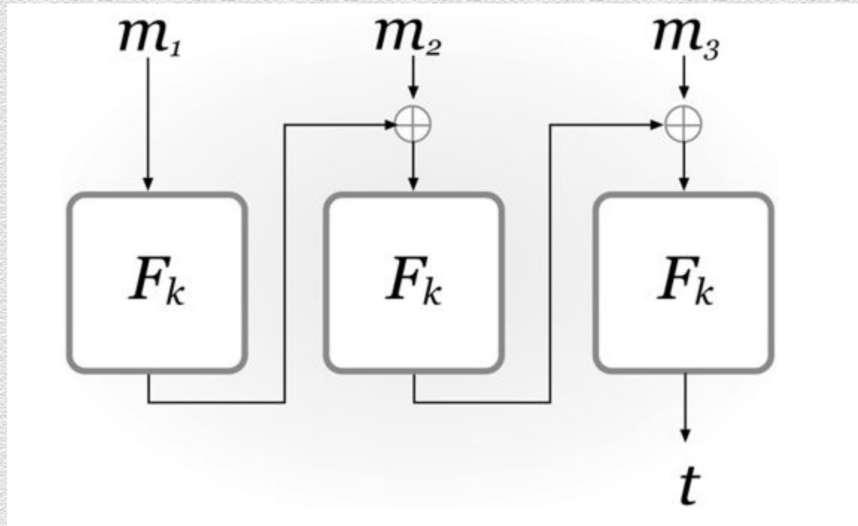
- assumes a secure MAC scheme for messages of size n
- set tag of message m of size δ at most $2^{n/4}$ as follows
  - choose fresh random nonce r of size n/4; view m as d blocks of size n/4 each
  - separately apply MAC on each block, authenticating also its index, δ and nonce r

Security

- extension is secure, if F_k is a secure PRF



3. CBC-MAC



Idea

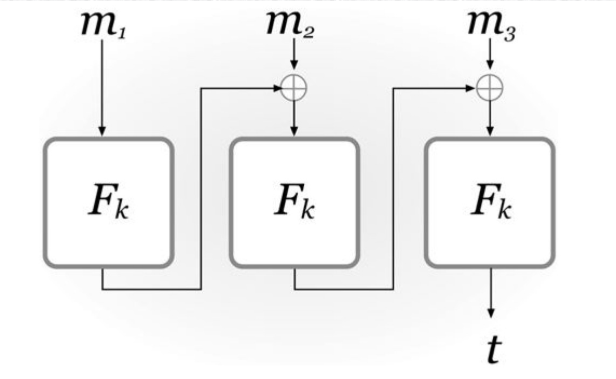
- employ a PRF in a manner similar to CBC-mode encryption

Security

- extension is secure, if
  - F_k is a secure PRF; and
  - only **fixed-length** messages are authenticated
- messages of length equal to any multiple of n can be authenticated
  - but this length need be fixed in advance
  - insecure, otherwise

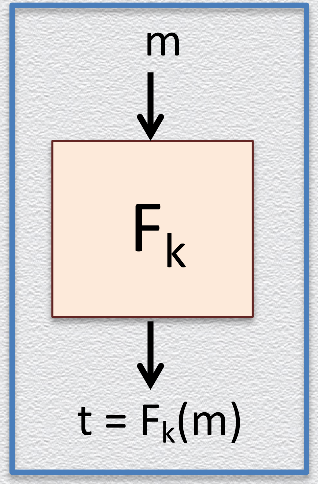
3. CBC-MAC Vs. previous schemes

- can authenticate longer messages than basic PRF-based scheme (1)

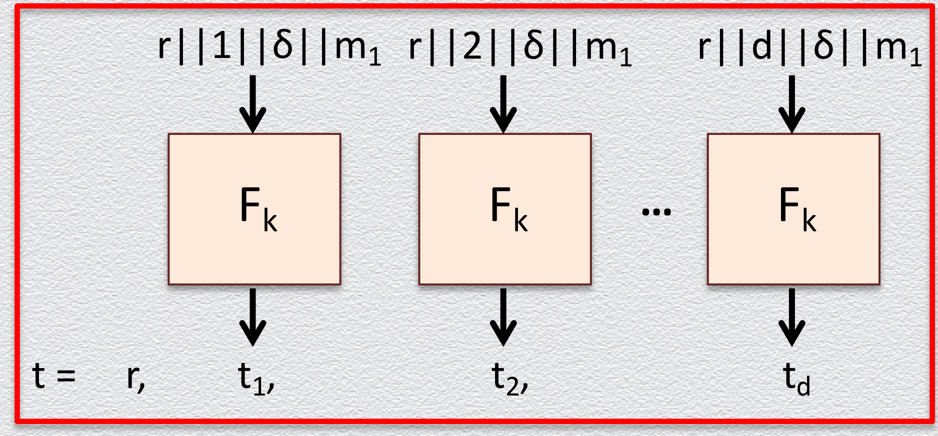


- more efficient than domain-extension MAC scheme (2)

Scheme (1)



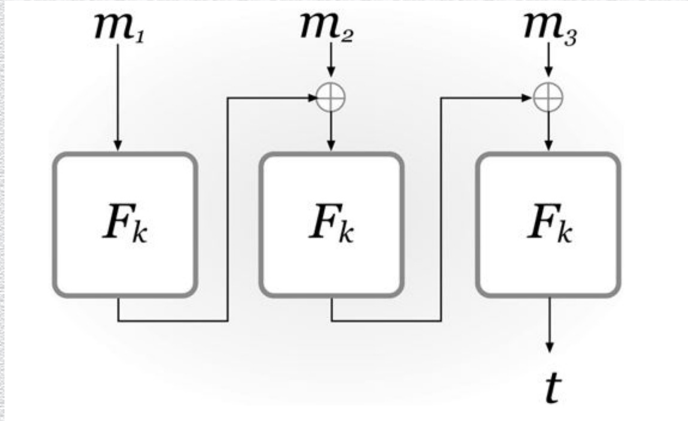
Scheme (2)



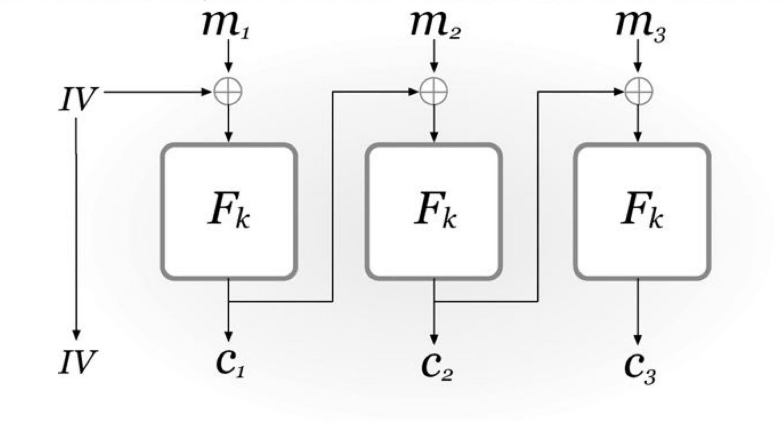
3. CBC-MAC Vs. CBC-mode encryption

- crucially for their security
  - CBC-MAC uses **no IV** (or uses an IV set to 0) and only the **last PRF output**
  - CBC-mode encryption uses a **random IV** and **all PRF outputs**
  - “simple”, innocent modification can be catastrophic…

CBC-MAC



CBC-mode encryption



5.3 Authenticated encryption

Recall: Two distinct properties

Secrecy

- **sensitive** information has value
  - if **leaked**, it can be **risky**
- **specific** scope / **general** semantics
- **prevention**
- does **not** imply integrity
  - e.g., bit-flipping "attack"

Integrity

- **correct** information has value
  - if **manipulated**, it can **harmful**
  - **random Vs. adversarial** manipulation
- **wider** scope / **context-specific** semantics
  - source Vs. content authentication
  - replay attacks
- **detection**
- does **not** imply secrecy
  - e.g., user knows cookies' "contents"

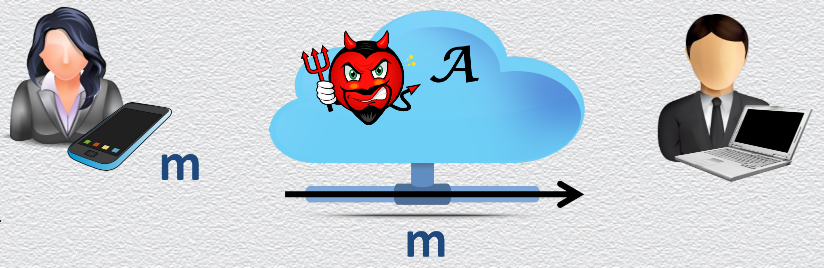
Recall: Yet, they are quite close…

Common setting

- communication (storage) over an "**open**," i.e., **unprotected**, channel (medium)

Fundamental security problems

- while in transit (at rest)
  - no message (file) should be **leaked** to $\mathcal{A}$
  - no message (file) should be **modified** by $\mathcal{A}$



Core cryptographic protections

- **encryption schemes** provide **secrecy / confidentiality**
- **MAC schemes** provide **integrity / unforgeability**

Can we achieve both at once in the symmetric-key setting? **Yes!**

Authenticated Encryption (AE): Catch 2 birds w/ 1 stone

Cryptographic primitive that realizes an "**ideally secure**" communication channel

- motivation
  - important in practice as real apps often **need both**
  - **good security hygiene**
    - even if a given app "asks" only/more for secrecy or integrity than the other, it's always better **to achieve both**!

Three generic AE constructions

Constructions of a **secure authenticated encryption** scheme Π_{AE}

- they all make use of
  - a **CPA-secure** encryption scheme Π_E = (Enc, Dec); and
  - a **secure MAC** Π_M = (Mac, Vrf)
  - which are instantiated using **independent** secret keys ke, km

  - …but the **order** with which these are used matters!

Generic AE constructions (1)

1. **encrypt-and-authenticate**

- $Enc_{ke}(m) \longrightarrow c$; $Mac_{km}(m) \longrightarrow t$; send ciphertext (c, t)
- if $Dec_{ke}(c) = m \neq$ `fail` and $Vrf_{km}(m,t)$ accepts, output m; else output `fail`
- **insecure scheme, generally**
  - e.g., MAC tag t may leak information about m
  - e.g., if MAC is deterministic (e.g., CBC-MAC) then Π_{AE} is not even CPA-secure
  - used in SSH

Generic AE constructions (2)

2. **authenticate-then-encrypt**

- $Mac_{km}(m) \rightarrow t$; $Enc_{ke}(m||t) \rightarrow c$; send ciphertext c
- if $Dec_{ke}(c) = m||t \neq$ `fail` and $Vrf_{km}(m,t)$ accepts, output m; else output `fail`
- **insecure scheme, generally**
  - used in TLS, IPsec

Generic AE constructions (3)

3. **encrypt-then-authenticate** (cf. “authenticated encryption”)

- $\mathrm{Enc}_{ke}(m) \longrightarrow c$; $\mathrm{Mac}_{km}(c) \longrightarrow t$; send ciphertext (c, t)
- if $\mathrm{Vrf}_{km}(c,t)$ accepts then output $\mathrm{Dec}_{ke}(c) = m$, else output `fail`
- **secure scheme, generally** (as long as Π_M is a **“strong” MAC**)
  - used in TLS, SSHv2, IPsec

Application: Secure communication sessions

An AE scheme Π_{AE} = (Enc, Dec) enables two parties to **communicate securely**

- session: period of time during which sender and receiver maintain state
- idea: send any message m as $c = Enc_k(m)$ & ignore received c that don't verify
- security: **secrecy & integrity are protected**
- remaining possible attacks
  - **re-ordering** attack — counters can be used to eliminate reordering/replays
  - **reflection** attack — directional bit can be used to eliminate reflections
  - **replay** attack — $c = Enc_k(b_{A \to B} | ctr_{A,B} || m)$; $ctr_{A,B}$++

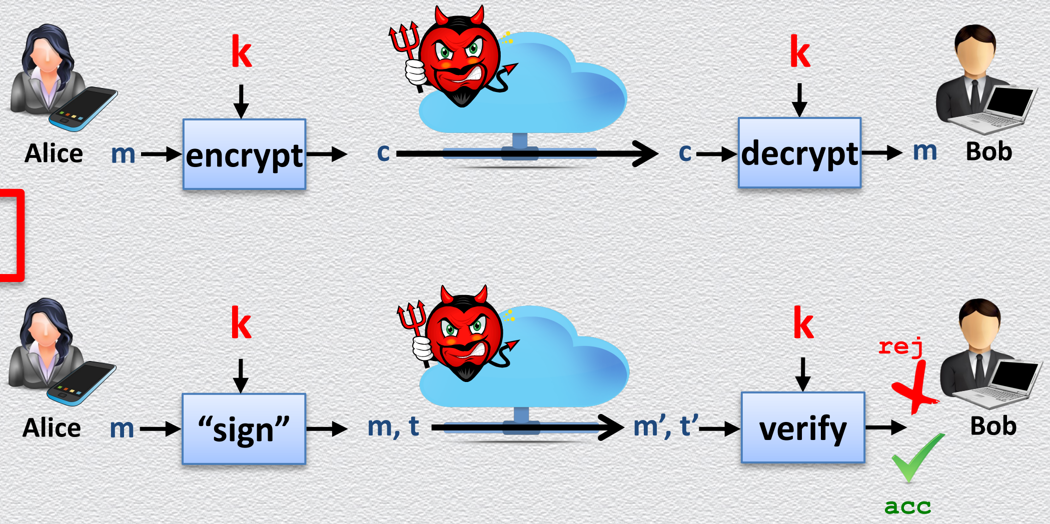
5.4 Public-key encryption & digital signatures

Recall: Principles of modern cryptography

(A) security definitions, **(B) precise assumptions**, (C) formal proofs

For **symmetric-key** message encryption/authentication

- adversary
  - types of attacks
- trusted set-up
  - secret key is distributed securely
  - secret key remains secret
- trust basis
  - underlying primitives are secure
  - **PRG, PRF, hashing, ...**
    - **e.g., block ciphers, AES, etc.**



On “secret key is distributed securely”

Alice & Bob (or 2 individuals) must **securely obtain** a **shared secret key**

- “securely obtain” ⇨ **strong assumption** to accept
  - need of a secure channel
- “shared secret key” ⇨ **challenging problem** to manage
  - too many keys

⇨ **Public-key cryptography to the rescue…**

On "secret key is distributed securely"

Alice & Bob (or 2 individuals) must **securely obtain** a **shared secret key**

- "securely obtain" → **(A) strong assumption** to accept
  - requires secure channel for key distribution (chicken & egg situation)
  - seems impossible for two parties having no prior trust relationship
  - not easily justifiable to hold a priori
- "shared secret key" → **(B) challenging problem** to manage
  - requires too many keys, namely $O(n^2)$ keys for n parties to communicate
  - imposes too much risk to protect all such secret keys
  - entails additional complexities in dynamic settings (e.g., user revocation)

Alternative approaches?

Need to securely distribute, protect & manage many **session-based** secret keys

- (A) for secure distribution, just "make another assumption…"
  - employ **"designated" secure channels**
    - physically protected channel (e.g., meet in a "sound-proof" room)
  - employ **"trusted" party**
    - entities authorized to distribute keys (e.g., key distribution centers (KDCs))
- (B) for secure management, just 'live with it!"

Public-key cryptography to the rescue…

Public-key (or asymmetric) cryptography

disclaimer on names
private = secret

Goal: devise a cryptosystem where key setup is "more" manageable

Main idea: **user-specific** keys (that come in pairs)

- user U generates two keys (U_{pk}, U_{sk})
  - **U_{pk} is public** – it can safely be known by everyone (even by the adversary)
  - **U_{sk} is private** – it must remain secret (even from other users)

Usage

- employ **public** key U_{pk} for certain "**public**" tasks (performed by **other users**)
- employ **private** key U_{sk} for certain "**sensitive/critical**" tasks (performed by **user U**)

Assumption

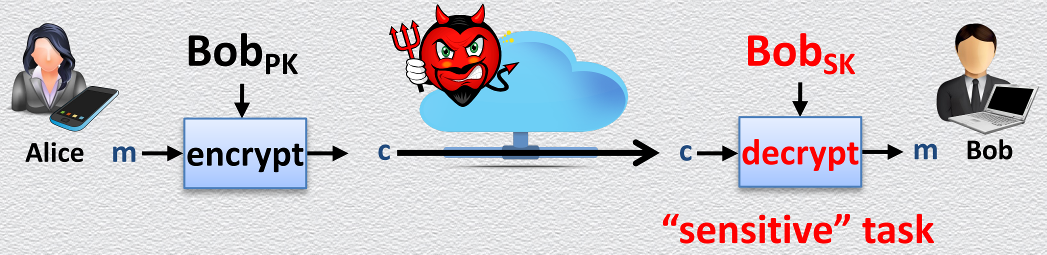
- **public-key infrastructure (PKI)**: public keys become **securely** available to users

From symmetric to asymmetric encryption

secret-key encryption

- main limitation
  - **session-specific** keys

public-key encryption



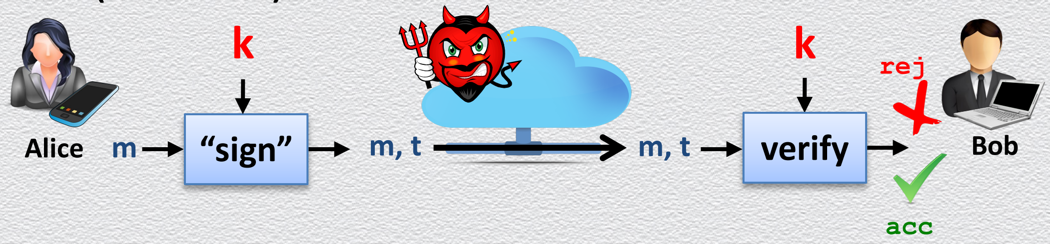
- main flexibility
  - **user-specific** keys

- messages encrypted by receiver's PK can (only) be decrypted by receiver's SK

From symmetric to asymmetric message authentication

secret-key message authentication (or MAC)

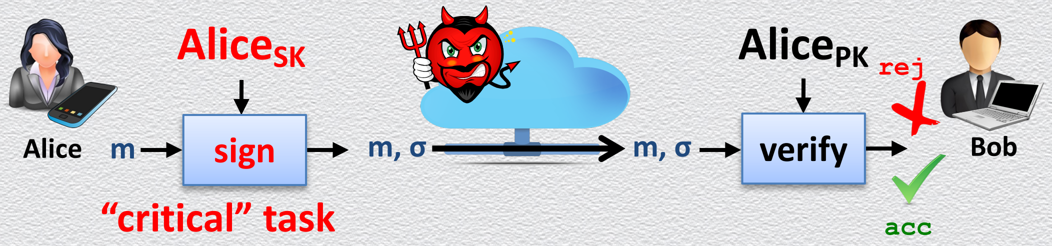
- main limitation
  - **session-specific** keys



public-key message authentication

(or **digital signatures**)

- main flexibility
  - **user-specific** keys



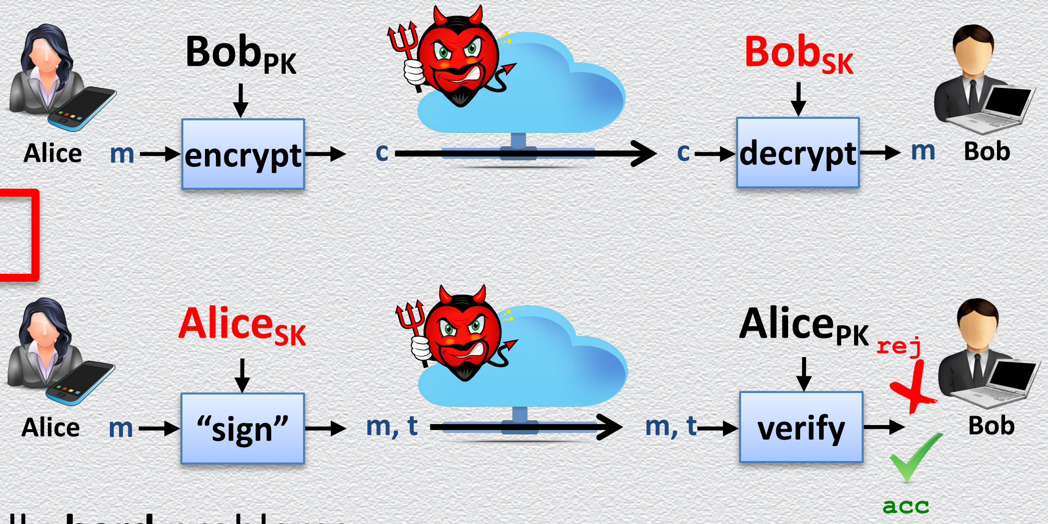
- (only) messages signed by sender’s SK can be verified by sender’s PK

Thus: Principles of modern cryptography

(A) security definitions, **(B) precise assumptions**, (C) formal proofs

For **asymmetric-key** message encryption/authentication

- adversary
  - types of attacks
- trusted set-up
  - PKI is needed
  - secret keys remain secret
- trust basis
  - underlying primitives are secure
  - typically, **algebraic** computationally-**hard** problems
    - e.g., **discrete log**, **factoring**, etc.



General comparison

Symmetric crypto

- key management
  - less scalable & riskier
- assumptions
  - secret & authentic communication
  - secure storage
- primitives
  - generic assumptions
  - more efficiently in practice

Asymmetric crypto

- key management
  - more scalable & simpler
- assumptions
  - authenticity (PKI)
  - secure storage
- primitives
  - math assumptions
  - less efficiently in practice (2-3 o.o.m.)

Public-key infrastructure (PKI)

A mechanism for securely managing, in a dynamic multi-user setting, user-specific public-key pairs (to be used by some public-key cryptosystem)

- **dynamic**, **multi-user**
  - the system is open to anyone; users can join & leave
- **user-specific public-key pairs**
  - each user U in the system is assigned a unique key pair (U_{pk}, U_{sk})
- **secure management** (e.g., authenticated public keys)
  - public keys are authenticated: current U_{pk} of user U is publicly known to everyone

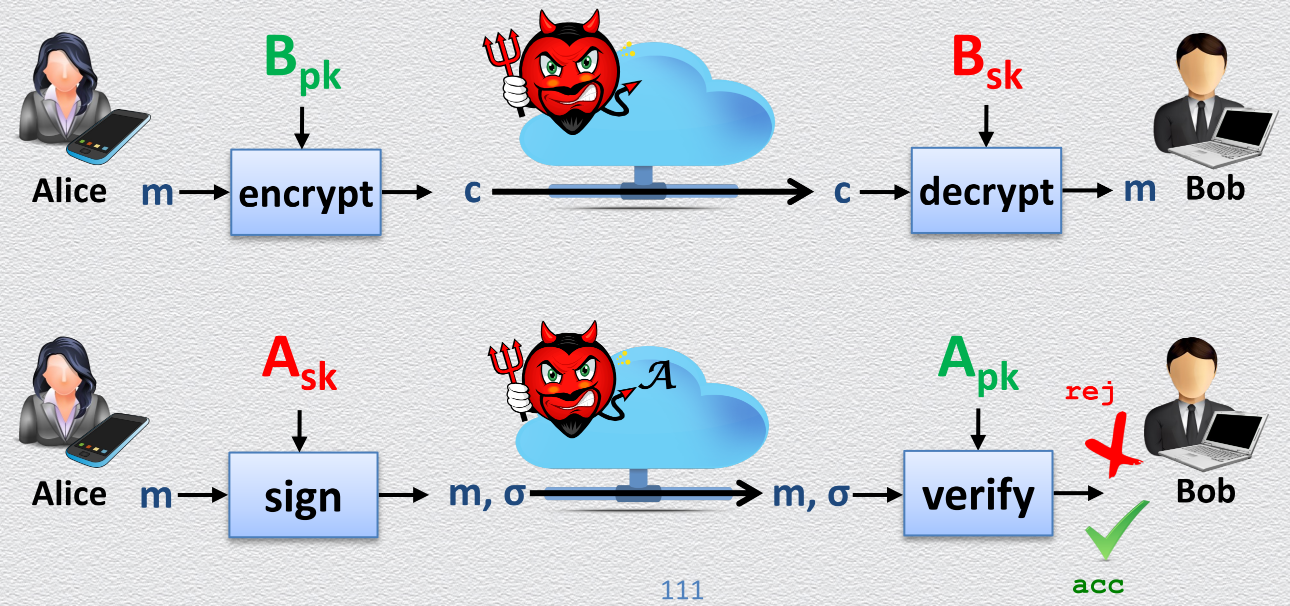
Very challenging to realize

- currently using **digital certificates**; ongoing research towards a better approach…

Overall: Public-key encryption & signatures

Assume a trusted set-up

- public keys are securely available (PKI) & secret keys remain secret



Secret-key vs. public-key encryption

	Secret Key (Symmetric)	Public Key (Asymmetric)
Number of keys	1	2
Key size (bits)	56–112 (DES), 128–256 (AES)	Unlimited; typically no less than 256; 1000 to 2000 currently considered desirable for most uses
Protection of key	Must be kept secret	One key must be kept secret; the other can be freely exposed
Best uses	Cryptographic workhorse. Secrecy and integrity of data, from single characters to blocks of data, messages and files	Key exchange, authentication, signing
Key distribution	Must be out-of-band	Public key can be used to distribute other keys
Speed	Fast	Slow, typically by a factor of up to 10,000 times slower than symmetric algorithms

Public-key cryptography: Early history

Proposed by Diffie & Hellman

- documented in “New Directions in Cryptography” (1976)
- solution concepts of public-key encryption schemes & digital signatures
- key-distribution systems
  - Diffie-Hellman key-agreement protocol
    - “reduces” symmetric crypto to asymmetric crypto

Public-key encryption was earlier (and independently) proposed by James Ellis

- classified paper (1970)
- published by the British Governmental Communications Headquarters (1997)
- concept of digital signature is still originally due to Diffie & Hellman