Cryptography Overview

Cryptography

- Is
  - A tremendous tool
  - The basis for many security mechanisms

- Is not
  - The solution to all security problems
  - Reliable unless implemented properly
  - Reliable unless used properly
  - Something you should try to invent or implement yourself

Kerckhoff's principle

A cryptosystem should be secure even if **everything** about the system, except the secret key, **is public knowledge**.

Goal 1:secure communication

Step 1: Session setup to exchange key

Step 2: encrypt data

HTTPS

Goal 2:   Protected files

Disk

Alice →→→ File 1 →→→ Alice

File 2

No eavesdropping
No tampering

Analogous to secure communication:
        Alice today sends a message to Alice tomorrow

Symmetric Cryptography

Assumes parties already share a secret key

Building block: sym. encryption

nonce

Alice

m, n → **E** → E(k,m,n)=c

Bob

c, n → **D** → D(k,c,n)=m

k (into E)

k (into D)
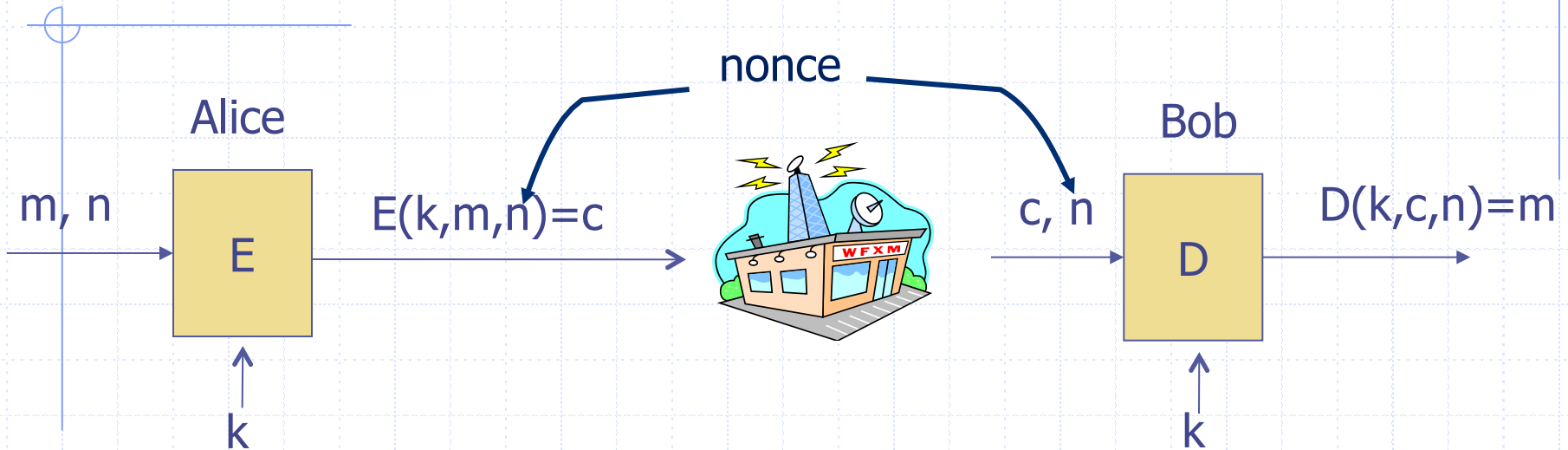
E, D: cipher   k: secret key (e.g. 128 bits)

m, c: plaintext, ciphertext   n: nonce   (aka IV)

Encryption algorithm is **publicly known**

- Never use a proprietary cipher

Use Cases

Single use key: (one time key)

- Key is only used to encrypt one message
    - encrypted email: new key generated for every email
- No need for nonce   (set to 0)

Multi use key: (many time key)

- Key used to encrypt multiple messages
    - files: same key used to encrypt many files

First example: One Time Pad
(single use key)

◆ **Vernam** (1917)

Key:	0	1	0	1	1	1	0	0	1	0

$\oplus$

Plaintext:	1	1	0	0	0	1	1	0	0	0

Ciphertext:	1	0	0	1	1	0	1	0	1	0

◆ **Shannon '49:**

- OTP is "secure" against ciphertext-only attacks

Stream ciphers (single use key)

Problem: OTP key is as long the message

Solution: Pseudo random key -- stream ciphers

$$c \leftarrow \text{PRG}(k) \oplus m$$
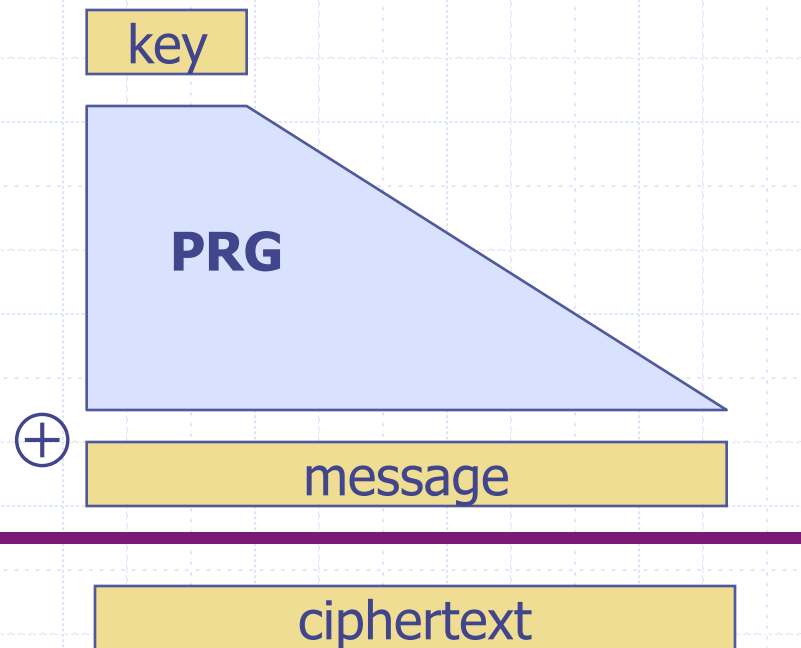
Stream ciphers: RC4 (126 MB/sec) , Salsa20/12 (643 MB/sec)

Dangers in using stream ciphers

One time key !!   "Two time pad" is insecure:

$$C_1 \leftarrow m_1 \oplus PRG(k)$$

$$C_2 \leftarrow m_2 \oplus PRG(k)$$

Eavesdropper does:

$$C_1 \oplus C_2 \rightarrow m_1 \oplus m_2$$

Enough redundant information in English that:

$$m_1 \oplus m_2 \rightarrow m_1 , m_2$$

Block ciphers: crypto work horse

n Bits

PT Block

E, D

n Bits

CT Block

Key

k Bits
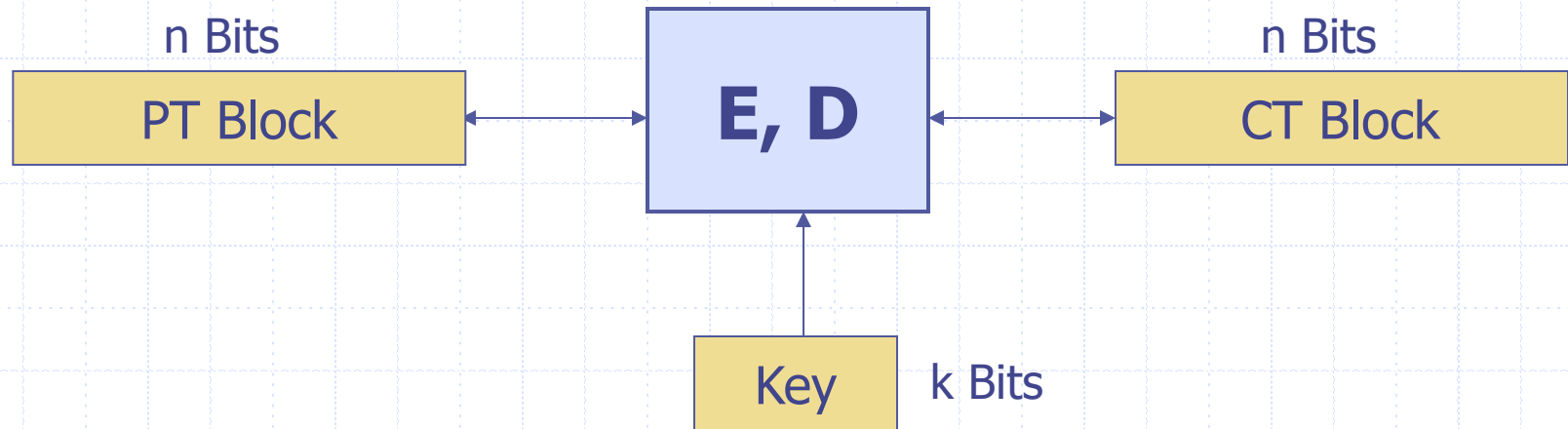
Canonical examples:

1. 3DES: n= 64 bits,   k = 168 bits

2. AES: n=128 bits,   k = 128, 192, 256 bits

IV handled as part of PT block

Building a block cipher

Input: (m, k)

Repeat simple "mixing" operation several times

- DES: Repeat 16 times:

$$\begin{cases} m_L \leftarrow m_R \\ m_R \leftarrow m_L \oplus F(k, m_R) \end{cases}$$

- AES-128:   Mixing step repeated 10 times

Difficult to design: must resist subtle attacks

- differential attacks, linear attacks, brute-force, …

Block Ciphers Built by Iteration
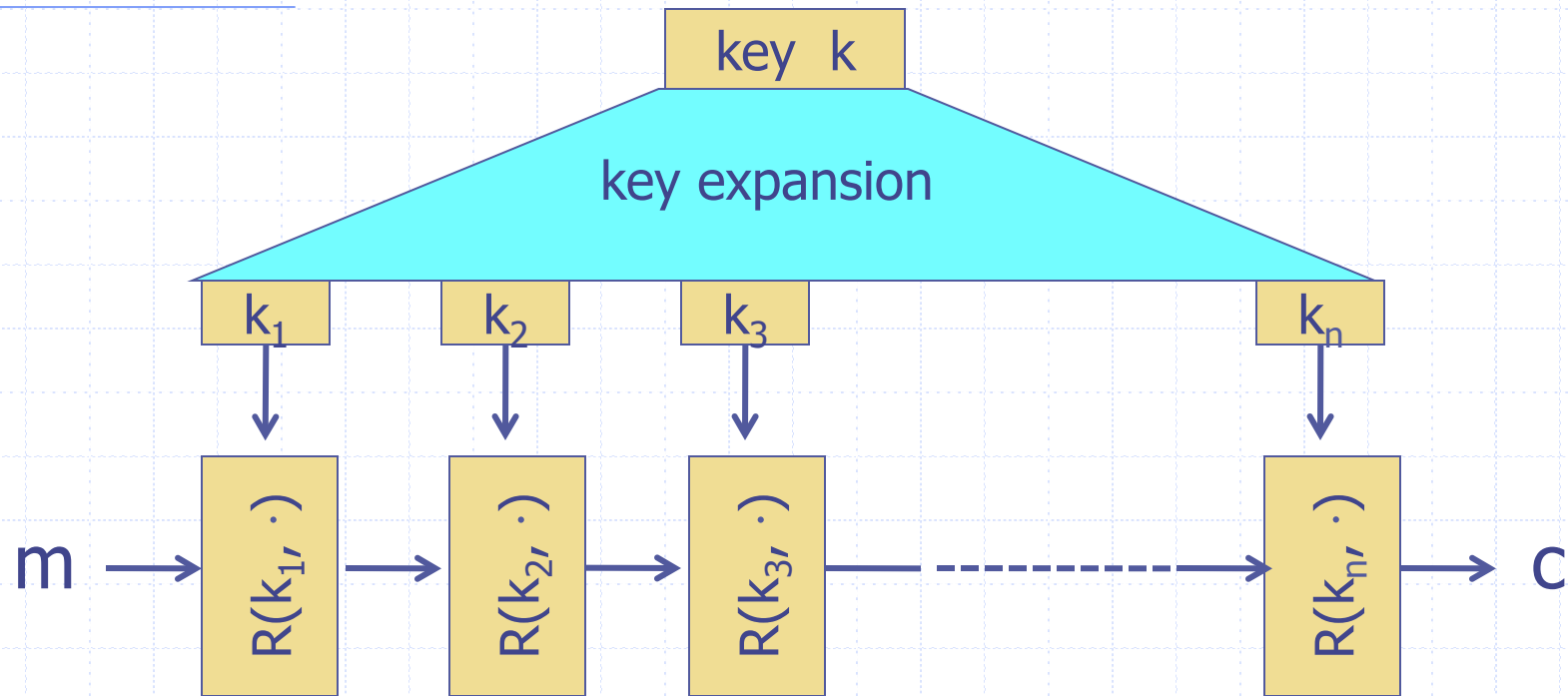
R(k,m):   round function
    for DES (n=16),   for AES-128 (n=10)

Incorrect use of block ciphers

Electronic Code Book (ECB):

PT: | | | m_1 | | m_2 | | | – · · – · | | | |

CT: | | | c_1 | | c_2 | | | – · · – · | | | |
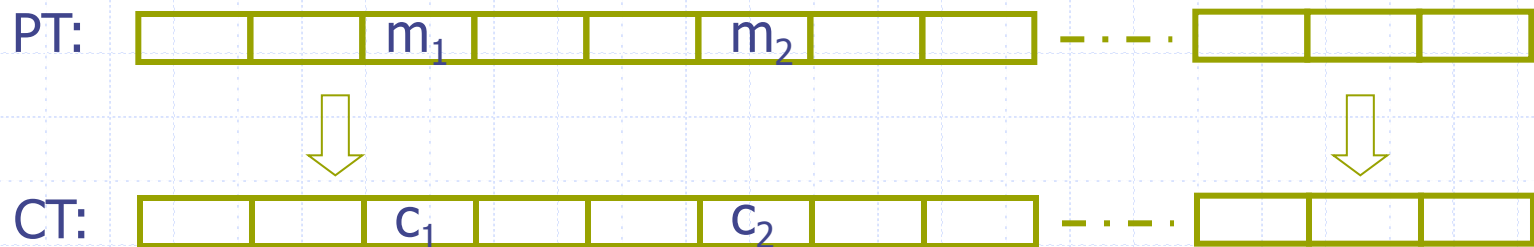
Problem:

- if $m_1 = m_2$ then $c_1 = c_2$

In pictures

An example plaintext

Encrypted with AES in ECB mode
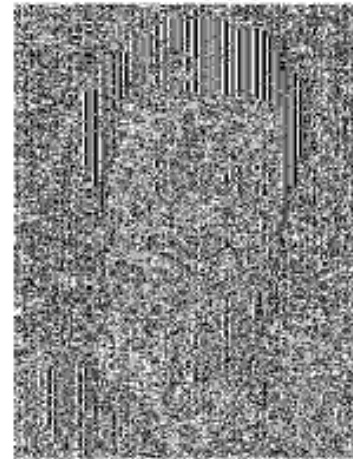
Correct use of block ciphers I: CBC mode

E a secure PRP.   Cipher Block Chaining with random IV:

ciphertext
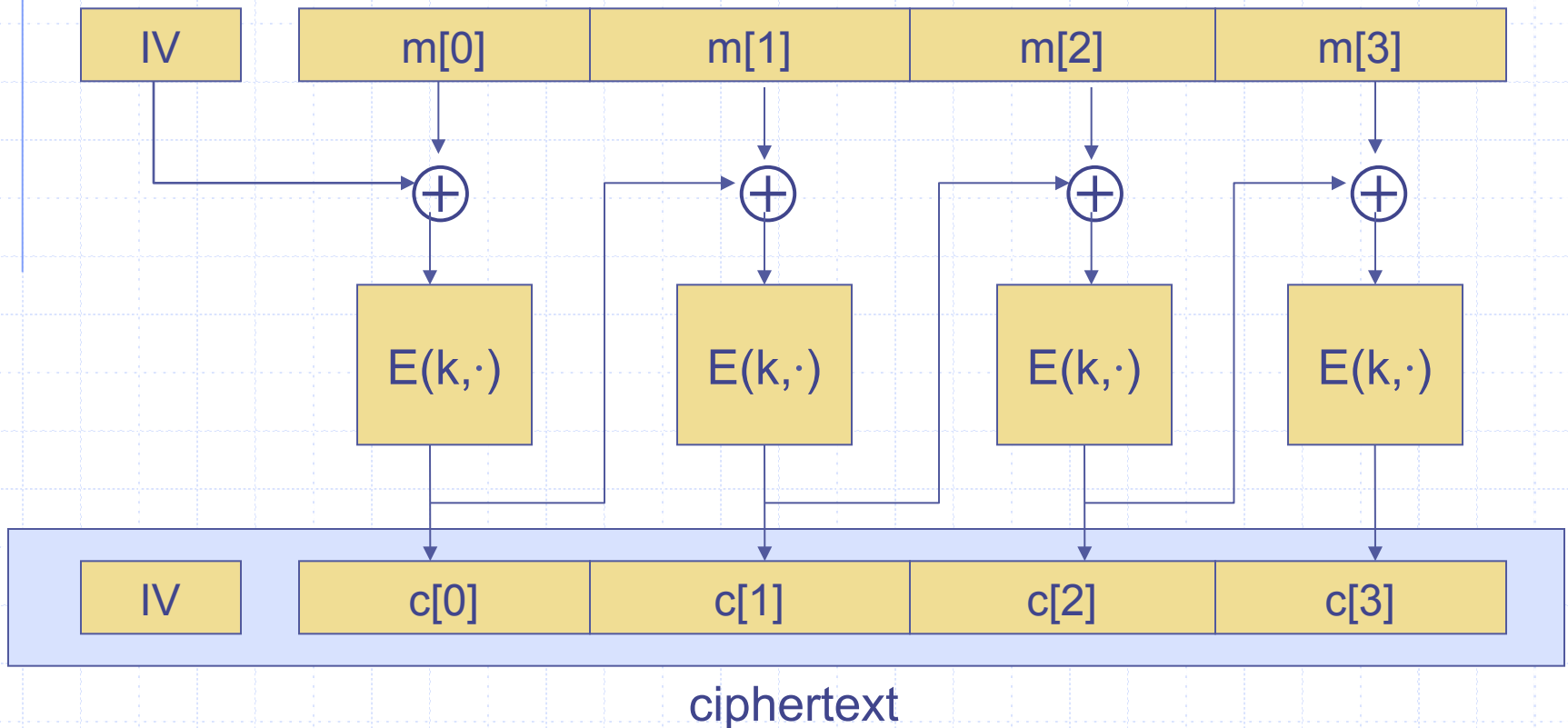
Q: how to do decryption?

Use cases: how to choose an IV

Single use key: no IV needed   (IV=0)

Multi use key: (CPA Security)

Best: use a fresh *random* IV for every message

Can use *unique* IV   (e.g counter)
but then first step in CBC <u>must be</u>   IV' ← $E(k_1, IV)$
benefit: may save transmitting IV with ciphertext

CBC with Unique IVs

unique IV means: (k,IV) pair is used for only one message.
generate unpredictable IV' as $E(k_1, IV)$

ciphertext
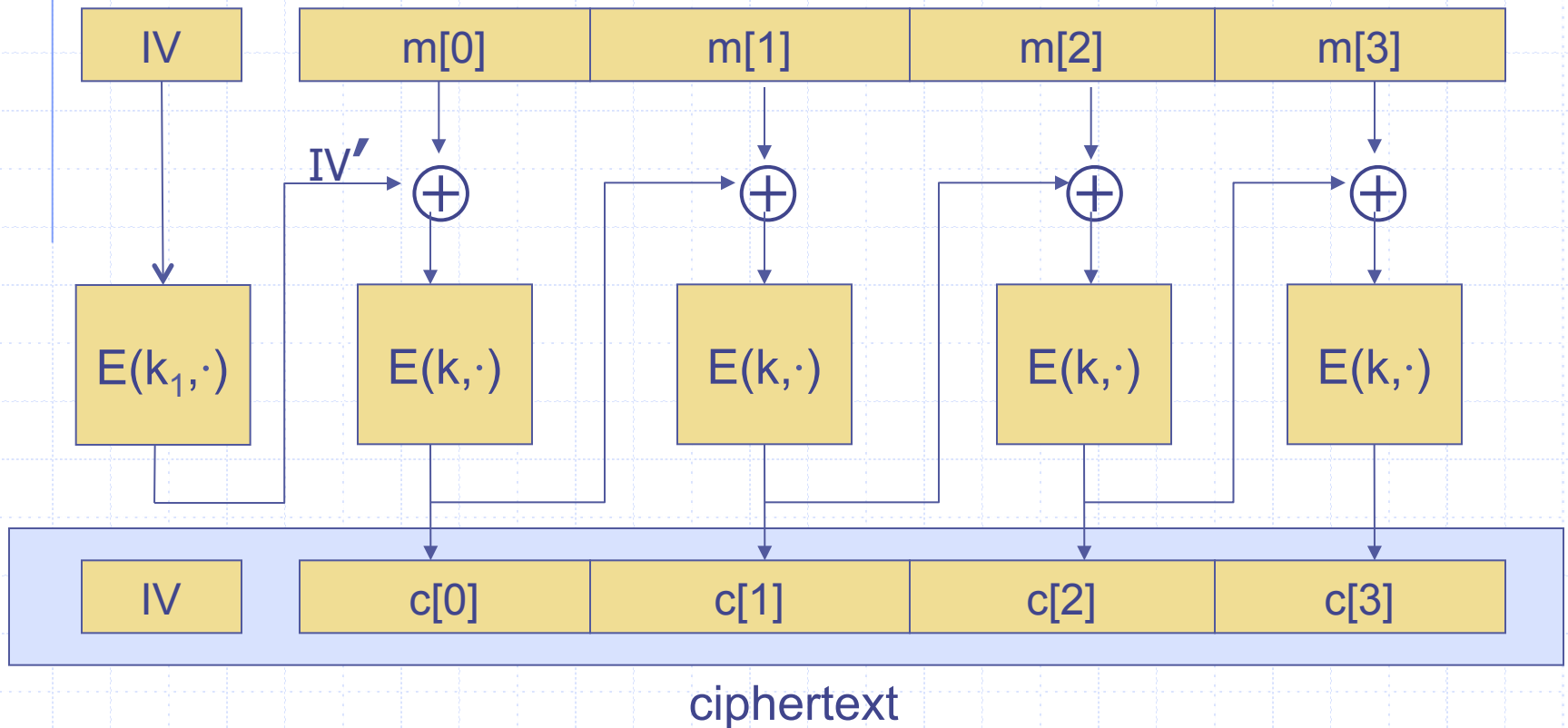
In pictures

An example plaintext

Encrypted with AES in CBC mode
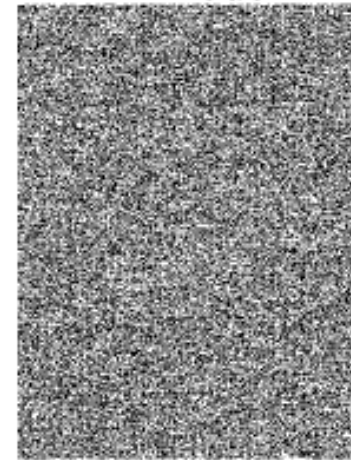
Correct use of block ciphers II: CTR mode

Counter mode with a random IV: (parallel encryption)

IV		m[0]	m[1]	…	m[L]

$\oplus$

	E(k,IV)	E(k,IV+1)	…	E(k,IV+L)

IV		c[0]	c[1]	…	c[L]

ciphertext
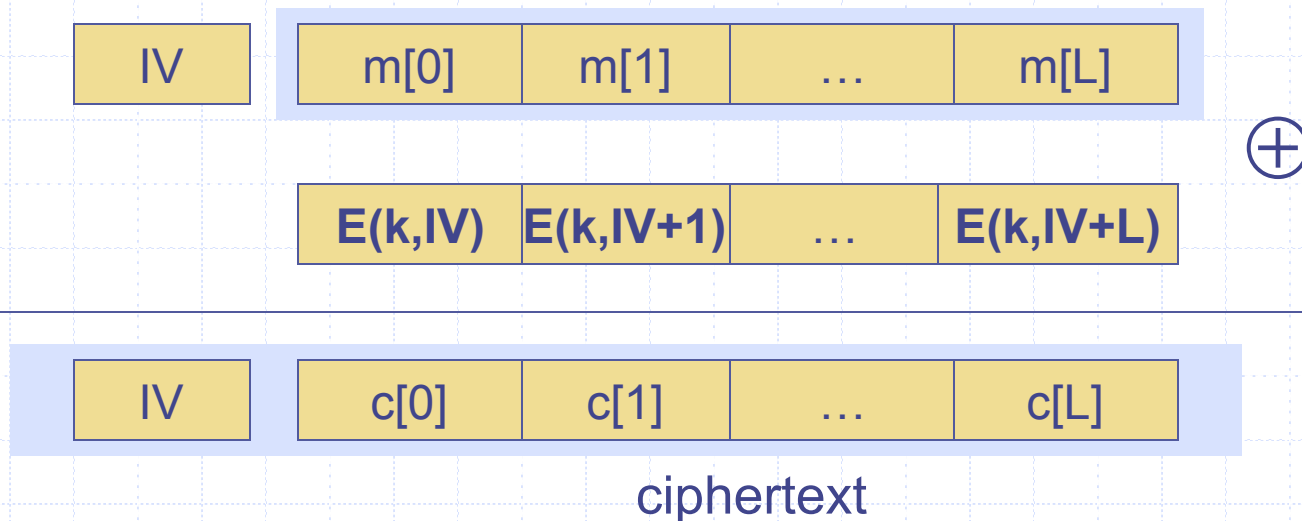
- Why are these modes secure?   not today.

2

Performance: Crypto++ 5.6.0 [Wei Dai]

Intel Core 2 (on Windows Vista)

Cipher	Block/key size	Speed (MB/sec)
RC4		126
Salsa20/12		643
3DES	64/168	10
AES/GCM	128/128	102

AES is about 8x faster with AES-NI : Intel Westmere and onwards

Data integrity

Message Integrity: MACs

◆ Goal: message integrity.   No confidentiality.
- ex: Protecting public binaries on disk.

k	Message m	tag	k
Alice		⟶	Bob

Generate tag: Verify tag: ?
  tag ← S(k, m)   V(k, m, tag) = `yes'
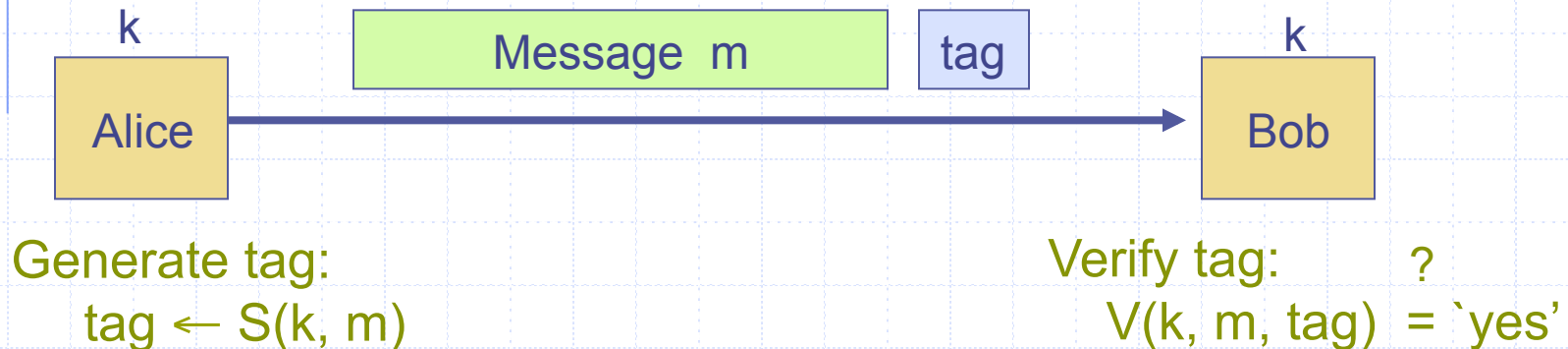
note: non-keyed checksum (CRC) is an insecure MAC !!

Secure MACs

◆ Attacker information: chosen message attack
  - for $m_1, m_2, \ldots, m_q$ attacker is given $t_i \leftarrow S(k, m_i)$

◆ Attacker's goal: existential forgery.
  - produce some **new** valid message/tag pair (m,t).

$$(m,t) \notin \{ (m_1, t_1), \ldots, (m_q, t_q) \}$$

◆ A secure PRF gives a secure MAC:
  - $S(k,m) = F(k,m)$
  - $V(k,m,t)$: `yes' if $t = F(k,m)$ and `no' otherwise.

Construction 1:   ECBC

Raw CBC

$$key = (k, k_1)$$

tag
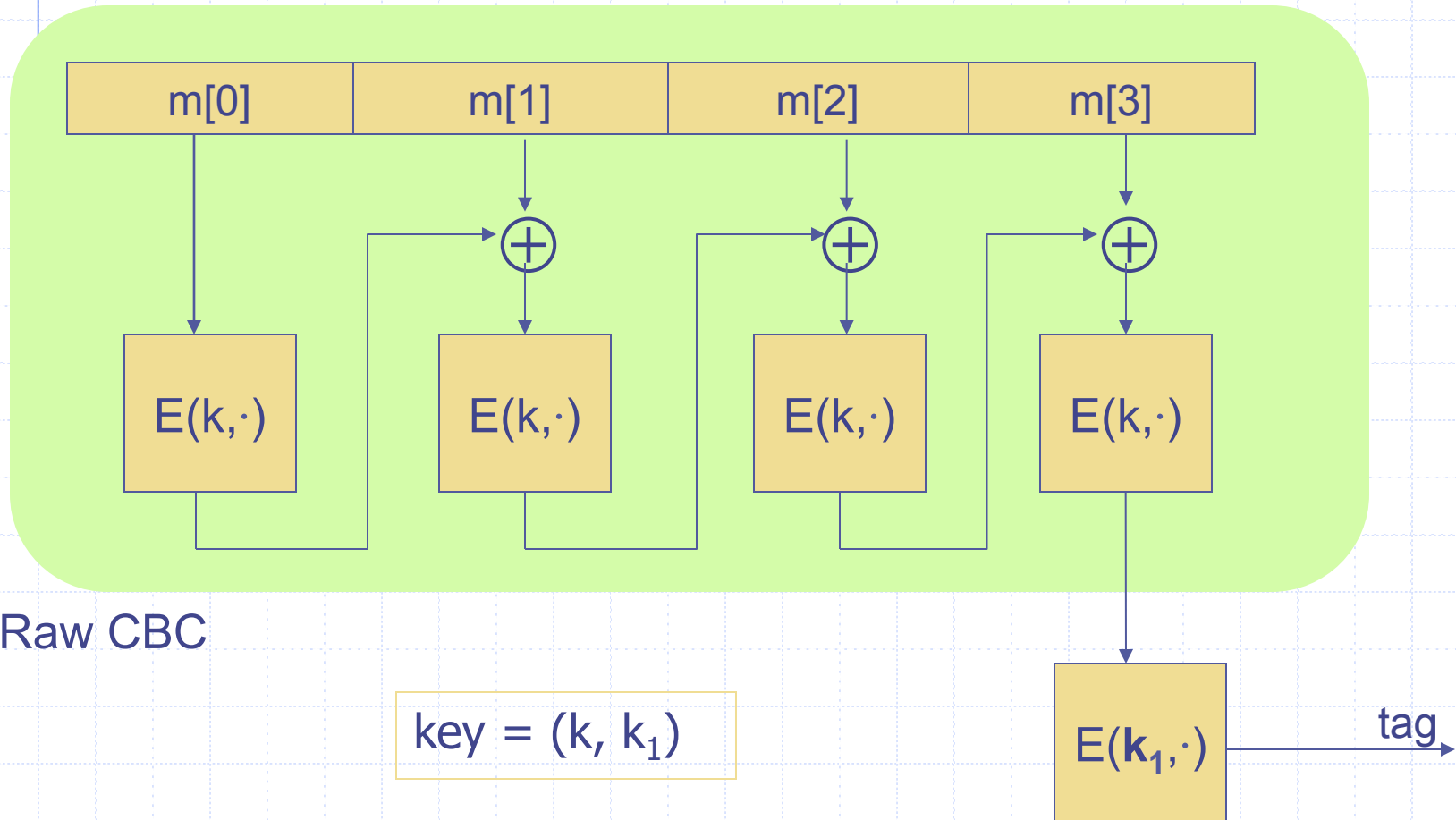
Construction 2:   HMAC (Hash-MAC)

Most widely used MAC on the Internet.

> H: hash function.
>   example: SHA-256   ;   output is 256 bits

Building a MAC out of a hash function:

> Standardized method: HMAC
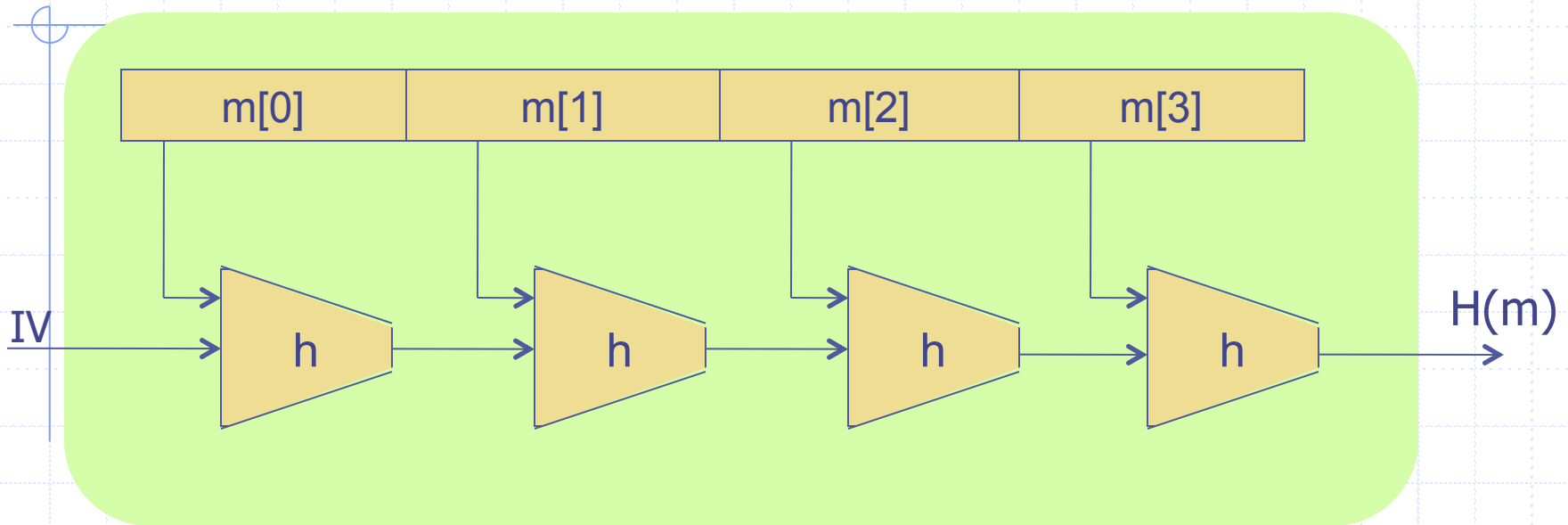> $S(k, m) = H(k \oplus opad \ \| \ \mathbf{H(k \oplus ipad \ \| \ m)})$

SHA-256:   Merkle-Damgard

h(t, m[i]): compression function

Thm 1:   if h is collision resistant then so is H

"Thm 2":   if h is a PRF then HMAC is a PRF

Construction 3: PMAC – parallel MAC

ECBC and HMAC are sequential.   PMAC:

m[0]	m[1]	m[2]	m[3]

$P(k,0) \to \oplus$ $P(k,1) \to \oplus$ $P(k,2) \to \oplus$ $P(k,3) \to \oplus$

$F(k,\cdot)$ $F(k,\cdot)$ $F(k,\cdot)$ $F(k,\cdot)$

$\oplus$

$F(\mathbf{k_1},\cdot) \to$ tag
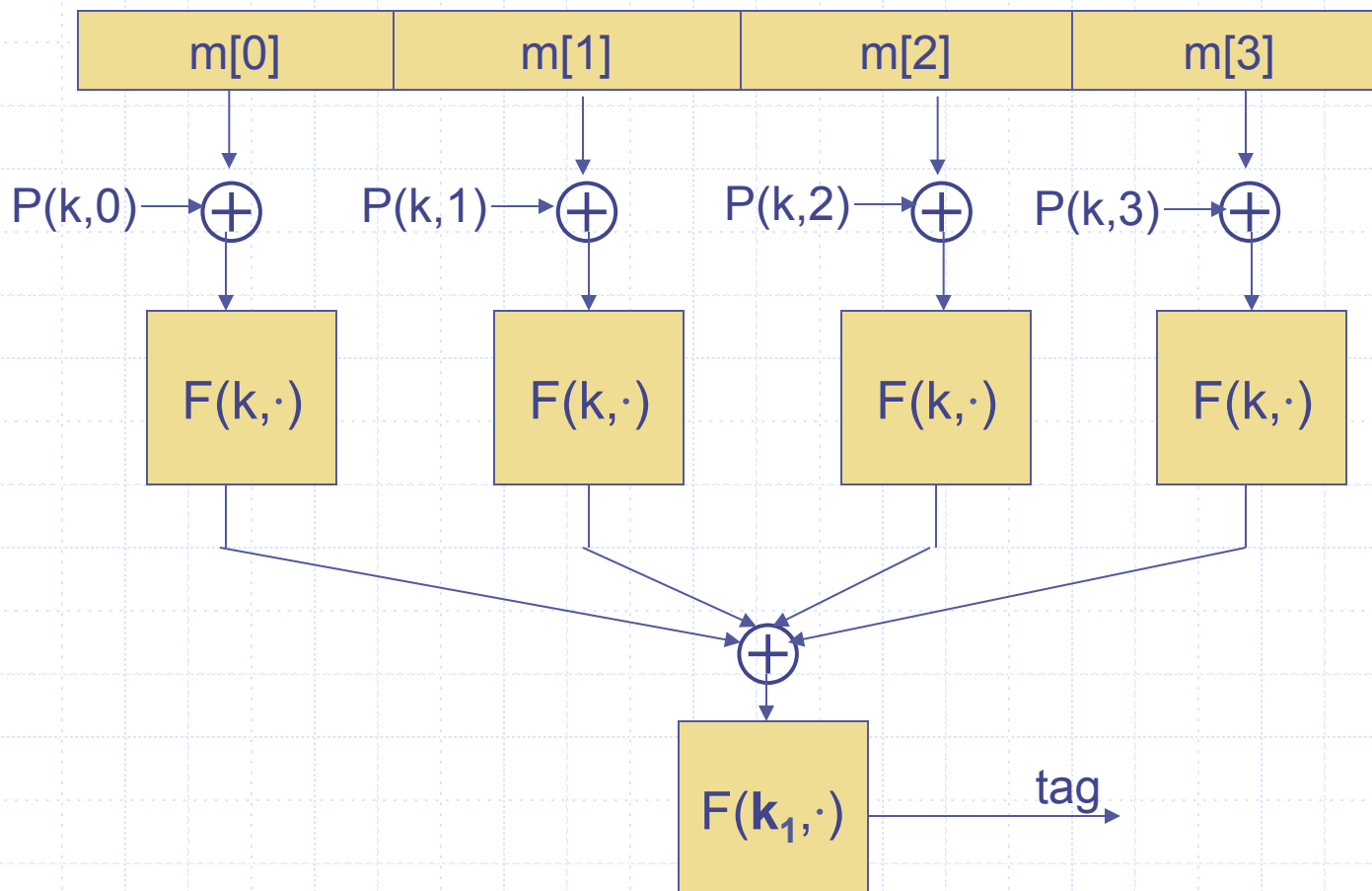
Why are these MAC constructions secure?

    … not today – take CS255

Why the last encryption step in ECBC?

- CBC (aka Raw-CBC) is not a secure MAC:
    - Given tag on a message m, attacker can deduce tag for some other message m'
    - How: good crypto exercise …

3

Authenticated Encryption:
Encryption + MAC

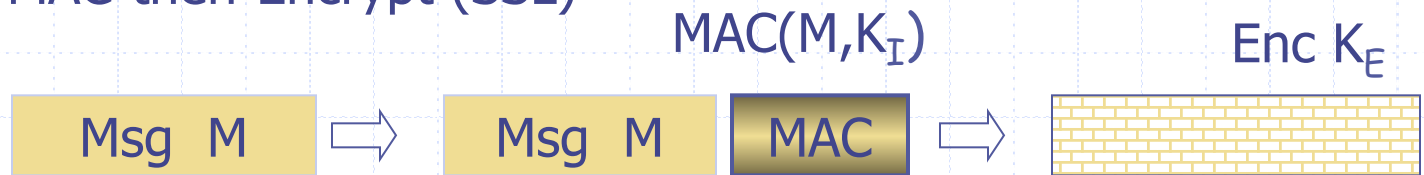# Combining MAC and ENC   (CCA)

Encryption key K_E MAC key = K_I

Option 1: MAC-then-Encrypt (SSL)

$MAC(M, K_I)$ Enc K_E

| Msg M | $\Rightarrow$ | Msg M | MAC | $\Rightarrow$ | |

Option 2: Encrypt-then-MAC (IPsec)

Enc K_E $MAC(C, K_I)$
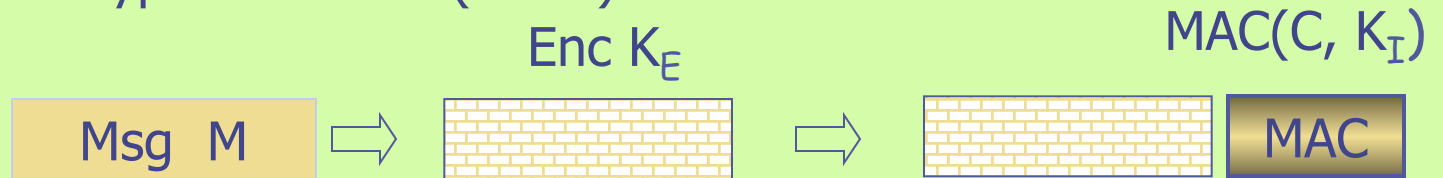
Secure for all secure primitives

| Msg M | $\Rightarrow$ | | $\Rightarrow$ | | MAC |

Option 3:   Encrypt-and-MAC (SSH)

Enc K_E $MAC(M, K_I)$

| Msg M | $\Rightarrow$ | | $\Rightarrow$ | | MAC |

OCB
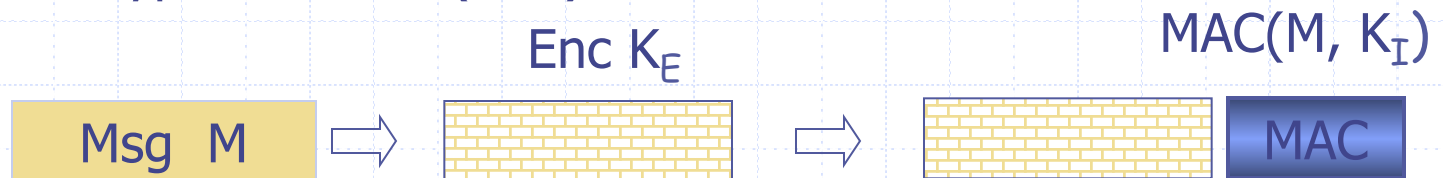
More efficient authenticated encryption
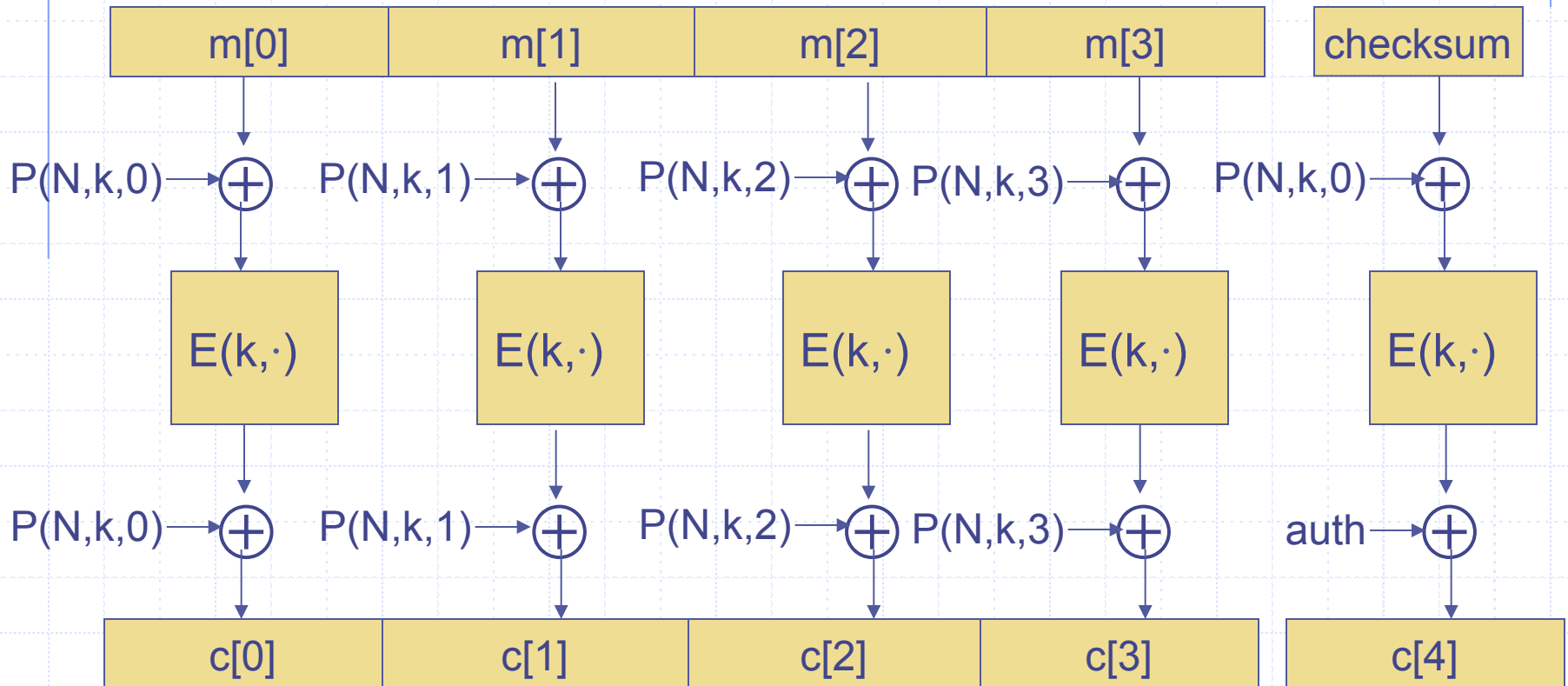
Rogaway, …

Public-key Cryptography

Public key encryption: (Gen, E, D)
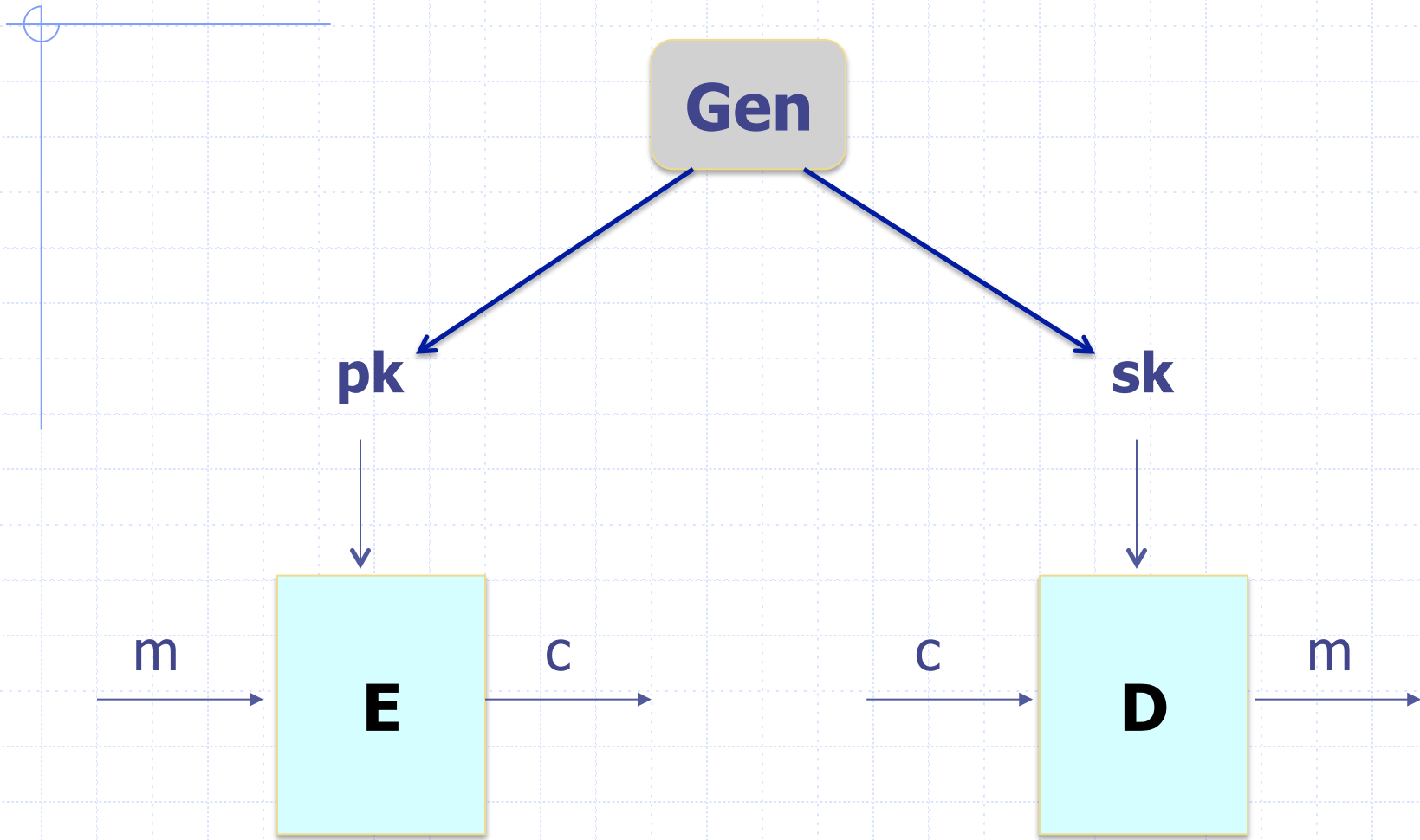
Applications

Session setup   (for now, only eavesdropping security)

Alice

Generate (pk, sk)

x

pk →

← E(pk, x)

Bob

choose random
x
(e.g. 48 bytes)
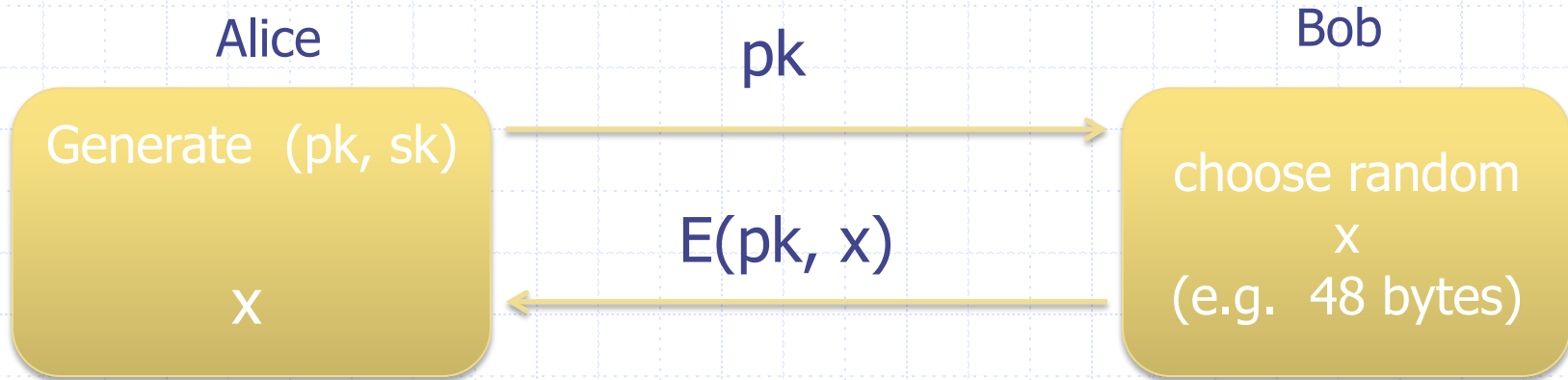
Non-interactive applications: (e.g. Email)

◆ Bob sends email to Alice encrypted using pk_{alice}

◆ Note: Bob needs pk_{alice} (public key management)

Applications

Encryption in non-interactive settings:

- ◆ Encrypted File Systems

write → Storage ← read

sk$_A$

Alice

Bob

File

$E(k_F,\ File)$

$E(pk_A,\ K_F)$

$E(pk_B,\ K_F)$
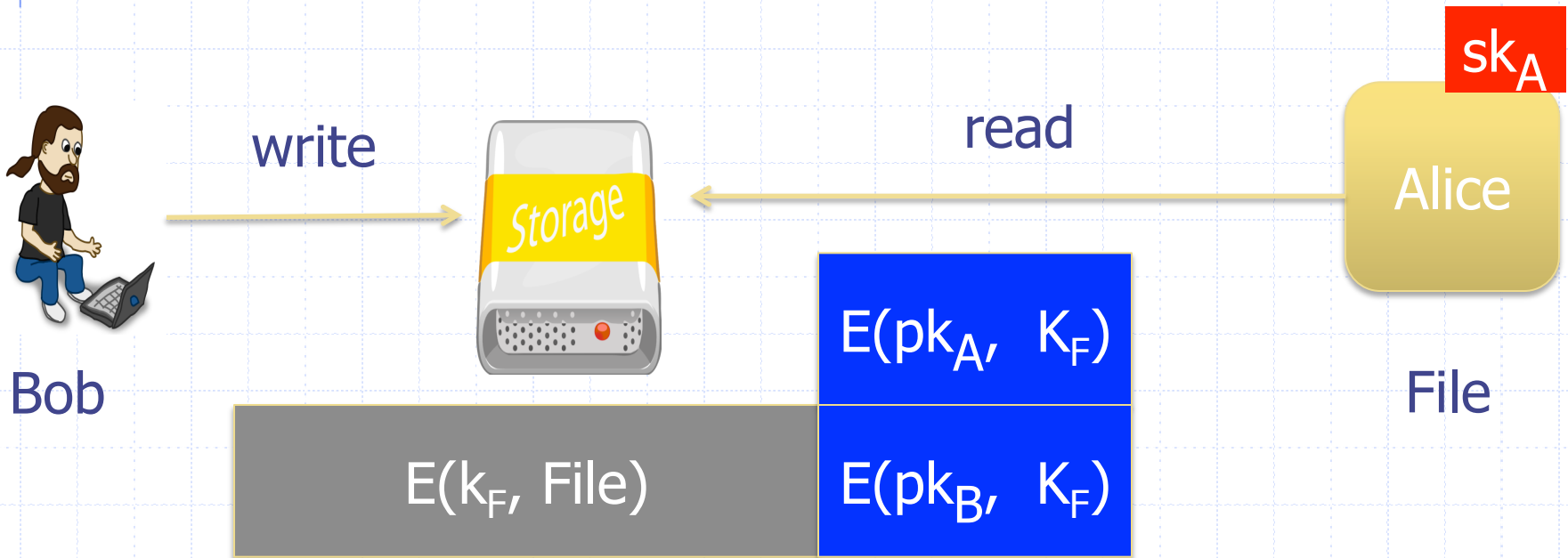
Applications

Encryption in non-interactive settings:

- ◆ Key escrow: data recovery without Bob's key

write

Storage

Bob

Escrow Service

sk_{escrow}

$E(pk_{escrow},\ K_F)$

$E(k_F, File)$

$E(pk_B,\ K_F)$
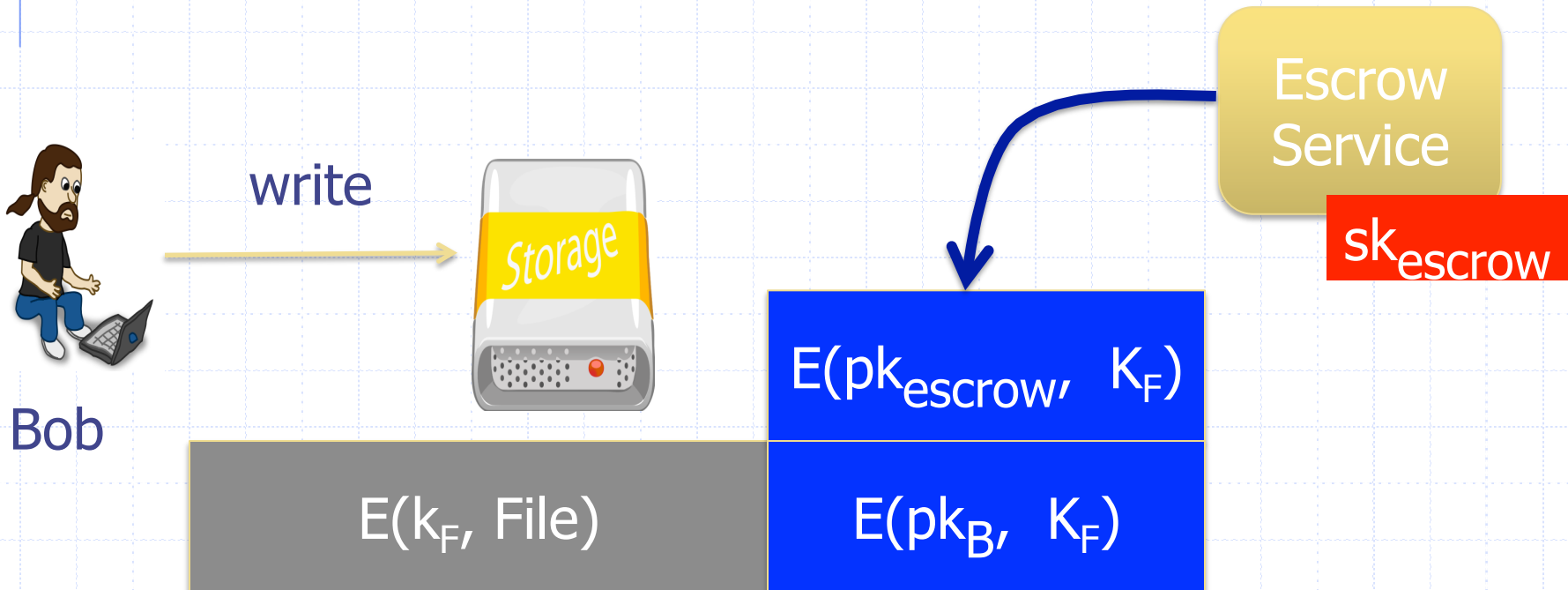
Trapdoor functions (TDF)

<u>Def</u>:   a trapdoor func. $X \longrightarrow Y$ is a triple of efficient algs. (G, F, F^{-1})

- G(): randomized alg. outputs key pair   (pk, sk)

- F(pk, ·):   det. alg. that defines a func. $X \longrightarrow Y$

- F^{-1}(sk, ·):   defines a func. $Y \longrightarrow X$ that
                          inverts   F(pk, ·)

Security: F(pk, ·) is one-way without sk

Public-key encryption from TDFs

- (G, F, F^{-1}): secure TDF $X \longrightarrow Y$
- (E_s, D_s) : symm. auth. encryption with keys in K
- H: $X \longrightarrow K$ a hash function

We construct a pub-key enc. system (G, E, D):

Key generation G: same as G for TDF

Public-key encryption from TDFs

- (G, F, F^{-1}):   secure TDF $X \longrightarrow Y$
- (E_s, D_s) :   symm. auth. encryption with keys in K
- $H: X \longrightarrow K$ a hash function

E(pk, m) :

$x \xleftarrow{R} X,$ $y \leftarrow F(pk, x)$

$k \leftarrow H(x),$ $c \leftarrow E_s(k, m)$

output (y, c)

D(sk, (y,c)) :

$x \leftarrow F^{-1}(sk, y),$

$k \leftarrow H(x),$ $m \leftarrow D_s(k, c)$

output   m

In pictures:

F(pk, x)	$E_s($ H(x), m $)$

header　　　　　　　　　　　　　body
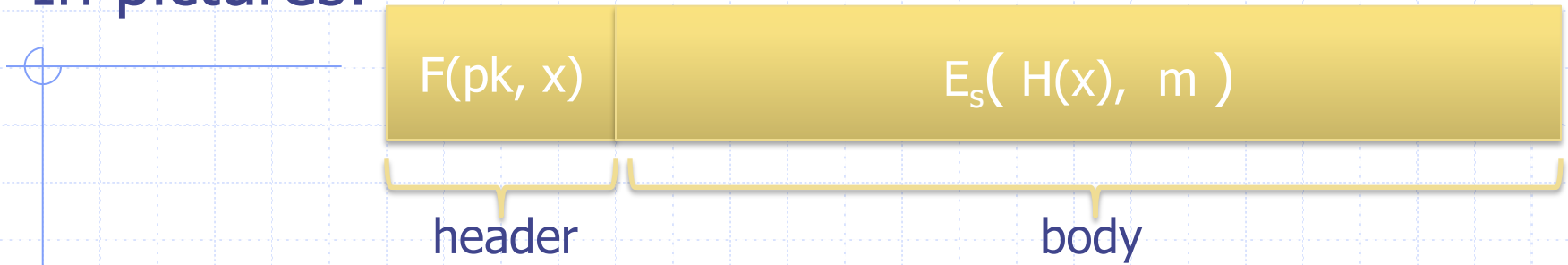
Security Theorem:

If **(G, F, F^{-1})** is a secure TDF,

(E$_s$, D$_s$) provides auth. enc.

and   **H:** $X \longrightarrow K$ is a   "random oracle"

then   **(G,E,D)**   is CCA^{ro} secure.

Digital Signatures

◆ Public-key encryption

  - Alice publishes encryption key

  - Anyone can send encrypted message

  - Only Alice can decrypt messages with this key

◆ Digital signature scheme

  - Alice publishes key for verifying signatures

  - Anyone can check a message signed by Alice

  - Only Alice can send signed messages

Digital Signatures from TDPs

◆ (G, F, F^{-1}):   secure TDP   X $\rightarrow$ X

◆ H: M $\rightarrow$ X   a hash function

Sign(sk, m$\in$X) :
  output

  $\text{sig} = F^{-1}\big(\text{sk, } H(m)\big)$

Verify(pk, m, sig) :
  output
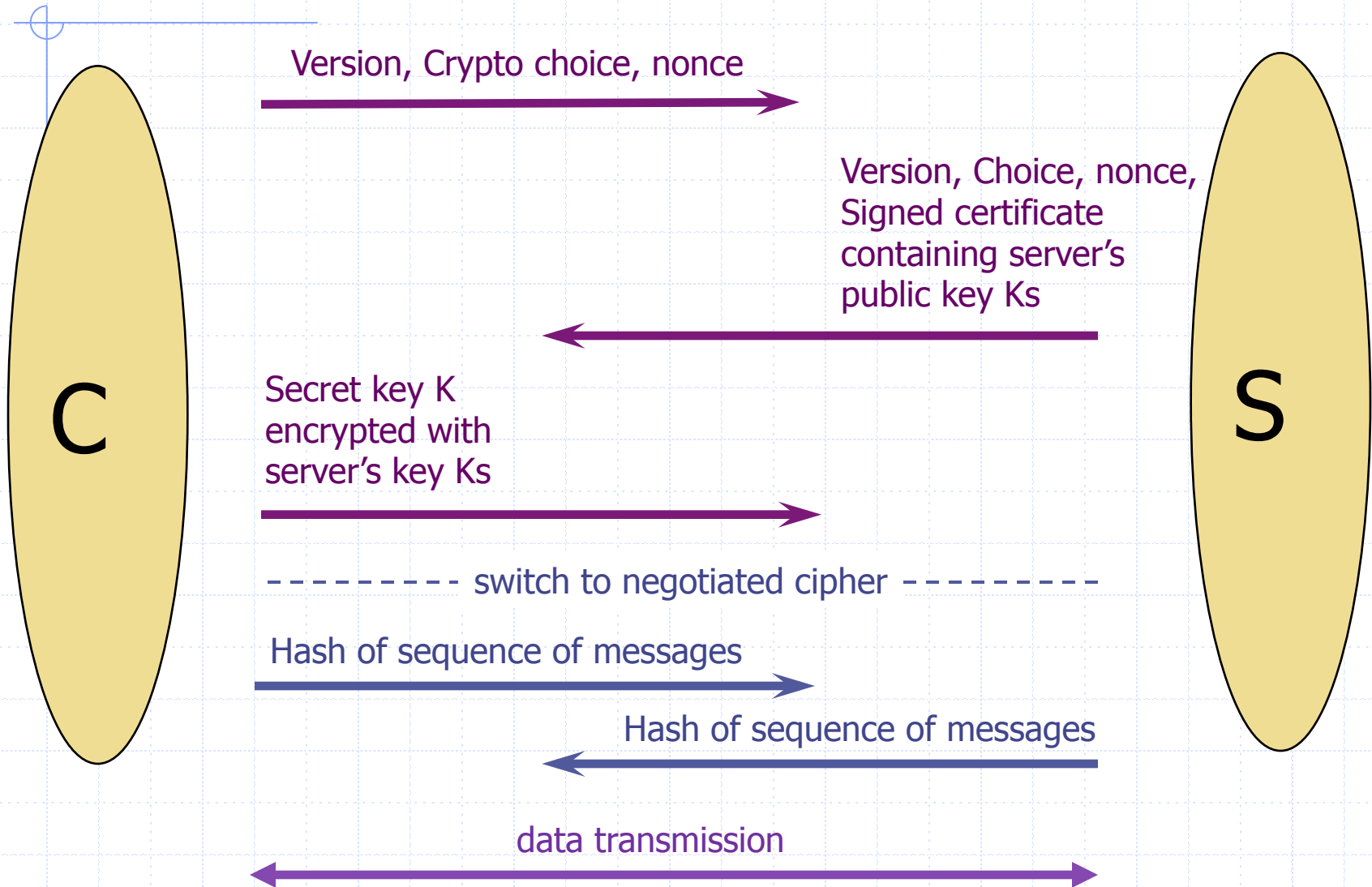
  $\begin{cases} 1 & \text{if } H(m) = F(pk, sig) \\ 0 & \text{otherwise} \end{cases}$

Security: existential unforgeability under a chosen message
        attack   (in the random oracle model)

Public-Key Infrastructure (PKI)

◆ Anyone can send Bob a secret message
- Provided they know Bob's public key

◆ How do we know a key belongs to Bob?
- If imposter substitutes another key, can read Bob's mail

◆ One solution: PKI
- Trusted root Certificate Authority (e.g. Symantec)
  - Everyone must know the verification key of root CA
  - Check your browser; there are hundreds!!
- Root authority signs intermediate CA
- Results in a certificate chain

Back to SSL/TLS

C → **S:** Version, Crypto choice, nonce

S → **C:** Version, Choice, nonce, Signed certificate containing server's public key K_s

C → **S:** Secret key K encrypted with server's key K_s

––––––––– switch to negotiated cipher –––––––––

C → **S:** Hash of sequence of messages

S → **C:** Hash of sequence of messages

C ↔ **S:** data transmission

Limitations of cryptography

Cryptography works when used correctly !!

... but is not the solution to all security problems

XKCD 538