CS155

Cryptography Overview

Acknowledgments: Lecture slides are from the Computer Security course taught by Dan Boneh and John Mitchell at Stanford University. When slides are obtained from other sources, a a reference will be noted on the bottom of that slide. A full list of references is provided on the last slide.

Cryptography

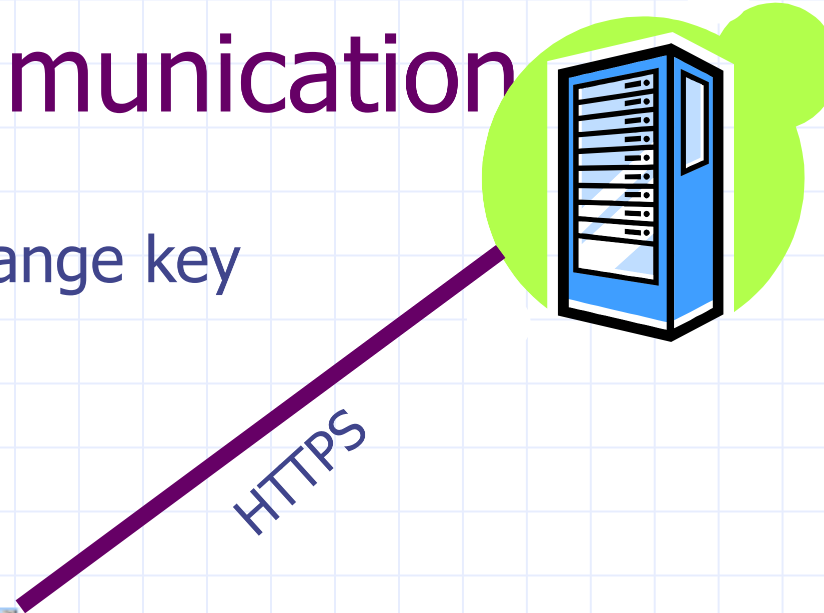
- Is
  - A tremendous tool
  - The basis for many security mechanisms
- Is not
  - The solution to all security problems
  - Reliable unless implemented properly
  - Reliable unless used properly
  - Something you should try to invent or implement yourself

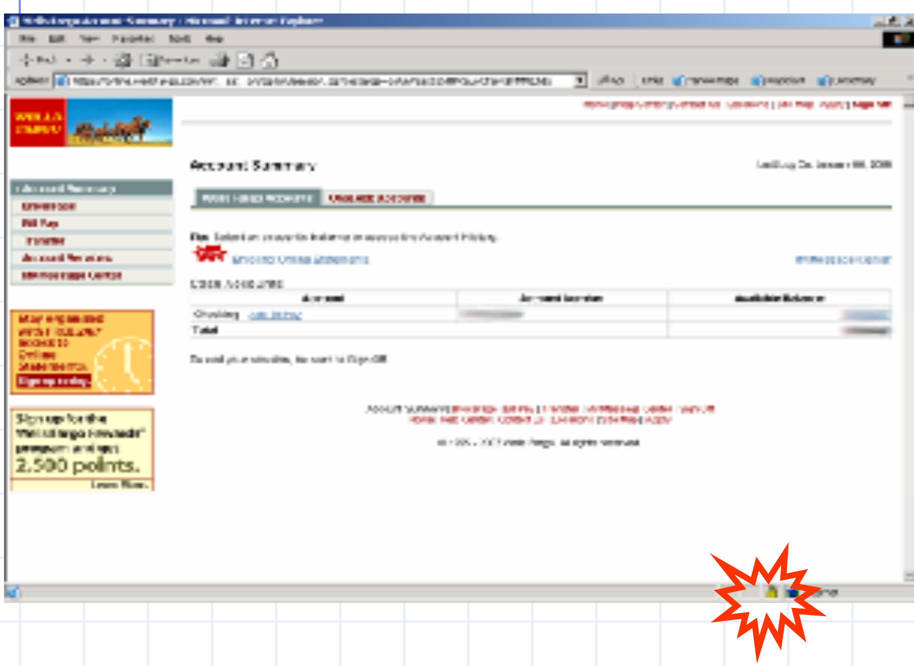
Kerckhoff’s principle

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

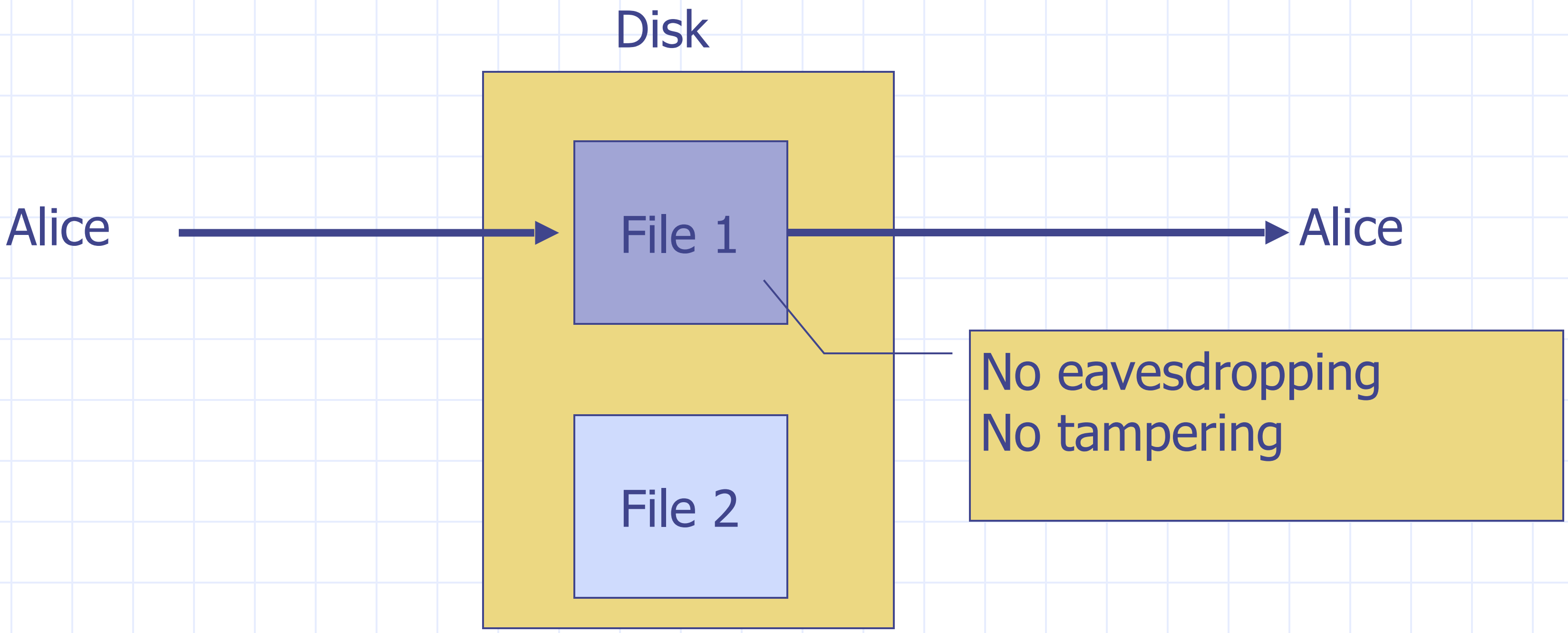
Goal 1:secure communication

- Step 1: Session setup to exchange key
- Step 2: encrypt data





Goal 2: Protected files



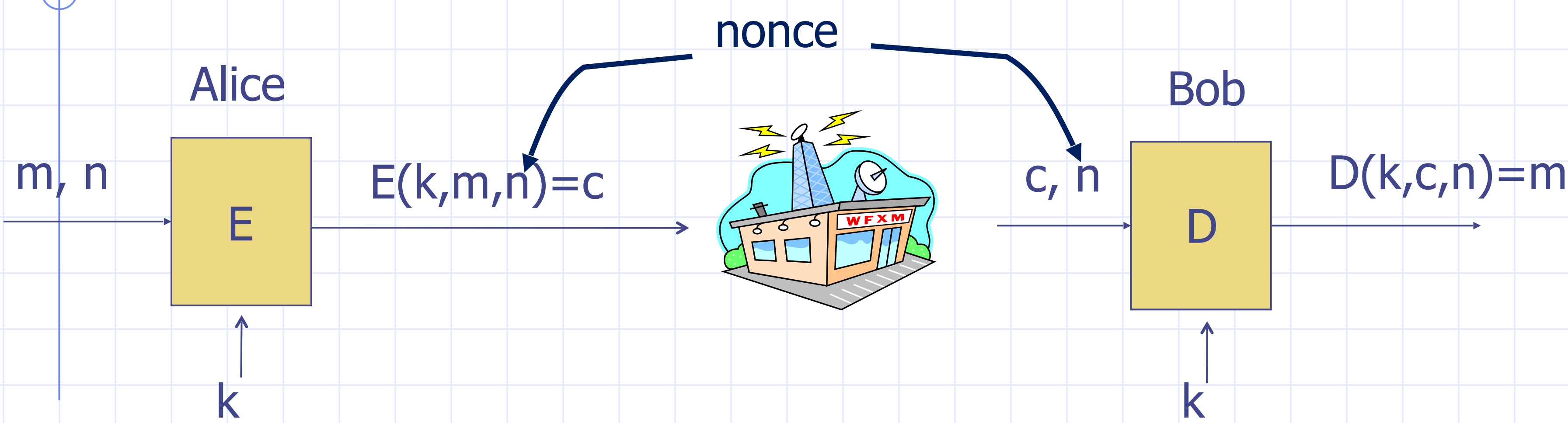
Analogous to secure communication:

Alice today sends a message to Alice tomorrow

Symmetric Cryptography

Assumes parties already share a secret key

Building block: sym. encryption



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

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

Encryption algorithm is **<u>publicly known</u>**

- 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
Plaintext:	1	1	0	0	0	1	1	0	0	0
Ciphertext:	1	0	0	1	1	0	1	0	1	0

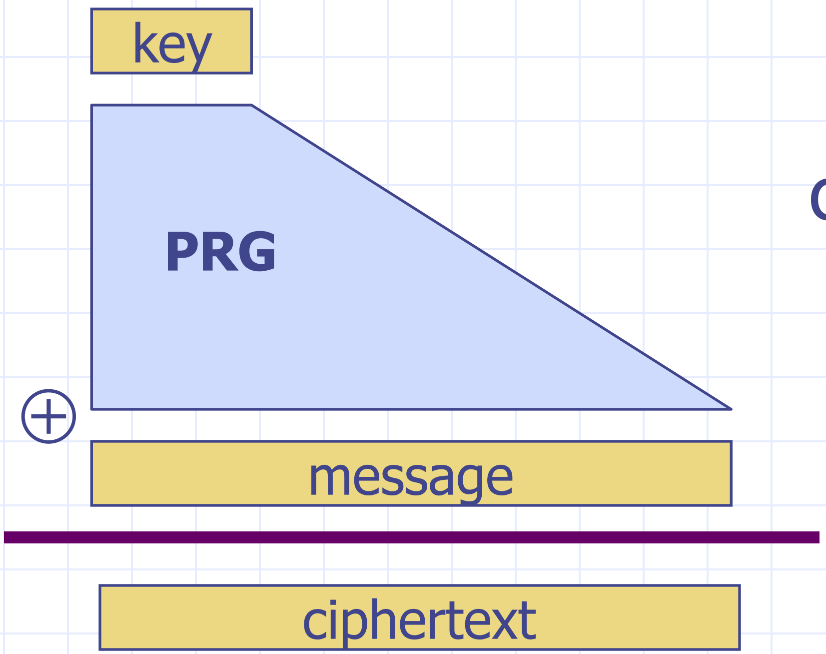
$\oplus$

- Shannon '49:
  - OTP is "secure" against ciphertext-only attacks

Stream ciphers (single use key)

Problem: OTP key is not as long as 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:

$$\begin{cases} C_1 \leftarrow m_1 \oplus PRG(k) \\ C_2 \leftarrow m_2 \oplus PRG(k) \end{cases}$$

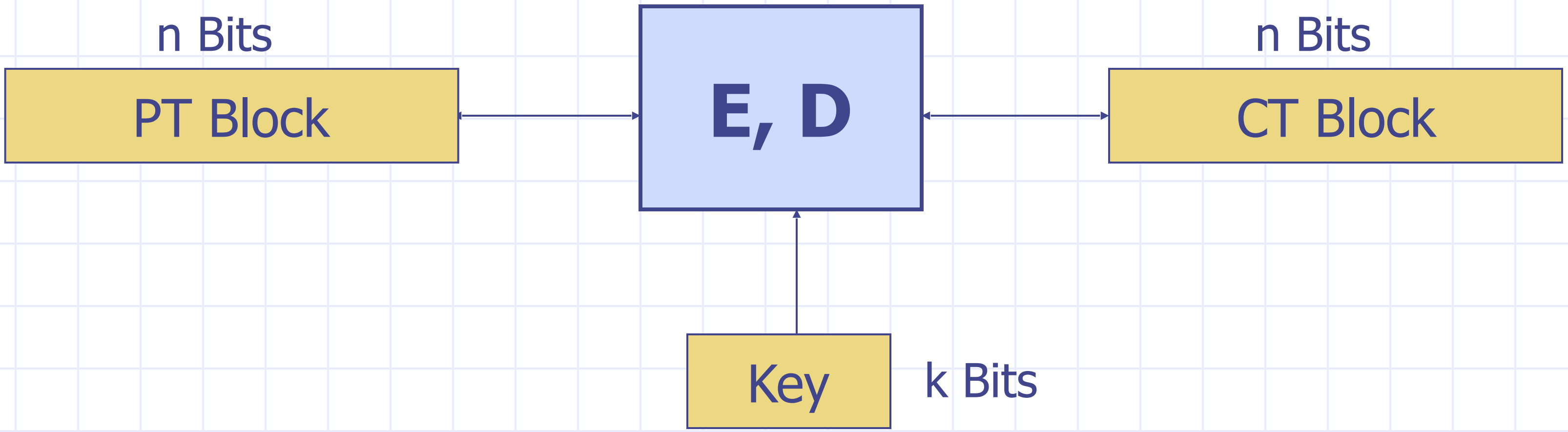
Eavesdropper does:

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

Enough redundant information in English that:

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

Block ciphers: crypto work horse



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:

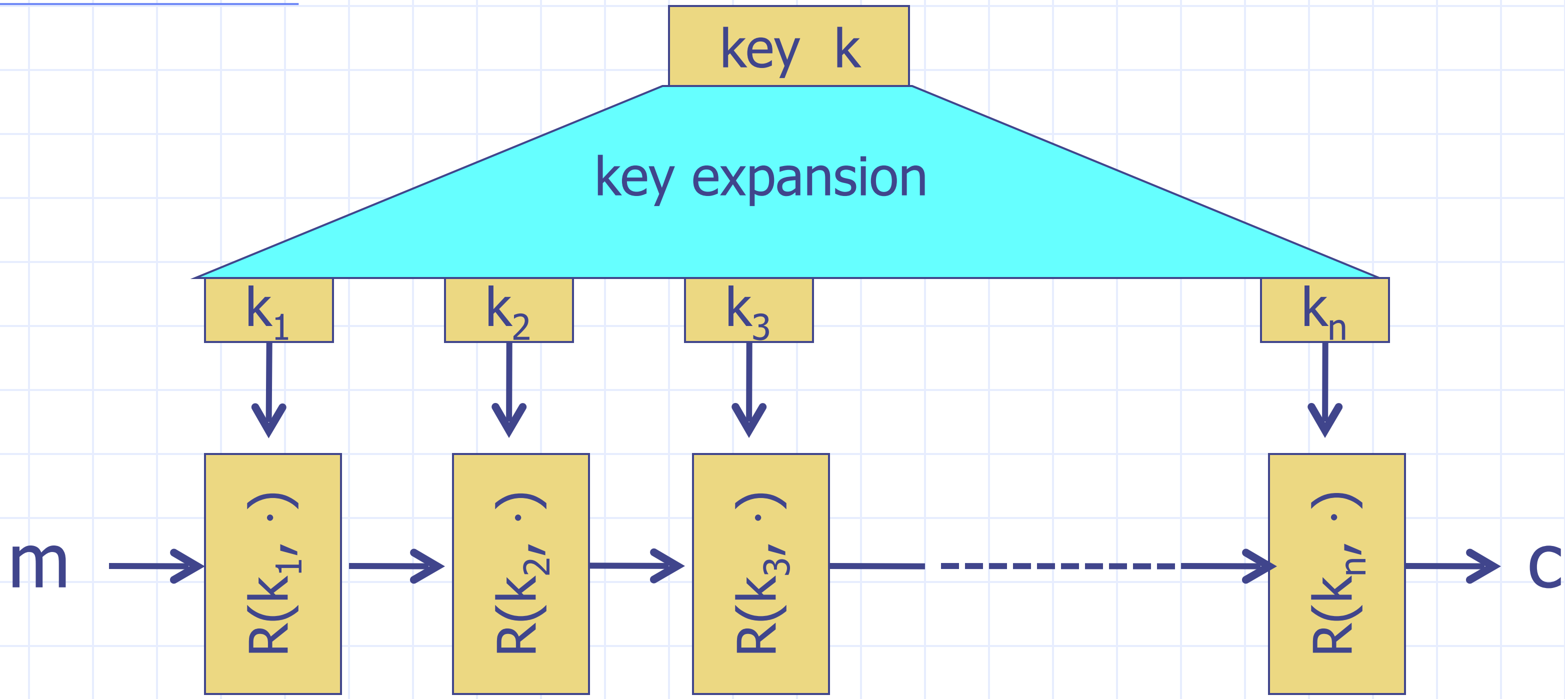
$$\begin{cases} m_L \leftarrow m_R \\ m_R \leftarrow m_L \oplus F(k, m_R) \end{cases}$$

- 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

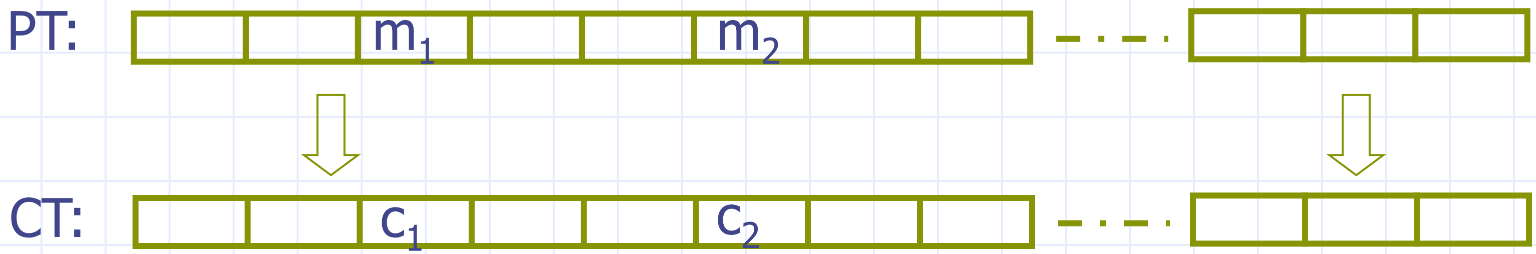


R(k,m): round function

for DES (n=16), for AES-128 (n=10)

Incorrect use of block ciphers

Electronic Code Book (ECB):



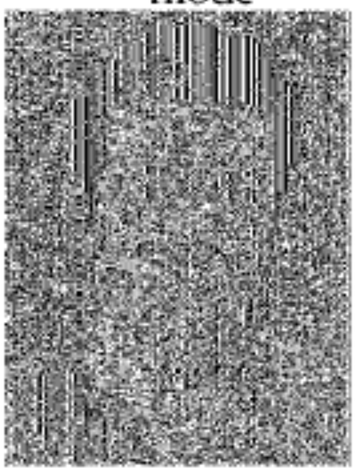
Problem:

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

In pictures

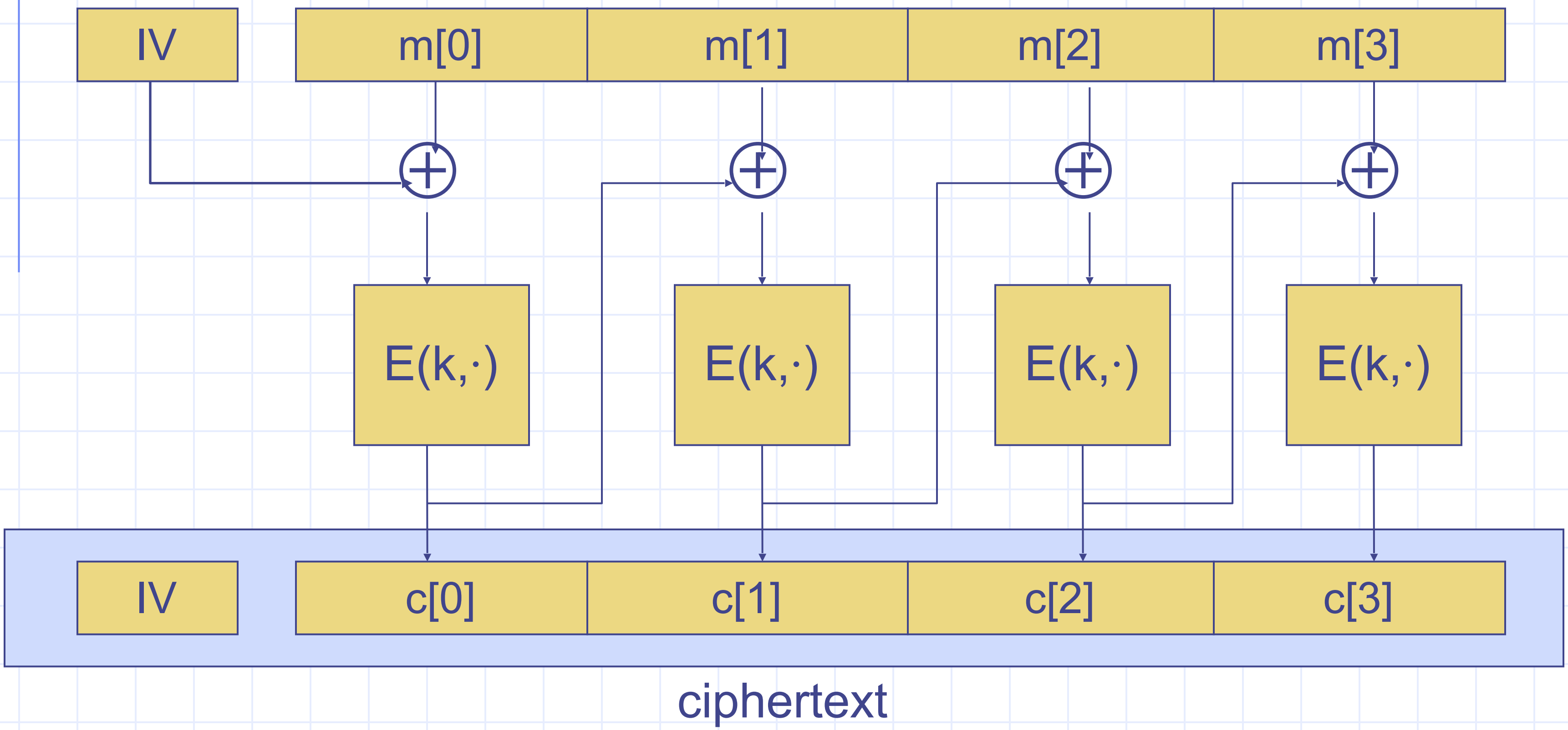
An example plaintext

Encrypted with AES in ECB mode



Correct use of block ciphers I: CBC mode

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



Q: how to do decryption?

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

Multi use key: (CPA Security)

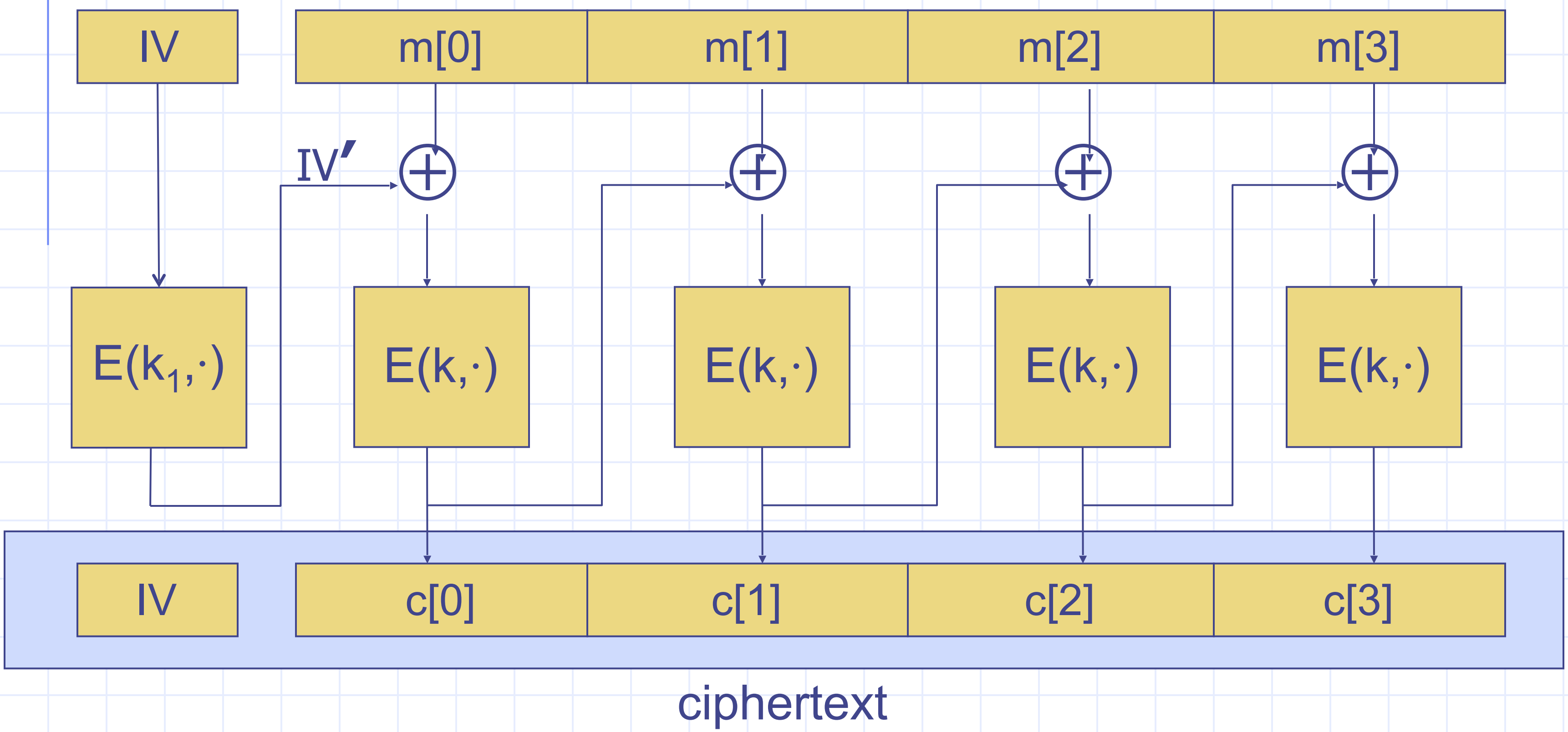
Best: use a fresh random IV for every message

Can use unique IV (e.g counter)

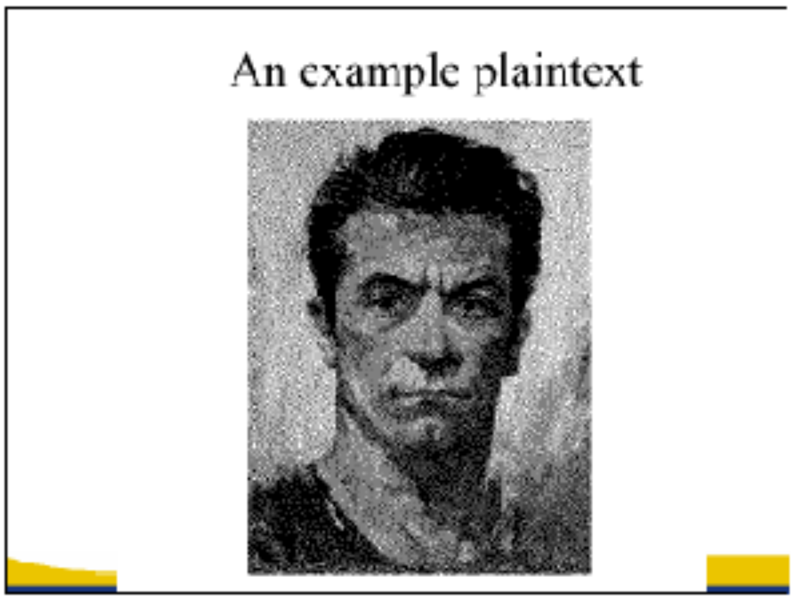
but then first step in CBC must be $IV' \leftarrow E(k_1, IV)$

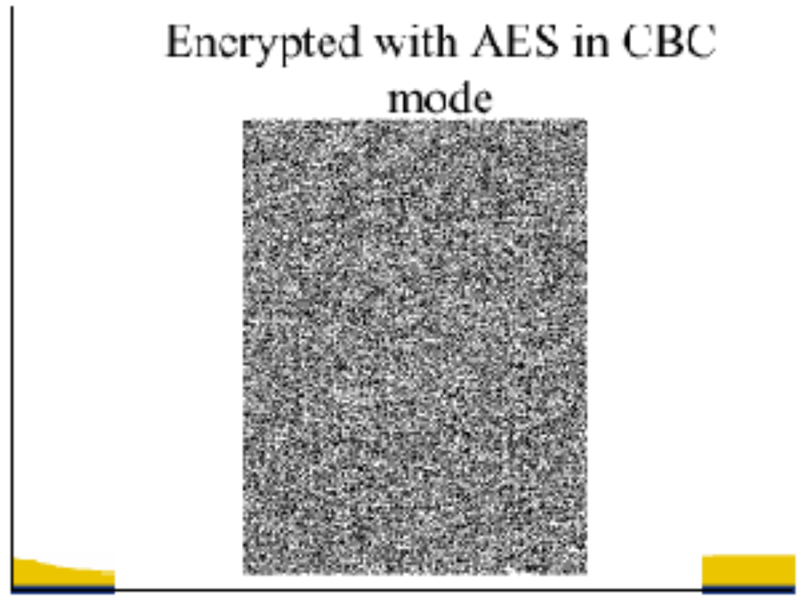
CBC with Unique IVs

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



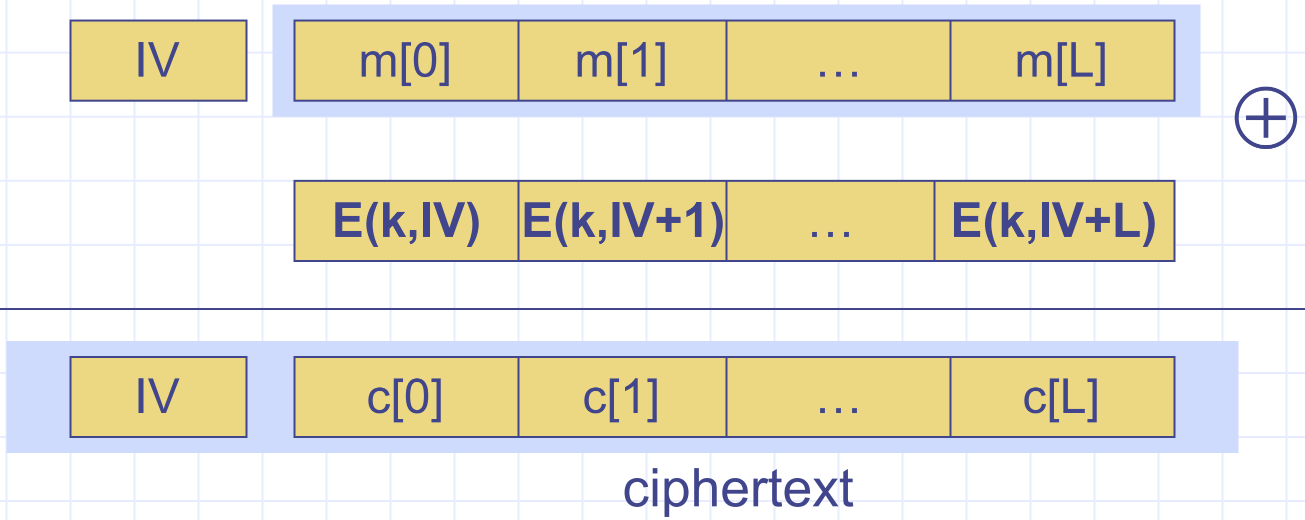
In pictures





Correct use of block ciphers II: CTR mode

Counter mode with a random IV: (parallel encryption)



Performance: Crypto++ 5.6.0 [Wei Dai]

Intel Core 2 (on Windows Vista)

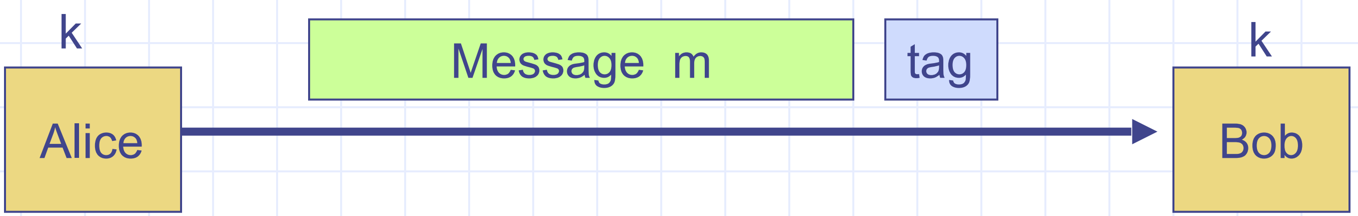
Cipher	Block/key size	Speed (MB/sec)
RC4		126
Salsa20/12		643
3DES	64/168	10
AES/GCM	128/128	102

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.



Generate tag:
$tag \leftarrow S(k, m)$

Verify tag:
$V(k, m, tag) \stackrel{?}{=}$ `yes'

note: non-keyed checksum (CRC) is an insecure MAC !!

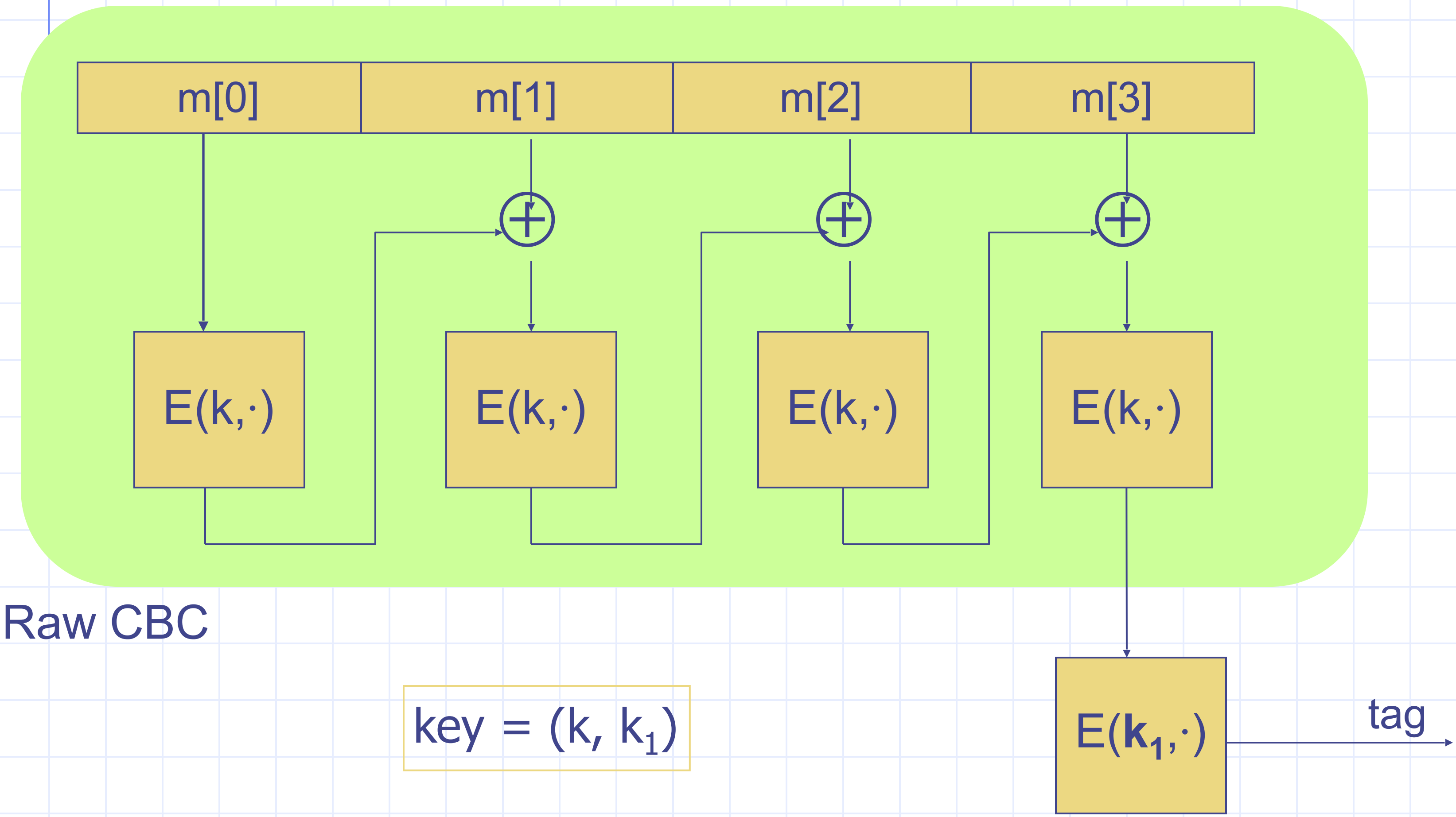
Secure MACs

- 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



key = (k, k_1)

Construction 2: HMAC (Hash-MAC)

Most widely used MAC on the Internet.

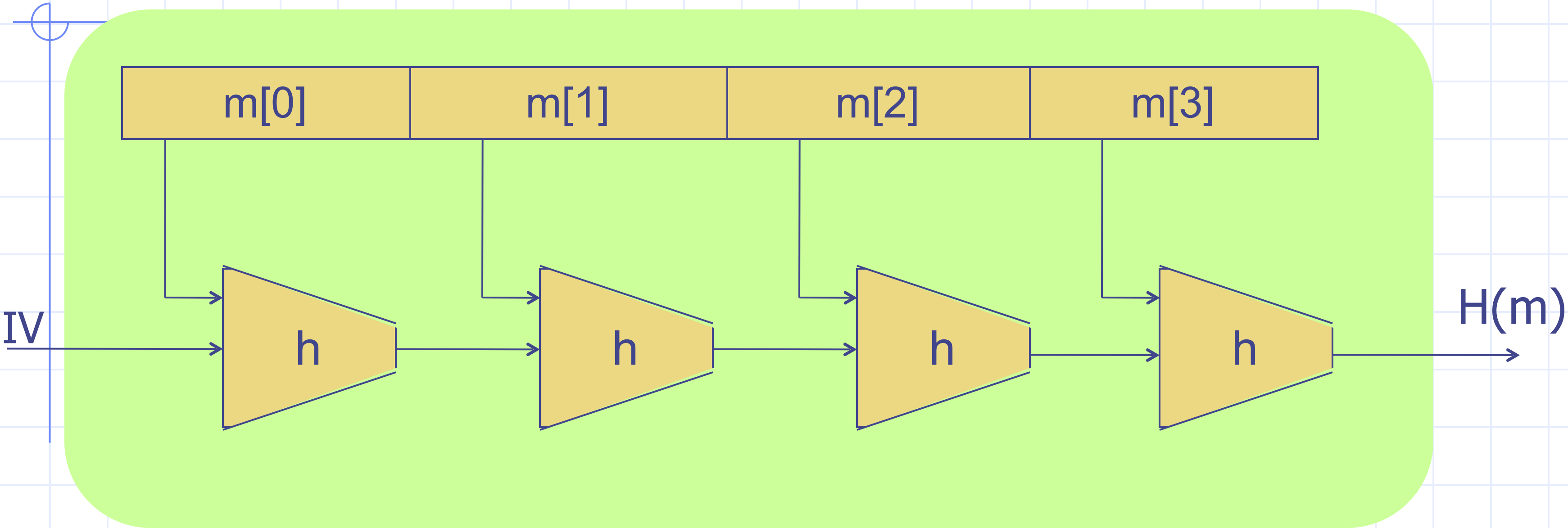
H: hash function.
example: SHA-256 ; output is 256 bits

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



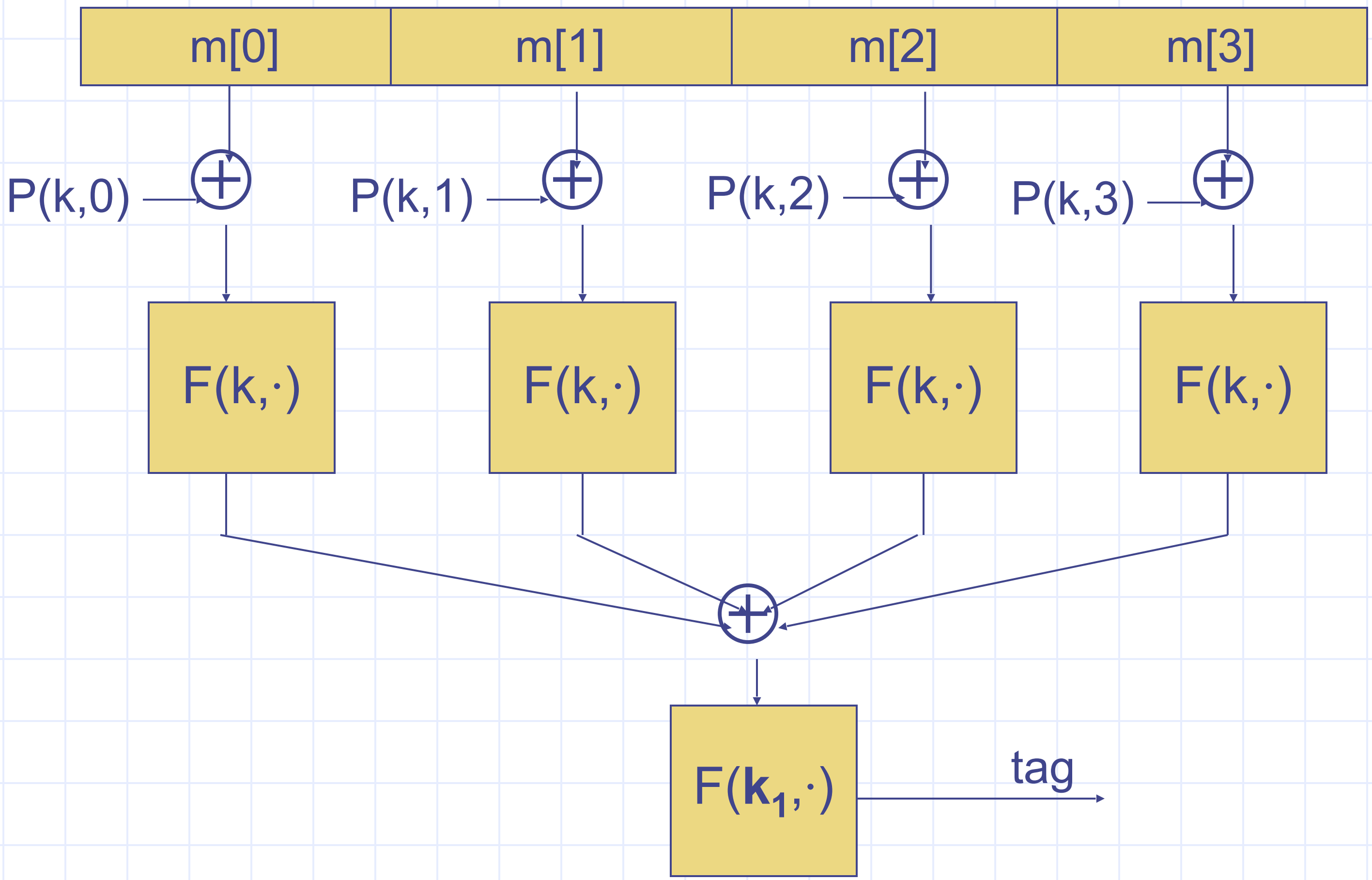
h(t, m[i]): compression function

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:



Why are these MAC constructions secure?

… not today – take 40-675

Why the last encryption step in ECBC?

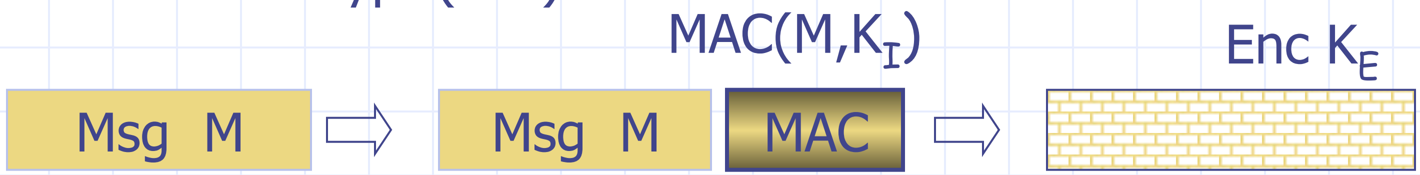
- CBC (aka Raw-CBC) is not a secure MAC:
  - Given tag on a message m, attacker can deduce tag for some other message m’
  - How: good crypto exercise … take 40-675 ;)

Authenticated Encryption: Encryption + MAC

Combining MAC and ENC (CCA)

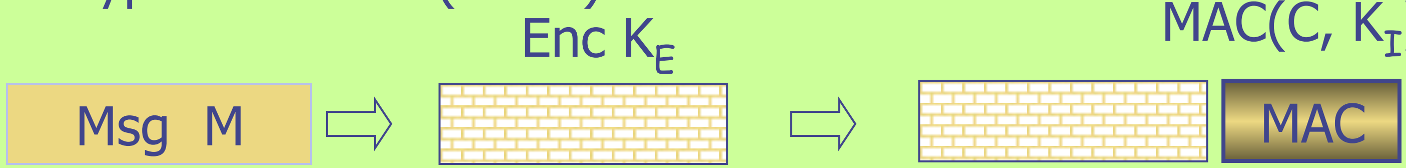
Encryption key K_E MAC key = K_I

Option 1: MAC-then-Encrypt (SSL)

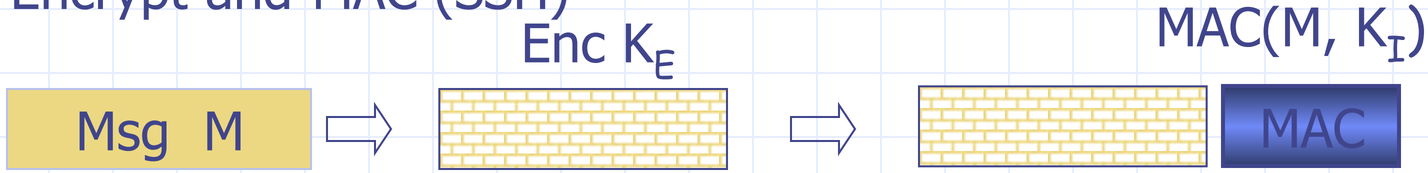


Option 2: Encrypt-then-MAC (IPsec)

Secure for all secure primitives



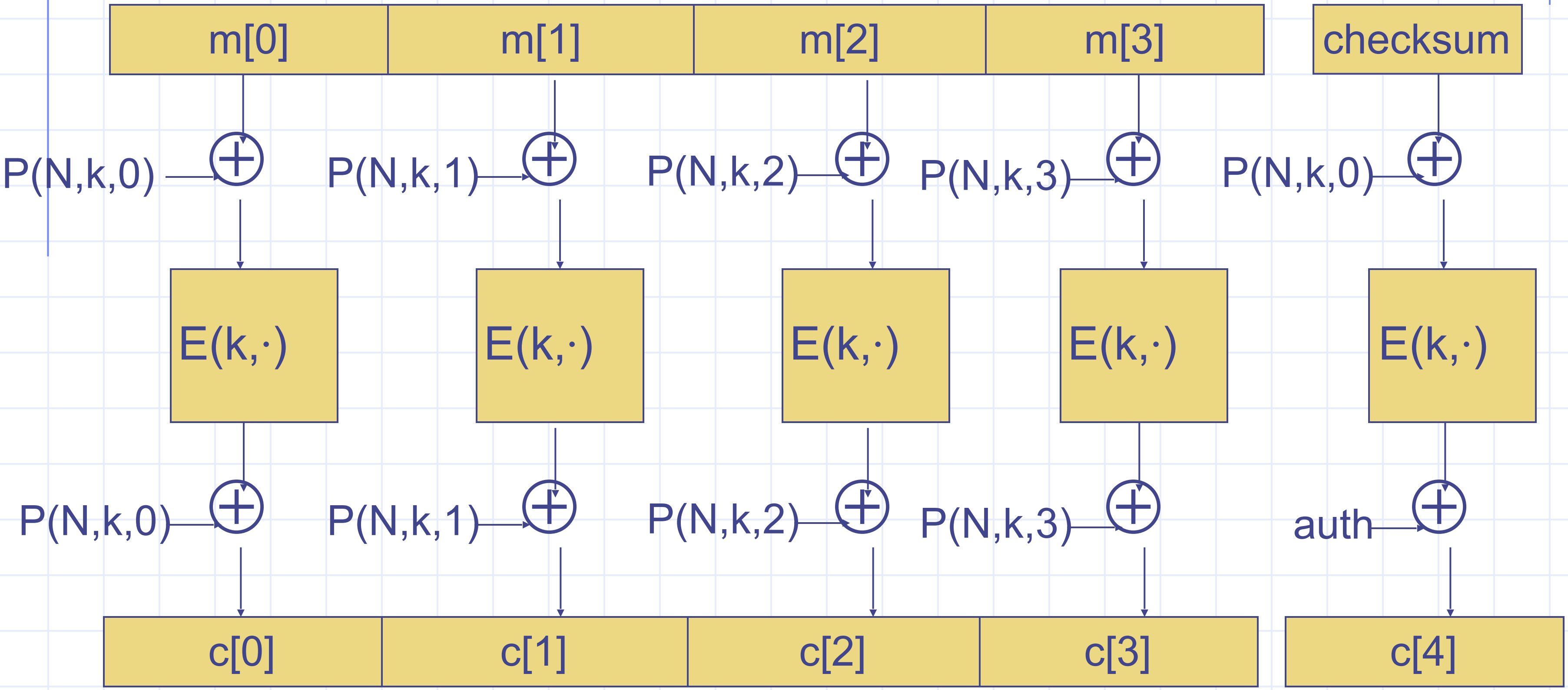
Option 3: Encrypt-and-MAC (SSH)



OCB

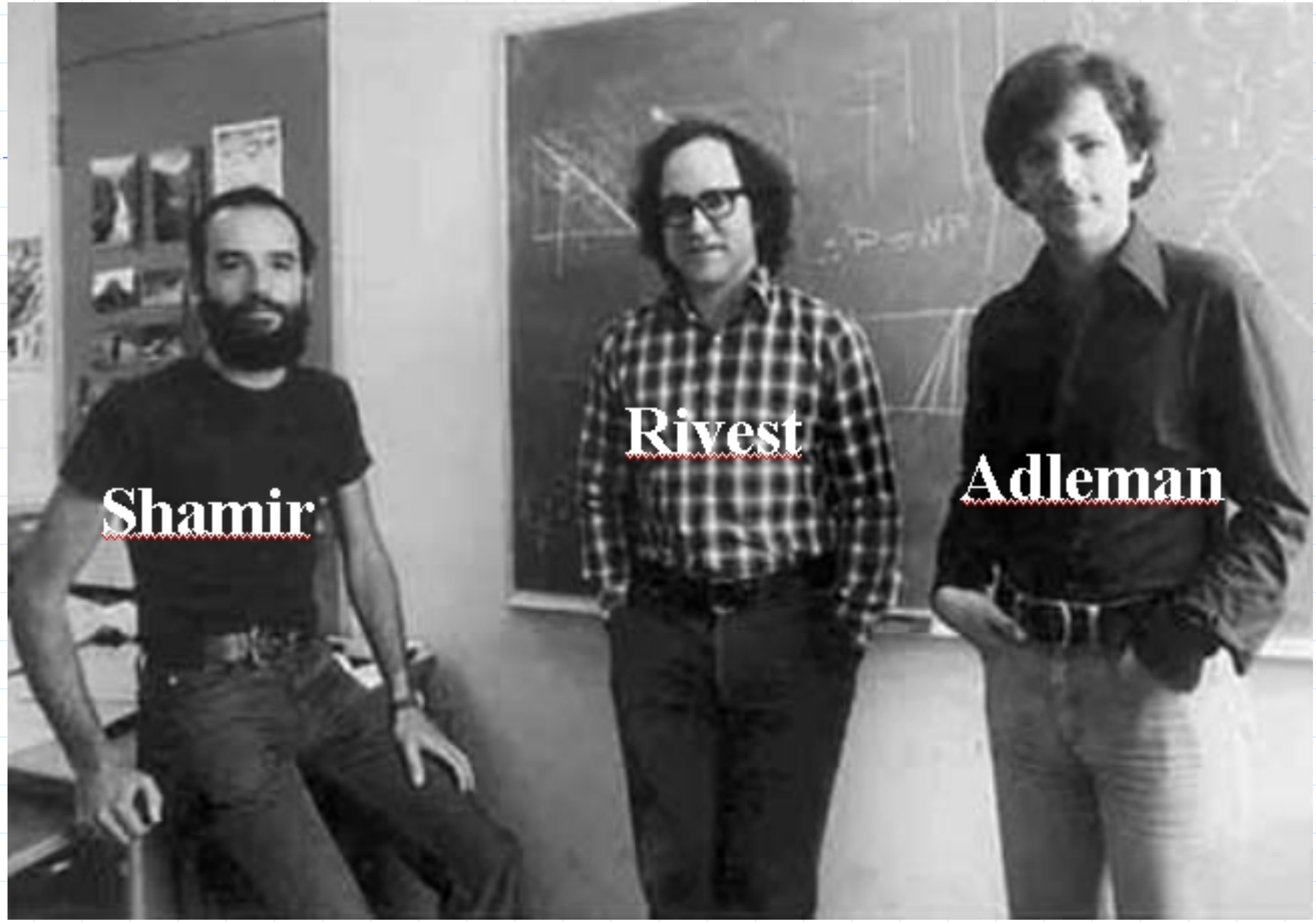
offset codebook mode

More efficient authenticated encryption

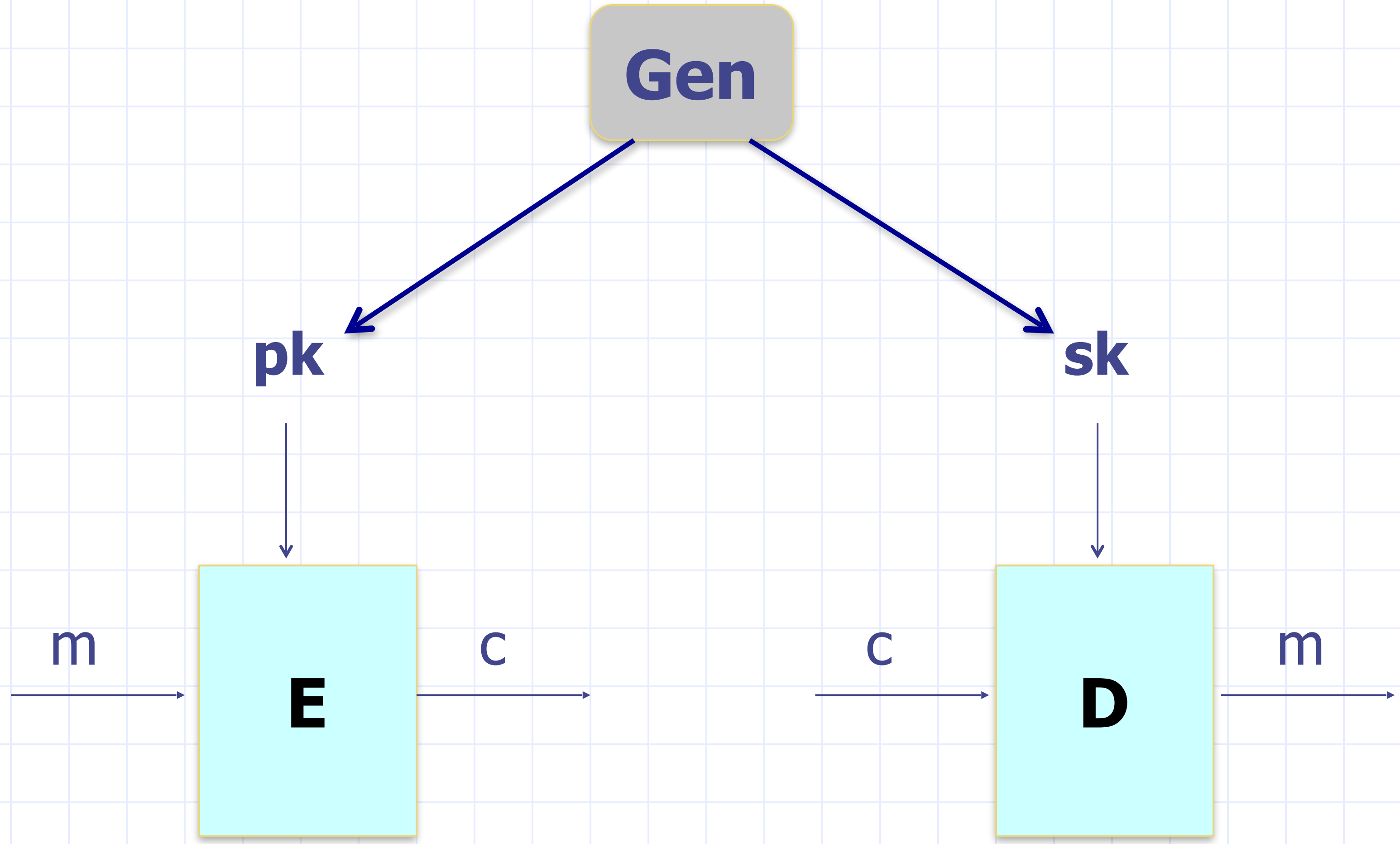


Rogaway, …

Public-key Cryptography

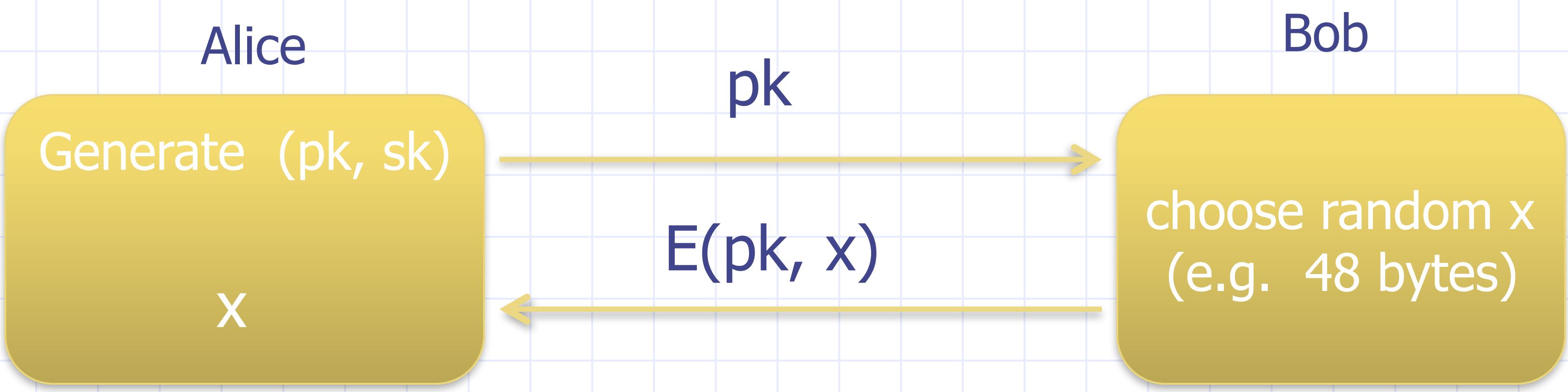

Shamir
Rivest
Adleman

Public key encryption: (Gen, E, D)



Applications

Session setup (for now, only eavesdropping security)



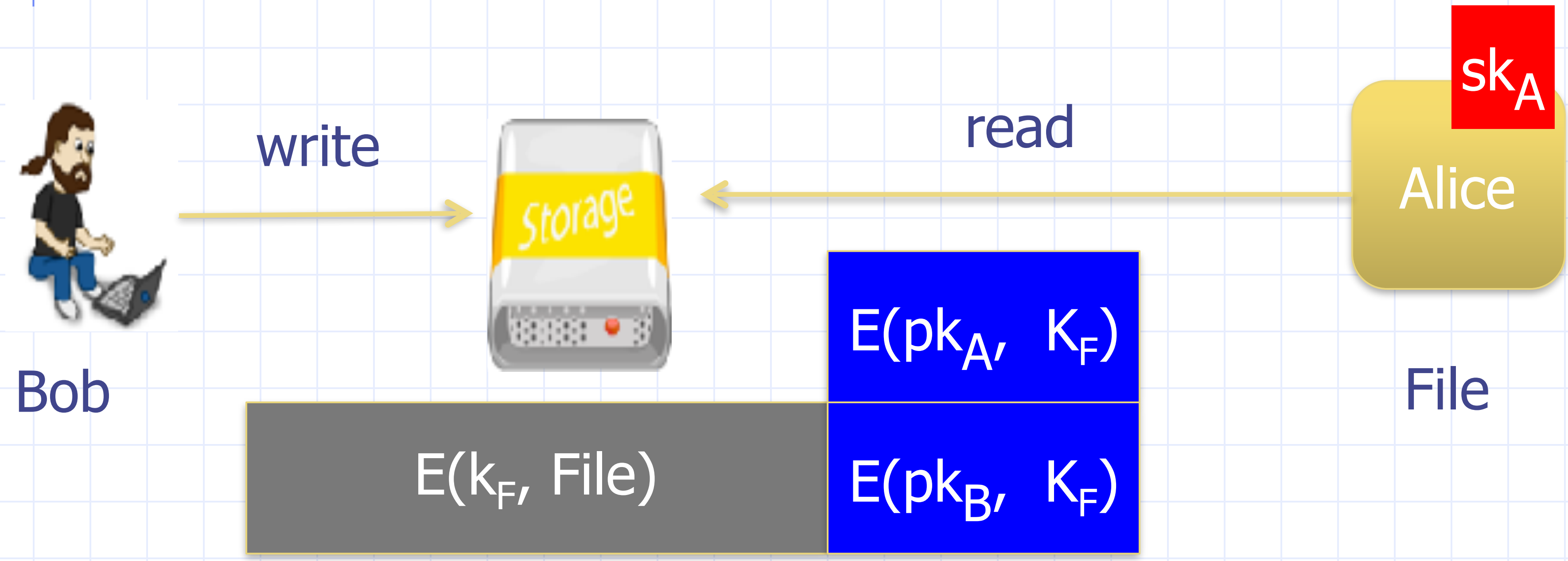
Non-interactive applications: (e.g. Email)

- Bob sends email to Alice encrypted using pk_{alice}
- Note: Bob needs pk_{alice} (public key management)

Applications

Encryption in non-interactive settings:

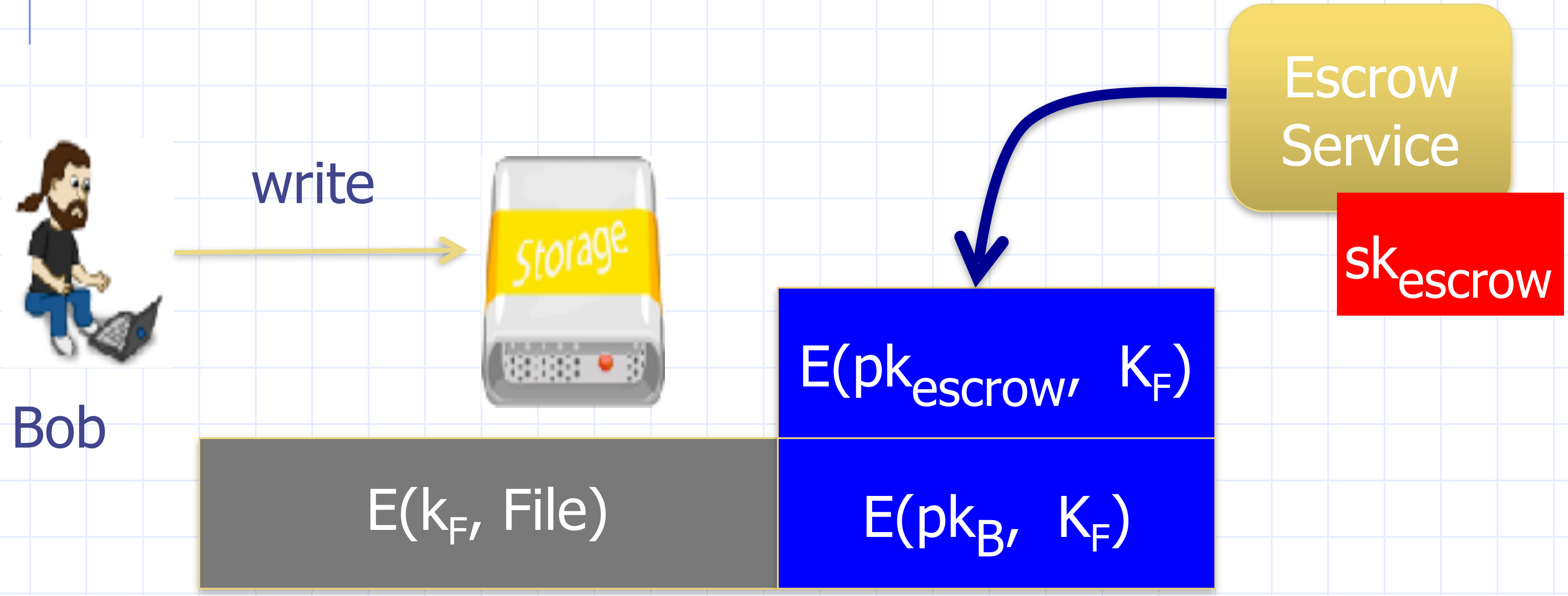
- Encrypted File Systems



Applications

Encryption in non-interactive settings:

- Key escrow: data recovery without Bob's key



Trapdoor functions (TDF)

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

- G(): randomized alg. outputs key pair (pk, sk)
- $F(pk, \cdot)$: det. alg. that defines a func. $X \longrightarrow Y$
- $F^{-1}(sk, \cdot)$: defines a func. $Y \longrightarrow X$ that inverts $F(pk, \cdot)$

Security: $F(pk, \cdot)$ 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) :

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

$k \longleftarrow H(x)$,

$c \longleftarrow E_s(k, m)$

output (y, c)

D(sk, (y,c)) :

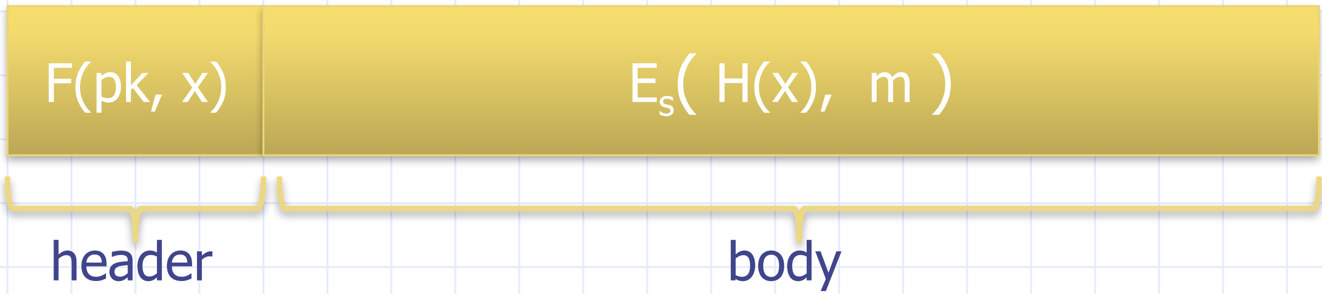
$x \longleftarrow F^{-1}(sk, y)$,

$k \longleftarrow H(x)$,

$m \longleftarrow D_s(k, c)$

output m

In pictures:



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 $X \longrightarrow X$
- $H: M \longrightarrow X$ a hash function

Sign(sk, m∈X) :

output

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

Verify(pk, m, sig) :

output

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

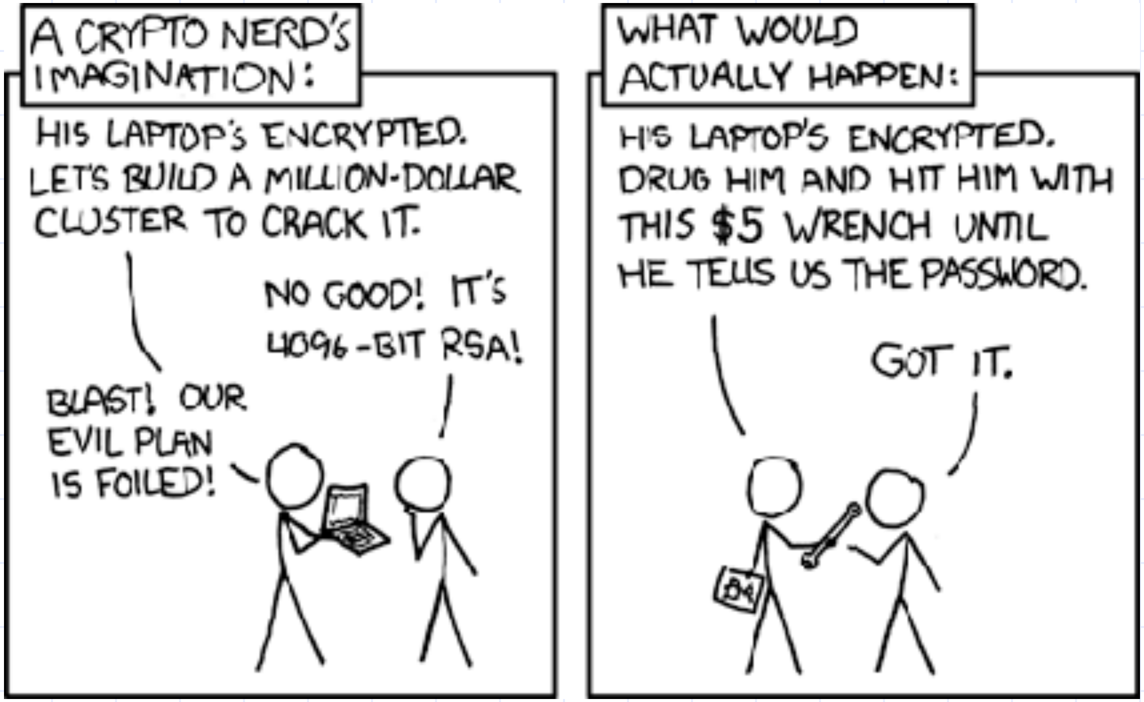
Public-Key Infrastructure (PKI)

- 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

Limitations of cryptography

Cryptography works when used correctly !!

… but is not the solution to all security problems



XKCD 538