# 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 yourself unless
    - you spend a lot of time becoming an expert
    - you subject your design to outside review

# Auguste Kerckhoffs

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

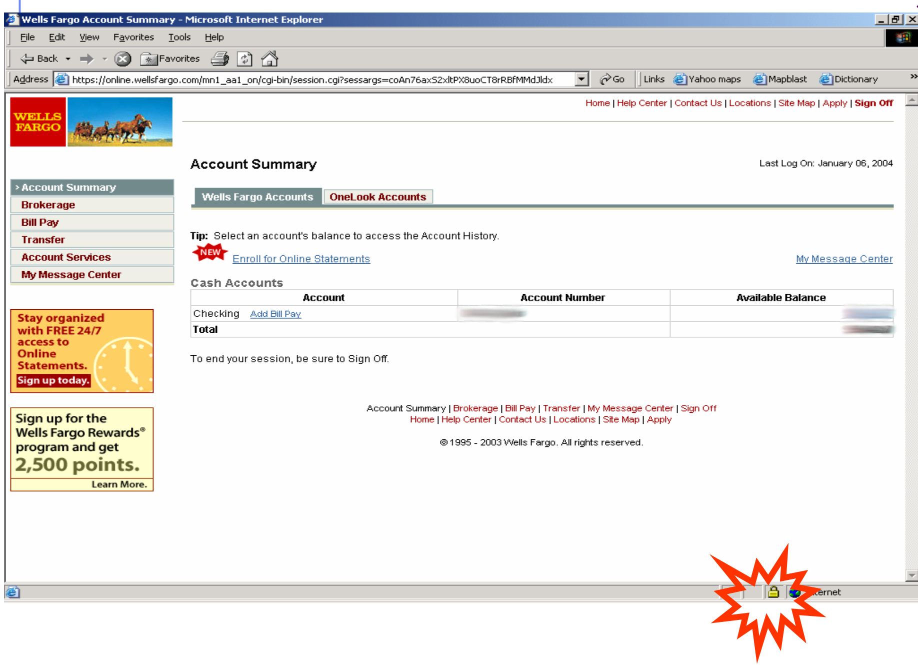
baptised as **Jean-Guillaume-Hubert-Victor-François-Alexandre-Auguste Kerckhoffs von Nieuwenhof**

# Goal 1:secure communication

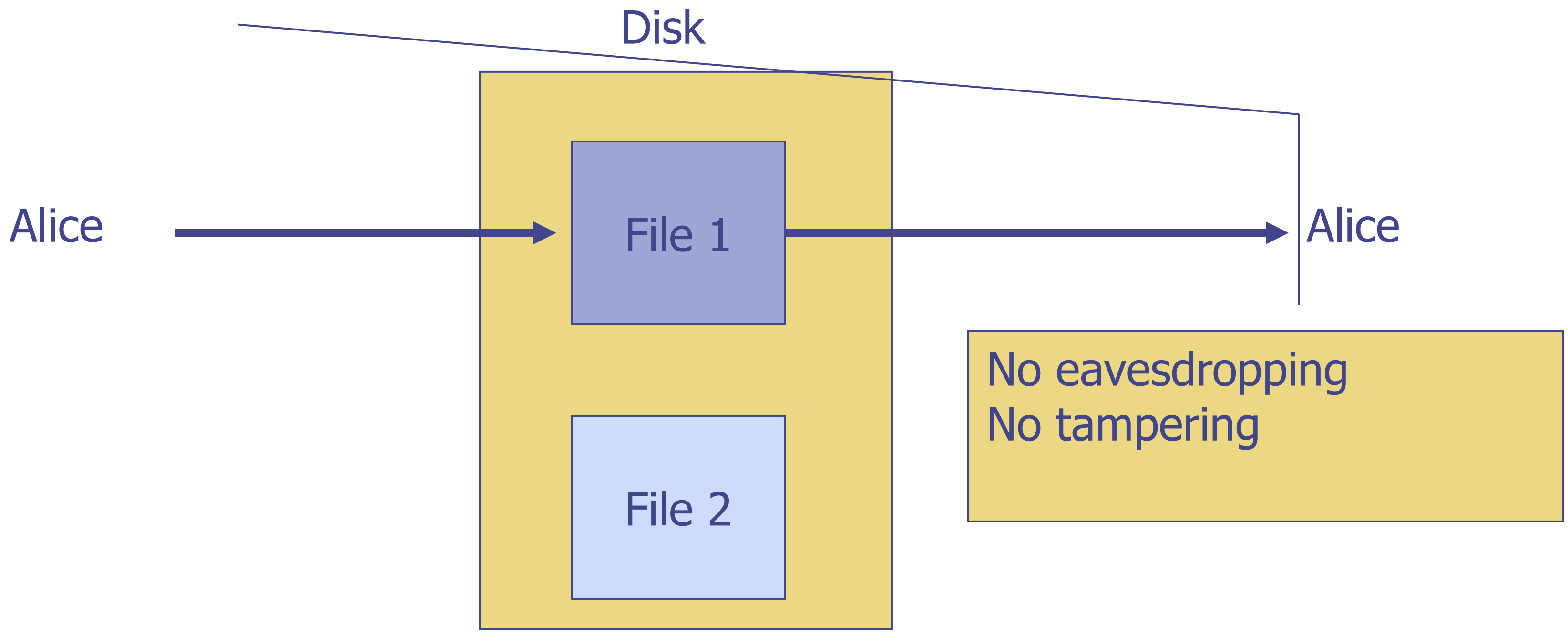
Step 1: Session setup to exchange key

Step 2: encrypt data

HTTPS



# Goal 2: Protected files



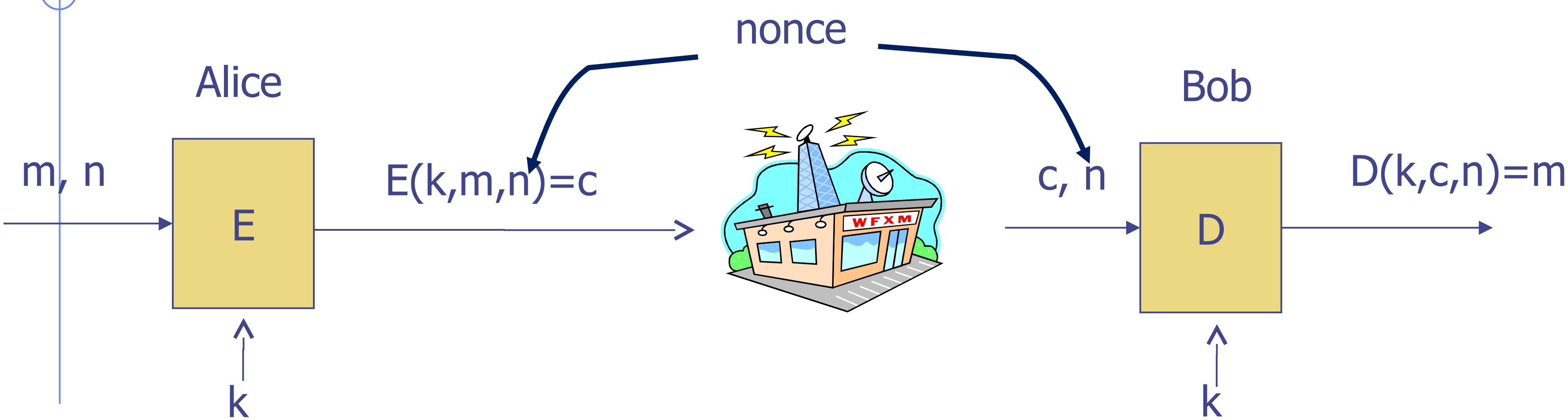

Analogous to secure communication:

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 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

  SSL: same key used to encrypt many packets
- Need either *unique* nonce or *random* nonce

# 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 |

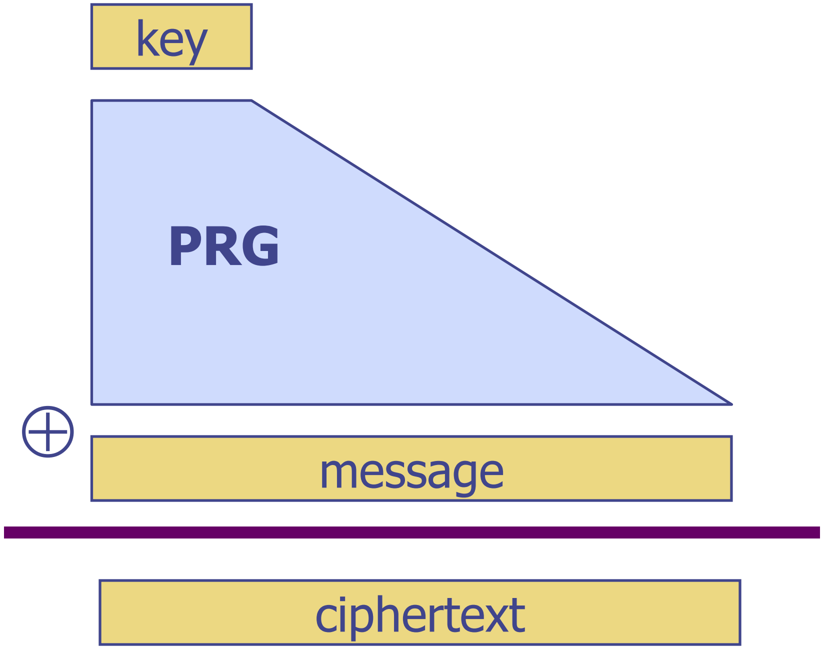
$\oplus$

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

# Stream ciphers (single use key)

Problem: OTP key is as long the message

Solution: Pseudo random key -- stream ciphers




$c \leftarrow \text{PRG}(k) \oplus m$

Stream ciphers: RC4 (113MB/sec) , SEAL (293MB/sec)

# Dangers in using stream ciphers

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

$$C1 \leftarrow m1 \oplus PRG(k)$$

$$C2 \leftarrow m2 \oplus PRG(k)$$

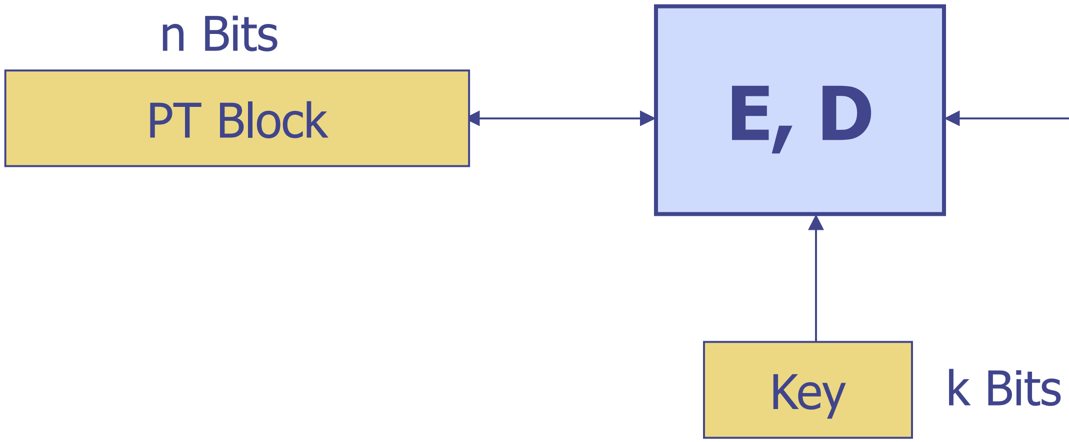
Eavesdropper does:

$$C1 \oplus C2 \quad \rightarrow \quad m1 \oplus m2$$

Enough redundant information in English that:

$$m1 \oplus m2 \rightarrow \quad m1 , \; m2$$

# 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:

  $mL \leftarrow mR$

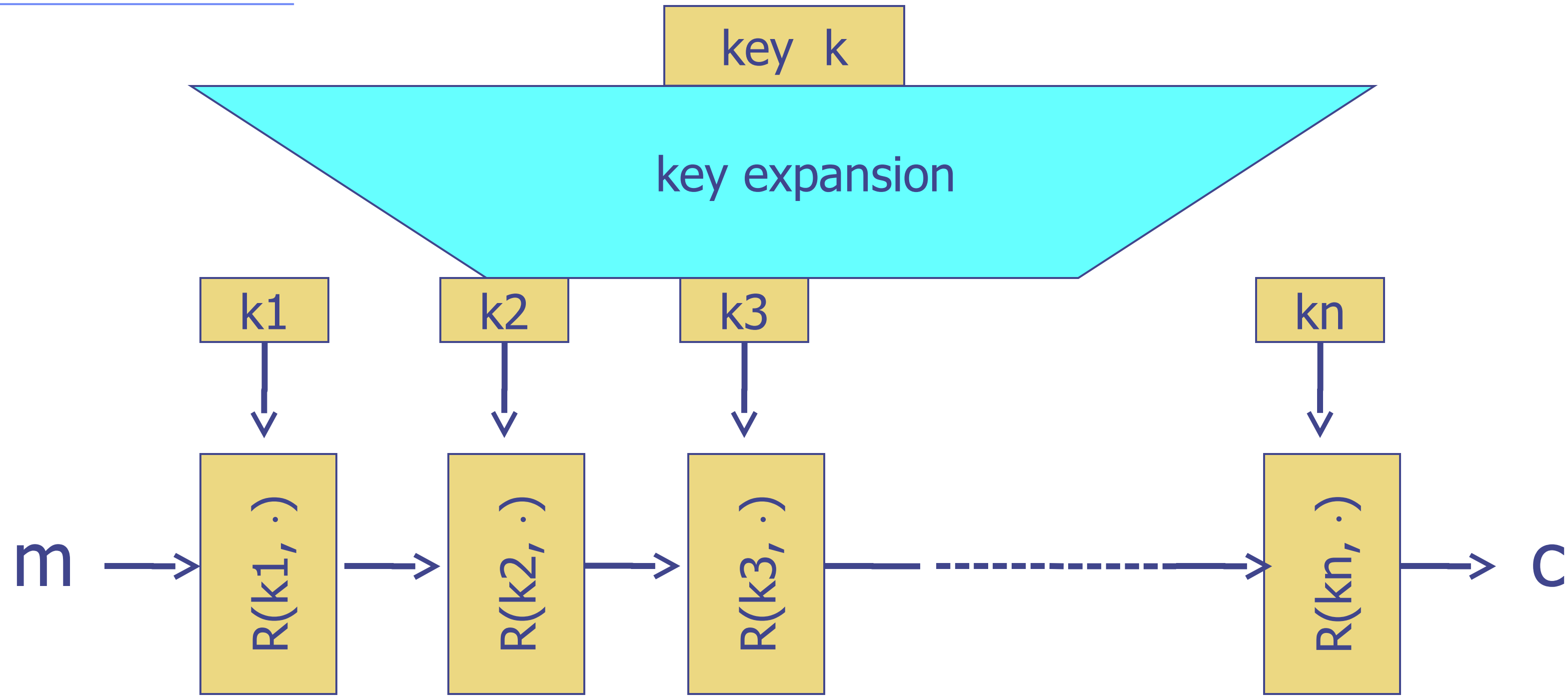
  $mR \leftarrow mL \oplus F(k, mR)$

- 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

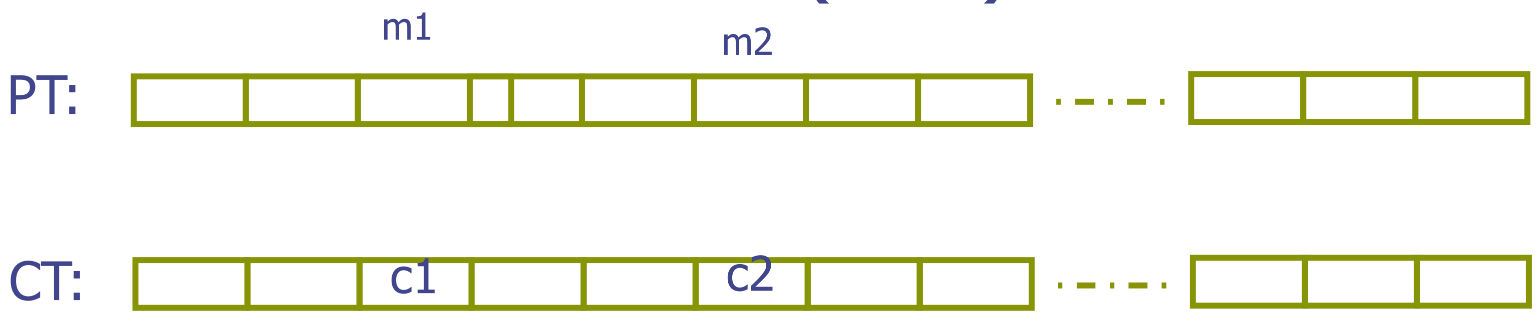



R(k,m): round function

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

# Incorrect use of block ciphers

## Electronic Code Book (ECB):




Parallel encryption of the various blocks through the same key

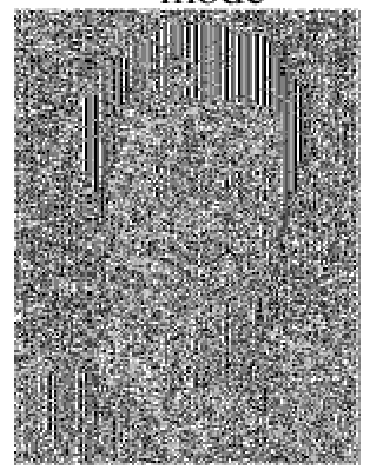
## <u>Problem</u>:

- if m1=m2 then c1=c2

# In pictures

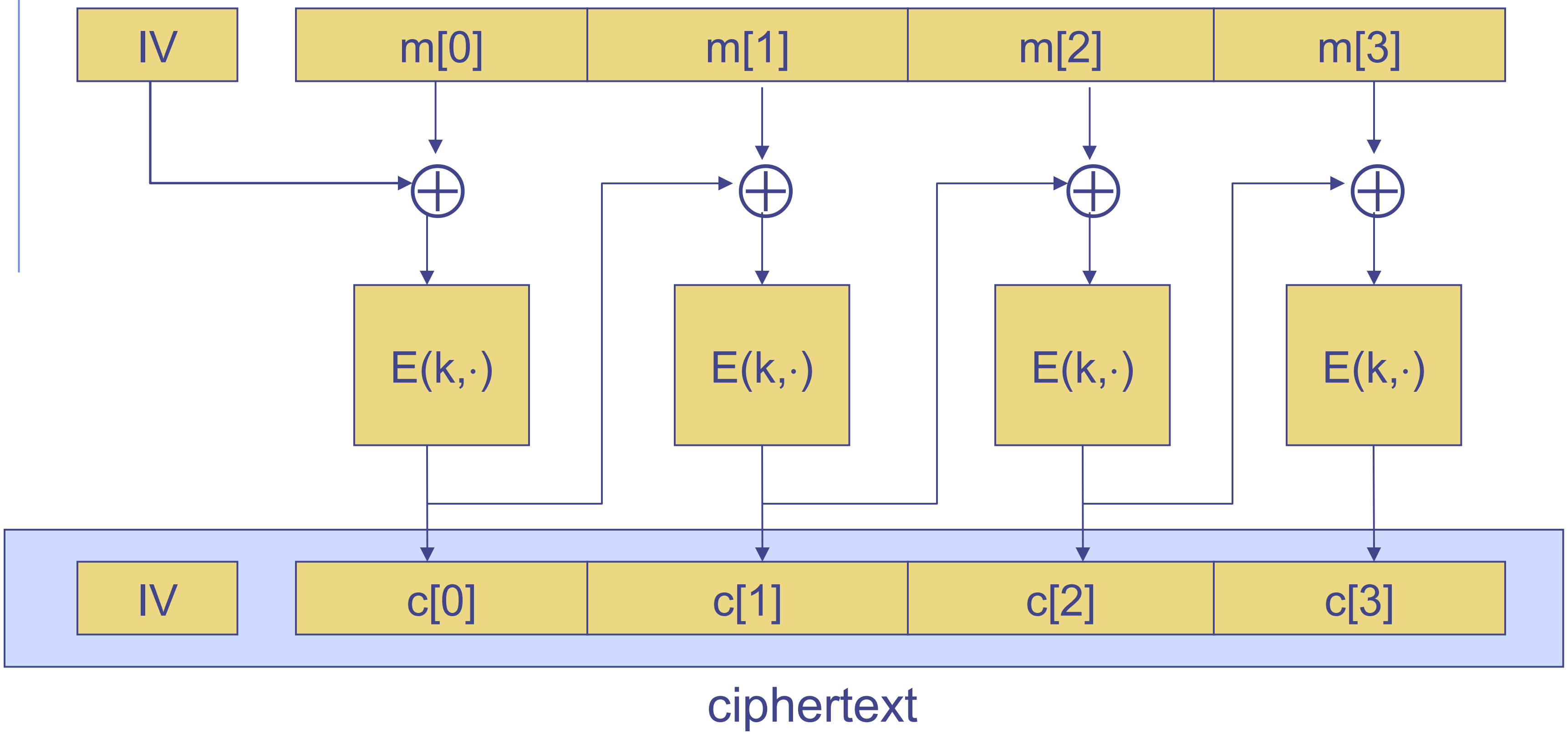
An example plaintext

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?

# Use cases: choosing an IV

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

Multi use key: (CPA Security)

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

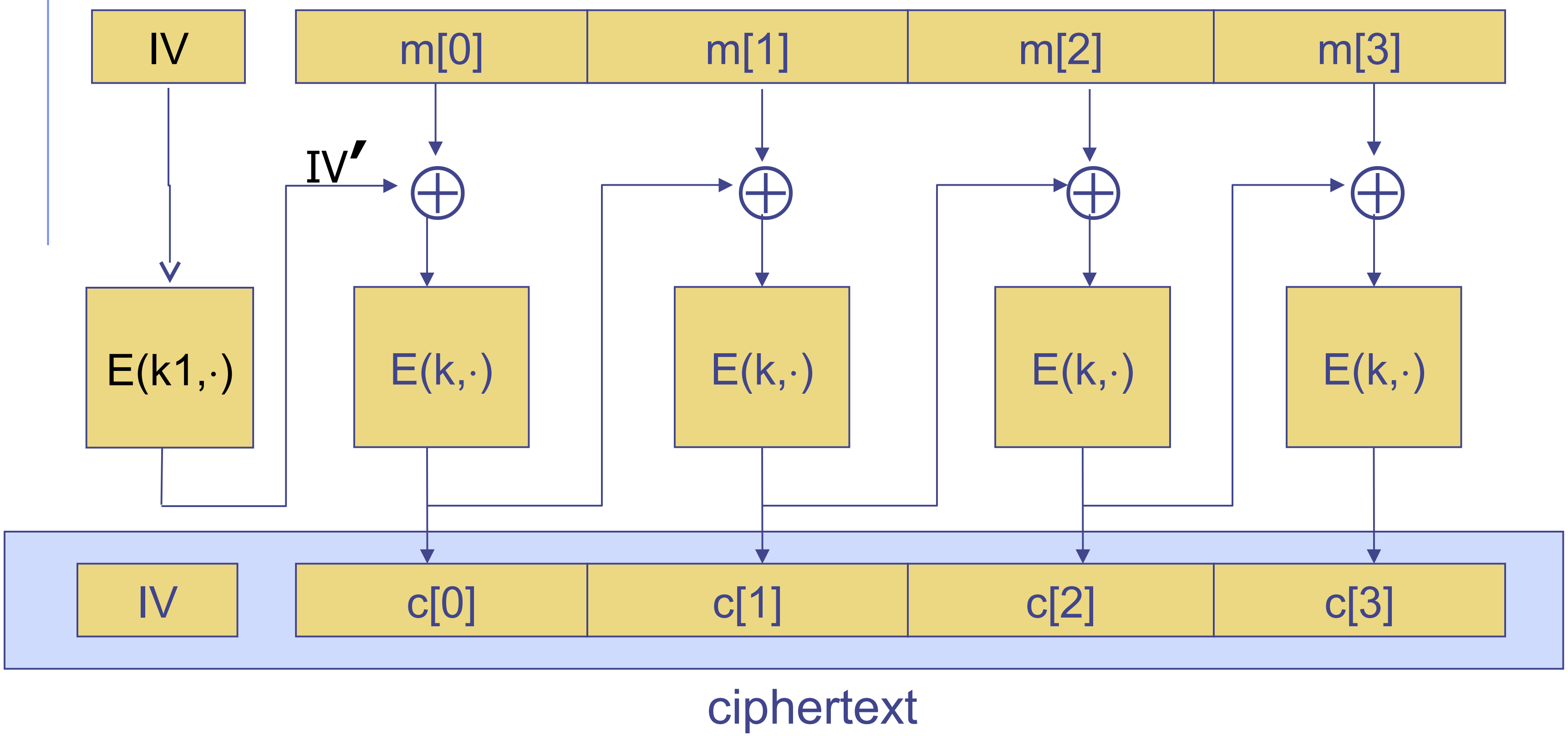
Can use *unique* IV (e.g counter)

but then first step in CBC must be $IV' \leftarrow E(k1, IV)$

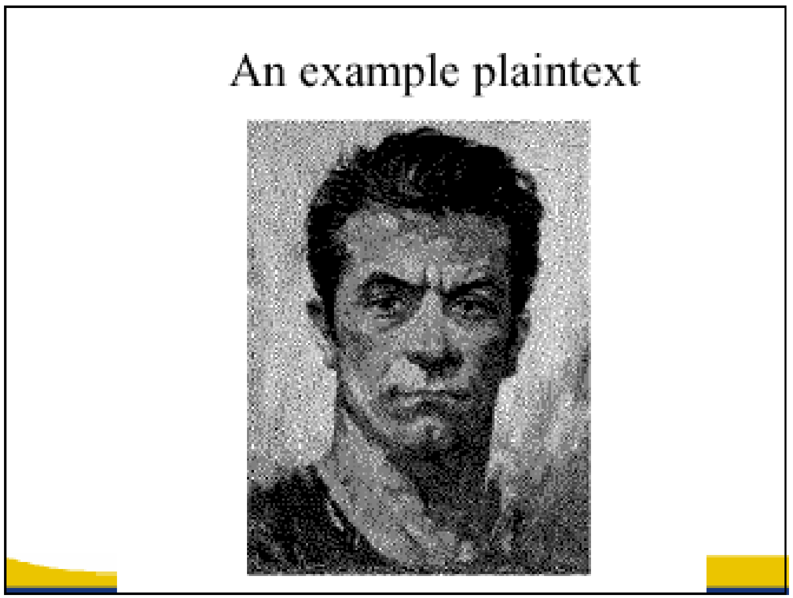
benefit: may save transmitting IV with ciphertext

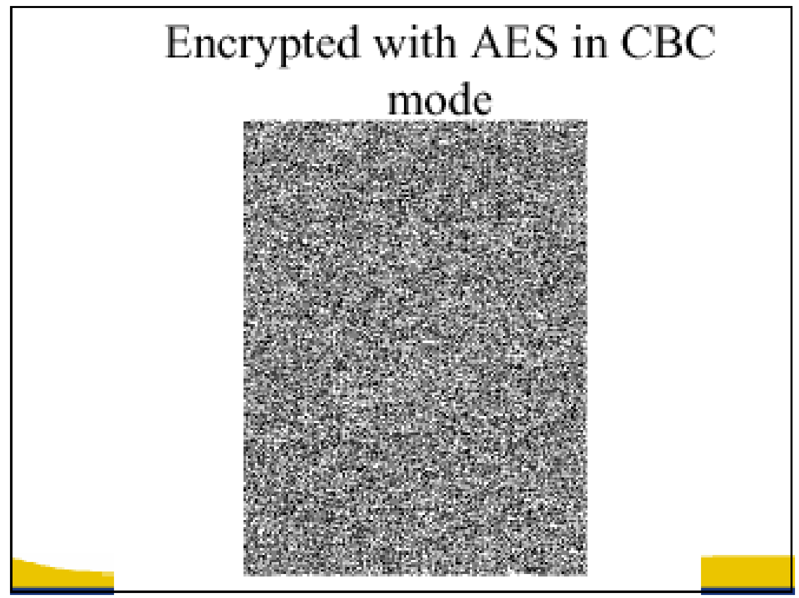
# CBC with Unique IVs

unique IV means: (k,IV) pair is used for only one message
may be predictable so use E(k1,·) as PRF



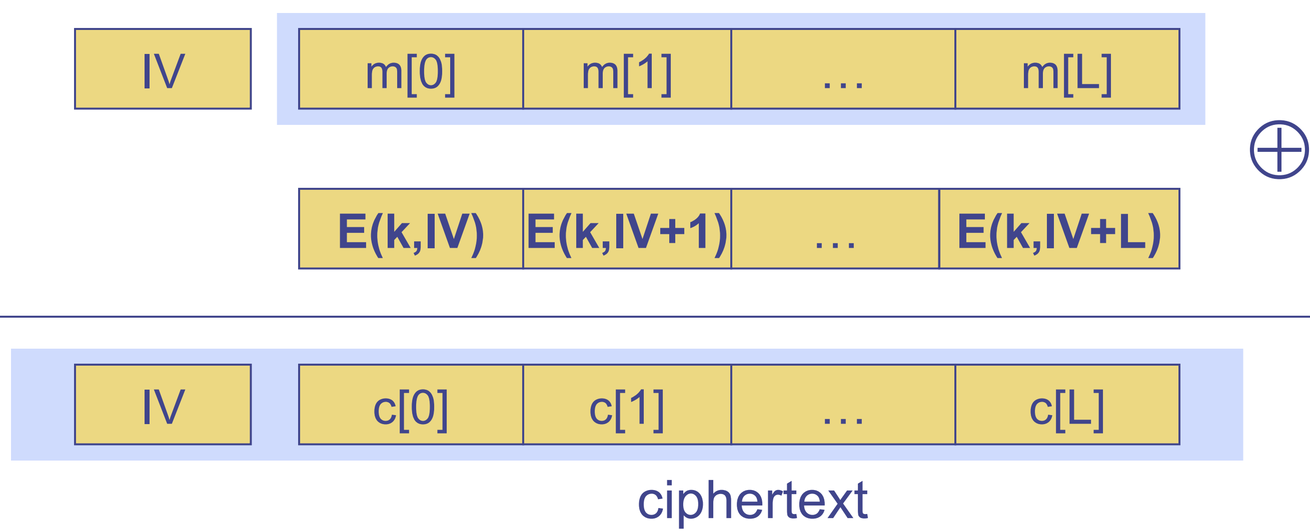
# In pictures



# Correct use of block ciphers II: CTR mode

Counter mode with a random IV: (parallel encryption)



# Performance: Crypto++ 5.2.1 [ Wei Dai ]

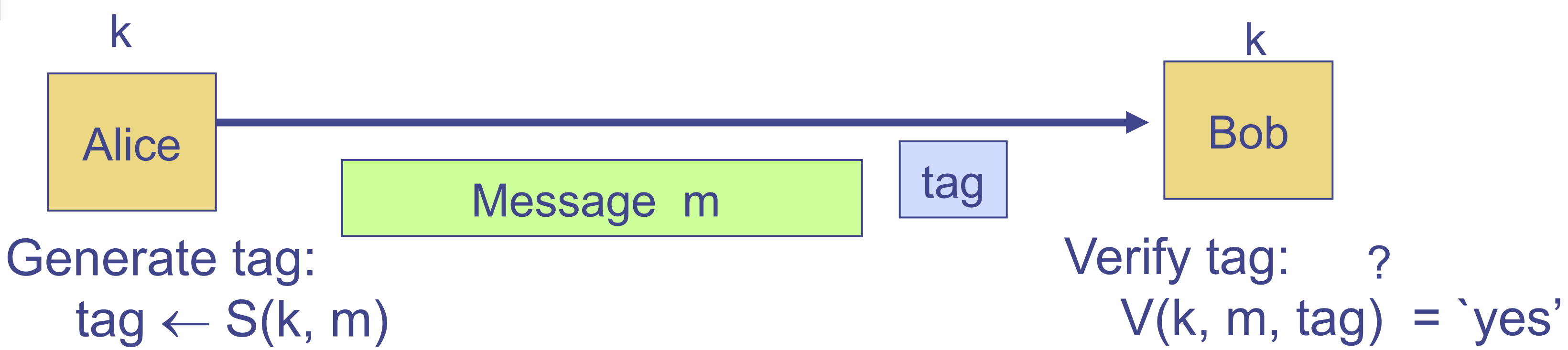
Pentium 4, 2.1 GHz ( on Windows XP SP1, Visual C++ 2003 )

| Cipher | Block/key size | Speed (MB/sec) |
|---|---|---|
| RC4 | | 113 |
| SEAL | | 293 |
| 3DES | 64/168 | 9 |
| AES | 128/128 | 61 |

# Data integrity

# Message Integrity: MACs

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

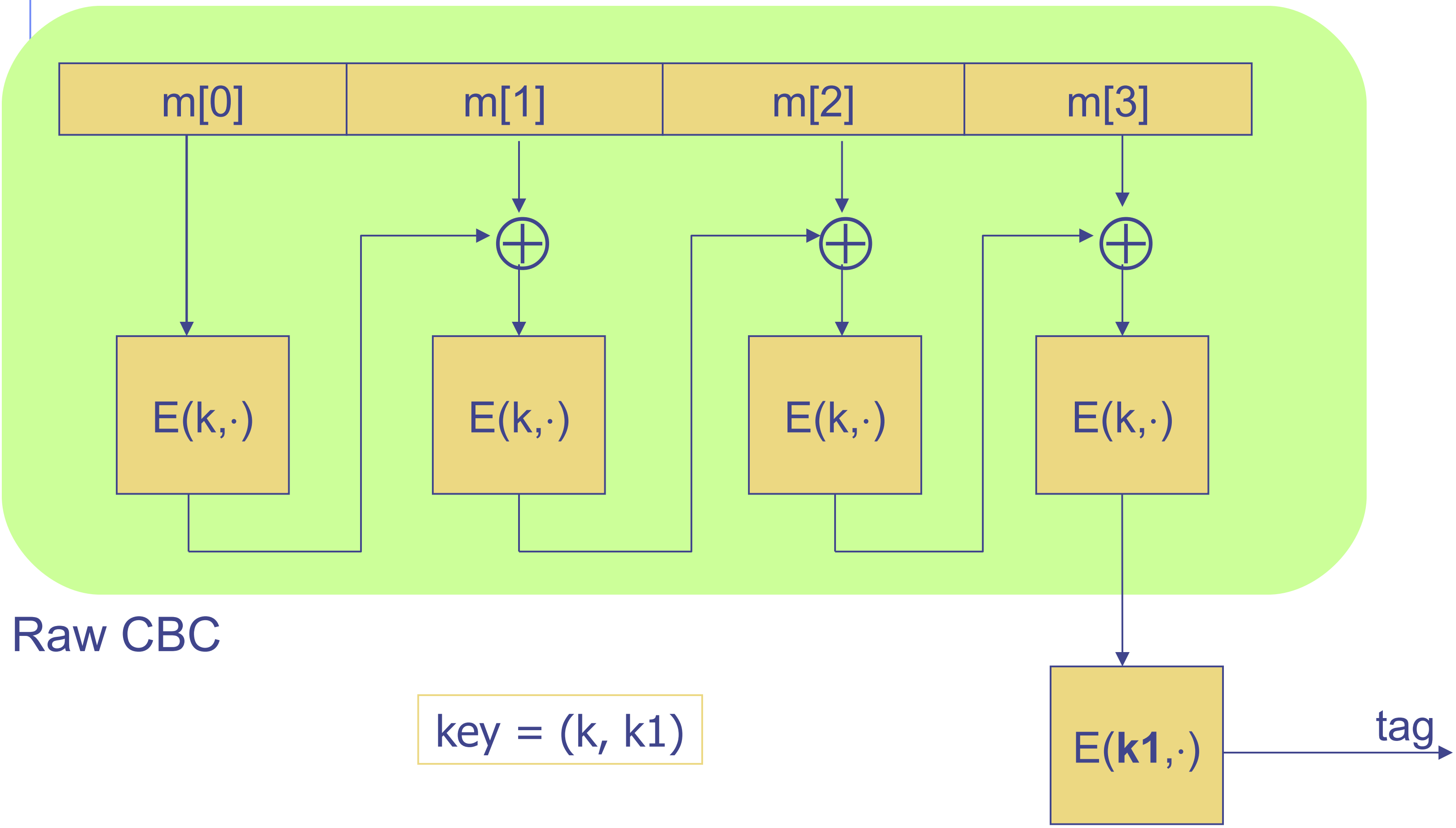



note: non-keyed checksum (CRC) is an insecure MAC !!

# Secure MACs

- Attacker information: chosen message attack
  - for m1,m2,…,mq   attacker is given   ti ← S(k,mi)
- Attacker's goal:   existential forgery.
  - produce some <u>**new**</u> valid message/tag pair  (m,t).
    (m,t)  $\notin$  { (m1,t1) , … , (mq,tq) }
- 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



# Construction 2: HMAC (Hash-MAC)

Most widely used MAC on the Internet.

H: hash function.

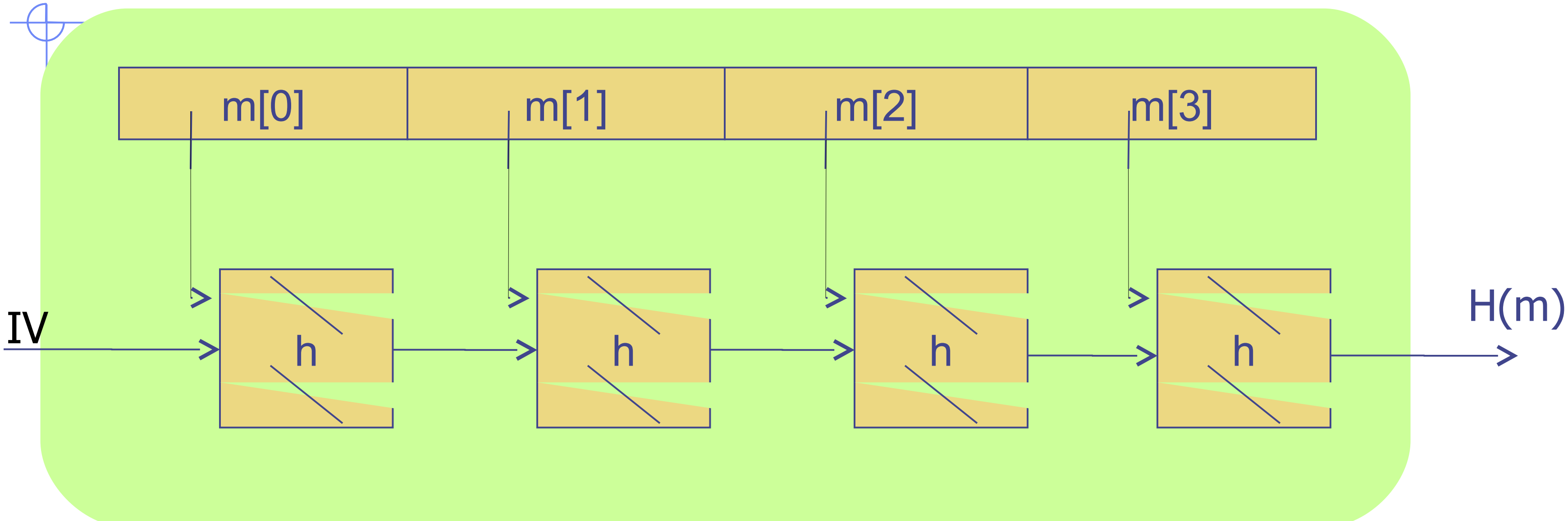
example: SHA-256 ; output is 256 bits

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

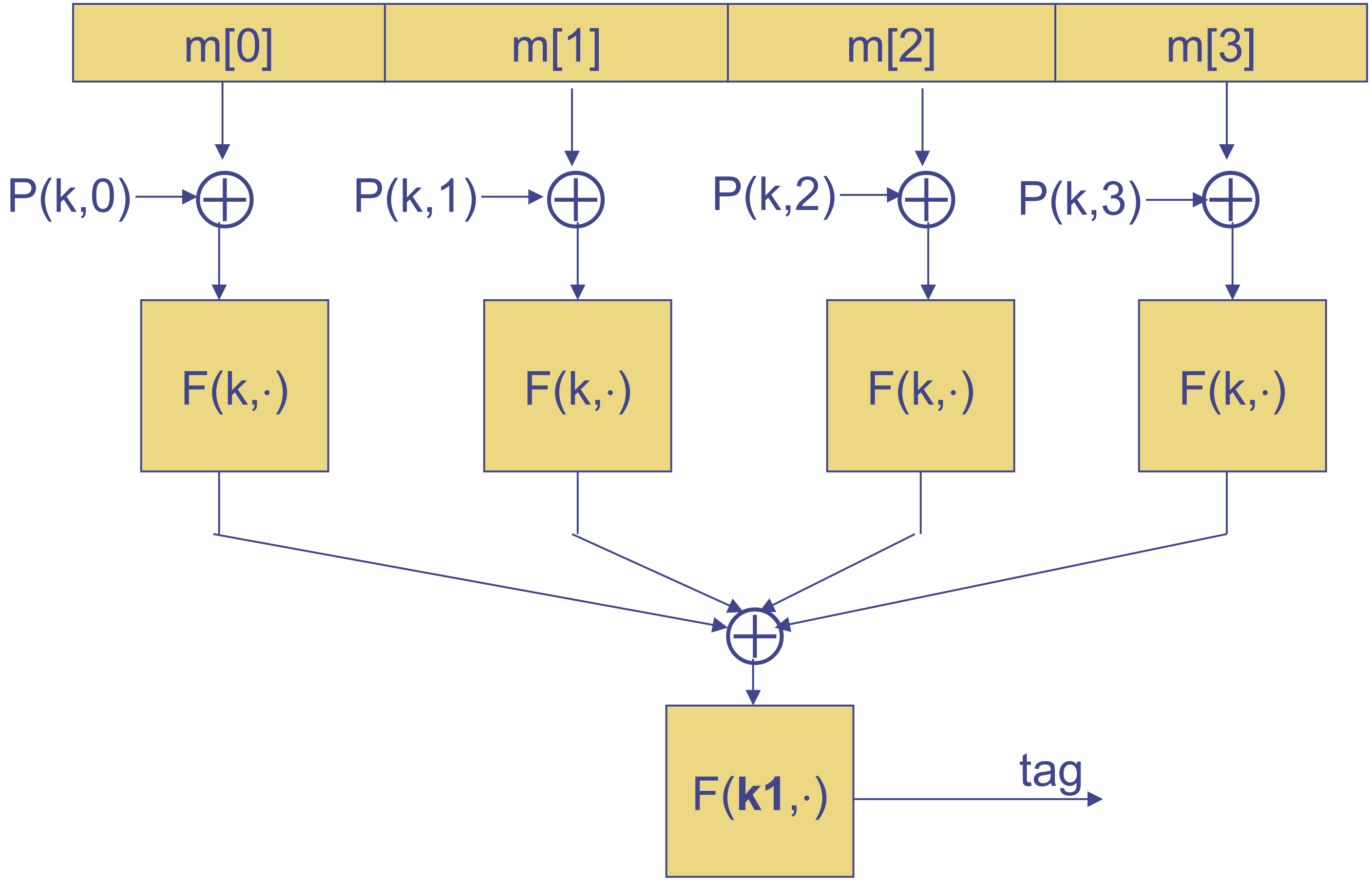
Thm 1: if h is collision resistant then so is H

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

PRF=pseudo random function

# Construction 3: PMAC – parallel MAC

ECBC and HMAC are sequential. PMAC:



- These MAC constructions are secure
  - No time to prove it

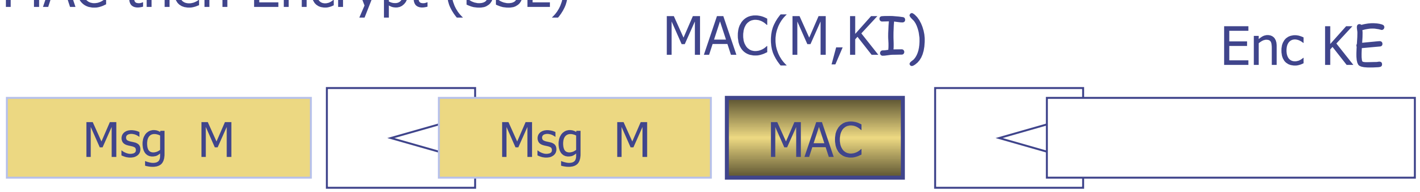
- Why the last encryption step in ECBC?
  - CBC (aka Raw-CBC) is not a secure MAC:
    - Given tag on a message m, attacker can deduce tag for some other message m'
    - How: good crypto exercise …

# Authenticated Encryption:
# Encryption + MAC

# Combining MAC and ENC (CCA)

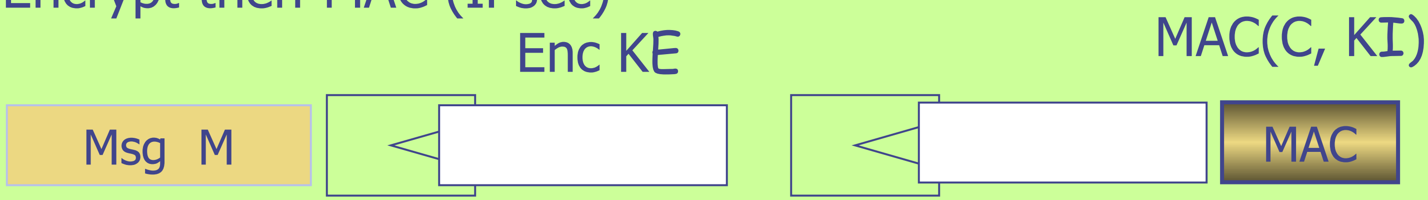
Encryption key KE    MAC key = KI

Option 1: MAC-then-Encrypt (SSL)

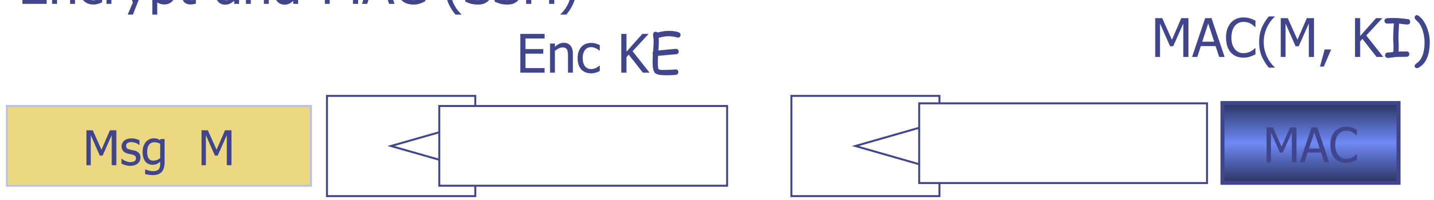



Option 2: Encrypt-then-MAC (IPsec)

Secure on general grounds



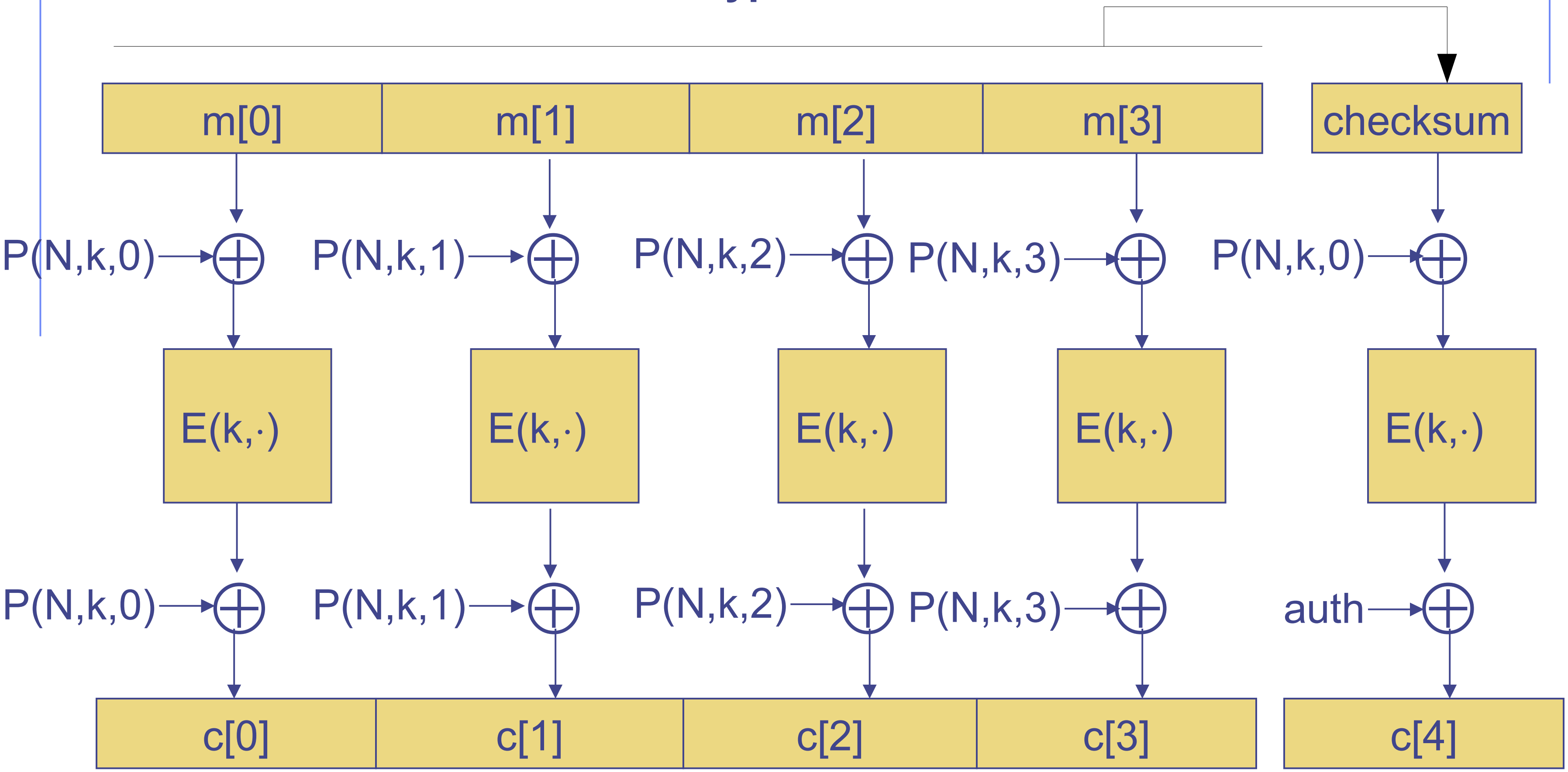

Option 3: Encrypt-and-MAC (SSH)



# OCB

offset codebook mode

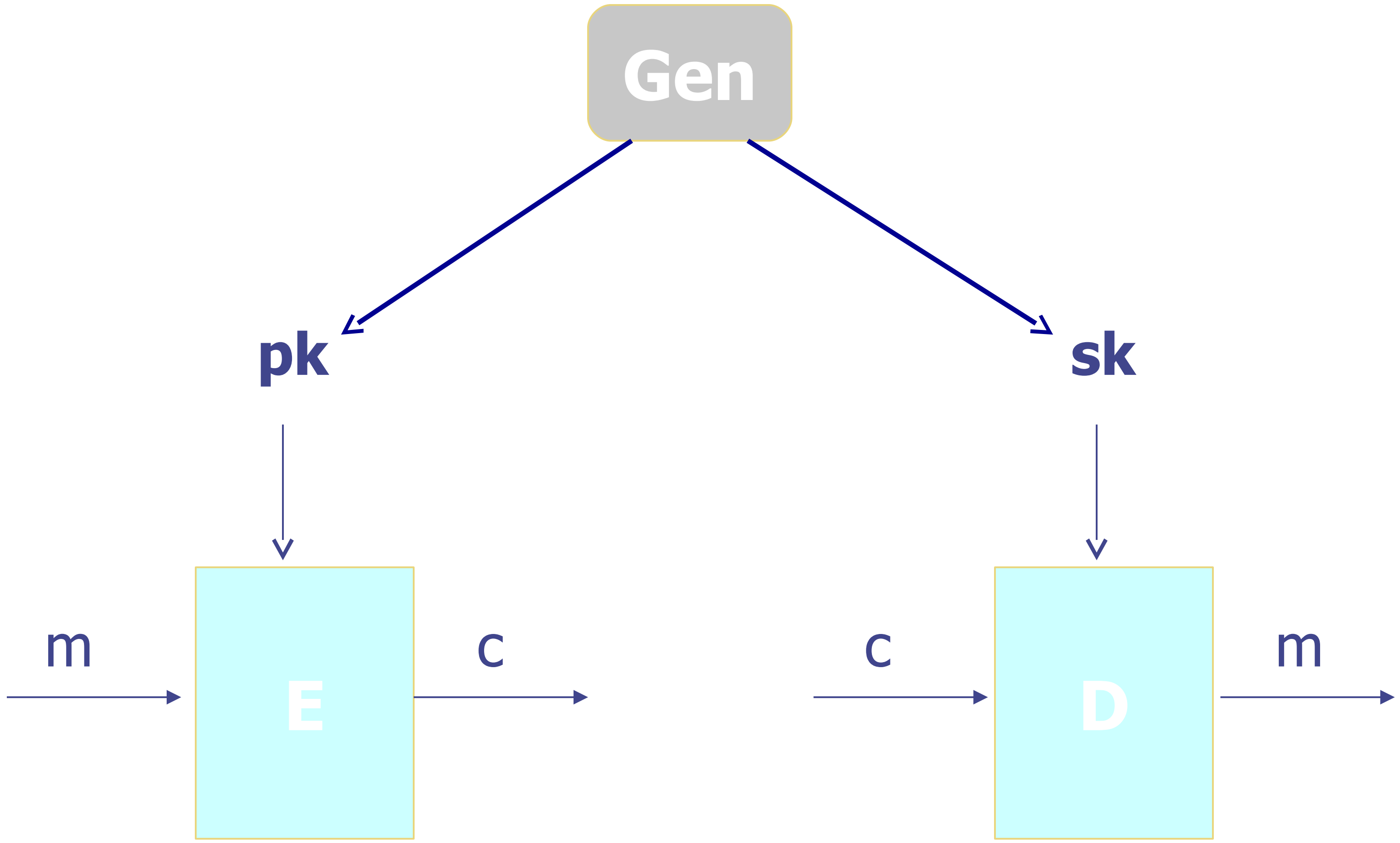
**More efficient authenticated encryption**




Rogaway, ...

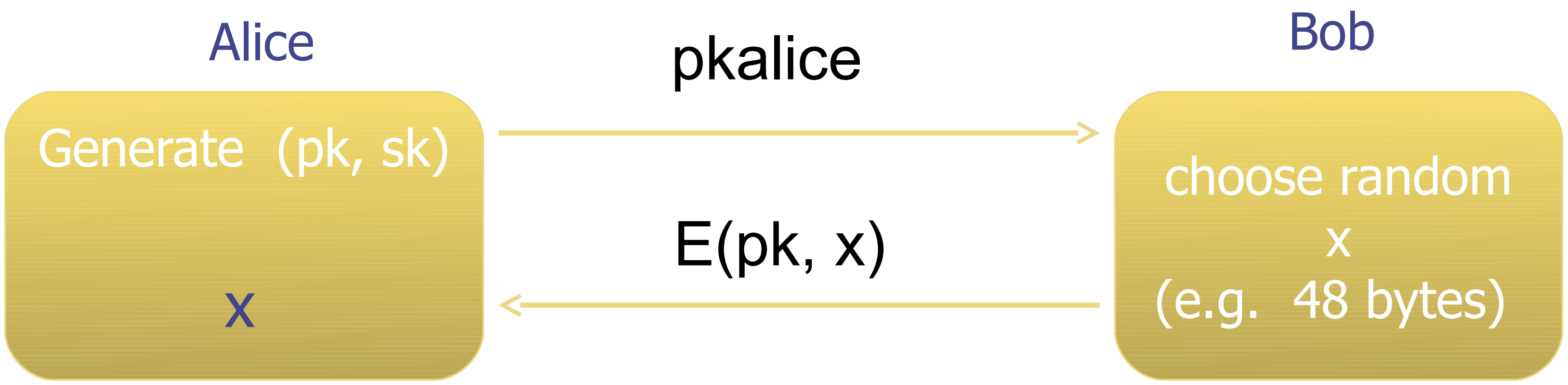
# Public-key Cryptography

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



# Applications

## Session setup (for now, only eavesdropping security)



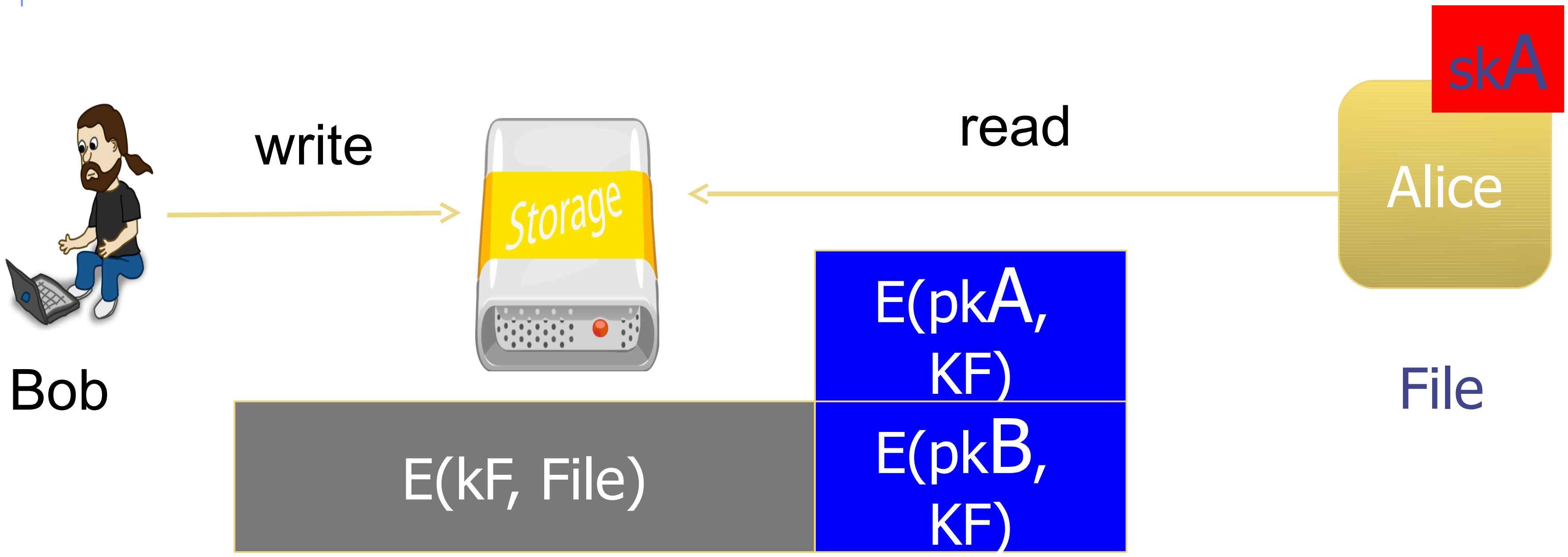

**Non-interactive applications**: (e.g. Email)

- Bob sends email to Alice encrypted using pkalice
- Note: Bob needs pkalice (public key management)

# Applications

Encryption in non-interactive settings:

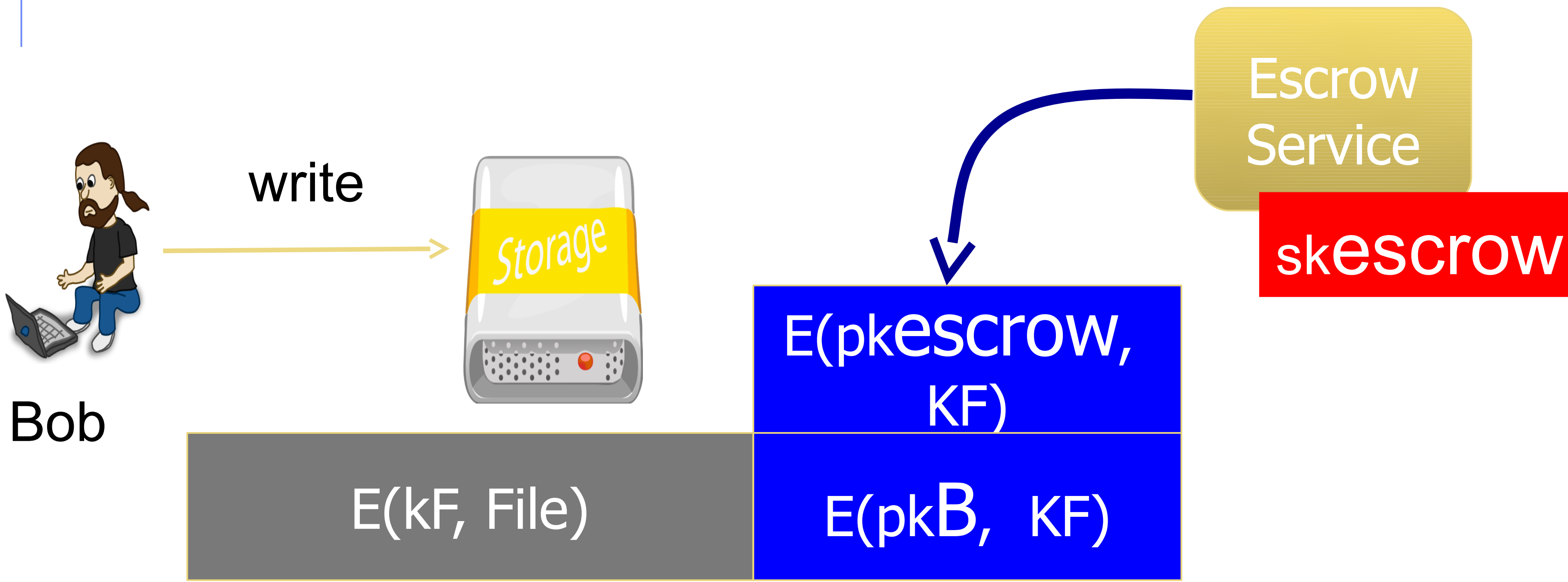
- Encrypted File Systems



# Applications

Encryption in non-interactive settings:

- Key escrow: data recovery without Bob's key



# Trapdoor functions (TDF)

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,⋅): $Y \longrightarrow X$ that inverts F(pk,⋅)

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

# Public-key encryption from TDFs

- (G, F, F-1):    secure TDF   $X \longrightarrow Y$
- (Es, Ds) :   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 $X \longrightarrow Y$
- (Es, Ds) : 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 $X \longrightarrow Y$
- (Es, Ds) : symm. auth. encryption with keys in K
- H: $X \longrightarrow K$ a hash function

**E( pk, m) :**

$x \longleftarrow X$, $y \longleftarrow F(pk, x)$

$k \longleftarrow H(x)$, $c \longleftarrow Es(k, m)$

output (y, c)

**D( sk, (y,c) ) :**

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

$k \longleftarrow H(x)$, $m \longleftarrow Ds(k, c)$

output m

# 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

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

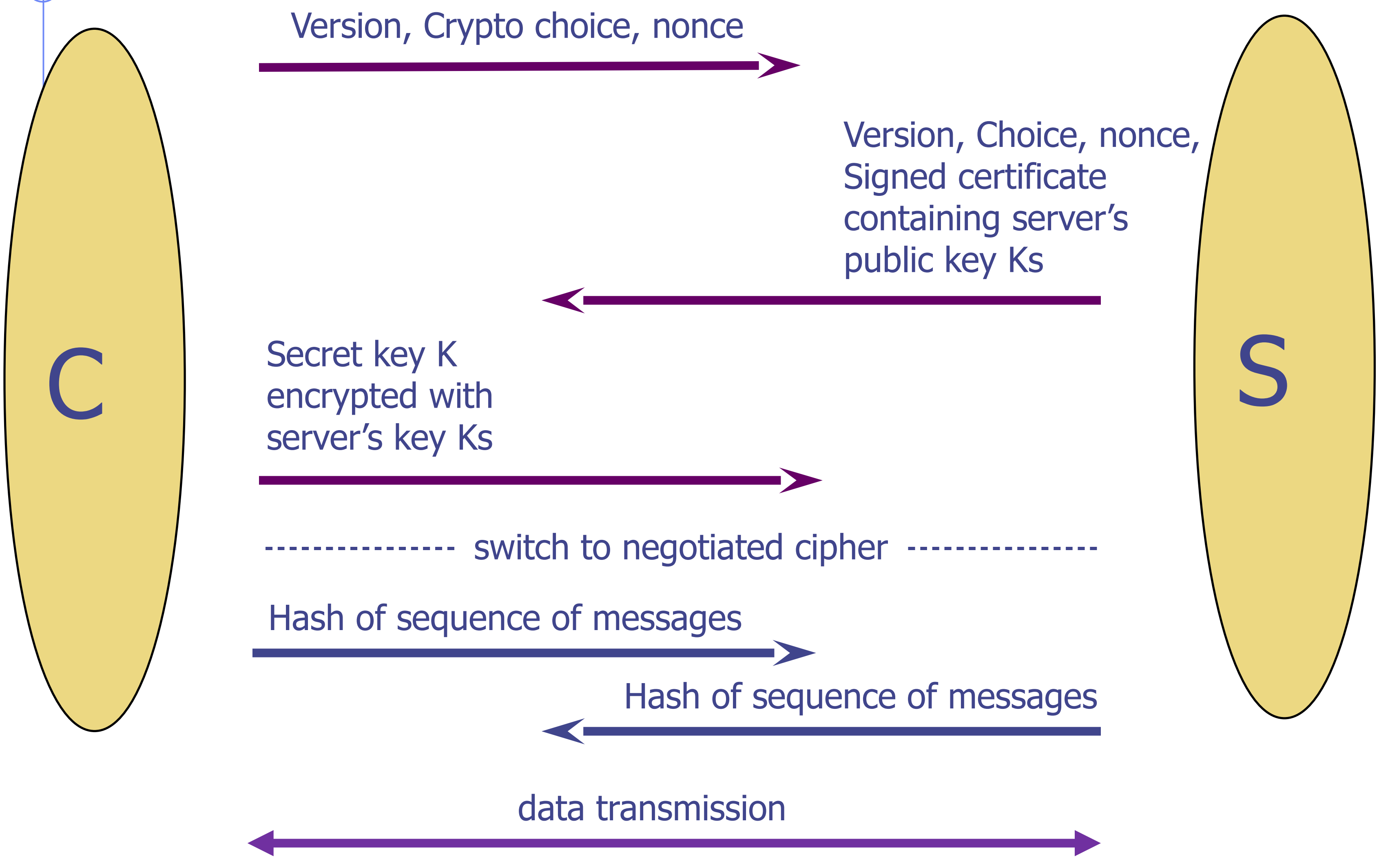
Security: existential unforgeability under a chosen message attack in the random oracle model

# 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

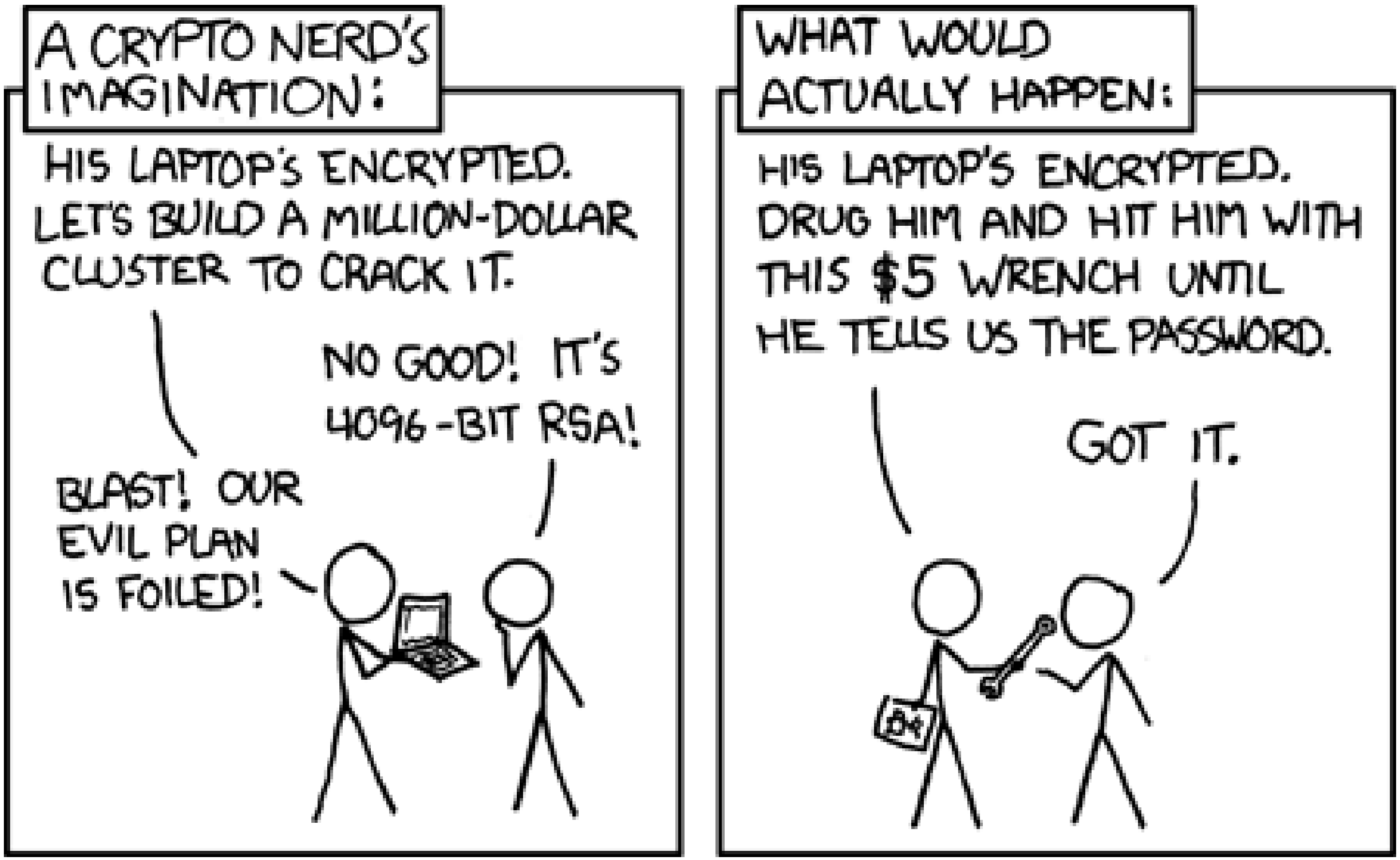
- One solution: PKI
  - Trusted root Certificate Authority (e.g. Symantec)
    - Everyone must know the verification key of root CA
    - Check your browser; there are hundreds!!
  - Root authority signs intermediate CA
  - Results in a certificate chain

# Back to SSL/TLS



# Limitations of cryptography

- Most security problems are not crypto problems
  - This is good
    - Cryptography works!
  - This is bad
    - People make other mistakes; crypto doesn't solve them
- Misuse of cryptography is fatal for security
  - WEP – ineffective, highly embarrassing for industry
  - Occasional unexpected attacks on systems subjected to serious review


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.