
# Introduction to Information Security

0368-3065, Spring 2015

## **Lecture 6:**
## Applied cryptography: symmetric

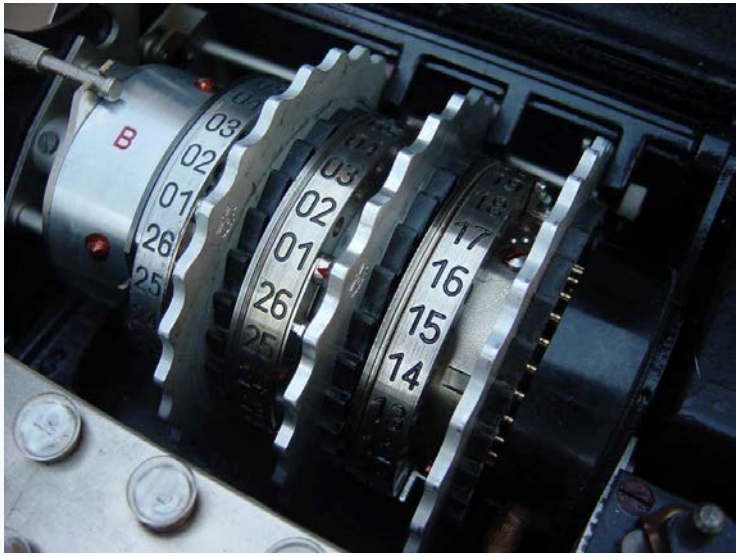
Eran Tromer

Slides credit:
John Mitchell, Stanford

# Cryptography overview

# History of crypto

- Ceaser cipher
- Electromechanical ciphers (e.g., Enigma)
- Information theory
- Complexity theory
- Modern cryptography



# 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



# Scenarios

- Storage
  - Store files privately
  - Protect files from tampering
- Communication
  - Avoid eavesdropping
  - Avoid corruption
  - “Secure channel”
- Authentication
- Many protocols



# Cryptographic primitives

Primitive: syntax and semantics of some cryptographic functionality.

A scheme/protocol/algorithm may be an instance of the primitive.

Example claim: "AES-128-CBC is a secure symmetric encryption scheme".

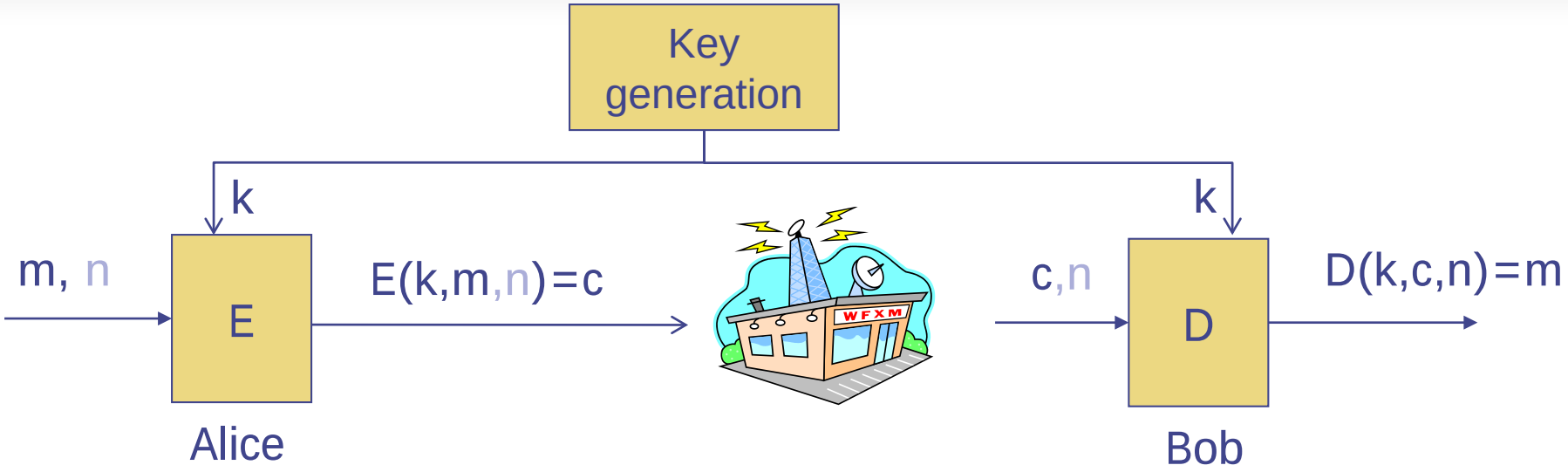
Most common cryptographic primitives (very informally):

- Encryption: confidentiality
  Adversary cannot learn anything about message from its encrypted form.
  - Symmetric (cipherblock ciphers, stream ciphers)
  - Asymmetric
- Digital Signatures: integrity/authentication
  Adversary cannot fake a signature that passes verification.
  - Symmetric (Message Authentication Code)
  - Asymmetric
- Hashing:
  Summarizing messages into "unique" short digests
  Adversary cannot find messages with the same digest.



# Symmetric cryptography schemes

# Symmetric encryption




E, D: cipher

m: plaintext

c: ciphertext

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

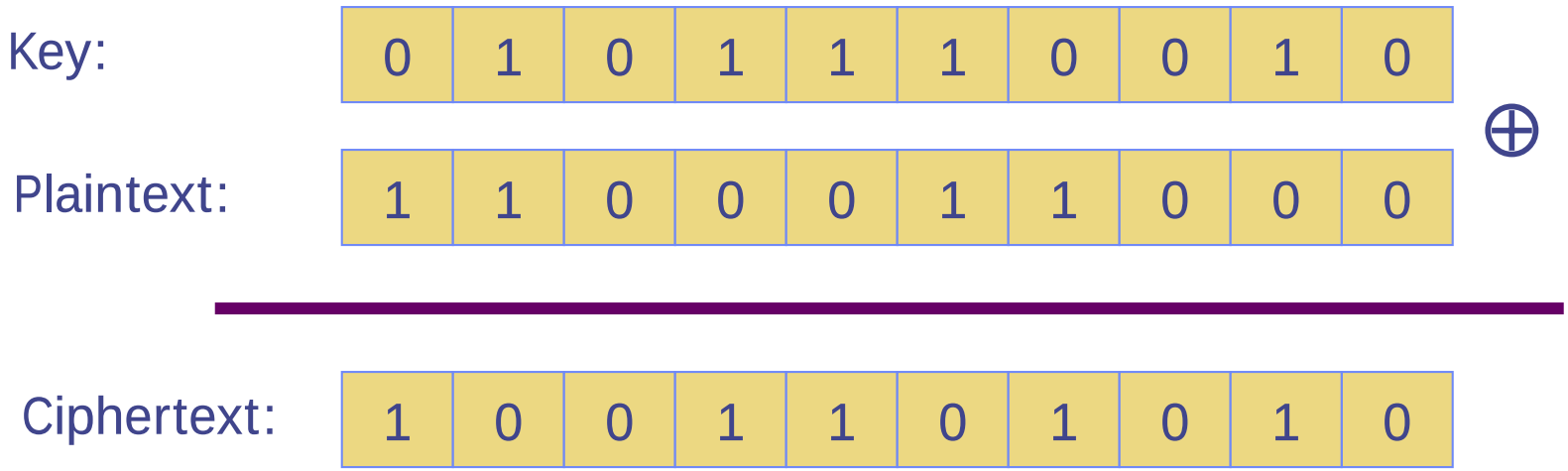
n: nonce / randomness / initial vector (IV)

Kerckhoff's Principle

A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.
(Never use a proprietary cipher)

# First example: One Time Pad (single use key)

- Vernam (1917)



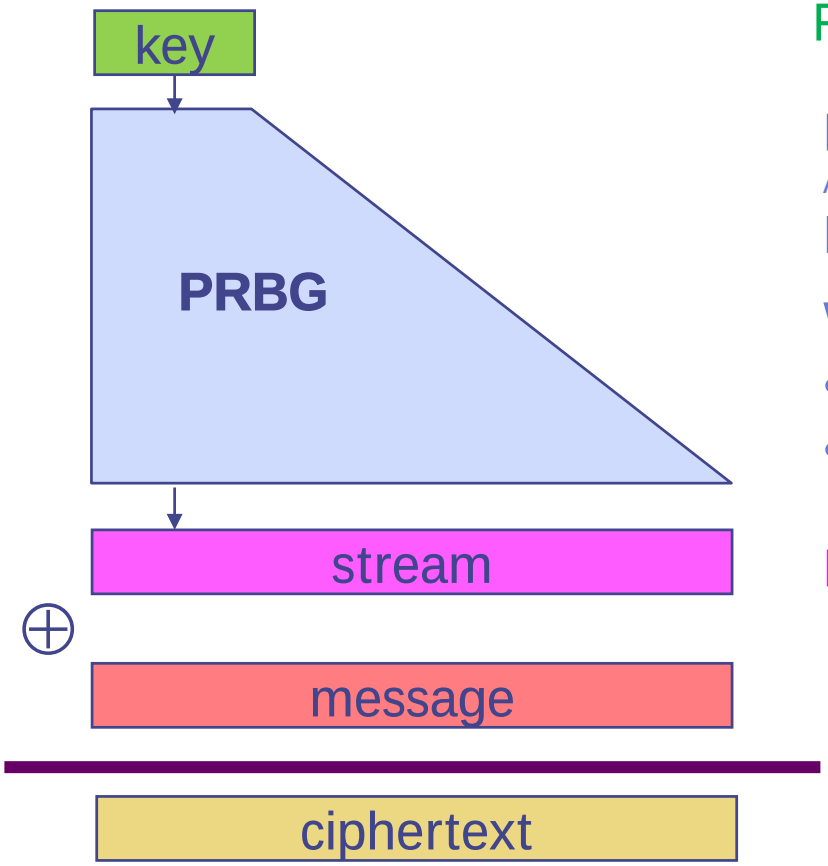

- Shannon (1949):
  - OTP is “secure” against ciphertext-only attacks

# Stream ciphers (single use key)

Problem: OTP key is as long the message

Solution: XOR with a *pseudorandom* seed:

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




Really random (physical source)

PRBG: "Pseudo-Random Bit Generator"
AKA
PRG: "Pseudo-Random Generator"

Well-known algorithms:

- RC4, RC5 (not secure anymore),
- AES in counter mode

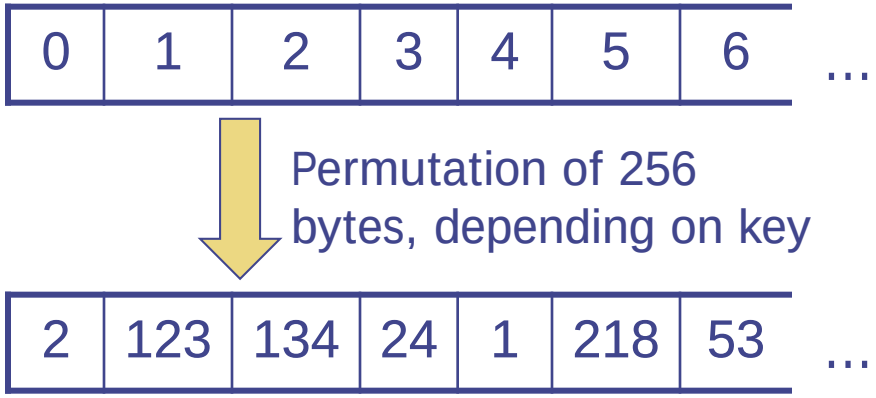
Pseudorandom string

# RC4 stream cipher [Rives 1987]

All values are bytes, all arithmetic is mod 256. S is a 256-byte state array.

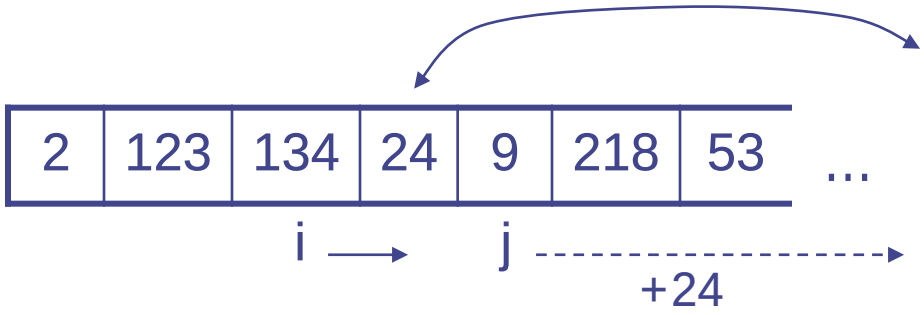
```
for i = 0..255:
    S[i] = i
j = 0
for i = 0 to 255
    j = j + S[i] + key[i]
    swap (S[i], S[j])
```

Key setup




```
i, j = 0
repeat:
    i = i + 1
    j = j + S[i]
    swap (S[i], S[j])
    output to stream: S[ S[i] + S[j] ]
```

Stream generator




Problem: statistical biases, especially in beginning of stream.

# Dangers in using stream ciphers

One time key!

"Two time pad" is insecure:

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

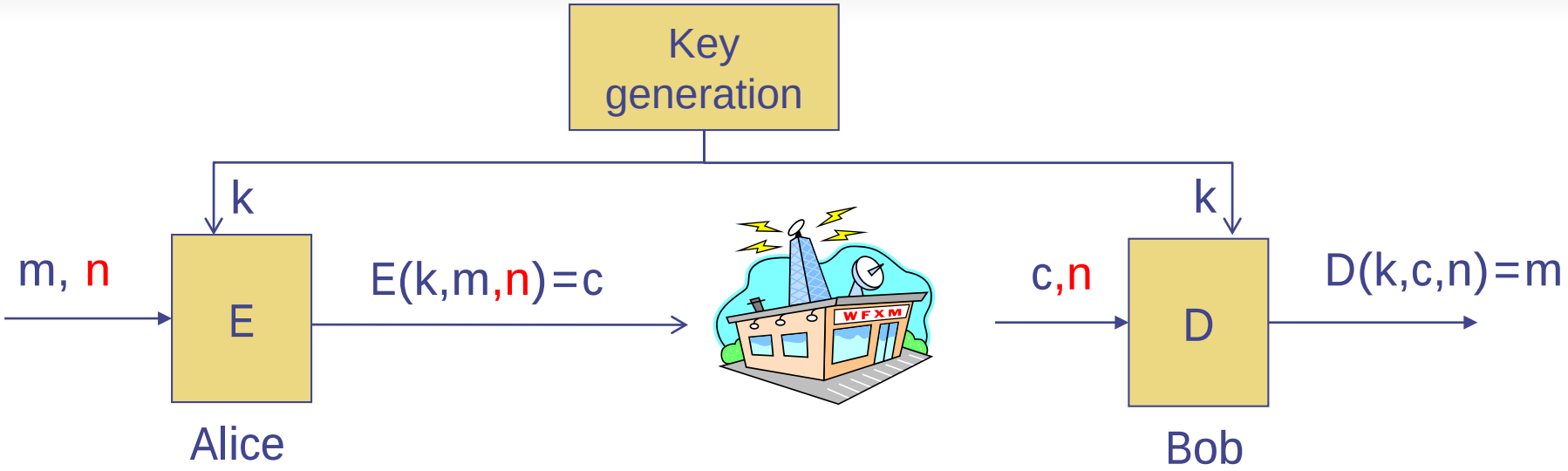
Eavesdropper does:

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

Enough redundant information in English that:

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

# Randomized symmetric encryption




E, D: cipher

m: plaintext

c: ciphertext

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

n: nonce / randomness / initial vector (IV)

(Notation: sometimes the nonce is included in c.)

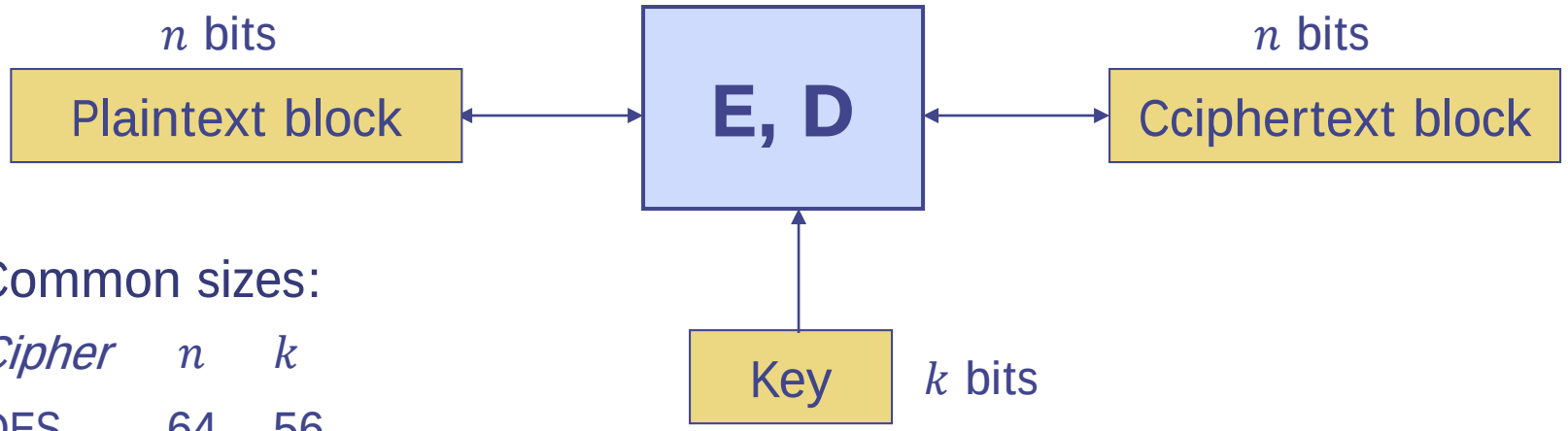
# Need for nonce

- 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:
  - Key used to encrypt multiple messages
    - SSL: same key used to encrypt many packets
  - Needs either unique nonce or random nonce
    - Otherwise, repeated messages can be identified
- Multi-use key, but all plaintexts are distinct:
  - Can eliminate nonce using special mode (SIV)



# Block ciphers: crypto work horse

Works on fixed-size blocks:




Common sizes:

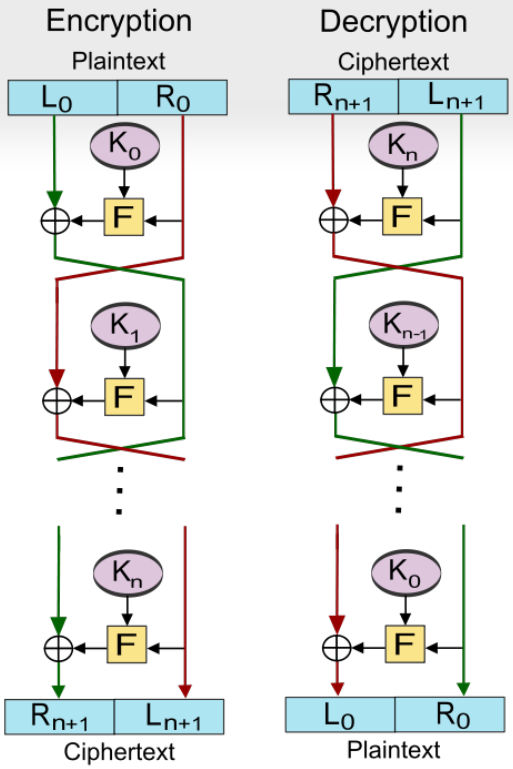
| *Cipher* | $n$ | $k$ |
|---|---|---|
| DES | 64 | 56 |
| 3DES | 64 | 168 |
| AES | 128 | 128/192/256 |

Deterministic; random IV can be supplied as part of plaintext block in a mode of operation (see next).).

Efficiency and security optimized for the specific block size.

Historically, more secure and easier to analyze than traditional stream ciphers.

# Building a block cipher




Input: (m, k)

- DES
  Data Encryption Standard [NIST 1977]
  Repeat simple mixing operation 16 times

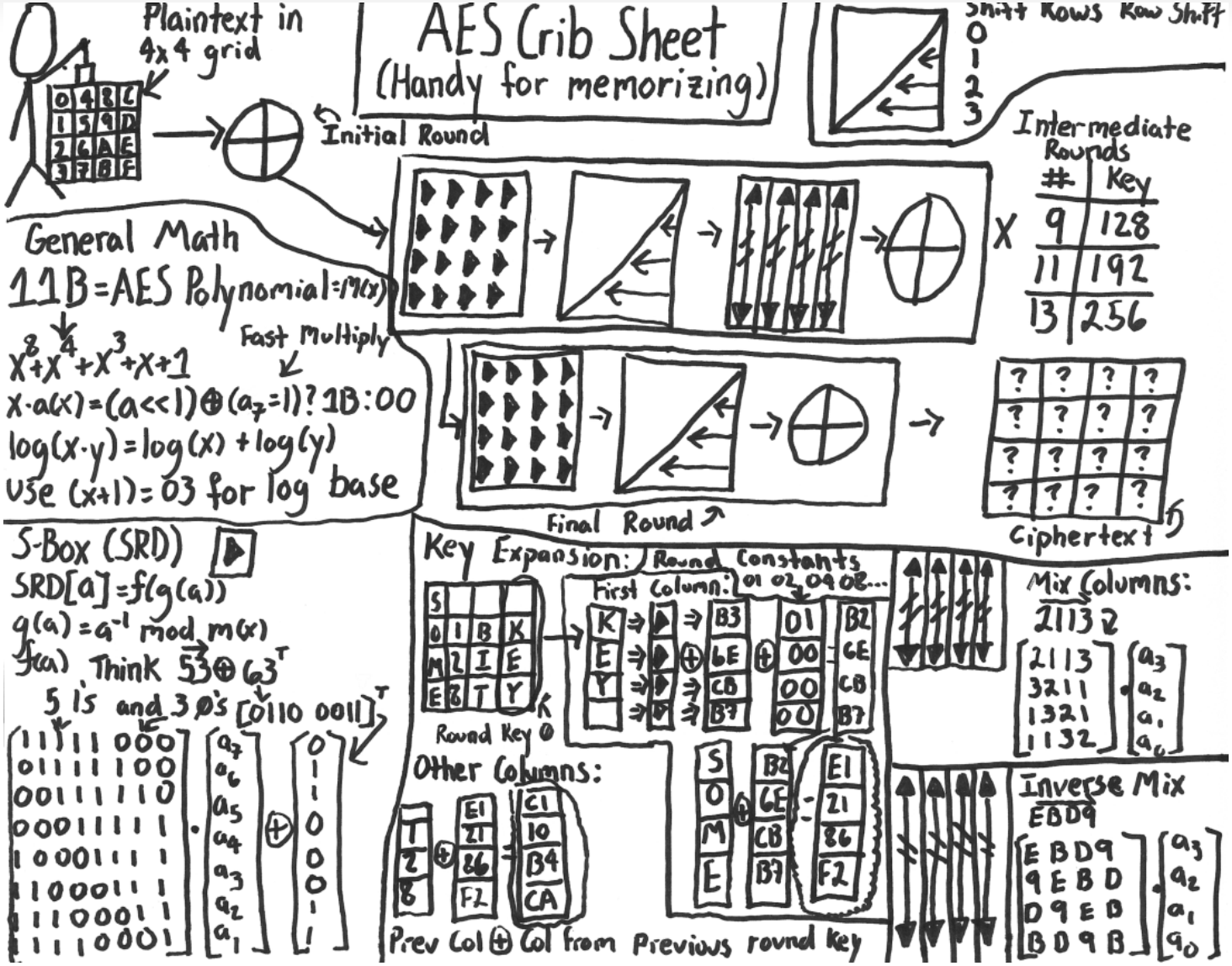
$$\begin{cases} m_L \leftarrow m_R \\ m_R \leftarrow m_L \oplus F(k, m_R) \end{cases}$$

- AES
  Advanced Encryption Standard
  [Rijmen Daemen 1998] [NIST 2001]
  Repeat a mixing step 10 times (for $k = 128$)

Difficult to design: must resist

- Brute force (guessing a key using $2^k$ attempts)
- Differential cryptanalysis, linear cryptanalysis, related-key attacks, side-channel attacks, ...

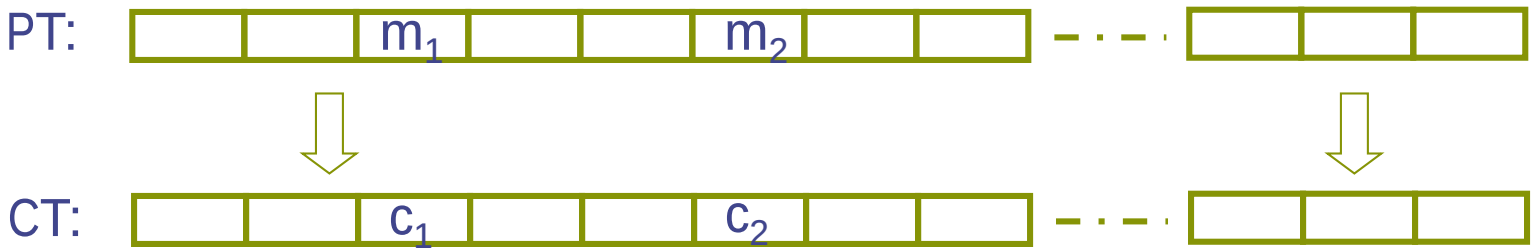
# AES Block Cipher (on a napkin)




Source: http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

# Incorrect use of block cipher: ECB (Electronic Codebook) mode

Break plaintext into blocks and encrypt independently:

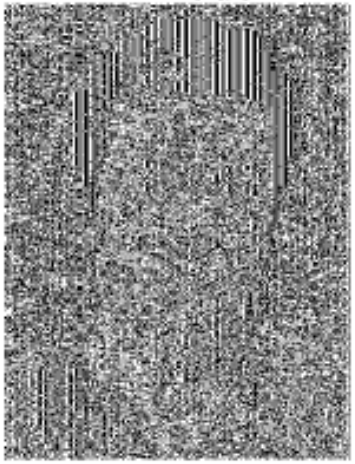



- Problem:
  - if $m_1 = m_2$ then $c_1 = c_2$

# In pictures

An example plaintext

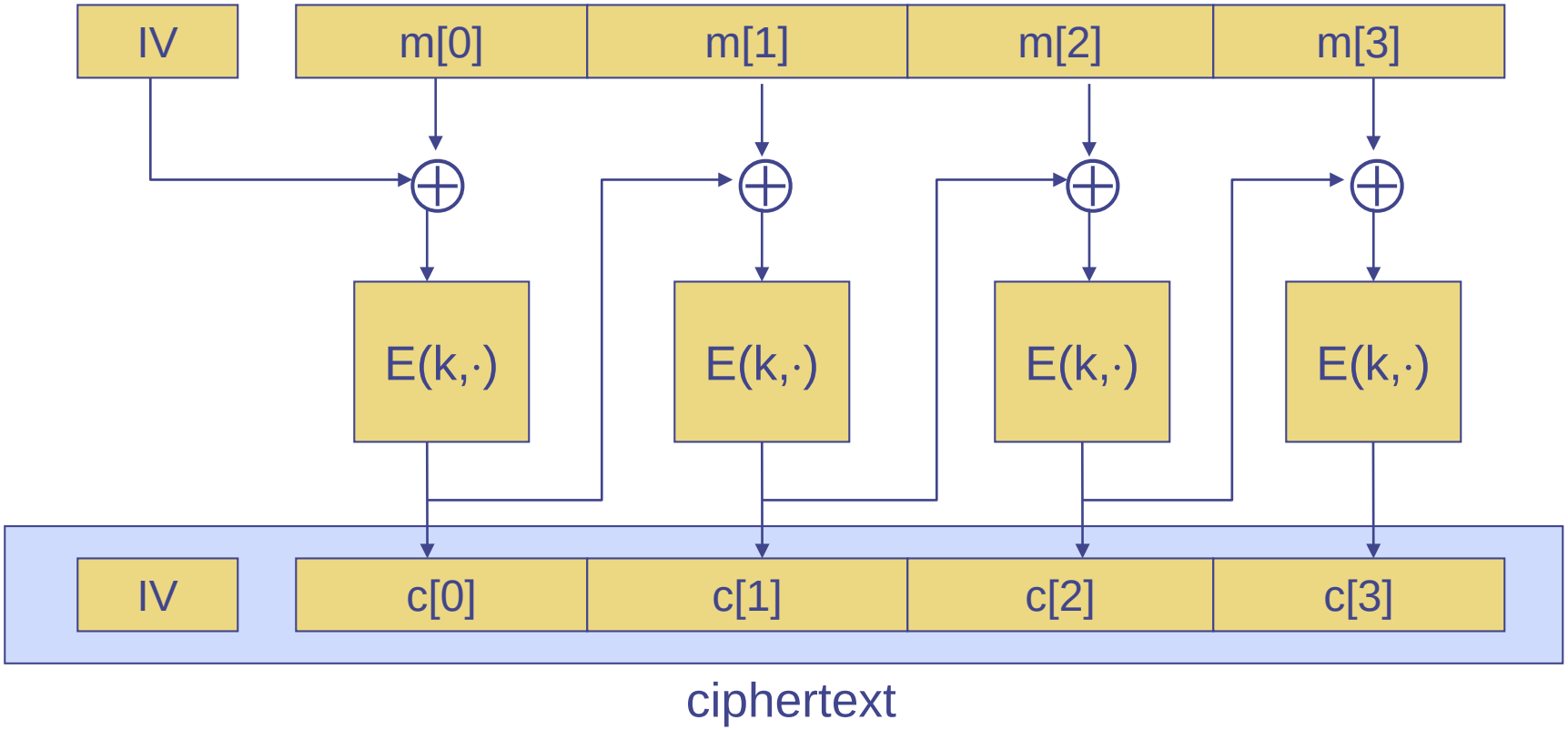
Encrypted with AES in ECB mode



[visual comparison by Bart Preneel]

# Correct use of block ciphers I: CBC (Cipher Block Chaining) mode

Assume E is a secure "pseudorandom permutation" (i.e., a good block cipher).
To encrypt a plaintext message m whose size is a multiple of the block size:



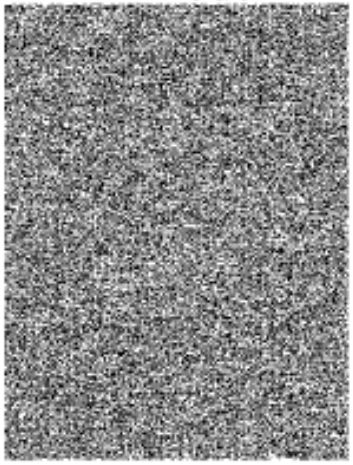

To handle messages of different length, use suitable *padding*.

Q: how to do decryption?

# In pictures

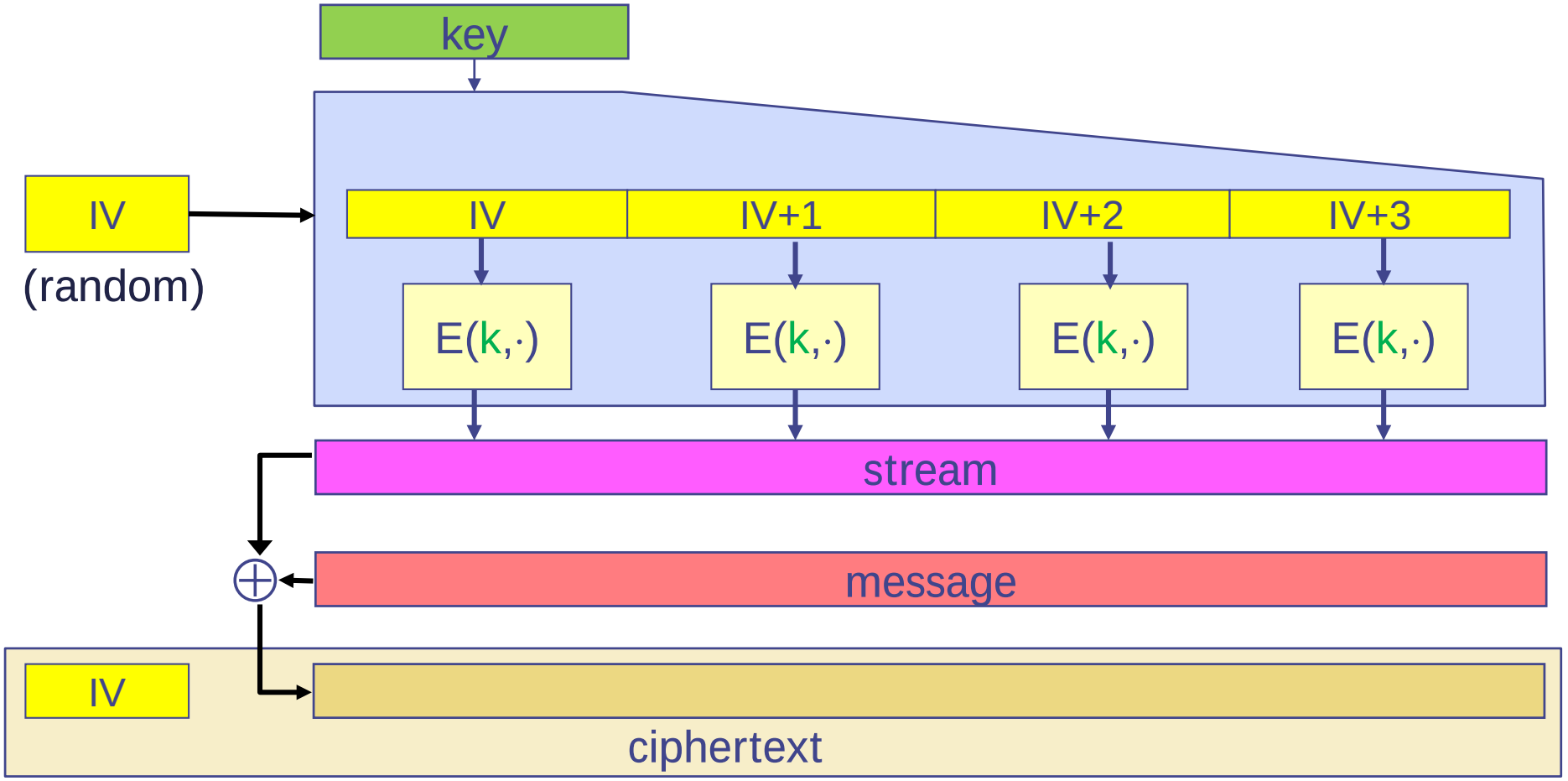
An example plaintext

Encrypted with AES in CBC mode



# Correct use of block ciphers II: CTR (Counter) mode

Encrypt a counter to create a pseudorandom string (similar to stream cipher).




Advantage: allows parallel encryption.

These “modes of operations” are proven secure (under suitable definitions).

# Performance of ciphers

| Algorithm | Speed (Mbyte/sec) | Block size (bits) | Key size (bits) |
|---|---|---|---|
| RC4 | 657 | | unlimited |
| DES | 71 | 64 | 56 |
| 3DES | 26 | 64 | 168 |
| IDEA | 91 | 64 | 128 |
| AES | 142 | 128 | 128 |

OpenSSL 1.0.1 on 2.1GHz Intel i7, 64-byte messages.

To benchmark on your computer:

```
$ openssl speed rc4 des-cbc des-ede3 idea-cbc aes-128-cbc
```

Fast, but still bottleneck for:

- Applications applying crypto at high bandwidth with little processing (web servers, bulk storage encryption, virtual private networks)
- Small devices (weak CPU, power-limited)

Cheaper/faster with dedicated hardware

- AES instructions in recent x86
- Dedicated “cryptographic accelerator” cards



# Attacks and security definitions

**What power does the adversary have?**

- Known-ciphertext attack
- Chosen-ciphertext attacks (CCA)
- Known-plaintext attack
- Chosen-plaintext attack (CPA)
  - Chosen before target ciphertext
  - Chosen after target ciphertext
- Related-key attack

**What secrecy is preserved?**

- Key remains secret
- Adversary can learn nothing about a message from observing its encryption
  - For random messages
  - Even if it's an encryption of one of two messages chosen by adversary, he can't tell which one it is



# Hash functions and message integrity

# Cryptographic hash functions

- Length-reducing function H
  - Map arbitrary strings to strings of fixed length
- One way ("preimage resistance")
  - Given y, hard to find x with H(x)=y
- Collision resistant
  - Hard to find any distinct m, m' with H(m)=H(m')
- Also useful: 2nd preimage resistance
  - Given x, hard to find x'≠x with H(x')=H(x)

Collision resistance ⇒ 2nd preimage resistance ⇒ preimage resistance



# Applications of cryptographic hashing

- Password files (one way)
- Digital signatures (collision resistant)
  - Sign hash of message instead of entire message
- Data integrity
  - Compute and securely store hash of some data
  - Check later by recomputing hash and comparing
- Keyed hash for message authentication
  - MAC – Message Authentication Code



# Performance of hash functions

| Algorithm | Speed (Mbyte/sec) | Block size (bits) | Key size (bits) |
|---|---|---|---|
| RC4 | 657 | | unlimited |
| DES | 71 | 64 | 56 |
| 3DES | 26 | 64 | 168 |
| IDEA | 91 | 64 | 128 |
| AES | 142 | 128 | 128 |
| SHA-1 | 197 | | |
| SHA-256 | 113 | | |

OpenSSL 1.0.1 on 2.1GHz Intel i7, 64-byte messages.

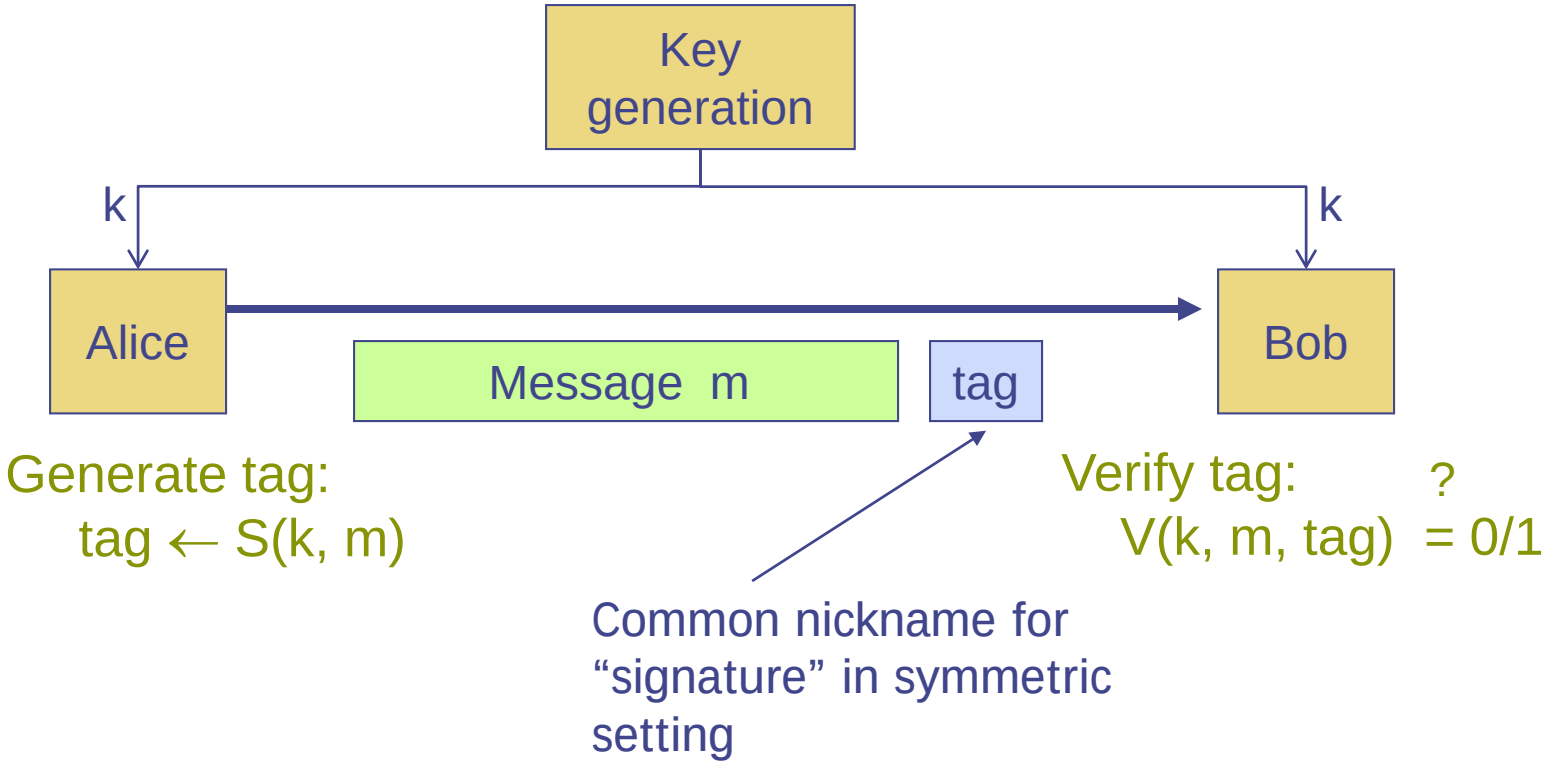
To benchmark on your computer:

```
$ openssl sha1 sha256
```



# Symmetric-key integrity: MAC (Message Authentication Code)

- Goal: message integrity. No confidentiality.




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

# Secure MACs

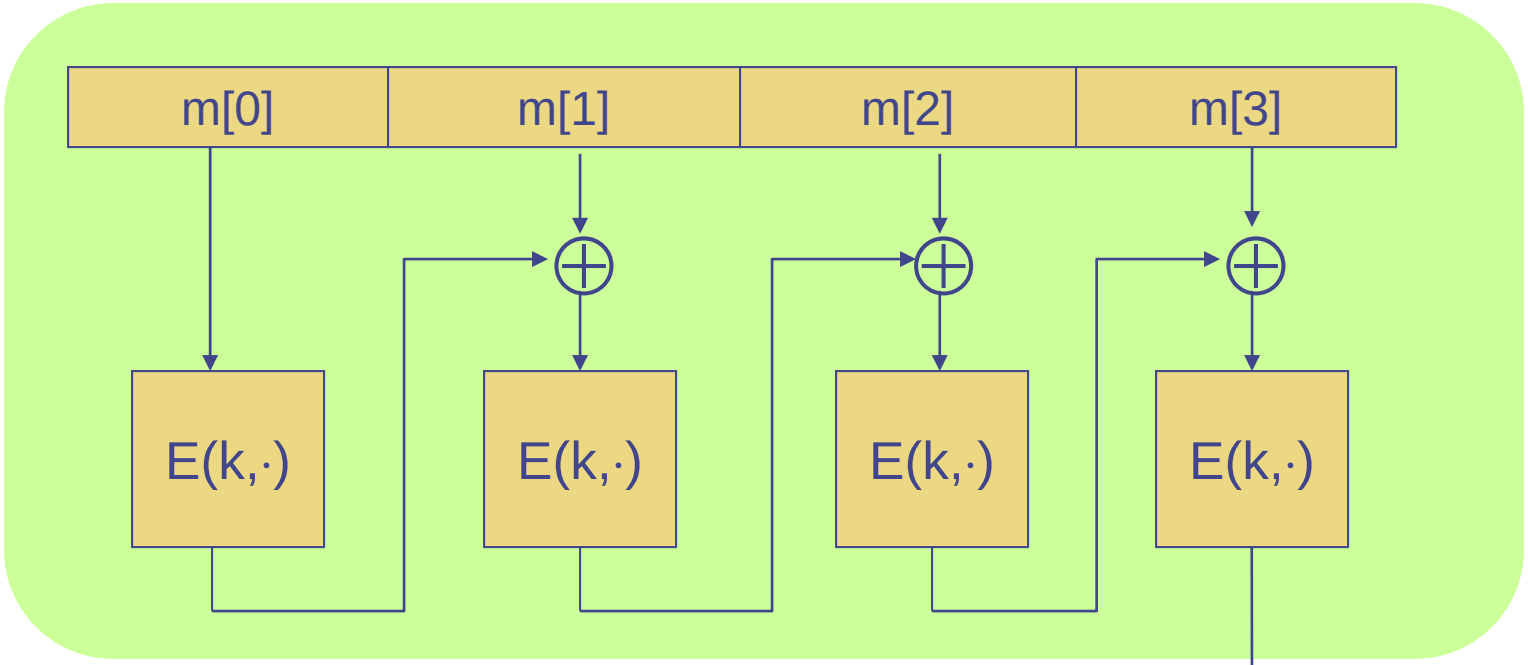
- Attacker's power: chosen message attack.
  - for $m_1, m_2, \ldots, m_q$ attacker is given $t_i \leftarrow S(k, mi)$
- Attacker's goal: existential forgery.
  - produce some **new** valid message/tag pair (m,t).

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

- *Existential unforgeability* security:
  no feasible attacker can witn the above game with more than negligible probability.

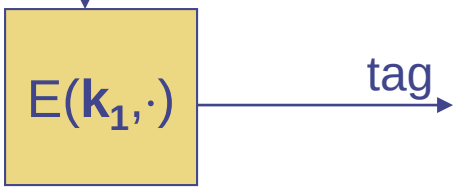


# MAC Construction 1: ECBC




Raw CBC

key = (k, $k_1$)



# MAC 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

$S(k, m) = H(k \oplus opad \,||\, \mathbf{H(k \oplus ipad \,||\, m)})$