Fondamenti di Cybersecurity – Modulo I

- 20h circa
- Docente: **Riccardo Treglia**
  - Email: **riccardo.treglia@unibo.it**

Piattaforma didattica

- Virtuale

  e verrà costantemente aggiornato con:
  - Informazioni
  - **Materiale didattico (slides)**
  - **Annunci**

Materiale didattico

- **Slide** caricate su Virtuale del corso
- Testi consigliati:
  - Jean-Philippe Aumasson,
    Serious Cryptography: A Practical Introduction to Modern Encryption.
  - Bruce Schneier,
    Applied Cryptography: Protocols, Algorithms, and Source Code in C.
  - Mark Stamp,
    Information Security: Principles and Practice.
  - William Stallings
    Crittografia
  - Dan Boneh, Victor Shoup,
    A Graduate Course in Applied Cryptography. (approccio matematico)

Esame

- Prova scritta
- Voto finale = Scritto + Successo laboratori

Scritto: 24/25 pt
Laboratori: max 8 pt
NO orali

- Date esami: consultare il sito del Dipartimento
  Due appelli a **Giugno**, uno a **Luglio** e uno a **Settembre**

Roadmap

0. What is Cryptography - History of Cryptography

1. Introduction Mathematics: Modular Arithmetic - Discrete Probability

2. One-time pad, Stream Ciphers and Pseudo Random Generators

3. Attacks on Stream Ciphers and The One-Time Pad

4. Real-World Stream Ciphers (weak(RC4), eStream,nonce, Salsa20)

5. Secret key cryptographic systems;

6. Public key cryptographic systems

7. DES protocols (just as an introduction), AES

8. Electronic Signatures, Public-key Infrastructure, Certificates and Certificate Authorities

9. Sharing of secrets; User authentication; Passwords

10. Tutor Training

Bonus. Legislation, Ethics and Management

Introduction

Welcome

Course **objectives**:

- Learn how crypto primitives work
- Learn how to use them correctly and reason about security

Che cos’è la Crittografia?

- **Crittografia**
  - *Kryptós*: nascosto
  - *Graphía:* scrittura
  - Metodi che consentano di **memorizzare**, **elaborare** e **trasmettere** informazioni in presenza di agenti ostili

- **Crittoanalisi**
  - Analisi di un testo cifrato nel tentativo di decifrarlo senza possedere la chiave

- **Crittologia:** Crittografia + Crittoanalisi

Cryptography is everywhere

Secure communication:
- web traffic: HTTPS
- wireless traffic: Wireless Network, GSM, Bluetooth

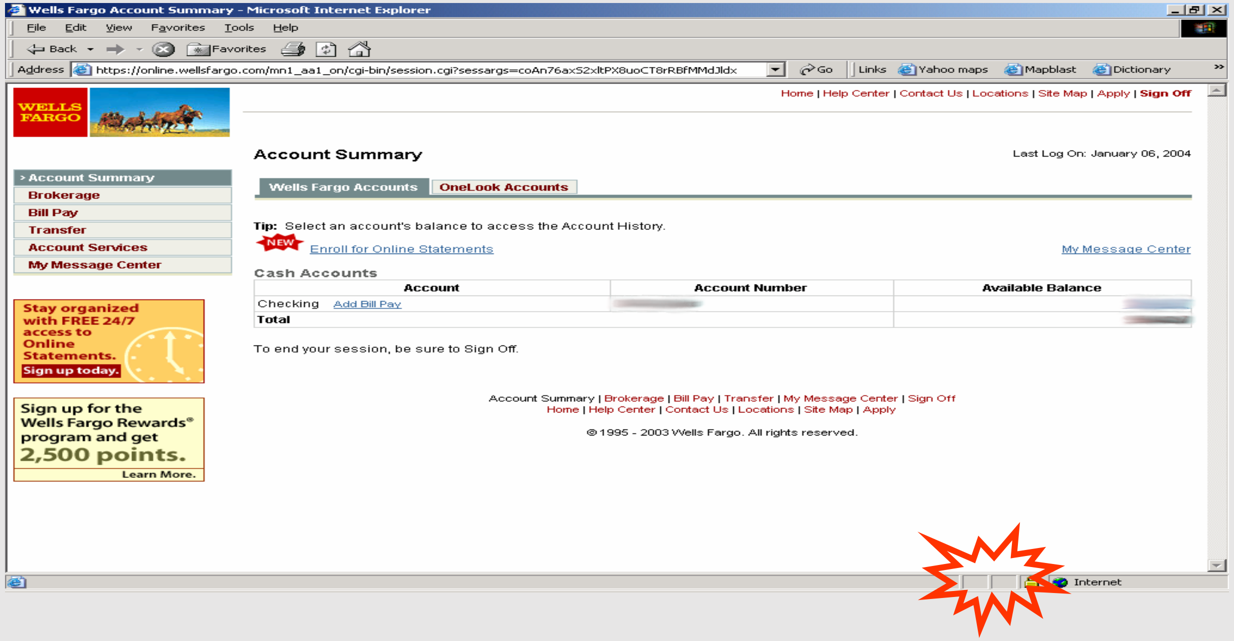
Encrypting files on disk

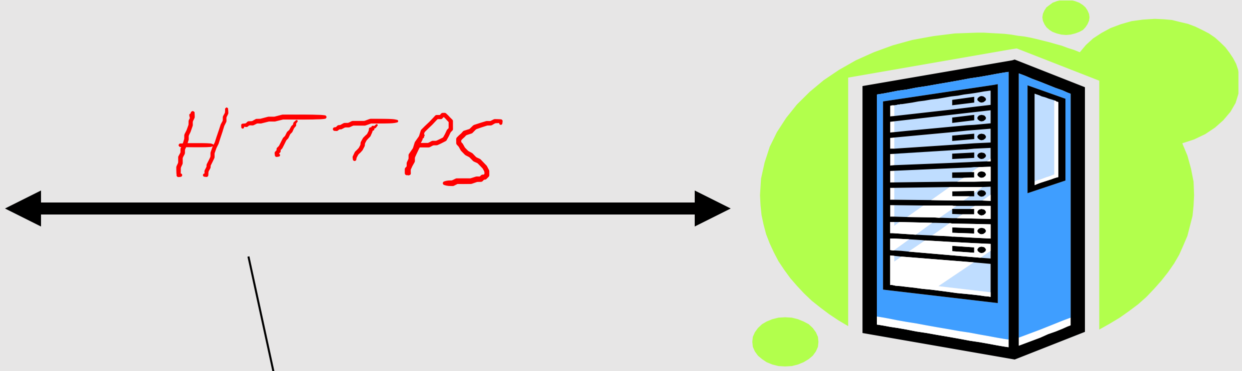
Content protection (e.g., DVD, Blu-ray)

User authentication

… and much much more (more “magical” applications later…)

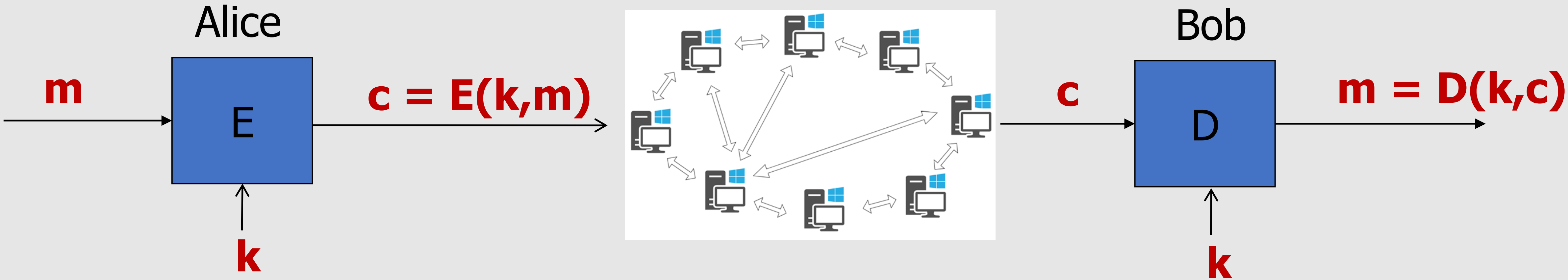
Secure communication





no eavesdropping
no tampering

Symmetric Encryption (confidentiality)



- **k:** **secret key (A SHARED SECRET KEY)**
- **m:** **plaintext**
- **c:** **ciphertext**
- **E:** **Encryption algorithm**
- **D:** **Decryption algorithm**
- **E, D:** **Cipher**

- **Confidentiality scenario**
- Other scenarios are possible, with the secret key used differently…
  - e.g., **MACs** (for **integrity**)

Algorithms are publicly known, never use a proprietary cipher

Use Cases

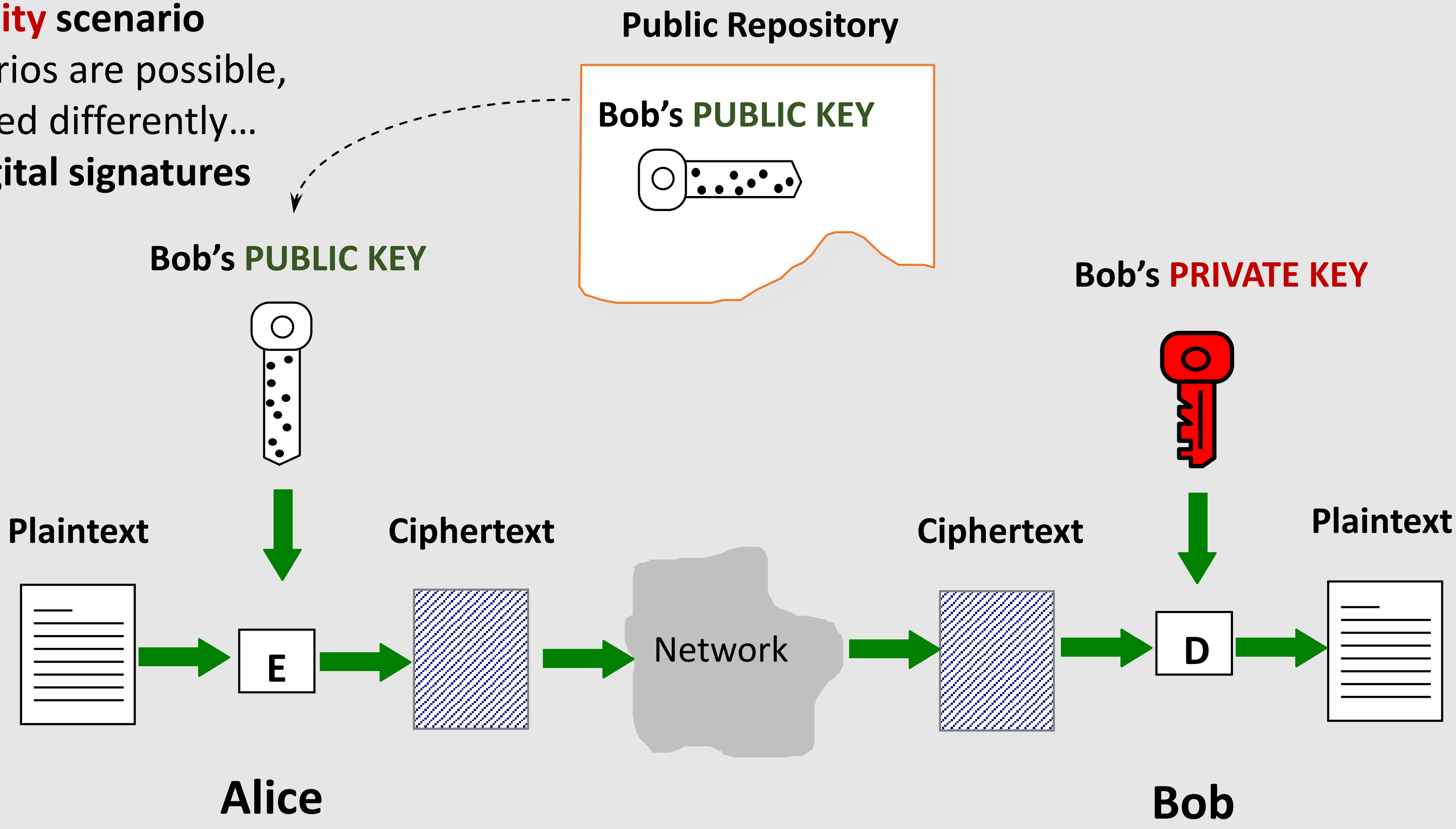
- **Single-use key**: (or **one-time key**):
  Key is only used to encrypt **one message**
  - encrypted email: new key generated for every email

- **Multi-use key**: (or **many-time key**):
  Same key used to encrypt **multiple messages**
  - encrypted files: same key used to encrypt many files

  Need more machinery than for one-time key

Asymmetric Encryption

- **Confidentiality scenario**
- Other scenarios are possible, with keys used differently…
  - e.g., **Digital signatures**



Things to remember

Cryptography is:

- A tremendous tool
- The basis for many security mechanisms

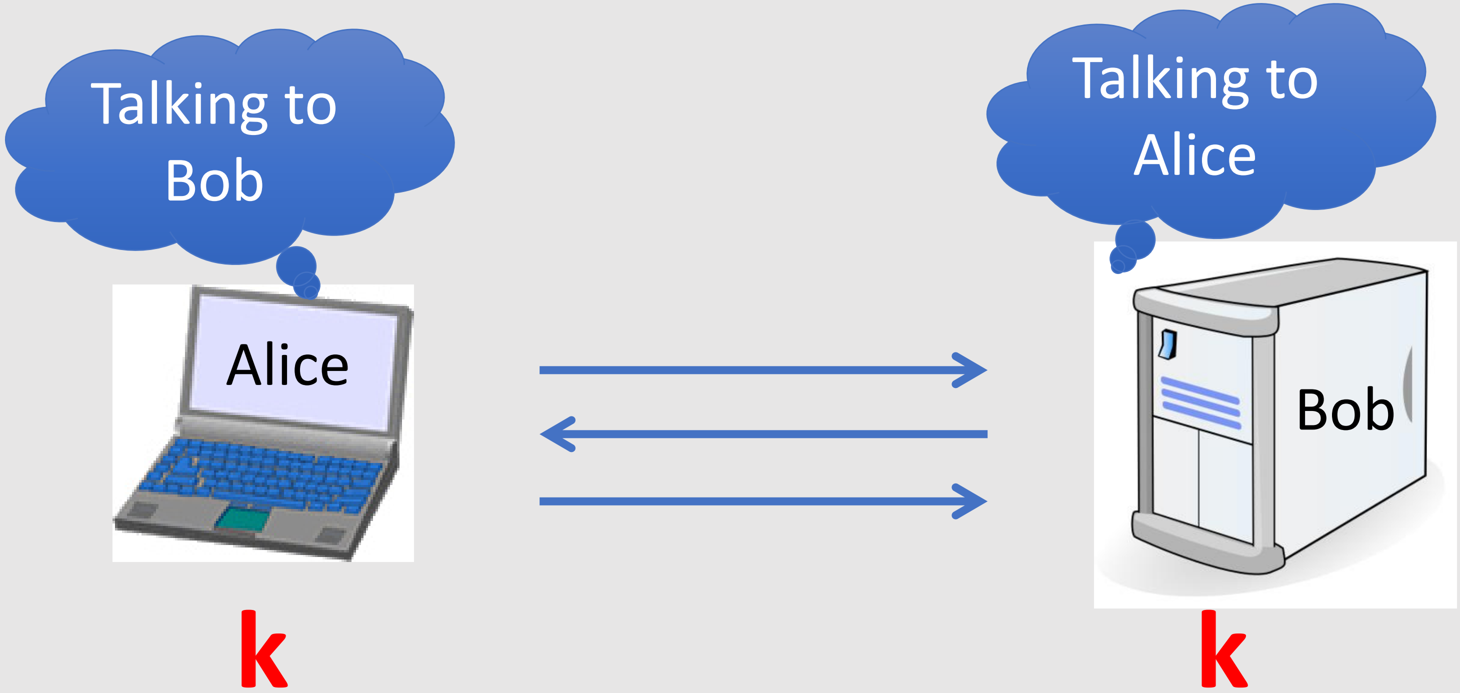
Cryptography is **not**:

- The solution to all security problems
- Reliable unless implemented and used properly
- Something you should try to invent yourself
    - many many examples of broken ad-hoc designs

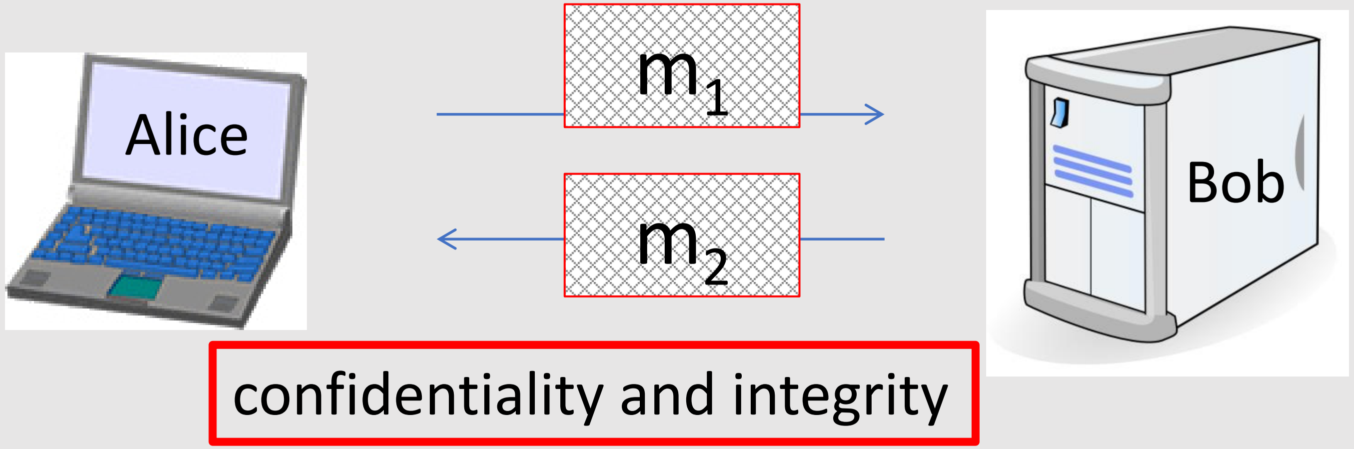
Some Applications

Secure communication

1. Secret key establishment:



2. Secure communication:



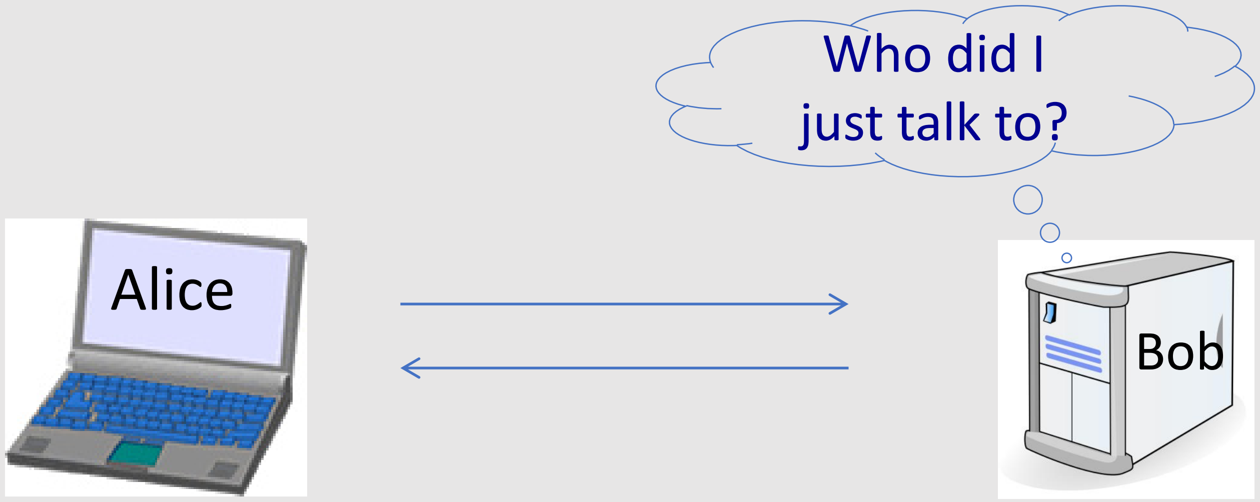
But crypto can do much more

- Digital signatures

- Signatures of the same person change over different documents
- Asymmetric Cryptography is used

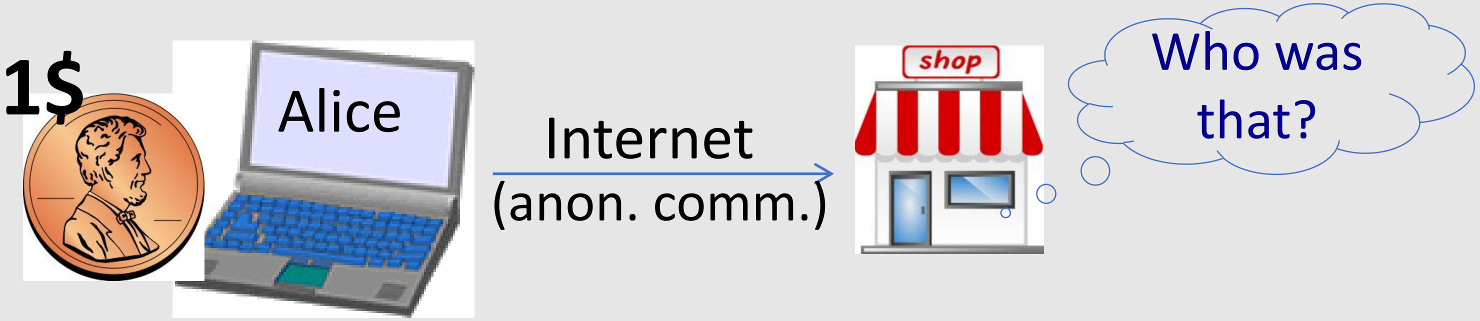
But crypto can do much more

- Anonymous communication
  (e.g., mix networks)



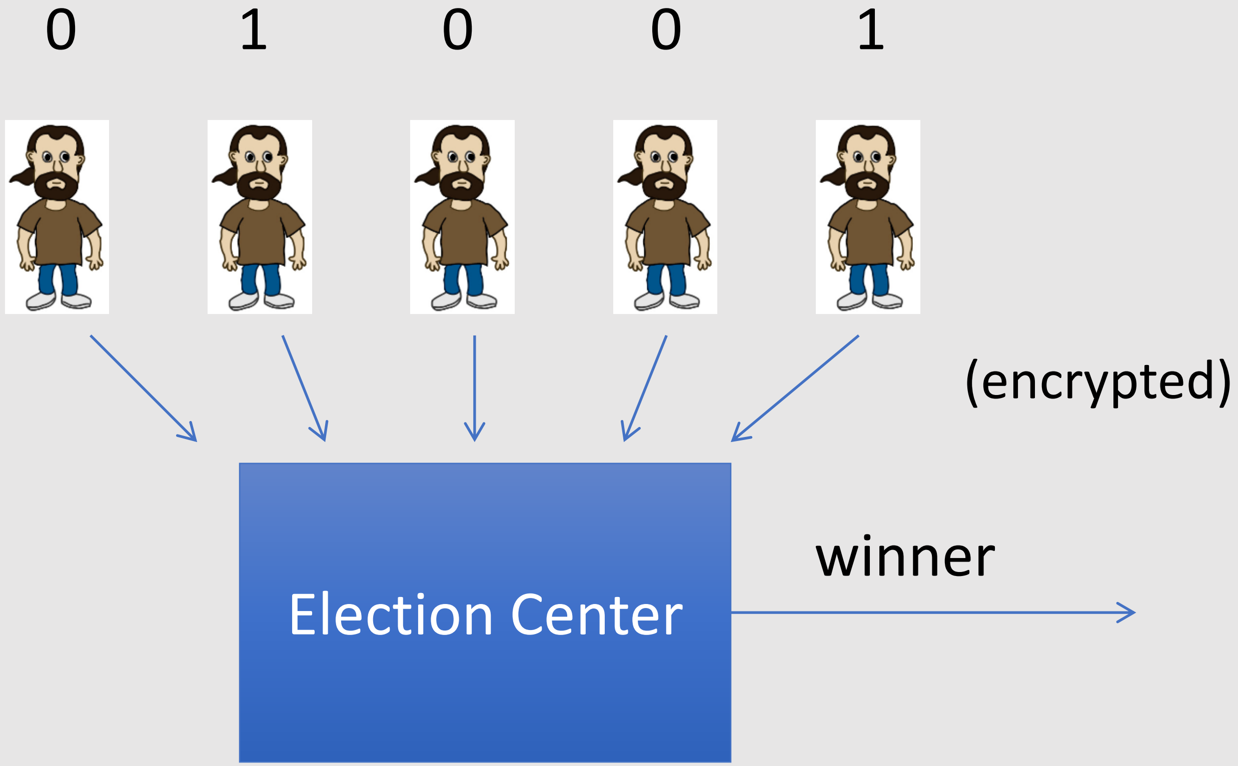
But crypto can do much more

- Anonymous **digital** cash
  - Can I spend a “digital coin” without anyone knowing who I am?
  - How to prevent double spending?



Protocols

- Elections
- Private auctions



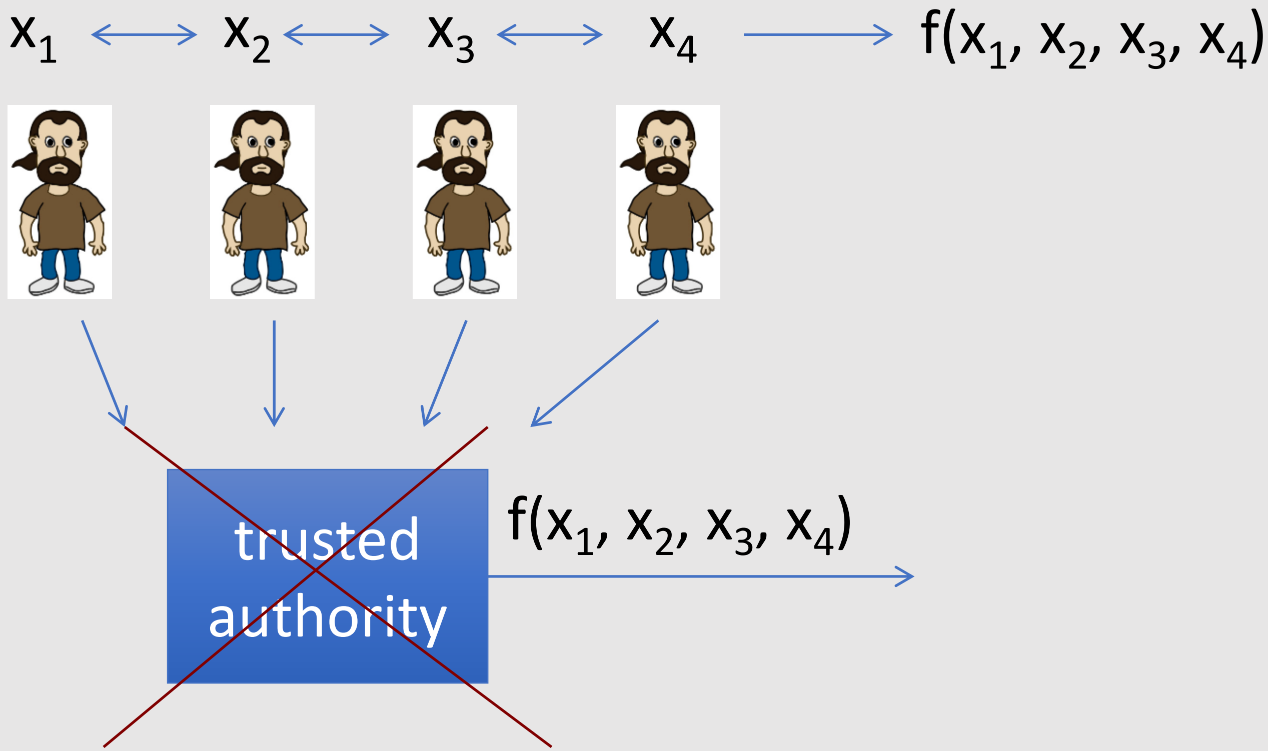
winner= majority [votes]

(Vickrey Auction)
Auction winner = highest bidder
pays 2^{nd} highest bid

Election Center must determine the winner
without knowing the individual votes!

Protocols

- Elections
- Private auctions



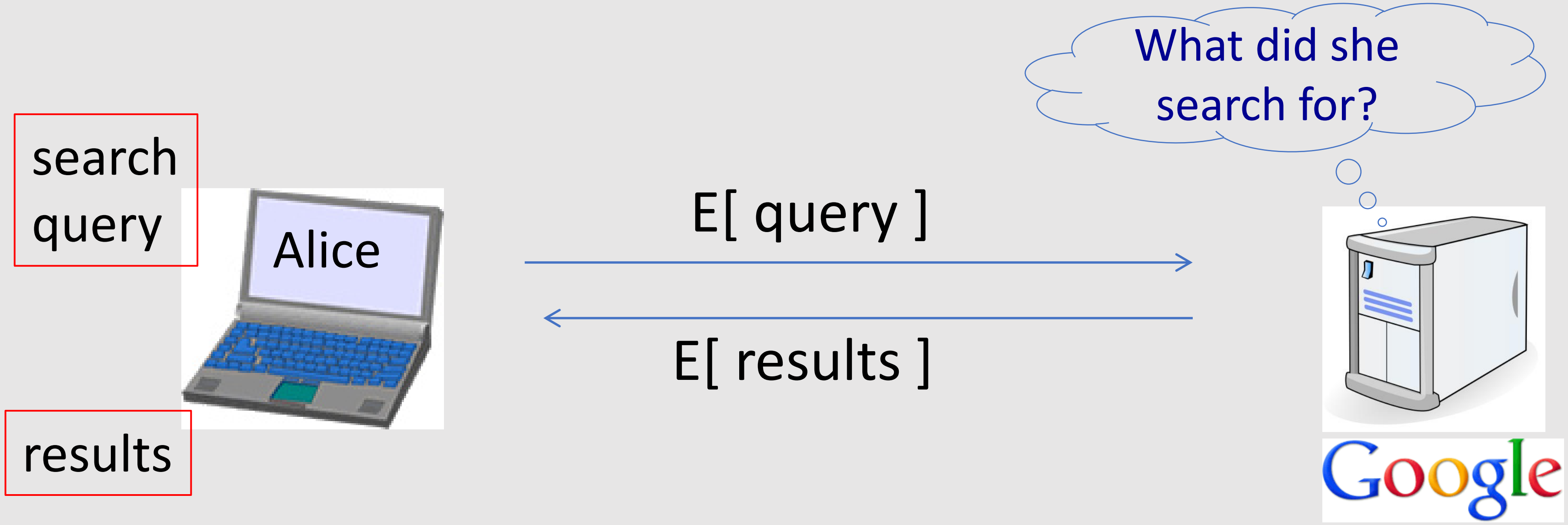
Secure multi-party computation

Goal: compute $f(x_1, x_2, x_3, x_4)$

"Thm:" anything that can done with trusted auth. can also be done without

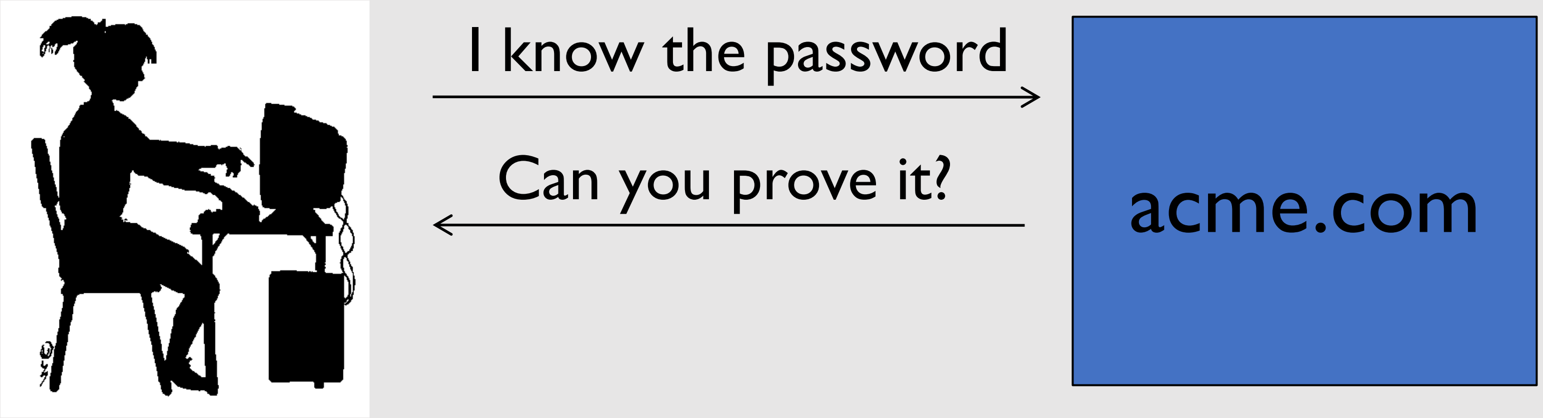
Crypto magic

- Privately outsourcing computation



Crypto magic

- Zero knowledge (proof of knowledge)



A rigorous science

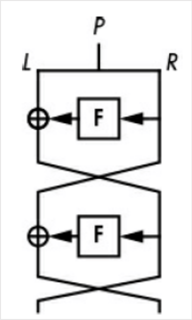
The three steps in cryptography:

- Precisely specify threat model
- Propose a construction
- Prove that breaking construction under threat model will solve an underlying hard problem

Brief History of Crypto

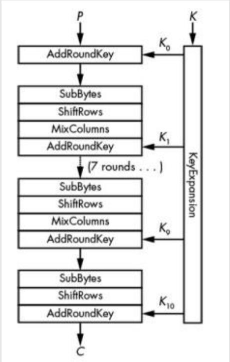
Che cos’è la Crittografia?

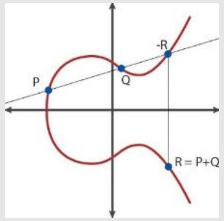
- Metodi per **memorizzare**, **elaborare** e **trasmettere** informazioni in maniera **sicura** in presenza di agenti ostili
- **Crittografia:** *Kryptós*: nascosto + *Graphía:* scrittura



$n = p \times q$

$p, q?$

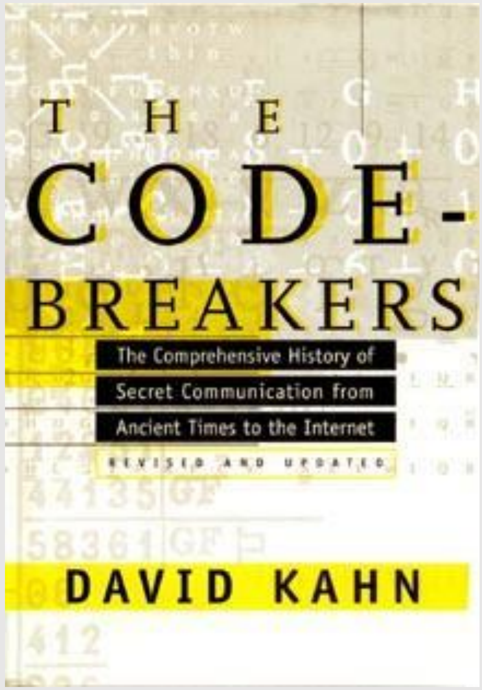




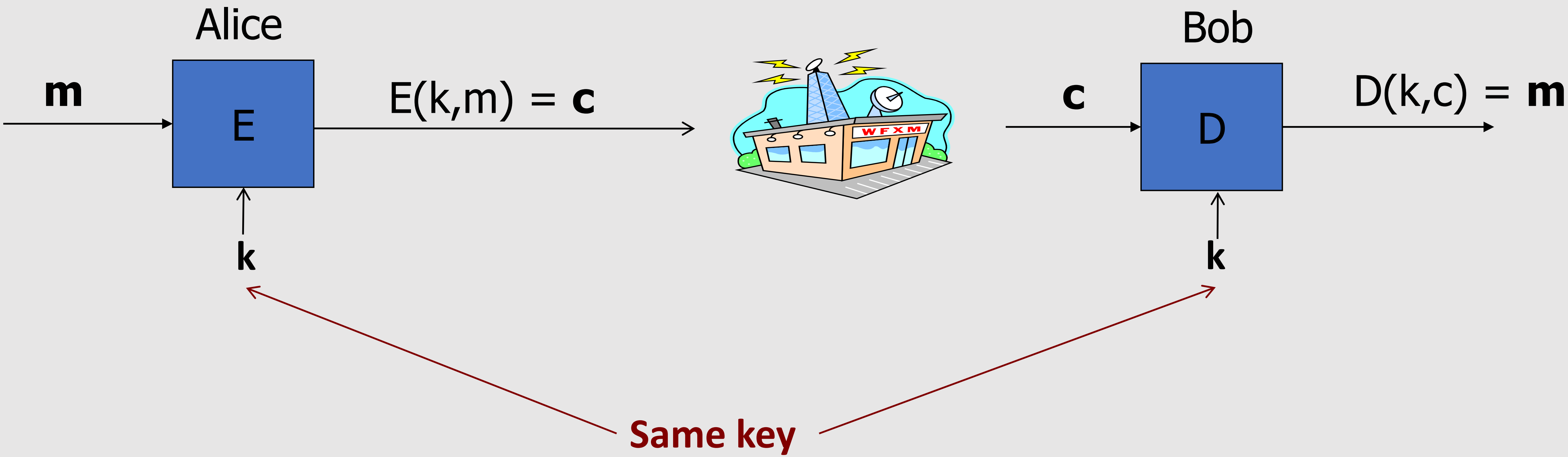
Scitala	Cifrario di Cesare	Enigma	DES	RSA	AES	Crittografia ellittica
400 aC	50 aC	1918	1975	1977	2001	2005

History

David Kahn, “The code breakers” (1996)



Symmetric Ciphers



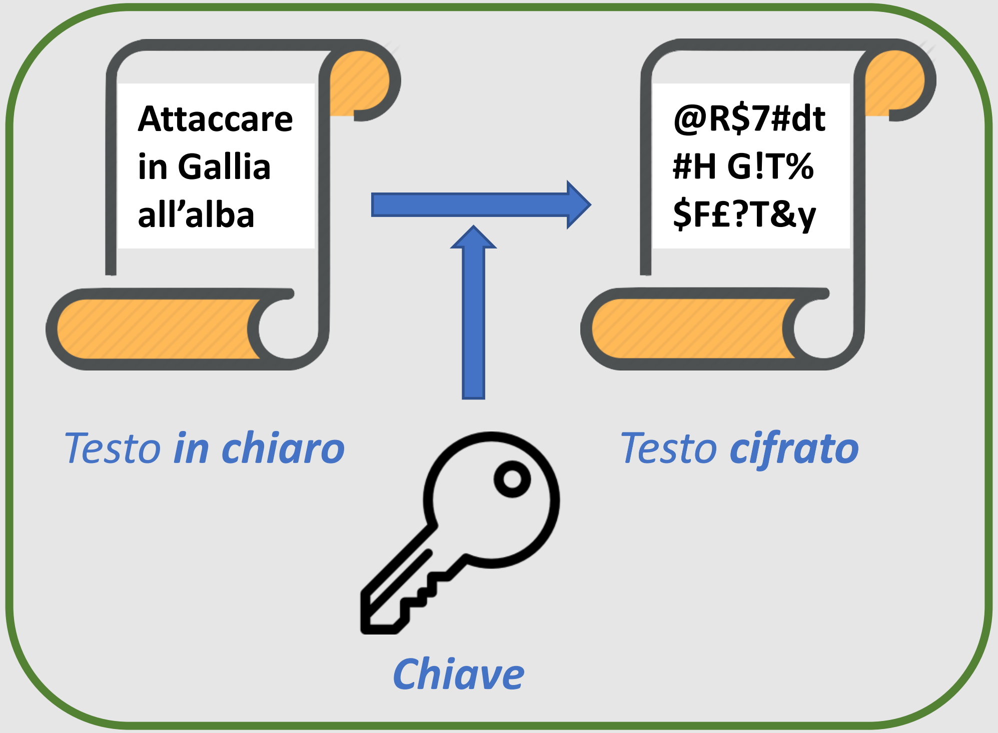
Cypher: (E, D)

Un classico scenario

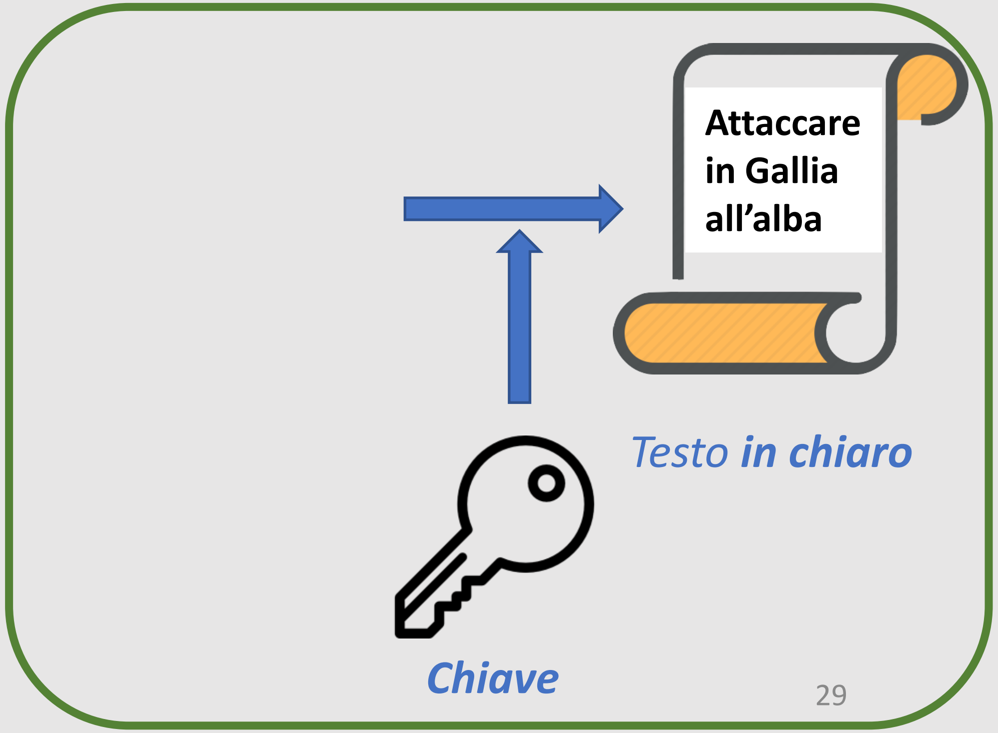
*Algoritmi di cifratura e decifratura: **pubblici***

Crittografia **simmetrica** e **asimmetrica**

Cifratura



Decifratura



Cifrario di Cesare

Chiave	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
	D	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	**A**	**B**	**C**

*Testo **in chiaro***

*Testo **cifrato***

(Cifrario a sostituzione)

Few Historic Examples (all badly broken)

1. Substitution cipher

$$c := E(k, \text{"bcza"}) = \text{"wnac"}$$

$$D(k, c) = \text{"bcza"}$$

$$k := \begin{array}{|ccc|} a & \longrightarrow & c \\ b & \longrightarrow & w \\ c & \longrightarrow & n \\ & \vdots & \\ z & \longrightarrow & a \end{array}$$

Caesar Cipher (no key)

Shift by 3

a → d
b → e
c → f
.
.
.
y → b
z → c

What is the size of key space in the substitution cipher assuming 26 letters?

$$|\mathcal{K}| = 26$$

$$|\mathcal{K}| = 26!$$

$$26! \approx 2^{88}$$

$$|\mathcal{K}| = 2^{26}$$

$$|\mathcal{K}| = 26^2$$

How to break a substitution cipher?

What is the most common letter in English text?

“X”

“L”

“E”

“H”

How to break a substitution cipher?

(1) Use frequency of English letters

e: 12,7%   **t**: 9,1%   **a**: 8,1%

(2) Use frequency of pairs of letters (digrams)

he, **an**, **in**, **th**

An Example

UKBYBIPOUZBCUFEEBORUKBYBHOBBRFESPVKBWFOFERVNBCVBZPRUBOFERVNBCVBPCYYFVU
FOFEIKNWFRFIKJNUPWRFIPOUNVNIPUBRNCUKBEFWWFDNCHXCYBOHOPYXPUBNCUBOYNRV
NIWNCPOJIOFHOPZRVFZIXUBORJRUBZRBCHNCBBONCHRJZSFWNVRJRUBZRPCYZPUKBZPUNV
PWPCYVFZIXUPUNFCPWRVNBCVBRPYYNUNFCPWWJUKBYBIPOUZBCUIPOUNVNIPUBRNCHOP
YXPUBNCUBOYNRVNIWNCPOJIOFHOPZRNCRVNBCUNENVVFZIXUNCHPCYVFZIXUPUNFCPWZP
UKBZPUNVR

B	36	→ E
N	34	
U	33	→ T
P	32	→ A
C	26	

NC	11	→ IN
PU	10	→ AT
UB	10	
UN	9	

digrams

UKB	6	→ THE
RVN	6	
FZI	4	

trigrams

2. Vigenère cipher (16'th century, Rome)

k = $\boxed{\text{C R Y P T O}}$ C R Y P T O C R Y P T

(+ mod 26)

m = W H A T A N I C E D A Y T O D A Y

c = Y Y Y I T B K T C S T M V F B P R

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

2. Vigenère cipher (16’th century, Rome)

k = C R Y P T O C R Y P T O C R Y P T

m = W H A T A N I C E D A Y T O D A Y

(+ mod 26)

c = Y Y Y I T B K T C S T M V F B P R

Polyalphabetic cypher

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

2. Vigenère cipher (16'th century, Rome)

k = C R Y P T O C R Y P T O C R Y P T

(+ mod 26)

m = W H A T A N I C E D A Y T O D A Y

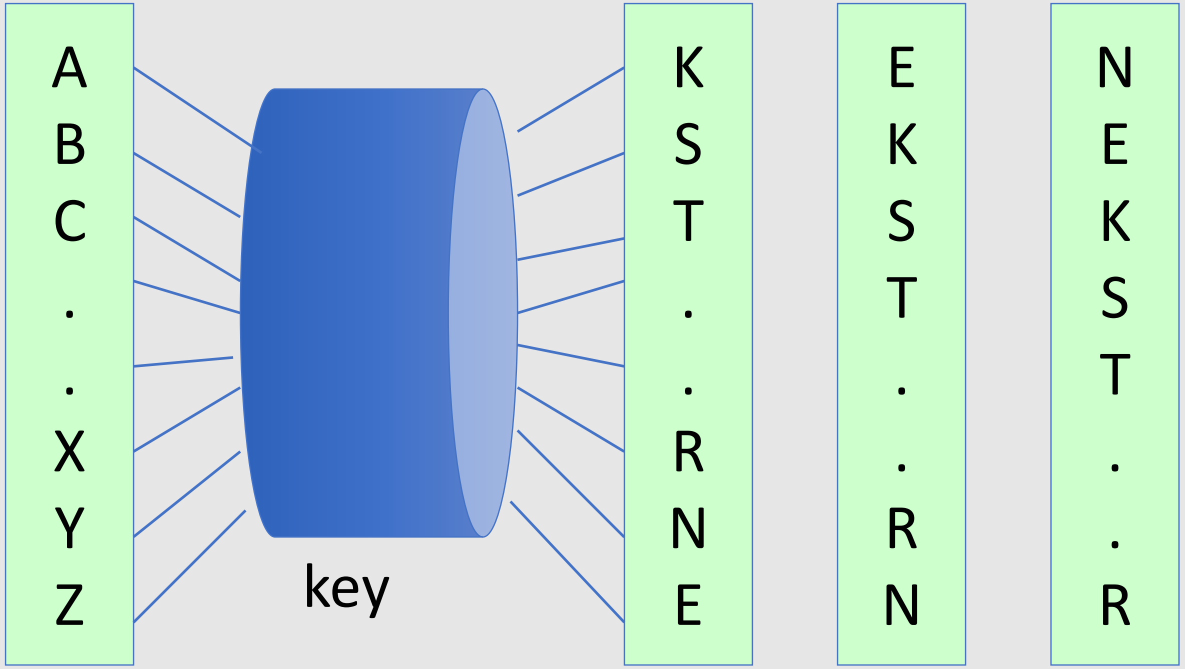
c = Y Y Y I T B K T C S T M V F B P R

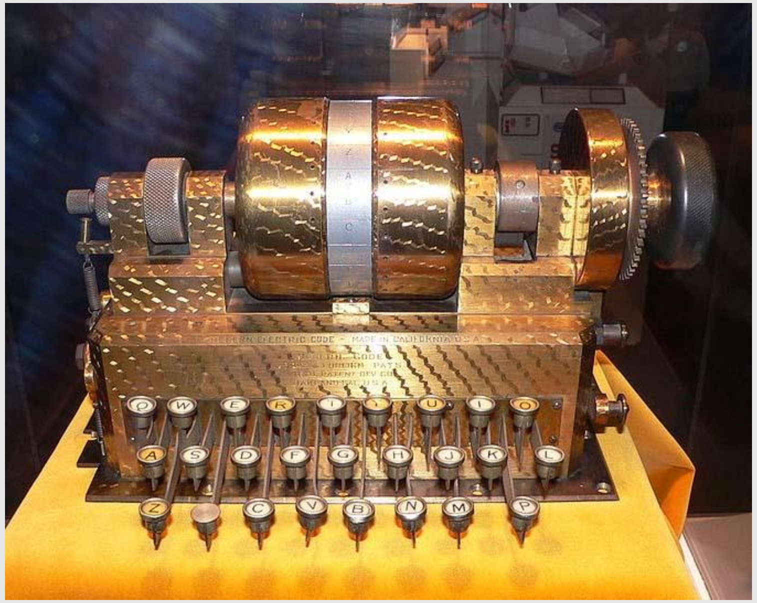
Suppose the most common letter is "G" ⟶ It is likely that "G" corresponds to "E"

⟶ **First letter of key = "G" – "E" = "C"** (c[i] = m[i] + k[i] ⇒ k[i] = c[i] – m[i])

3. Rotor Machines (1870-1943)

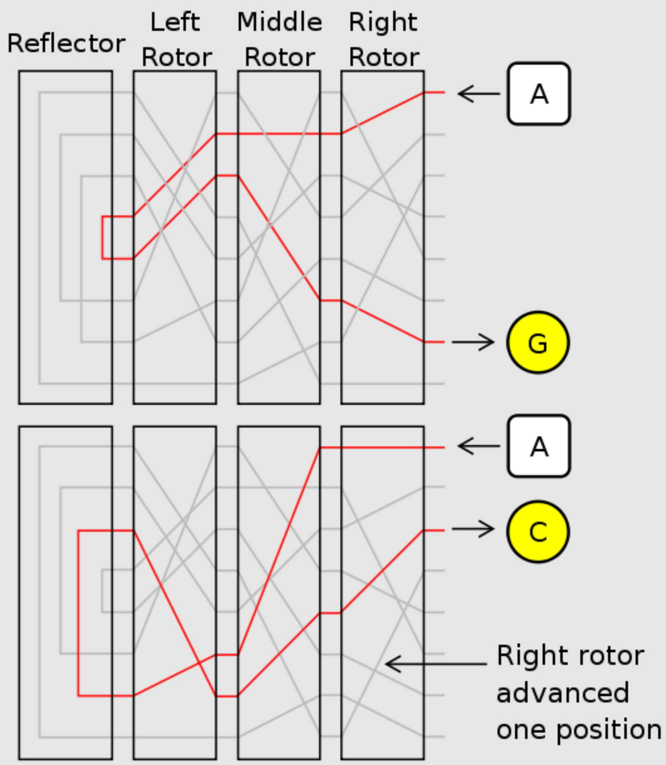
Early example: the Hebern machine (single rotor)





Rotor Machines (cont.)

Most famous: the Enigma (3-5 rotors)



4. Data Encryption Standard (1974)

~~DES: # keys = 2^{56} , block size = 64 bits~~

<u>Today</u>: AES (2001), Salsa20 (2008) (and many others)

Discrete Probability
(crash course)

Probability distribution

- **U**: **finite set**, called **Universe** or **Sample space**

  Examples:
  - Coin flip: **U = { heads, tail }** or **U = { 0, 1 }**
  - Rolling a dice: **U = { 1, 2, 3, 4, 5, 6 }**

- A **Probability distribution** P over U is a function $\mathrm{P} : \mathrm{U} \longrightarrow [0,1]$

  such that $\sum_{\mathrm{x} \in \mathrm{U}} \mathrm{P}(\mathrm{x}) = 1$

  Examples:
  - Coin flip: **P(heads) = P(tail) = 1/2**
  - Rolling a dice: **P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6**

Probability distribution

- **U**: **finite set**, called **Universe** or **Sample space**
- A **Probability distribution** P over U is a function **P : U ⟶ [0,1]**

  such that $\sum_{\mathrm{x} \in \mathrm{U}} \mathrm{P}(\mathrm{x}) = 1$

- Notation: U = $\{0,1\}^n$
- **Example:**

  Universe **U** = $\{0,1\}^2$ = {00, 01, 10, 11}

  Probability distribution **P** defined as follows:

  P(00)= 1/2 P(01)= 1/8 P(10)= 1/4 P(11)= 1/8

Probability distributions

Examples:

1. Uniform distribution: for all $x \in U$: $P(x) = 1/|U|$
2. Point distribution at x_0: $P(x_0) = 1, \quad \forall x \neq x_0$: $P(x) = 0$

… and many others

Events

Let us consider a universe **U** and a probability distribution **P** over U.

- An **event is a subset A of U,** that is, A ⊆ U
- The **probability of A** is $\mathbf{Pr[A] = \sum_{x \in A} P(x)}$

Note: Pr[U] = 1

Example

- Universe U = { 1, 2, 3, 4, 5, 6 }
- Probability distribution P s.t. P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6
- **A = {1, 3, 5}**
- **P[A] = 1/6 + 1/6 + 1/6 = 1/2**

Events

Let us consider a universe **U** and a probability distribution **P** over U.

- An **event is a subset A of U,** that is, $A \subseteq U$
- The **probability of A** is $\mathbf{Pr[A]} = \sum_{\mathbf{x \in A}} \mathbf{P(x)}$

Example

- Universe $U = \{0,1\}^8$
- Uniform distribution P over U, that is, $P(x) = 1/2^8$ for every $x \in U$
- $\mathbf{A = \{\text{ all x in U such that } lsb_2(x)=11 \} \subseteq U}$
- **Pr[A] = ¼**

Hints: $Pr[A] = 1/2^8 \times |A|$
each element in A is of the form _ _ _ _ _ _ 1 1

Union of Events

Given events $\mathbf{A_1}$ and $\mathbf{A_2}$,

$\mathbf{A_1 \cup A_2}$ is an event.

- $\Pr[\, A_1 \cup A_2 \,] \;=\; \Pr[A_1] + \Pr[A_2] - \Pr[A_1 \cap A_2\,]$
- $\Pr[\, A_1 \cup A_2 \,] \;\leq\; \Pr[A_1] + \Pr[A_2]$ (“Union bound”)
- $A_1 \cap A_2 = \emptyset \Rightarrow \Pr[\, A_1 \cup A_2 \,] \;=\; \Pr[A_1] + \Pr[A_2]$

Random Variables

Def: a **random variable** X is a function $\mathbf{X : U \longrightarrow V}$

Example (Rolling a dice):

U = { 1, 2, 3, 4, 5, 6 }

Uniform distribution P over U: P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6

Random variable **X : U ⟶ { "even", "odd" }**

X(2) = X(4) = X(6) = "even"

X(1) = X(3) = X(5) = "odd"

Pr[X="even"] = 1/2 , Pr[X="odd"] = 1/2

More generally: **X *induces* a distribution on V**

The **uniform** random variable

Let S be some set,   e.g. $S = \{0,1\}^n$

We write **$r \longleftarrow S$** to denote a **uniform random variable** over S

for all $a \in S$: $\Pr[\, r=a \,] \;=\; 1/|S|$

Defining a random variable in terms of another

- Let r be a uniform random variable on $\{0,1\}^2$
- Define the random variable $X = r_1 + r_2$

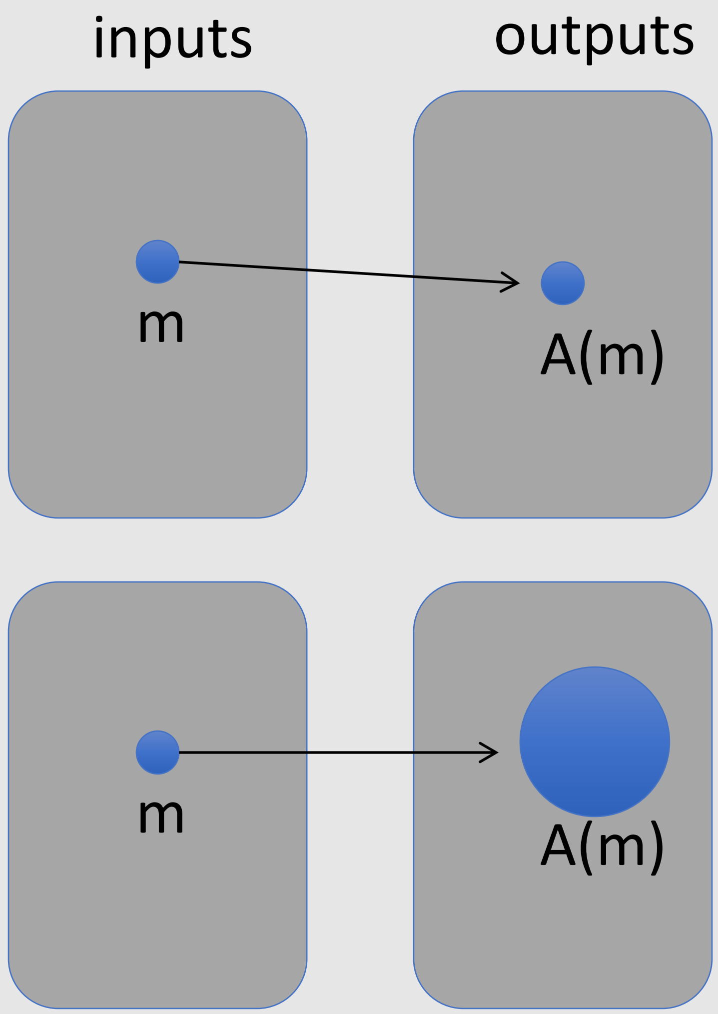
- Then $\Pr[X=2] = ¼$

- Hint: $\Pr[X=2] = \Pr[\, r=11 \,]$

Randomized algorithms

- **Deterministic** algorithm: $y \longleftarrow A(m)$

- **Randomized** algorithm
  output is a random variable $y \longleftarrow A(\ m\ )$



Recap

- U: Universe or Sample space (e.g., $U = \{0,1\}^n$)
- A Probability distribution P over U is a function $P : U \longrightarrow [0,1]$ such that $\sum_{x \in U} P(x) = 1$
- An event is a subset A of U, that is, $A \subseteq U$
- The probability of event A is $\Pr[A] = \sum_{x \in A} P(x)$
- A random variable is a function $X : U \longrightarrow V$
  X takes values in V and defines a distribution on V

Independence

Definition. **Independent events**

Events A and B are **independent** if

$\Pr[\ A \cap B\ ] = \Pr[A] \cdot \Pr[B]$

Definition. **Independent random variables**

Random variables X and Y taking values in V are **independent** if

$\forall a,b \in V: \quad \Pr[\ X=a \text{ and } Y=b] = \Pr[X=a] \cdot \Pr[Y=b]$

XOR

XOR of two strings in $\{0,1\}^n$ is their bit-wise addition mod 2

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

$$
\begin{array}{r}
0\ 1\ 1\ 0\ 1\ 1\ 1 \\
\oplus\quad 1\ 0\ 1\ 1\ 0\ 1\ 0 \\
\hline
1\ 1\ 0\ 1\ 1\ 0\ 1
\end{array}
$$

An important property of XOR

Theorem:

1. **X:** a random variable over $\{0,1\}^n$ with a **uniform distribution**
2. **Y:** a random variable over $\{0,1\}^n$ with an **arbitrary distribution**
3. **X** and **Y** are **independent**

- Then **$Z := Y \oplus X$** is a **UNIFORM** random variable over $\{0,1\}^n$

Proof: (for n=1)

Pr[Z=0] =

Pr[(X,Y)=(0,0) or (X,Y)=(1,1)] =

Pr[(X,Y)=(0,0)] + Pr[(X,Y)=(1,1)] =

$p_0/2 + p_1/2 = ½$

Therefore Pr[Z=1] = ½

Y	Pr
0	p_0
1	p_1

X	Pr
0	1/2
1	1/2

X	Y	Pr
0	0	$p_0/2$
0	1	$p_1/2$
1	0	$p_0/2$
1	1	$p_1/2$

The birthday paradox

Let $r_1, \ldots, r_n \in U$ be **independent identically distributed** random variables

Theorem: when $\mathbf{n} = 1.2 \times |\mathbf{U}|^{1/2}$ then $\Pr[\, \exists i \neq j: \; r_i = r_j \,] \geq ½$

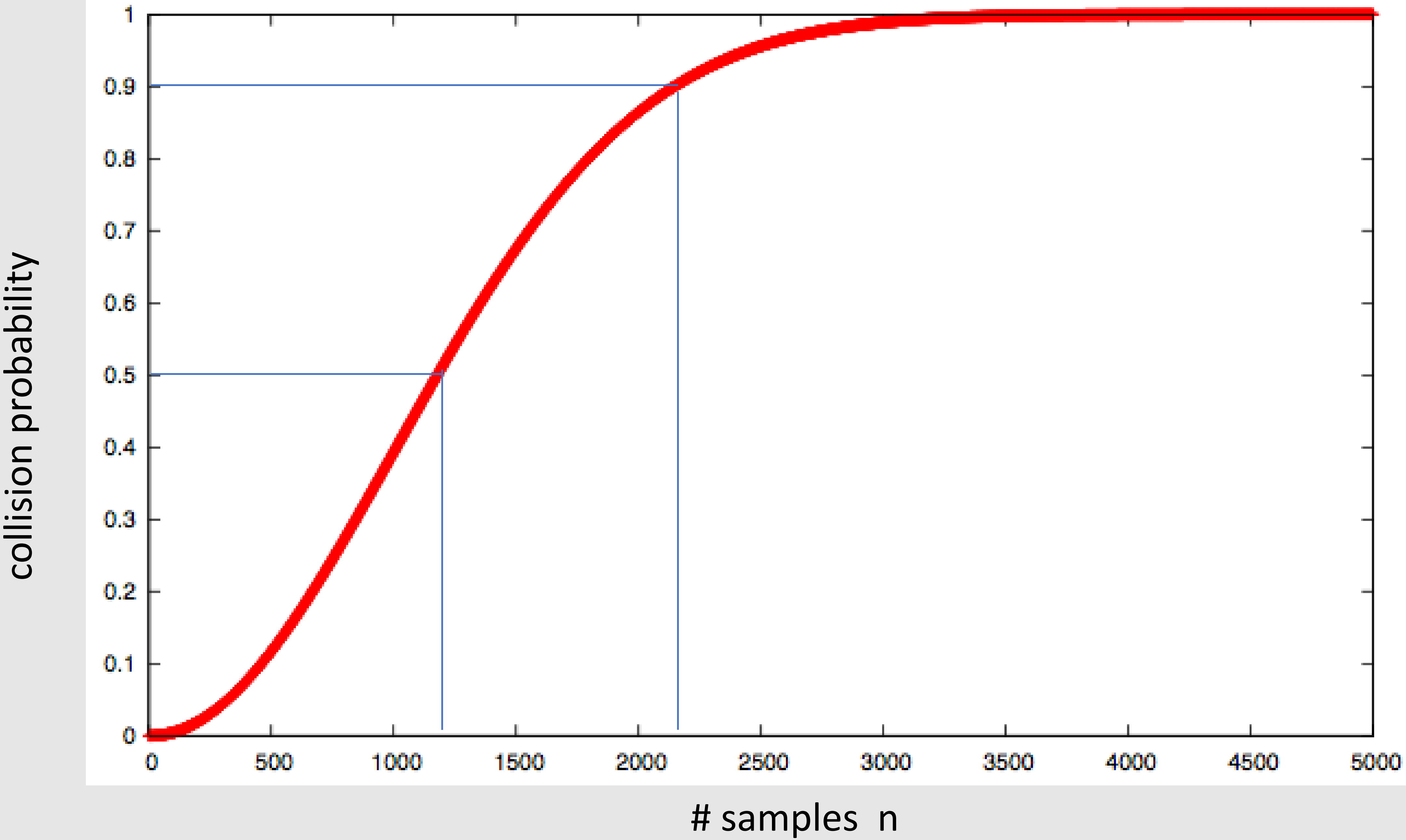
Example:

- U = {1, 2, 3, ..., 366}
- When $\boldsymbol{n} = 1.2 \times \sqrt{366} \approx \mathbf{23}$, two people have the same birthday with probability ≥ ½

Example:

- Let $U = \{0,1\}^{128}$
- After sampling about 2^{64} random messages from U, some two sampled messages will likely be the same

$|U|=10^6$



collision probability
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
samples n

Stream Ciphers

Outline

- One-Time Pad
- Perfect Secrecy
- Pseudorandom Generators (PRGs) and Stream Ciphers
- Attacks
- Security of PRGs
- Semantic Security

Symmetric Ciphers

Definition.

A (symmetric) **cipher** defined over (K, M, C)

is a pair of "efficient" algorithms **(E,D)** where

- **E:** $K \times M \to C$
- **D:** $K \times C \to M$

such that $\forall m \in M, \forall k \in K$: $\mathbf{D(k, E(k,m)) = m}$

- E is often **randomized**.
- D is **always deterministic**.

The One-Time Pad (Vernam 1917)

First example of a "secure" cipher

- $K = M = C = \{0,1\}^n$
- $E(k, m) = k \oplus m$
- $D(k, c) = k \oplus c$
- k used <u>**only once**</u>
- k is a **random** key (i.e., <u>**uniform**</u> distribution over K)

$$
\begin{array}{lccccccc}
m: & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
k: & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
\hline
c: & 1 & 1 & 0 & 1 & 1 & 0 & 1
\end{array}
\quad \oplus
$$

The One-Time Pad (Vernam 1917)

The one-time pad is a **cipher**:

- $D(k, E(k,m)) =$
- $D(k, k \oplus m) =$
- $k \oplus (k \oplus m) =$
- $(k \oplus k) \oplus m =$
- $0 \oplus m =$
- m

One-time pad definition:

- $E(k, m) = k \oplus m$
- $D(k, c) = k \oplus c$

The One-Time Pad (Vernam 1917)

- **Pro:**
  - Very **fast** encryption and decryption
- **Con:**
  - **Long keys** (as long as the plaintext),
    If Alice wants to send a message to Bob,
    she first has to transmit a key of the same length to Bob **in a secure way**.
    If Alice has a secure mechanism to transmit the key, she might use that same mechanism to transmit the message itself!

Is the OTP secure? **What is a secure cipher?**

What is a secure cipher?

Attacker's abilities: **CT only attack** (for now)

Possible security requirements:

attempt #1: **attacker cannot recover secret key**

$E(k, m) = m$ would be secure

attempt #2: **attacker cannot recover all of plaintext**

$E(k, m_0 \| m_1) = m_0 \| k \oplus m_1$ would be secure

Shannon's idea:

CT should reveal no "info" about PT

Information Theoretic Security (Shannon 1949)

Definition.

A cipher (E, D) over (K, M, C) has **perfect secrecy** if

$\forall \mathbf{m_0}, \mathbf{m_1} \in M$ with $\mathbf{len(m_0) = len(m_1)}$ and $\forall \mathbf{c} \in C$

$$\Pr[E(k, m_0)=c] = \Pr[E(k, m_1)=c]$$

where **k is uniform in K** $(k \longleftarrow K)$

Information Theoretic Security

- Given CT, can't tell if PT is m_0 or m_1 (for all m_0, m_1)
- Most powerful adversary learns nothing about PT from CT
- No CT only attack! (but other attacks are possible…)

Is OTP ''secure''?

OTP has perfect secrecy.

Proof:

$$\forall m, c \qquad \Pr_k[E(k,m)=c] \;=\; \frac{\#keys\ k \in K\ \ s.t.\ E(k,m)=c}{|K|}$$

$$\text{So if } \forall m, c \;\; \#\{k \in K : E(k,m)=c\} = const.$$

$$\Rightarrow \text{Cipher has perfect secrecy}$$

Let **m** ∈ M and **c** ∈ C.

How many OTP keys map **m** to **c** ?

- None
- 1
- 2
- It depends on **m**

$$
\begin{array}{lccccccc}
m: & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
k: & ? & ? & ? & ? & ? & ? & ? \\
\hline
c: & 1 & 1 & 0 & 1 & 1 & 0 & 1
\end{array}
\quad \oplus
$$

Is OTP "secure"?

OTP has perfect secrecy.

Proof:

$$\forall m, c \quad \Pr_k[E(k,m) = c] = \frac{1}{|K|}$$

$$\text{So if } \forall m, c \;\; \#\{k \in K : E(k,m) = c\} = const.$$

$$\boxed{\Rightarrow \text{Cipher has perfect secrecy}}$$

The bad news …

- OTP drawback: **key-length=msg-length**

- Are there ciphers with perfect secrecy that use shorter keys?

  Theorem: perfect secrecy ⇒ $|K| \geq |M|$

  i.e. perfect secrecy ⇒ key-length ≥ msg-length

- Hard to use in practice!!!!

Pseudorandom Generators and Stream Ciphers

Review

Cipher over (K,M,C): a pair of "efficient" algorithms (E, D) s.t.

$\forall\, m \in M,\ \forall\, k \in K: \quad D(k, E(k, m)) = m$

Weak ciphers: substitution cipher, Vigener, …

A good cipher: **OTP** $\quad M = C = K = \{0,1\}^n$

$$\mathbf{E(k, m) = k \oplus m \quad , \quad D(k, c) = k \oplus c}$$

OTP has perfect secrecy (i.e., no CT only attacks)

Bad news: perfect-secrecy $\Rightarrow$ key-len $\geq$ msg-len

Stream Ciphers: making OTP practical

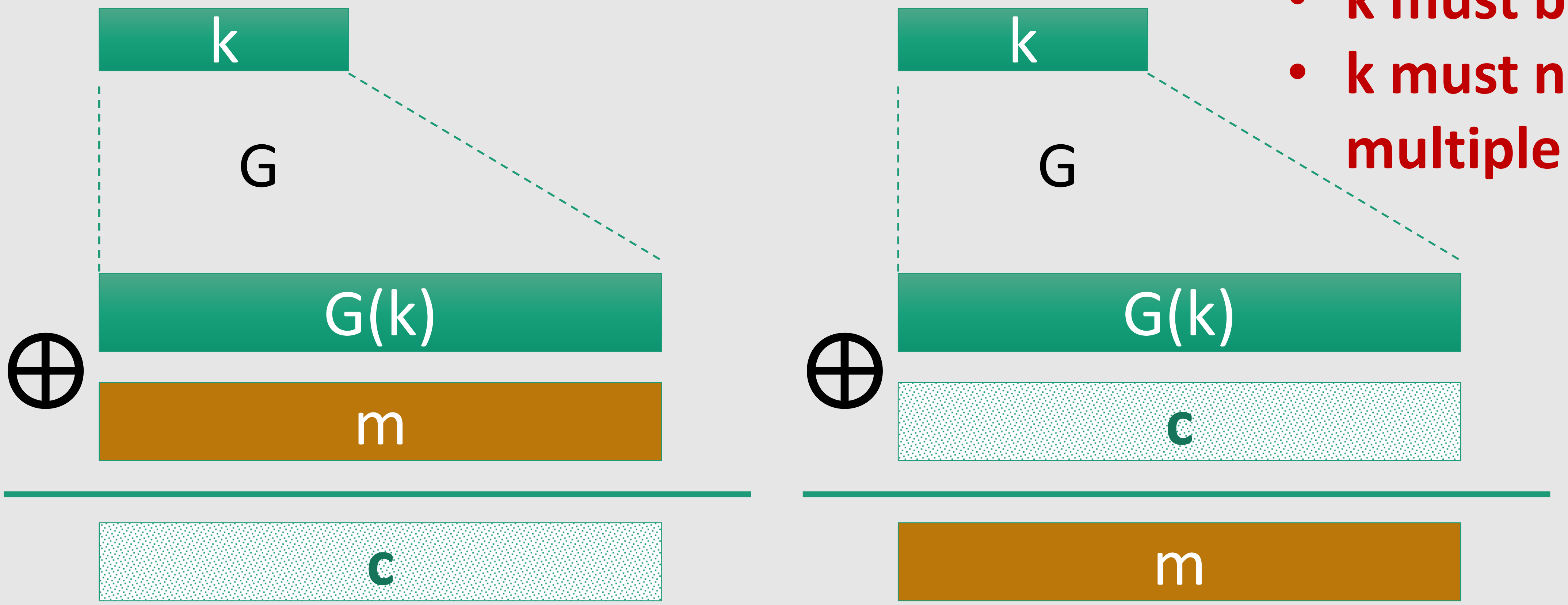
Idea: replace "**random**" key by "**pseudorandom**" key

Pseudorandom Generator (PRG):

PRG is a function $G: \underbrace{\{0,1\}^s}_{\textbf{seed space}} \to \{0,1\}^n \quad n \gg s$

(efficiently computable by a <u>deterministic</u> algorithm)

Stream Ciphers: making OTP practical



- **k must be random**
- **k must not be used multiple times**

$E(k, m) = G(k) \oplus m$

$D(k, c) \ = \ G(k) \oplus c$

Can a stream cipher have perfect secrecy?

- Yes, if the PRG is really “secure”
- No, there are no ciphers with perfect secrecy
- Yes, every cipher has perfect secrecy
- No, since the key is shorter than the message

Can a stream cipher have perfect secrecy?

- Yes, if the PRG is really “secure”
- No, there are no ciphers with perfect secrecy
- Yes, every cipher has perfect secrecy
- No, since the key is shorter than the message

Stream Ciphers: making OTP practical

Stream ciphers cannot have perfect secrecy !!

- Need a different definition of security
- Security will **depend on specific PRG**

Weak PRGs (do not use for crypto)

Linear congruential generator with parameters a, b, p:
(a, b are integers, p is a prime)

r[0] := seed
r[i] ← a r[i-1] + b mod p
output few bits of r[i]
i++

has some good statistical properties
But it's easy to predict

glibc random():

$r[i] \leftarrow (\ r[i-3] + r[i-31]\ )\ \%\ 2^{32}$
output r[i] >> 1

Do not use random() for crypto
(e.g., Kerberos v4)

Attacks on OTP and Stream Ciphers

Review

- **One-time pad**:
  - $E(k,m) = \mathbf{k} \oplus m$
  - $D(k,c) = \mathbf{k} \oplus c$

- **k** is random (**uniform**)
- **k** used only once

- **Stream ciphers**
  making OTP practical using a **PRG** $G: K \longrightarrow \{0,1\}^n$
  - $E(k,m) = \mathbf{G(k)} \oplus m$
  - $D(k,c) = \mathbf{G(k)} \oplus c$

Attack 1: **two time** pad is insecure !!

Never use stream cipher **key more than once** !!

$$c_1 \leftarrow m_1 \oplus \text{PRG}(k)$$
$$c_2 \leftarrow m_2 \oplus \text{PRG}(k)$$

Eavesdropper does:

$$c_1 \oplus c_2 \quad \rightarrow$$

Enough redundancy in English and ASCII encoding that:

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

Real-world examples

- Project Venona (1941 – 1946)

Real-world examples

- Project Venona (1941 – 1946)

- MS-PPTP   (windows NT):

$k = (k_{C \longrightarrow S} \, , \, k_{S \longrightarrow C})$

k 

m_1

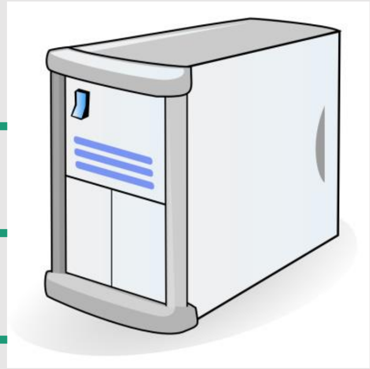
m_2

m_3

s_1

s_2

s_3

 **k**

$[\, m_1 \,||\, m_2 \,||\, m_3 \,] \oplus \boxed{\mathrm{PRG}(k)}$

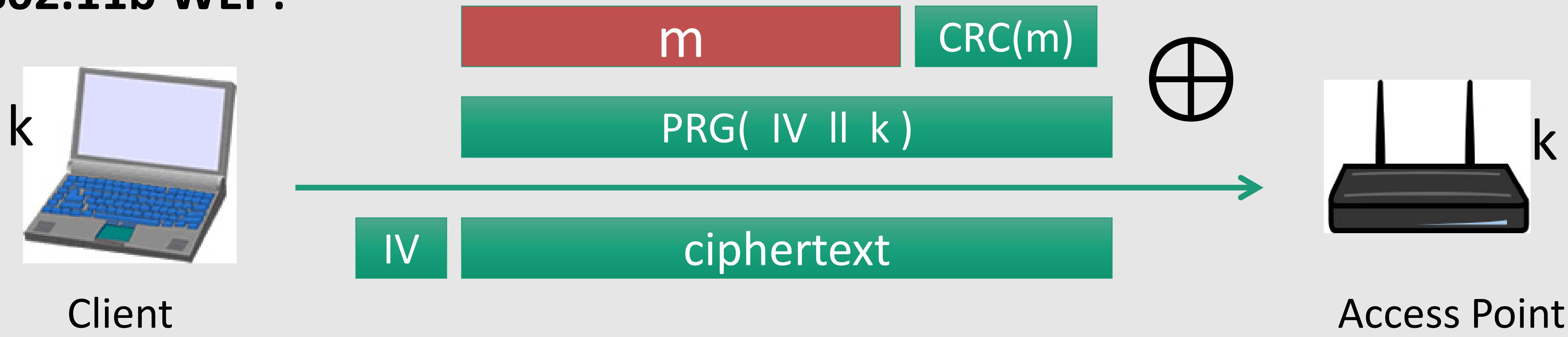
$[\, s_1 \,||\, s_2 \,||\, s_3 \,] \oplus \boxed{\mathrm{PRG}(k)}$

Need different keys for C⟶S and S⟶C

Real-world examples

k: LONG-TERM KEY

802.11b WEP:



Length of IV: 24 bits

- Repeated IV after $2^{24} \approx$ 16M frames
- On some 802.11 cards: IV resets to 0 after power cycle

Avoid related keys

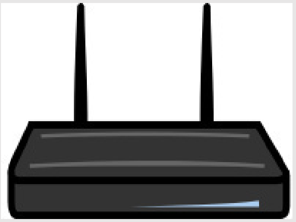
802.11b WEP:

k 

m | CRC(m)

$\oplus$

PRG(IV ll k)

 k

IV | ciphertext

24 bits 104 bits

key for frame #1: (1 ll k)

key for frame #2: (2 ll k)

⋮

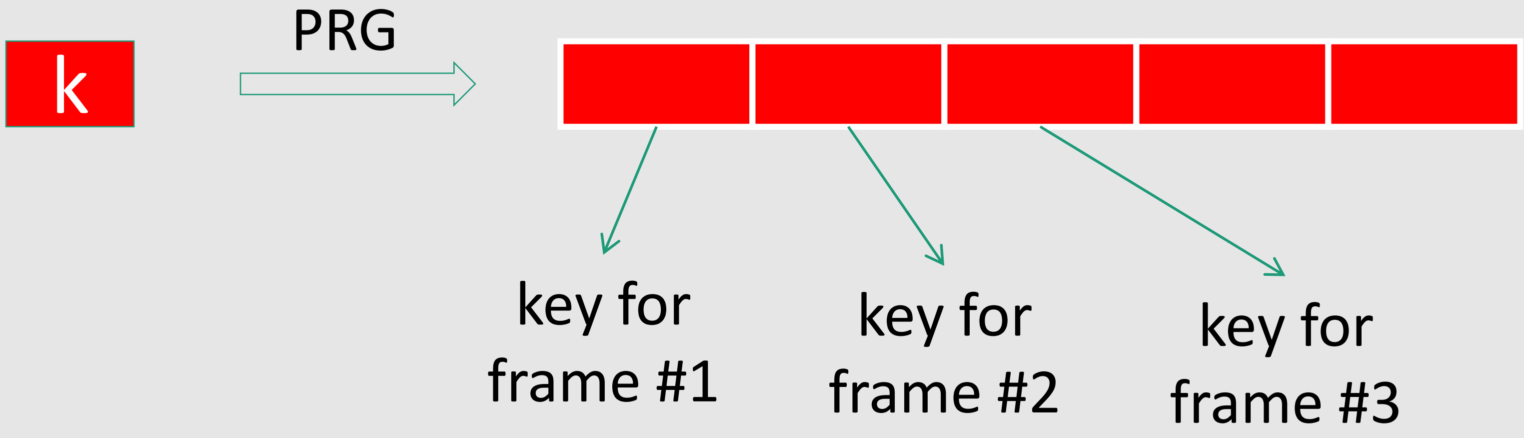
Very related keys!!
Not random keys!

The PRG used in WEP (called RC4) is not secure for such related keys

- Attack that can recover k after 10^6 frames (FMS 2001)
- Recent attack => 40.000 frames

A better construction

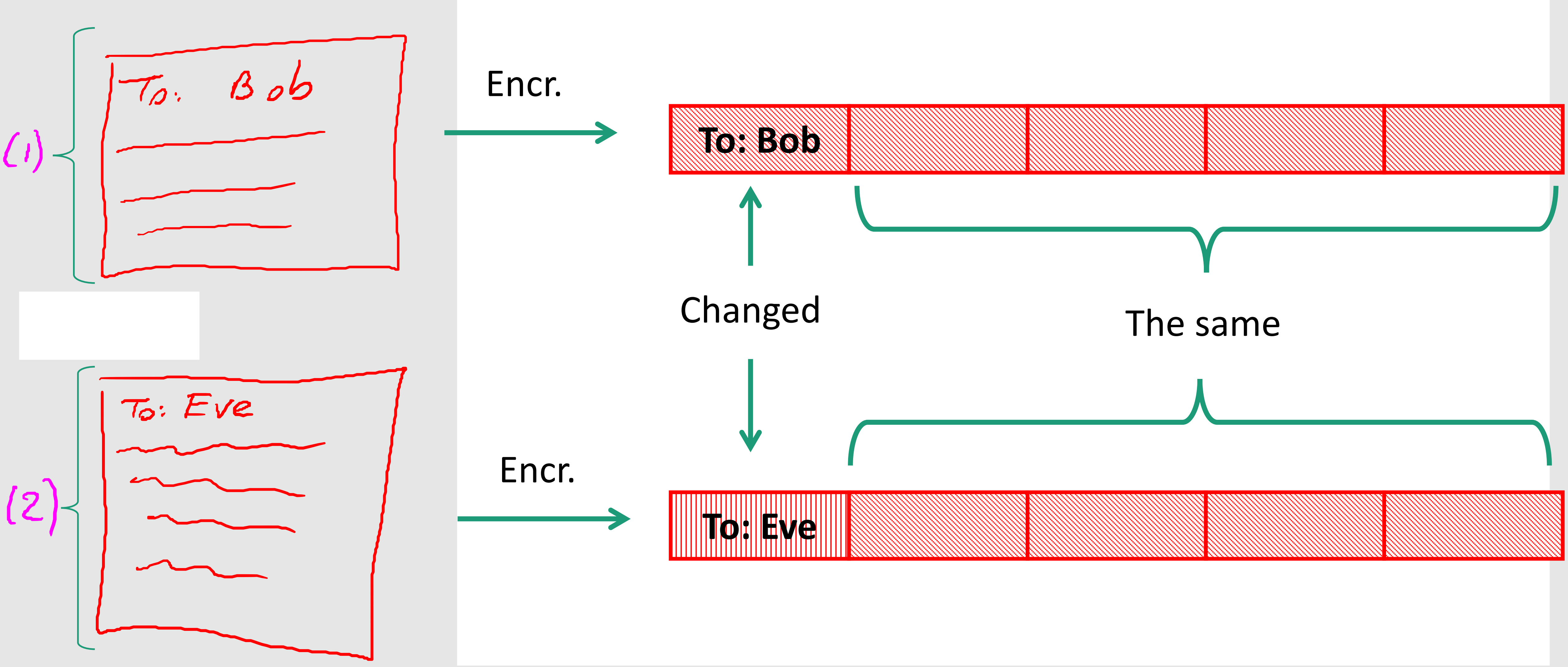
k



⇒ now each frame has a pseudorandom key

better solution: use stronger encryption method (as in WPA2)

Yet another example: disk encryption



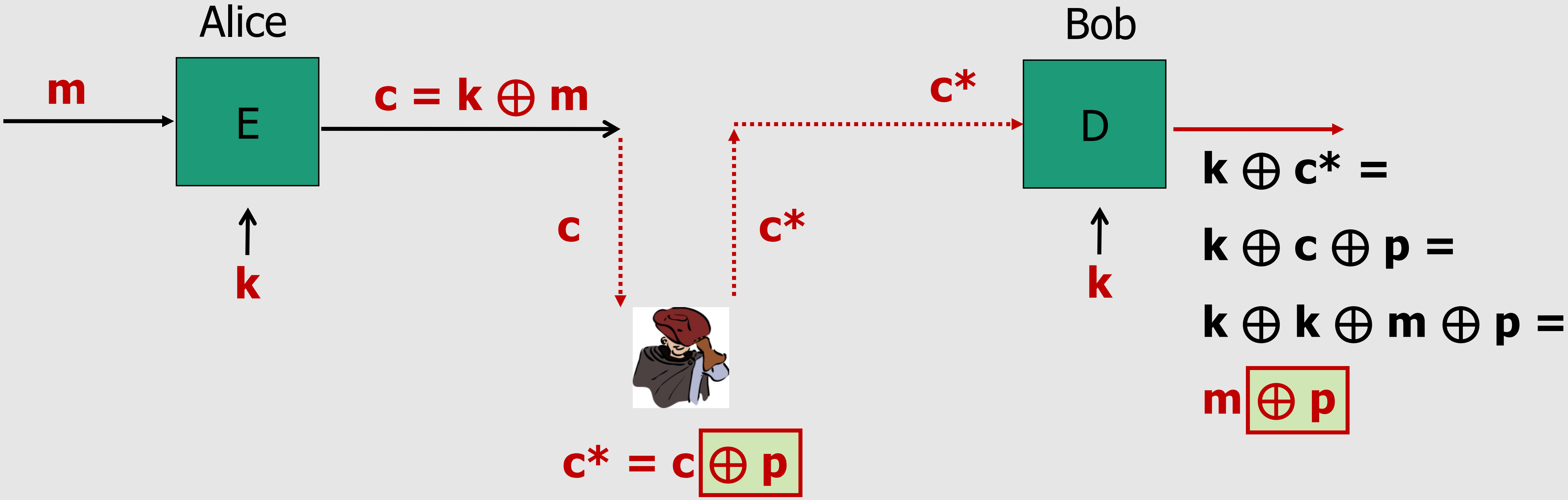
Two time pad: summary

Never use stream cipher key **more than once** !!

- Network traffic: negotiate new key for every session (e.g. TLS)
  - One key (or ‘’sub-key’’) for traffic **from Client to Server**
  - One key (or ‘’sub-key’’) for traffic **from Server to Client**

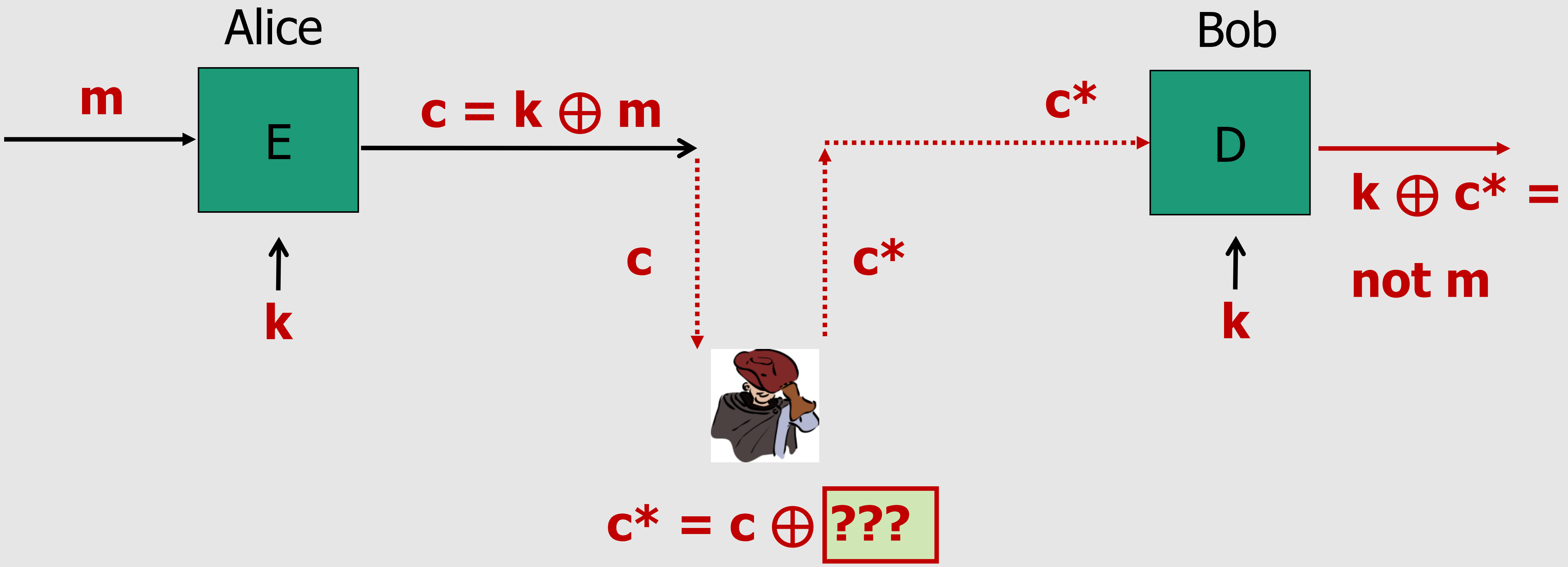
- Disk encryption: typically do not use a stream cipher

Attack 2: no integrity (OTP is **malleable**)



Modifications to ciphertext are **undetected** and
have **predictable** impact on plaintext

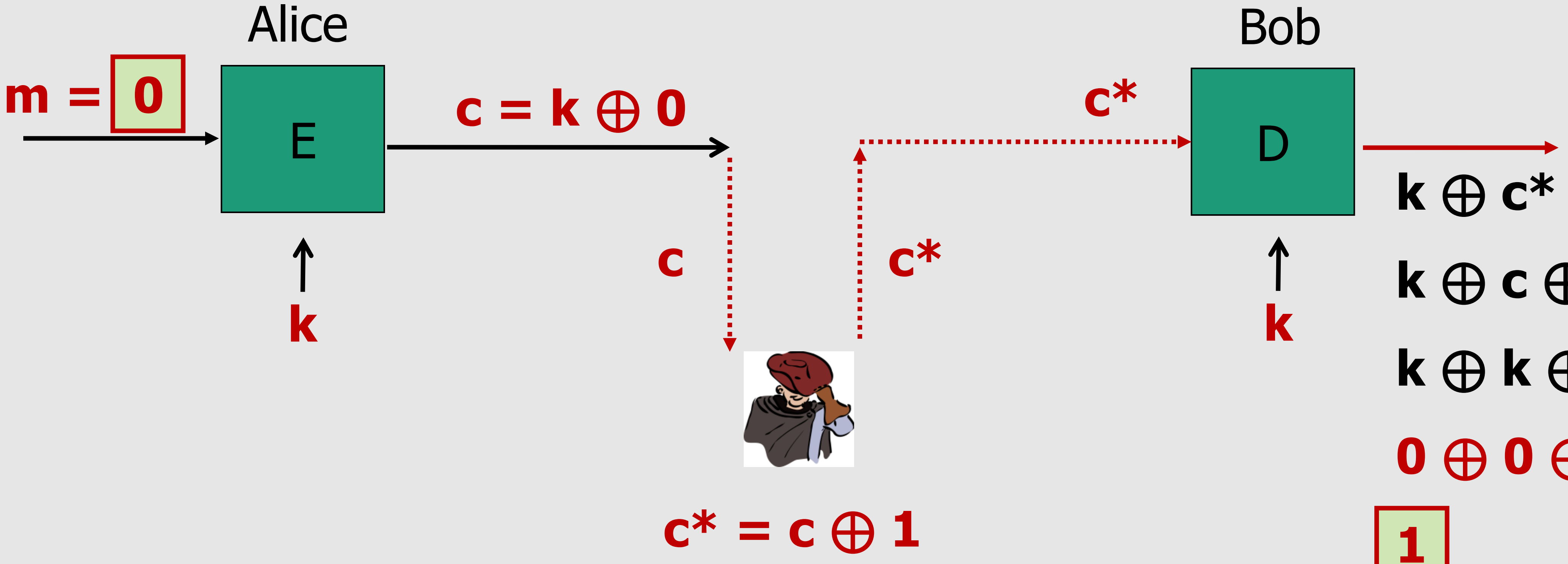
Attack 2: no integrity (OTP is **malleable**)



- Alice has to answer yes (**1**) or no (**0**) to Bob's invitation. She'll encrypt the answer with OTP.
- The attacker cannot recover Alice's answer from CT.
- **Still, can the attacker "flip" Alice's answer?**

  Yes !! Apply ⊕ 1 to the intercepted CT

Attack 2: no integrity (OTP is **malleable**)



$k \oplus c^* =$

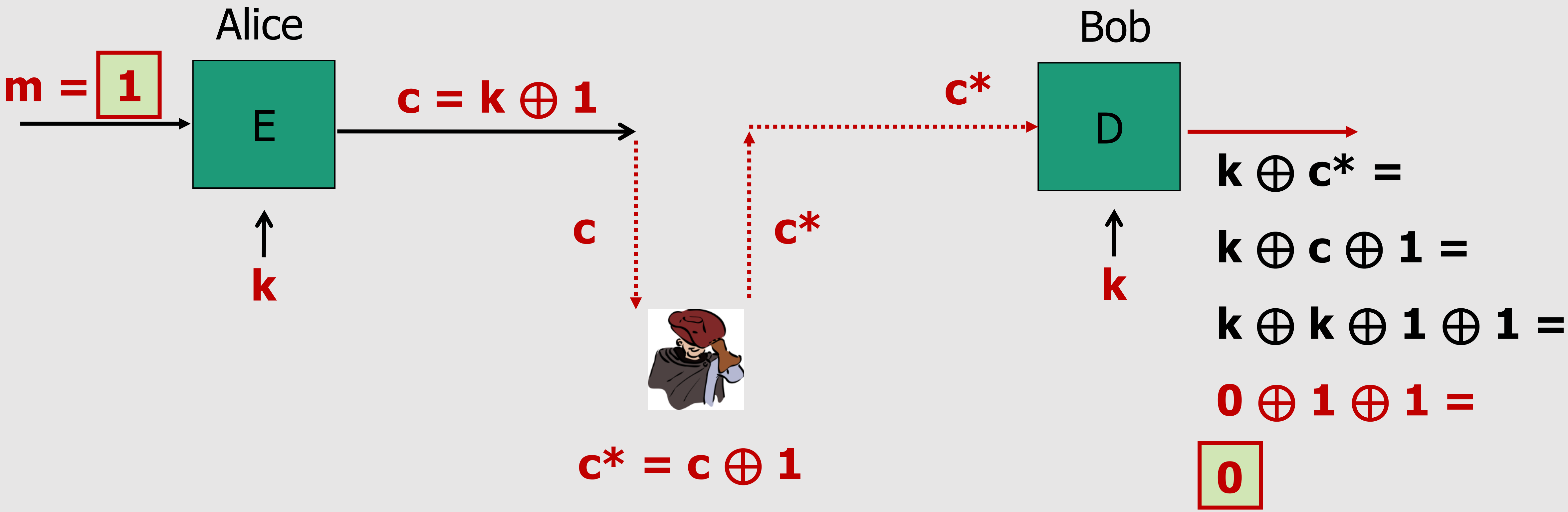
$k \oplus c \oplus 1 =$

$k \oplus k \oplus 0 \oplus 1 =$

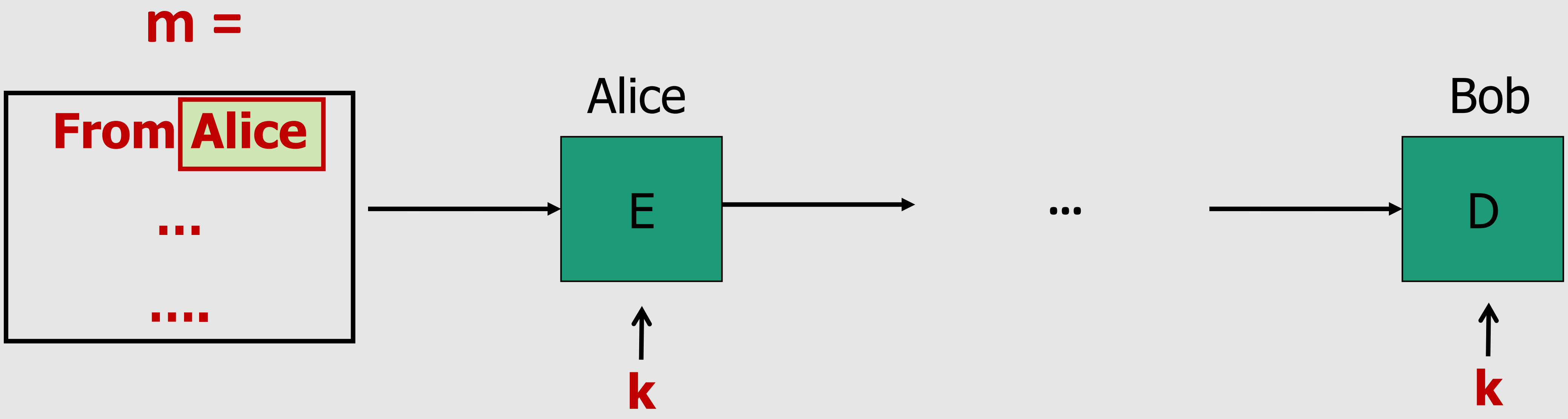
$0 \oplus 0 \oplus 1 =$

$\boxed{1}$

Attack 2: no integrity (OTP is **malleable**)



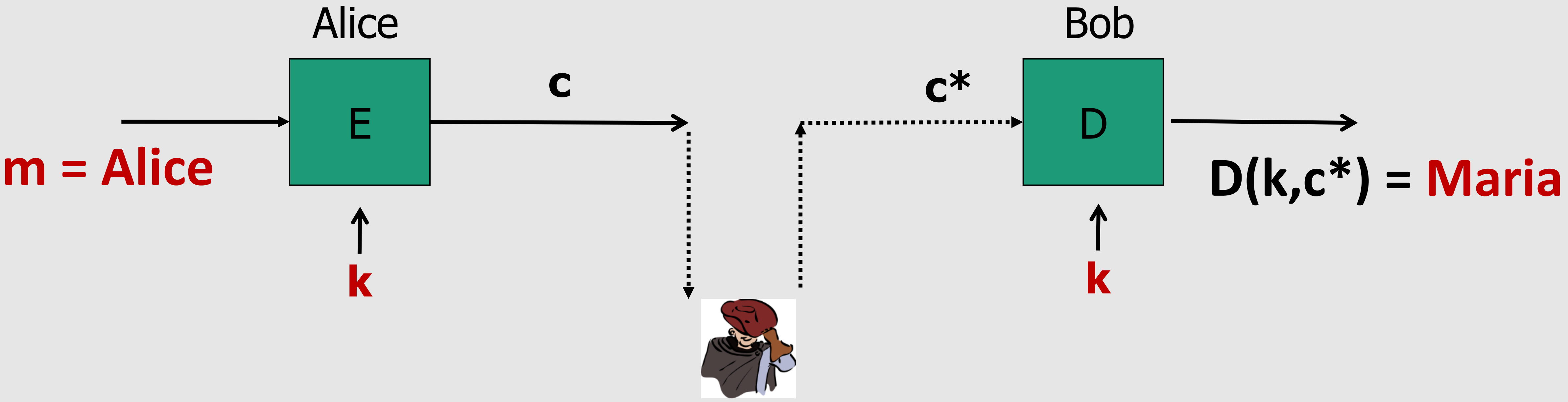
Attack 2: no integrity (OTP is **malleable**)



Attacker wants to change **Alice** into **Maria**.

Can he do that?

Attack 2: no integrity (OTP is **malleable**)

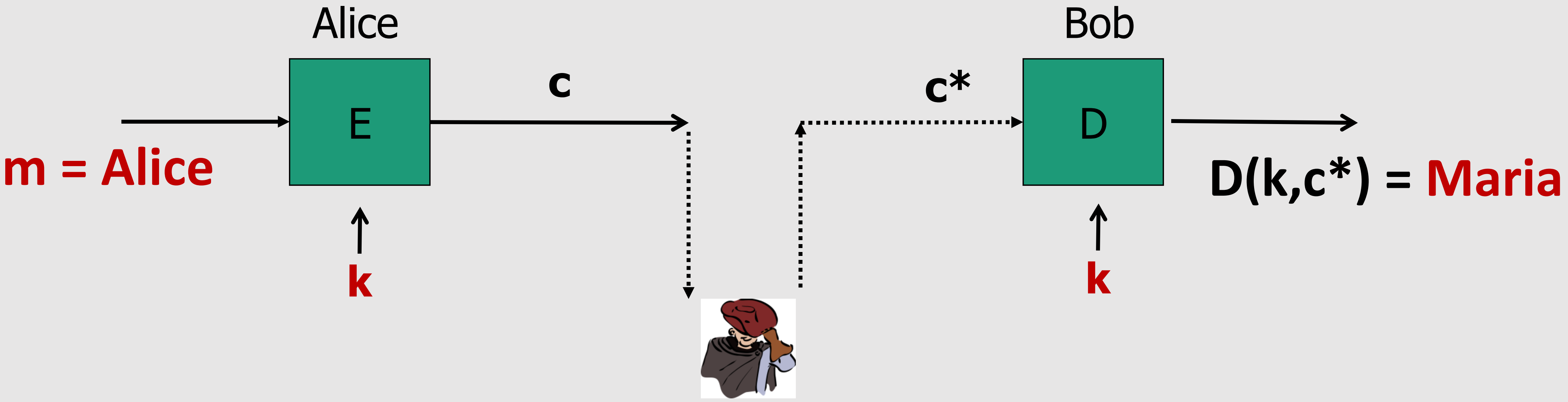


$c^* = c \oplus \boxed{???}$

Attacker wants to change **Alice** into **Maria**.

Can he do that?

Attack 2: no integrity (OTP is **malleable**)

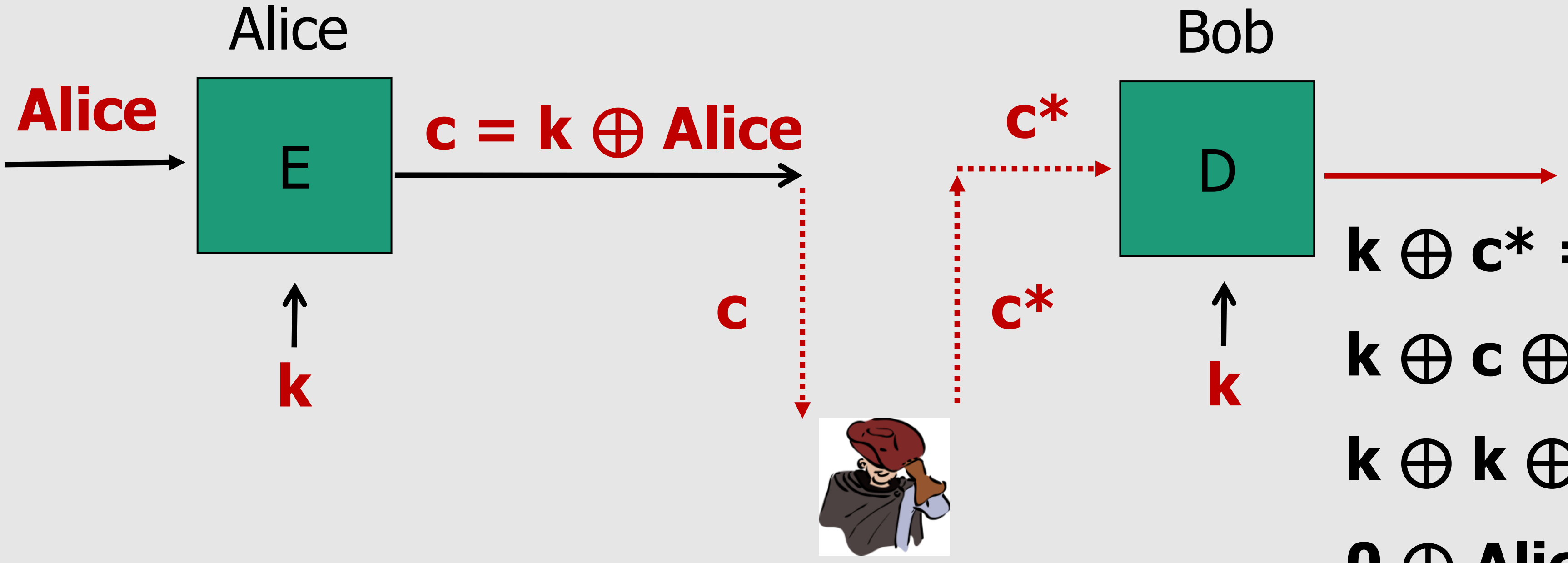


$$c^* = c \oplus \boxed{\text{Alice} \oplus \text{Maria}}$$

Attacker wants to change **Alice** into **Maria**.

Can he do that?

Attack 2: no integrity (OTP is **malleable**)



$$c^* = c \oplus \text{Alice} \oplus \text{Maria}$$

$$k \oplus c^* =$$

$$k \oplus c \oplus \text{Alice} \oplus \text{Maria} =$$

$$k \oplus k \oplus \text{Alice} \oplus \text{Alice} \oplus \text{Maria} =$$

$$0 \oplus \text{Alice} \oplus \text{Alice} \oplus \text{Maria} =$$

$$0 \oplus 0 \oplus \text{Maria} =$$

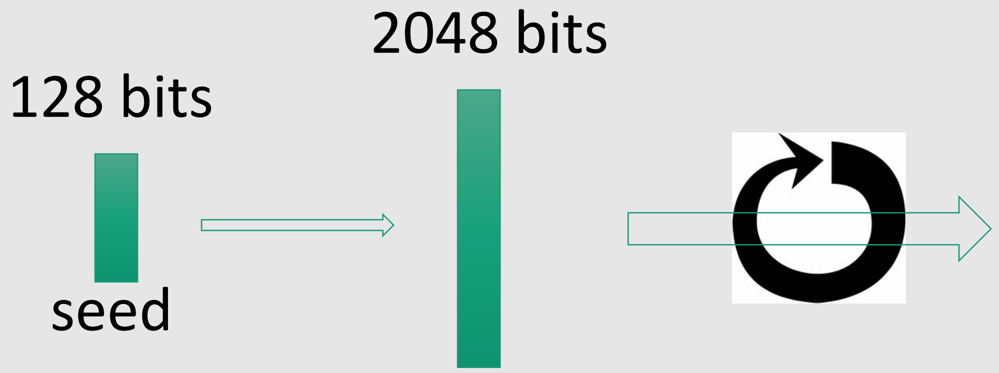
Maria

Consider the bank account number in a wire transfer…

Real-world Stream Ciphers

Old example (software): RC4 (1987)

Variable size seed
(e.g., 128 bits)



1 byte
per round

- Used in HTTPS and WEP

RC4 PRG

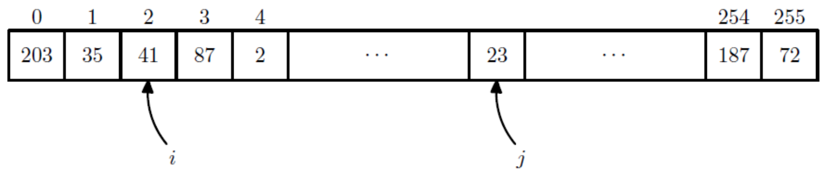


Figure 3.12: An example RC4 internal state

The RC4 stream cipher key s is a seed for the PRG and is used to initialize the array S to a pseudo-random permutation of the numbers 0 : : : 255. Initialization is performed using the following **setup algorithm**:

input: string of bytes s

for $i \leftarrow 0$ to 255 do: $\quad S[i] \leftarrow i$

$j \leftarrow 0$

for $i \leftarrow 0$ to 255 do

$\quad k \leftarrow s\big[i \bmod |s|\big] \quad //\quad$ *extract one byte from seed*

$\quad j \leftarrow \big(\, j + S[i] + k \,\big) \bmod 256$

$\quad \mathrm{swap}(S[i], S[j])$

During the loop the index i runs linearly through the array while the index j jumps around. At each iteration the entry at index i is swapped with the entry at index j.

RC4 PRG

Once the array S is initialized, the PRG generates pseudo-random output one byte at a time using the following **stream generator**:

$i \leftarrow 0, \quad j \leftarrow 0$

repeat

$\quad i \leftarrow (i+1) \bmod 256$

$\quad j \leftarrow (j + S[i]) \bmod 256$

$\quad \text{swap}(S[i], S[j])$

$\quad \text{output} \;\; S\big[\; (S[i] + S[j]) \bmod 256 \;\big]$

forever

The procedure runs for as long as necessary. Again, the index i runs linearly through the array while the index j jumps around. Swapping S[i] and S[j] continuously shuffles the array S.

Security of RC4

Weaknesses:

1. Bias in initial output: let us assume that the RC4 **setup algorithm is perfect** and generates a uniform permutation from the set of all 256! permutations.
Mantin and Shamir showed that, even assuming perfect initialization, the output of RC4 is biased: Pr[2^{nd} byte = 0] = 2/256 → RC4-drop[n]

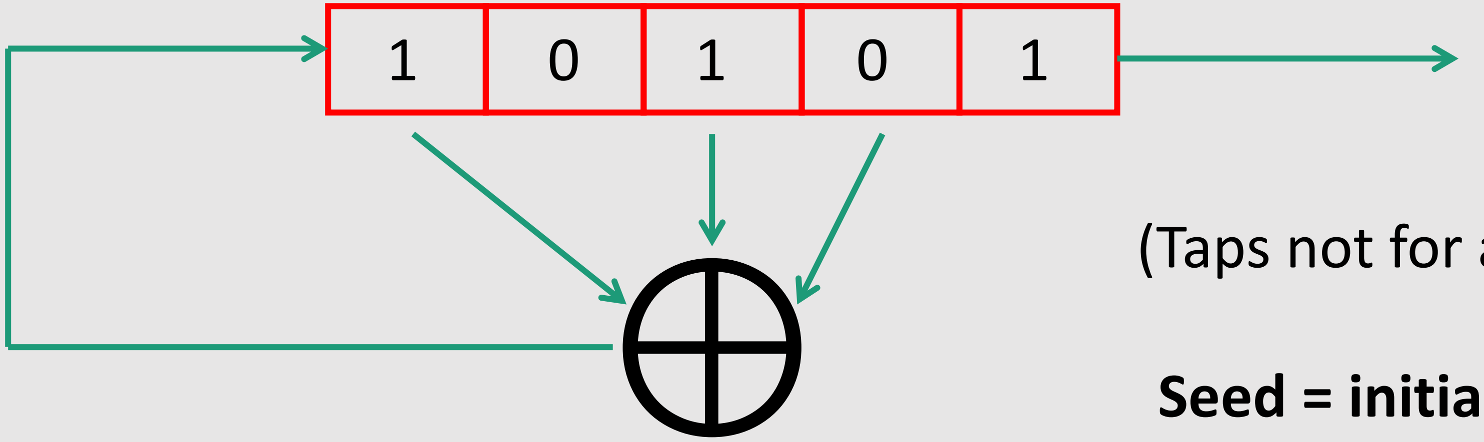
2. Fluhrer and McGrew: Prob. of (0,0) is $1/256^2 + 1/256^3$

3. Related key attacks: attack on WEP

Old example (hardware): CSS (badly broken)

Content Scrambling System

Linear feedback shift register (LFSR):



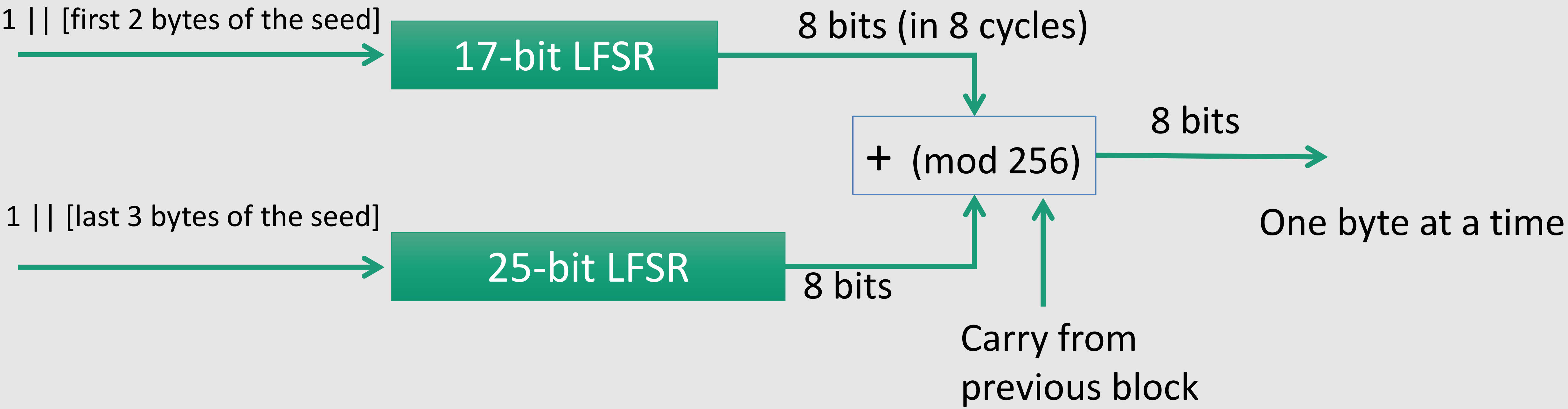
(Taps not for all cells)

Seed = initial state of the LFSR

DVD encryption (CSS):	2 LFSRs	all broken
GSM encryption (A5/1,2):	3 LFSRs	
Bluetooth (E0):	4 LFSRs	

Old example (hardware): CSS (badly broken)

CSS: seed = 5 bytes = 40 bits



Easy to break in time $\approx 2^{17}$

Modern stream ciphers: eStream

PRG: $\{0,1\}^s \times R \longrightarrow \{0,1\}^n$ n>>s

Seed ***Nonce***

Nonce: a non-repeating value for a given key, that is

a pair (k,r) is never used more than once

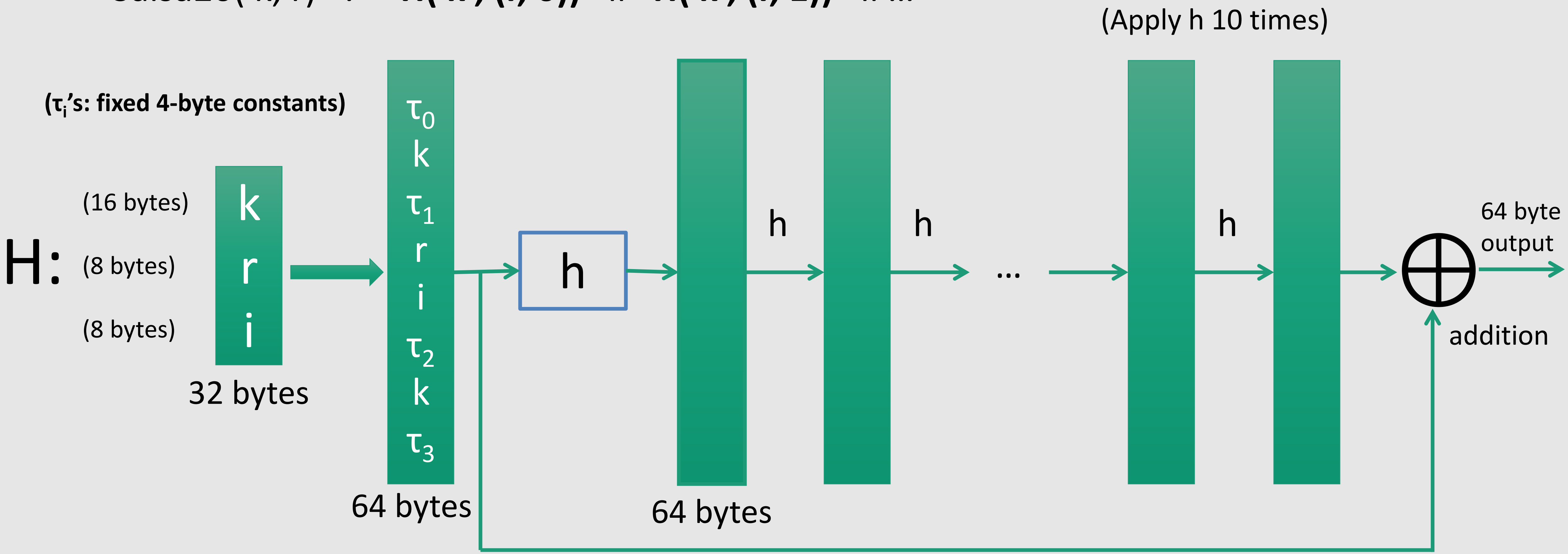
=> can re-use the key as long as the nonce changes

$E(k, m, r) = m \oplus PRG(k, r)$

eStream: Salsa 20 (SW+HW)

Salsa20: $\{0,1\}^{128 \text{ or } 256} \times \{0,1\}^{64} \longrightarrow \{0,1\}^n$ (max n = 2^{73} bits)

Salsa20(k, r) := **H(k , (r, 0))** ‖ **H(k , (r, 1))** ‖ …



h: invertible function. designed to be fast on x86 (SSE2)

Performance: Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

	PRG	Speed (MB/sec)
	RC4	126
eStream	Salsa20/12	643
	Sosemanuk	727

When is a PRG ‘‘secure’’?

When is a PRG ‘‘secure’’?

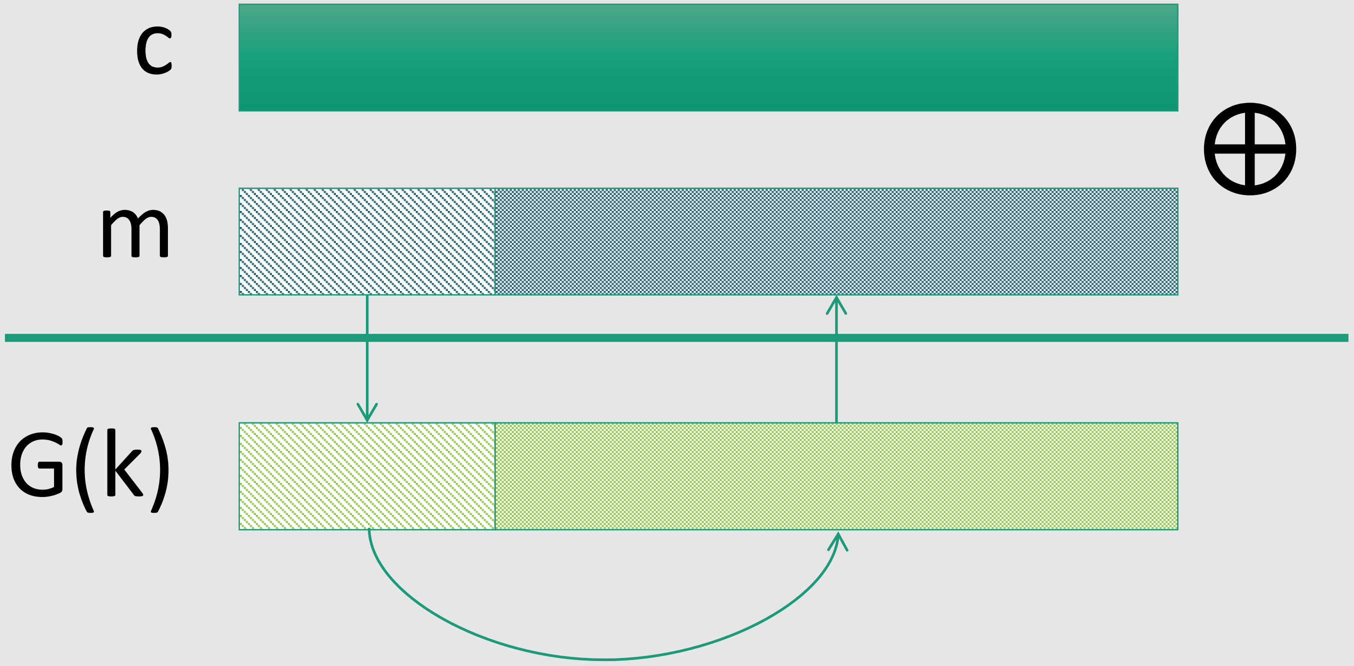
1. **Unpredictable** PRG
2. **Secure** PRG

We’ll see that they are **equivalent** notions

PRG must be unpredictable

Suppose PRG is **predictable**:

$$\exists i : \quad G(k)\big|_{1,\ldots,i} \xrightarrow{Alg} G(k)\big|_{i+1,\ldots,n}$$



Even

$$G(k)\big|_{1,\ldots,i} \xrightarrow{Alg} G(k)\big|_{i+1}$$

is a problem

PRG must be unpredictable

We say that G: K $\longrightarrow \{0,1\}^n$ is **predictable** if:

$$\exists \text{ "efficient" algorithm } A \text{ and } \exists 1 \leq i \leq n-1 \text{ s.t.}$$

$$\Pr_{k \leftarrow K}\left[A(G(k)|_{1,\ldots,i}) = G(k)|_{i+1}\right] > \frac{1}{2} + \epsilon$$

$$\text{for non-negligible } \epsilon \text{ (e.g., } \epsilon = \frac{1}{2^{30}})$$

PRG is **unpredictable** if it **is not predictable**

⇒ ∀i: no "efficient" adversary can predict bit (i+1) for "non-neg" ε

- Suppose $G: K \longrightarrow \{0,1\}^n$ is such that for all **k**: **XOR(G(k)) = 1**
- Is G predictable ??

1. Yes, given the first bit I can predict the second
2. No, G is unpredictable
3. Yes, given the first (n-1) bits I can predict the n-th bit
4. It depends

- Suppose G:K $\longrightarrow \{0,1\}^n$ is such that for all **k**: **XOR(G(k)) = 1**
- Is G predictable ??

1. Yes, given the first bit I can predict the second
2. No, G is unpredictable
3. Yes, given the first (n-1) bits I can predict the n-th bit
4. It depends

One more definition of ''secure'' PRG

Let **G:K $\longrightarrow$ {0,1}$^{\mathbf{n}}$** be a PRG

Goal:
define what it means that

$$[k \leftarrow K, \quad \text{output } G(k)]$$

is "indistinguishable" from

$$[r \leftarrow \{0,1\}^n, \ \text{ouput } r]$$

G: $\{0,1\}^{10} \longrightarrow \{0,1\}^{1000}$

$[k \longleftarrow \{0,1\}^{10}$, output G(k)]

$[r \longleftarrow \{0,1\}^{1000}$, output r]

Note

A minimum security requirement for a PRG is that

the length **s** of the random seed should be **sufficiently large**

so that a search over $\mathbf{2^s}$ elements (the total number of possible seeds) is infeasible for the adversary.

Statistical Tests

Statistical test on $\{0,1\}^n$:

An algorithm A s.t. A(x) outputs "0" or "1",
that is $\mathbf{A : \{0,1\}^n \longrightarrow \{0,1\}}$

Examples:

1. A(x)=1 iff $|\#0(x) - \#1(x)| \leq 10\sqrt{n}$
2. A(x)=1 iff $|\#00(x) - n/4| \leq 10\sqrt{n}$
3. A(x)=1 iff max-run-of-0(x) $< 10 \log_2(n)$

.....

Advantage

- Let **G:K ⟶{0,1}^{n}** be a **PRG**
- Let **A: {0,1}^{n} ⟶ {0,1}** be a **statistical test** on {0,1}^{n}

Define: $$Adv_{PRG}[A,G] = \left| \Pr_{k \leftarrow K}[A(G(k))=1] - \Pr_{r \leftarrow \{0,1\}^n}[A(r)=1] \right| \in [0,1]$$

- Adv close to 0 => A cannot distinguish G from random
- Adv non-negligible => A can distinguish G from random
- Adv close to 1 => A can distinguish G from random very well

A silly example: A(x) = 0 ⇒ Adv_{PRG} [A,G] =

Example of Advantage

- Suppose $G: K \longrightarrow \{0,1\}^n$ satisfies **msb(G(k)) = 1** for 2/3 of keys in K
- Define statistical test $A(x)$ as:

if [msb(x)=1] output "1" else output "0"

Then

$$\mathrm{Adv}_{\mathrm{PRG}}[A,G] = \left| \Pr[A(G(k))=1] - \Pr[A(r)=1] \right| =$$

$$\left| 2/3 - 1/2 \right| = 1/6$$

A breaks G with advantage 1/6 (which is not negligible)
hence **G is not a good PRG**

Secure PRGs: crypto definition

Definition:

We say that $\mathbf{G : K \longrightarrow \{0,1\}^n}$ is a **secure PRG** if

for every "*efficient*" statistical test **A**, **$\mathbf{Adv_{PRG}[A,G]}$ is "negligible"**

Are there provably secure PRGs? Unknown (=> P ≠ PN)

A secure PRG is unpredictable

We show: PRG predictable $\Rightarrow$ PRG is insecure

Suppose A is an efficient algorithm s.t.

$$\Pr_{k \leftarrow K}\left[A(G(k)|_{1,\ldots,i}) = G(k)|_{i+1}\right] > \frac{1}{2} + \epsilon$$

for non-negligible ε (e.g. ε = 1/1000)

A secure PRG is unpredictable

Define statistical test B as:

$$B(X) = \begin{cases} \text{if } A(X|_{1,\ldots,i}) = X_{i+1} \text{ output } 1 \\ \text{else output } 0 \end{cases}$$

$$k \leftarrow K: \; Pr[B(G(k)) = 1] > \frac{1}{2} + \epsilon$$

$$r \leftarrow \{0,1\}^n: \; Pr[B(r) = 1] = \frac{1}{2}$$

$$\Rightarrow Adv_{PRG}[B,G] = |Pr[B(G(k)) = 1] - Pr[B(r) = 1]| > \epsilon$$

Thm (Yao'82): an unpredictable PRG is secure

Let **G : K** ⟶ $\{0,1\}^n$ be **PRG**

"Thm": if **∀ i ∈ {0, … , n-1}** **G** is **unpredictable** at position **i**

then **G** is a **secure PRG**.

If next-bit predictors cannot distinguish G from random
then no statistical test can !!

More Generally

Let $\mathbf{P_1}$ and $\mathbf{P_2}$ be two distributions over $\{0,1\}^n$

We say that $\mathbf{P_1}$ and $\mathbf{P_2}$ are **computationally indistinguishable** (denoted $P_1 \approx_p P_2$)

$$\text{if } \forall \text{ "efficient" statistical test } A$$

$$\left| \Pr_{X \leftarrow P_1}[A(X) = 1] - \Pr_{X \leftarrow P_2}[A(X) = 1] \right| < \text{negligible}$$

Example: a PRG is secure if $\{ k \longleftarrow K : G(k) \} \approx_p \text{uniform}(\{0,1\}^n)$

Semantic Security

What is a secure cipher?

Attacker's abilities: **CT only attack: obtains one ciphertext**

Possible security requirements:

attempt #1: **attacker cannot recover secret key**

$E(k, m) = m$ would be secure

attempt #2: **attacker cannot recover all of plaintext**

$E(k, m_0 \| m_1) = m_0 \| k \oplus m_1$ would be secure

Shannon's idea:

CT should reveal no "info" about PT

Recall Shannon's perfect secrecy

Let (E,D) be a cipher over (K,M,C)

Shannon's perfect secrecy:

(E,D) has perfect secrecy if $\forall\, m_0, m_1 \in M$ ($|m_0| = |m_1|$)

$$\{ E(k,m_0) \} = \{ E(k,m_1) \} \quad \text{where} \quad k \leftarrow K$$

- The two distributions must be **identical**
- Too strong definition
- It requires long keys
- Stream Ciphers can't satisfy it

Weaker Definition:

(E,D) has perfect secrecy if $\forall\, m_0, m_1 \in M$ ($|m_0| = |m_1|$)

$$\{ E(k,m_0) \} \approx_p \{ E(k,m_1) \} \quad \text{where} \quad k \leftarrow K$$

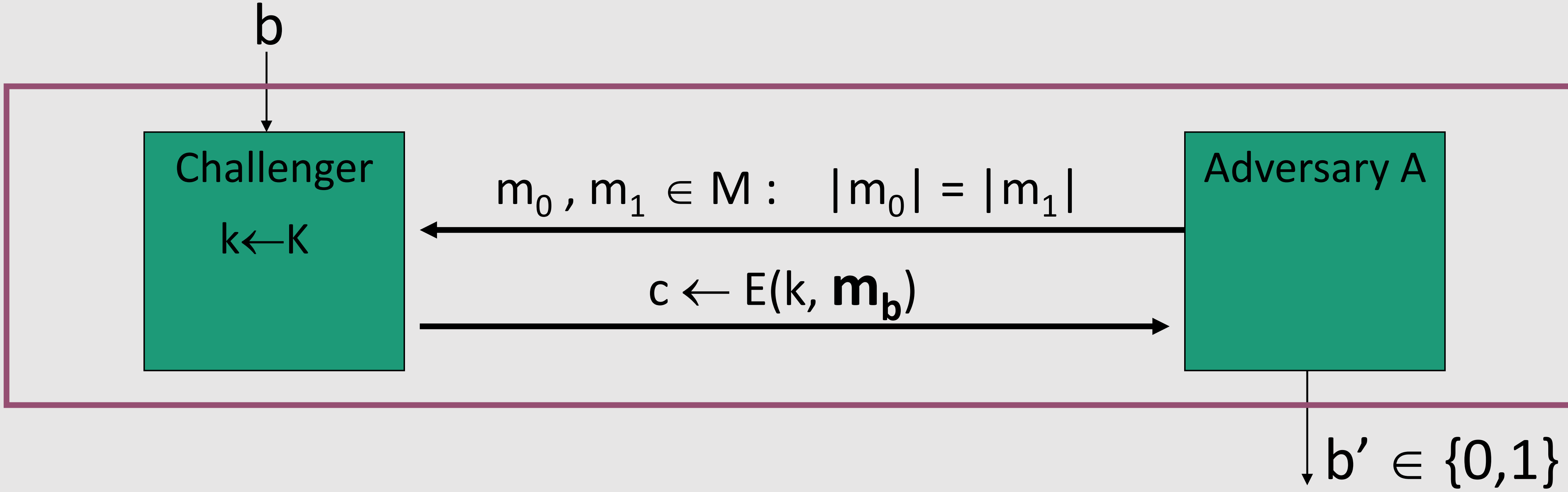
Rather than requiring the two distributions to be identical, we require them to be **COMPUTATIONALLY INDISTINGUISHABLE**

(One more requirement) … but also need adversary to exhibit $m_0, m_1 \in M$ explicitly

Semantic Security (one-time key)

For a cipher **Q = (E,D)** and an adversary **A** define a game as follows.

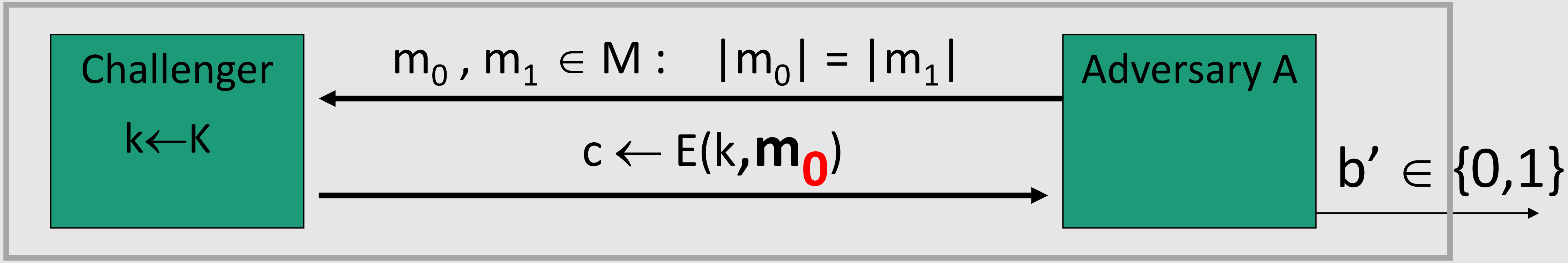
For b=0,1 define experiments EXP(0) and EXP(1) as:



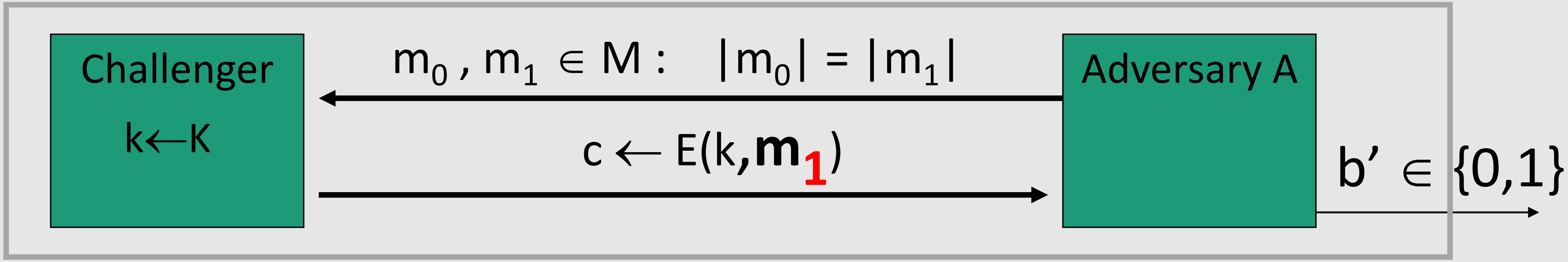
$\mathbf{Adv_{SS}[A,Q] := |\ Pr[EXP(0)=1\ ] - \ Pr[\ EXP(1)=1\ ]\ |}$

Semantic Security (one-time key)

EXP(0):



EXP(1):



$\mathrm{Adv}_{SS}[A,Q] = \big|\ \Pr[\ \mathbf{EXP(0)}{=}1\ ] - \Pr[\ \mathbf{EXP(1)}{=}1\ ]\ \big|$ should be "negligible" for all "efficient" A

Semantic Security (one-time key)

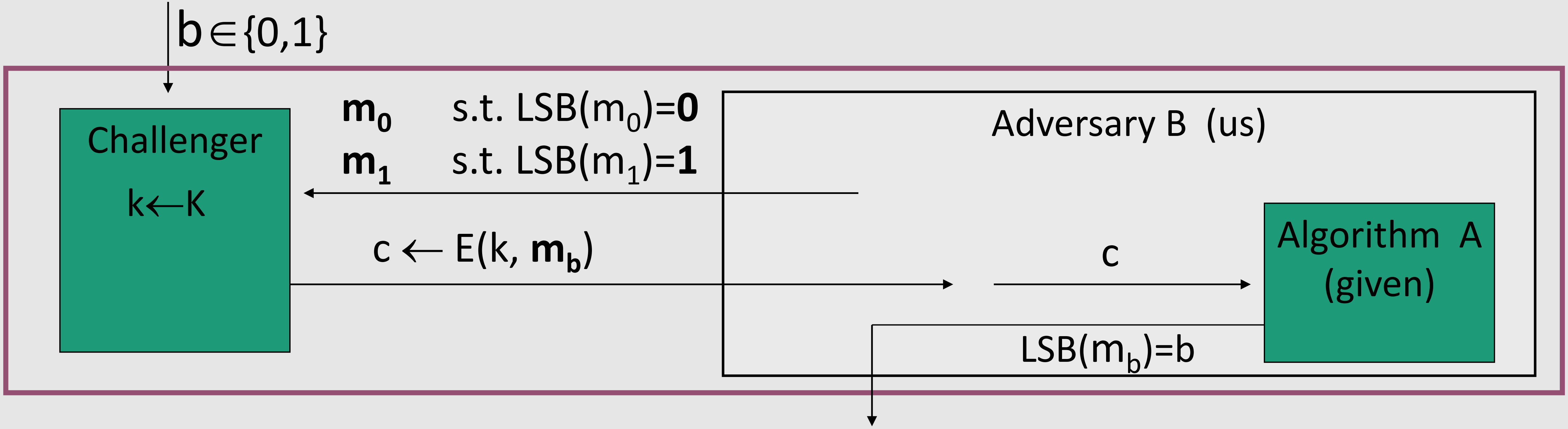
Definition:

Q is **semantically secure** if for all “efficient” **A**,

$$\mathbf{Adv_{SS}[A,Q]} \textbf{ is “negligible”.}$$

Example

Suppose efficient **A can always deduce LSB of PT from CT**
$\Rightarrow$ **Q** is **not** semantically secure.



Then $\mathbf{Adv_{SS}[B,Q]} = \left| \Pr[\,\mathbf{EXP(0)}=1\,] - \Pr[\,\mathbf{EXP(1)}=1\,] \right| =$

Stream ciphers are semantically secure

Theorem:

G is a **secure PRG** $\Rightarrow$ stream cipher **Q** derived from G is **semantically secure**

In particular:

∀ semantic security adversary **A**, ∃ a PRG adversary **B** (i.e., a statistical test) s.t.

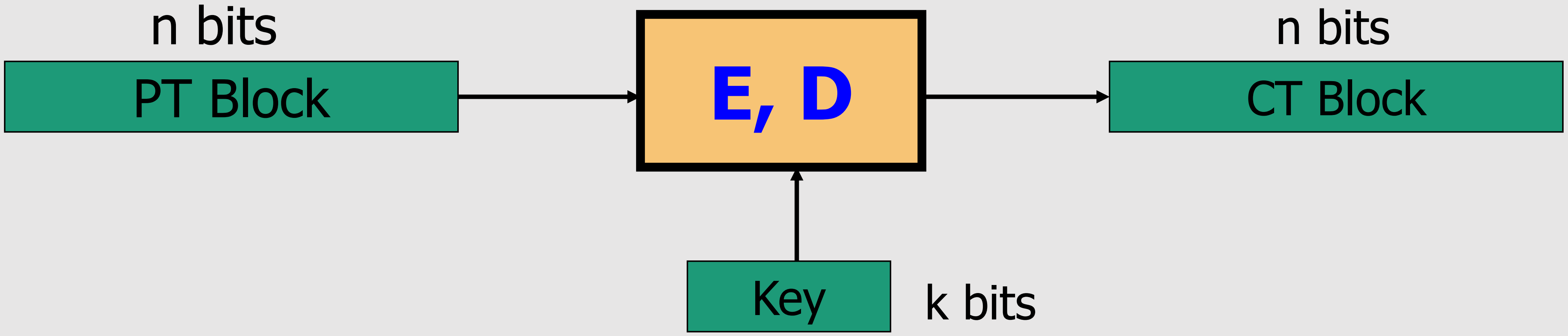
$$\mathbf{Adv_{SS}[A,Q] \leq 2 \cdot Adv_{PRG}[B,G]}$$

Block Ciphers

Outline

- Block Ciphers
- Pseudo Random Functions (PRFs)
- Pseudo Random Permutations (PRPs)
- DES – Data Encryption Standard
- AES – Advanced Encryption Standard
- PRF $\Rightarrow$ PRG
- PRG $\Rightarrow$ PRF

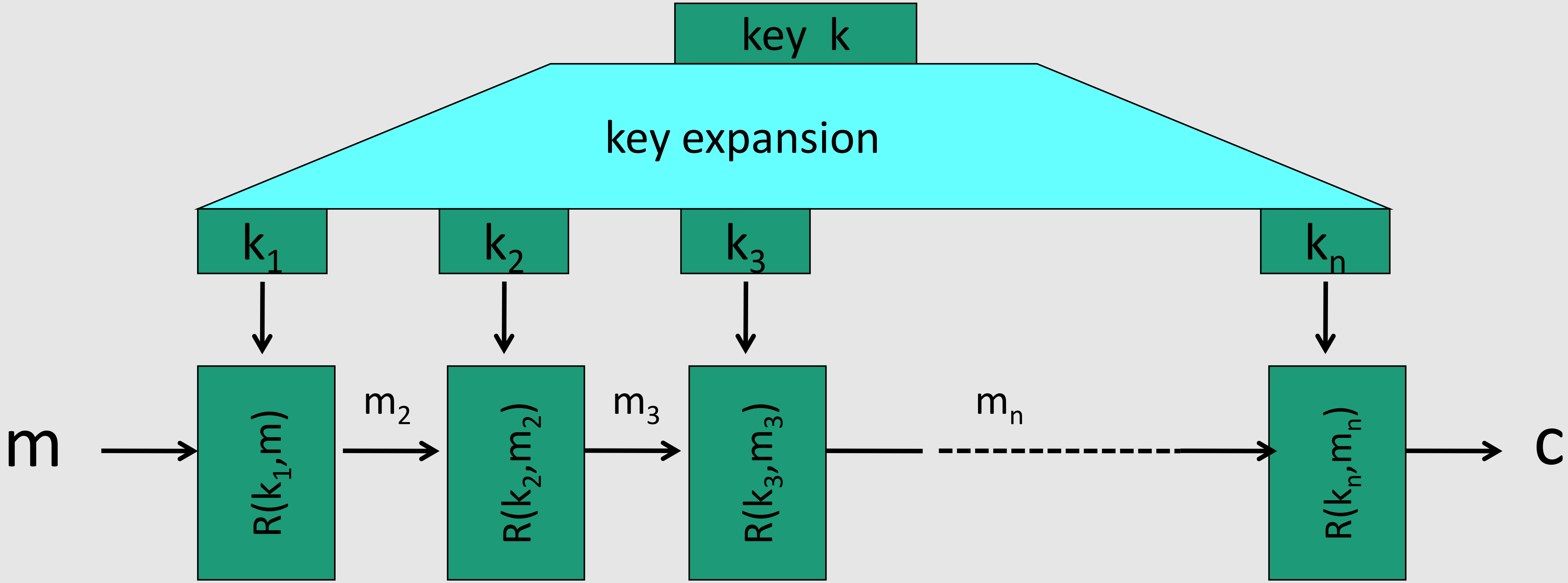
Block Ciphers: crypto work horse



Canonical examples:

- **DES**: n= 64 bits, k = 56 bits
- **3DES**: n= 64 bits, k = 168 bits
- **AES**: n=128 bits, k = 128, 192, 256 bits

Block Ciphers Built by Iteration



R(k,m) is called a **round function**

for 3DES (n=48), for AES-128 (n=10)

Performance: Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

	Cipher	Block/key size	Speed (MB/sec)
stream	RC4		126
	Salsa20/12		643
	Sosemanuk		727
block	3DES	64/168	13
	AES-128	128/128	109

Abstractly: PRPs and PRFs

- **Pseudo Random Function (PRF)** defined over (K,X,Y):

$$F:\ K \times X \rightarrow Y$$

such that there exists **"efficient" algorithm** to evaluate F(k,x)

- **Pseudo Random Permutation (PRP)** defined over (K,X):

$$E:\ K \times X \rightarrow X$$

such that:

1. There exists **"efficient"** **deterministic** algorithm to evaluate E(k,x)
2. The function $E(k, \cdot)$ is **one-to-one** (for every k)
3. There exists "efficient" **inversion algorithm** D(k,y)

Running example

- Example PRPs: 3DES, AES, …

  AES: $K \times X \rightarrow X$ where $K = X = \{0,1\}^{128}$

  3DES: $K \times X \rightarrow X$ where $X = \{0,1\}^{64}$, $K = \{0,1\}^{168}$

- Functionally, any **PRP is also a PRF.**
  - A PRP is a PRF where X=Y and is efficiently invertible.

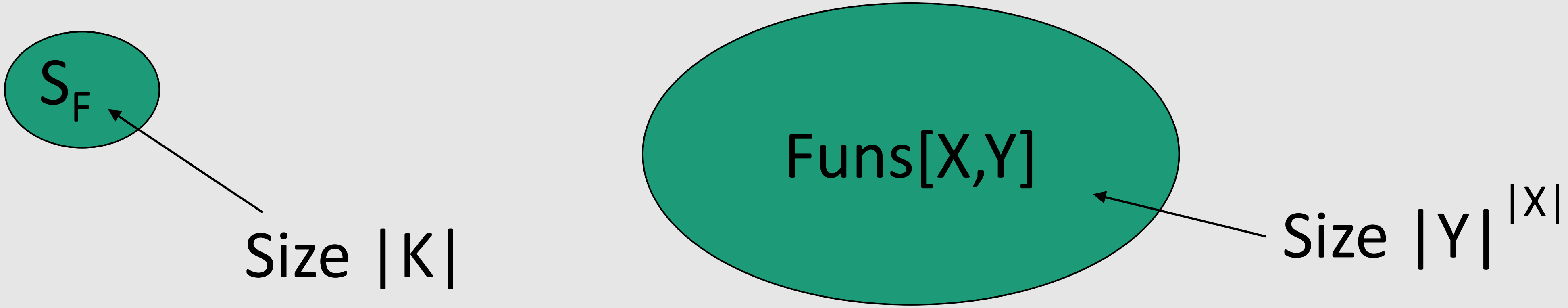
Secure PRFs

- Let $F: K \times X \rightarrow Y$ be a PRF. Set some notation:

  Funs[X,Y]: the set of **all** functions from X to Y

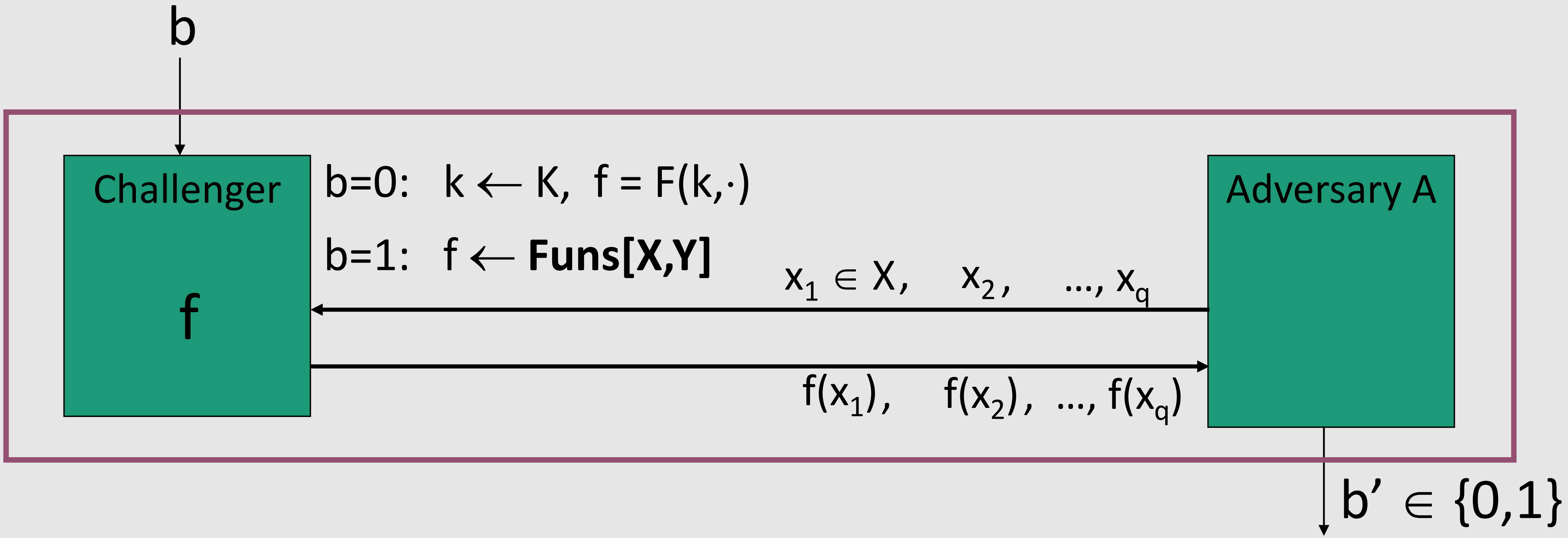
  $S_F = \{ F(k,\cdot) \text{ s.t. } k \in K \} \subseteq \text{Funs}[X,Y]$

- **Intuition:** a PRF is **secure** if a random function in Funs[X,Y] is "indistinguishable" from a random function in S_F



Secure PRF: definition

- Consider a PRF $\mathbf{F : K \times X \to Y}$. For b=0,1 define experiment EXP(b) as:



Definition: **F** is a **secure PRF** if for all "efficient" adversary A:

$$\mathbf{Adv_{PRF}[A,F] := \left| \Pr[EXP(0)=1] - \Pr[EXP(1)=1] \right|}$$ is "negligible".

Secure PRPs (secure block cipher)

- Let E: $K \times X \to X$ be a PRP

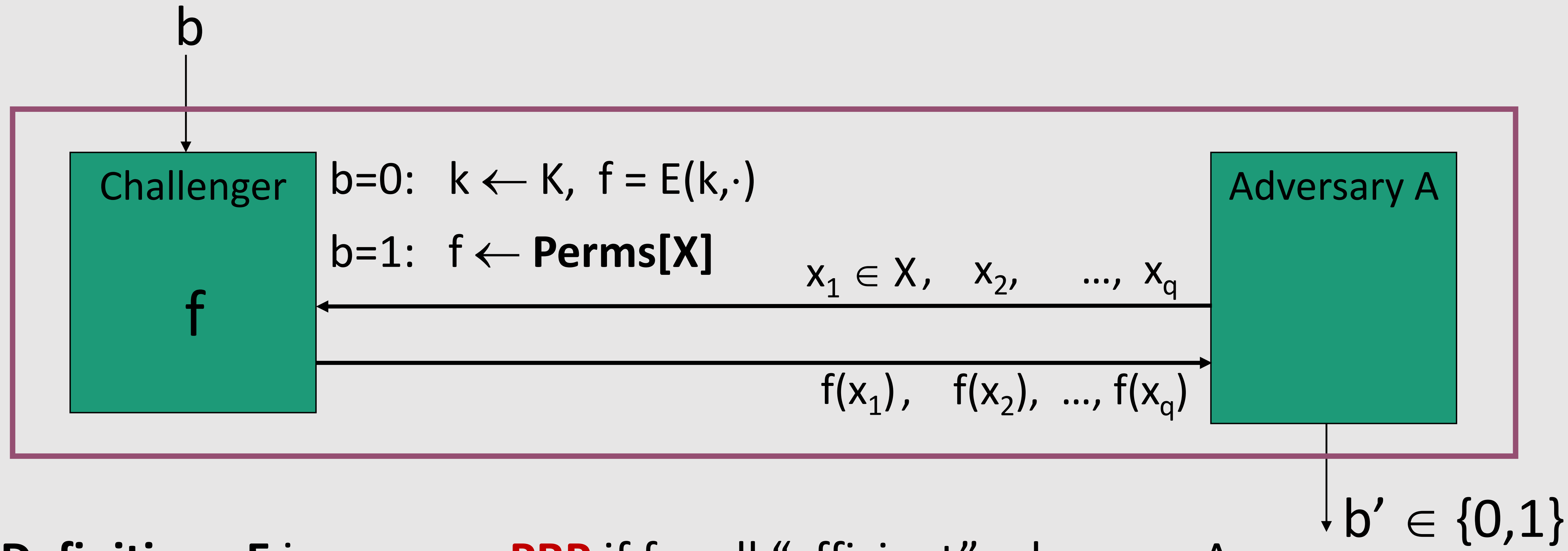
  Perms[X]: the set of **all one-to-one** functions from X to X (i.e., **permutations**)

  $S_E = \{ E(k,\cdot) \text{ s.t. } k \in K \} \subseteq \text{Perms}[X]$

- **Intuition:** a PRP is **secure** if a random function in Perms[X] is "indistinguishable" from a random function in S_E

Secure PRP (secure block cipher)

- Consider a PRP **E : K × X → X.** For b=0,1 define experiment EXP(b) as:



Definition. E is a **secure PRP** if for all "efficient" adversary A:

$$\mathbf{Adv_{PRP}[A,E]} = \left| \mathbf{Pr[EXP(0)=1] - Pr[EXP(1)=1]} \right| \text{ is "negligible".}$$

Data Encryption Standard (DES)

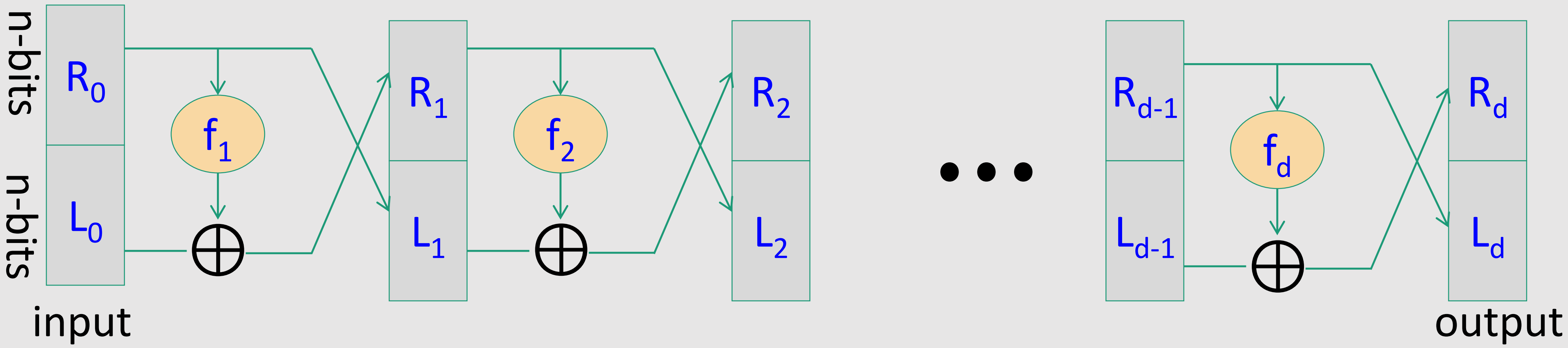
The Data Encryption Standard (DES)

- Early 1970s: Horst Feistel designs Lucifer at IBM
  key-length = 128 bits ; block-length = 128 bits
- 1973: NBS (nowadays called NIST) asks for block cipher proposals.
  IBM submits variant of Lucifer.
- 1976: NBS adopts DES as a federal standard
  key-length = 56 bits ; block-length = 64 bits
- 1997: DES broken by exhaustive search
- 2000: NIST adopts Rijndael as AES to replace DES

DES: core idea – Feistel Network

Given functions $f_1, \ldots, f_d$: $\{0,1\}^n \longrightarrow \{0,1\}^n$ (not necessarily invertible)

Goal: build **invertible** function F: $\{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$



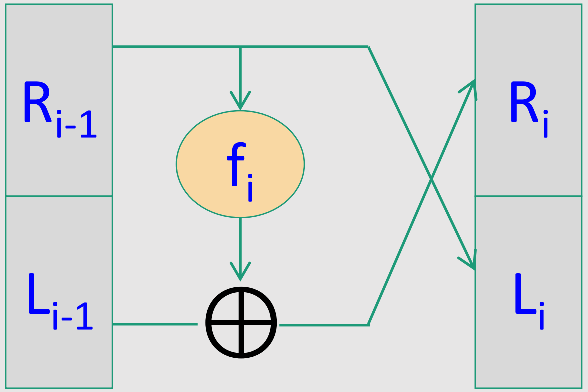
In symbols:
$$R_i = f_i(R_{i-1}) \oplus L_{i-1}$$
$$L_i = R_{i-1}$$

Feistel network is invertible

Claim: for all (**arbitrary**) $f_1, \ldots, f_d$: $\{0,1\}^n \longrightarrow \{0,1\}^n$

Feistel network $F: \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ is **invertible**

Proof: construct inverse



inverse

$R_{i-1} = L_i$

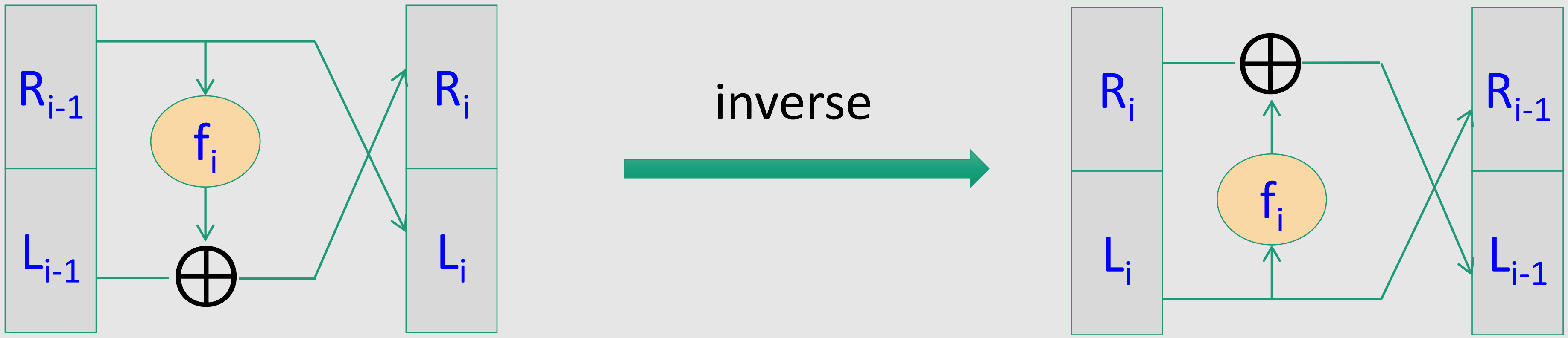
$L_{i-1} =$

Feistel network is invertible

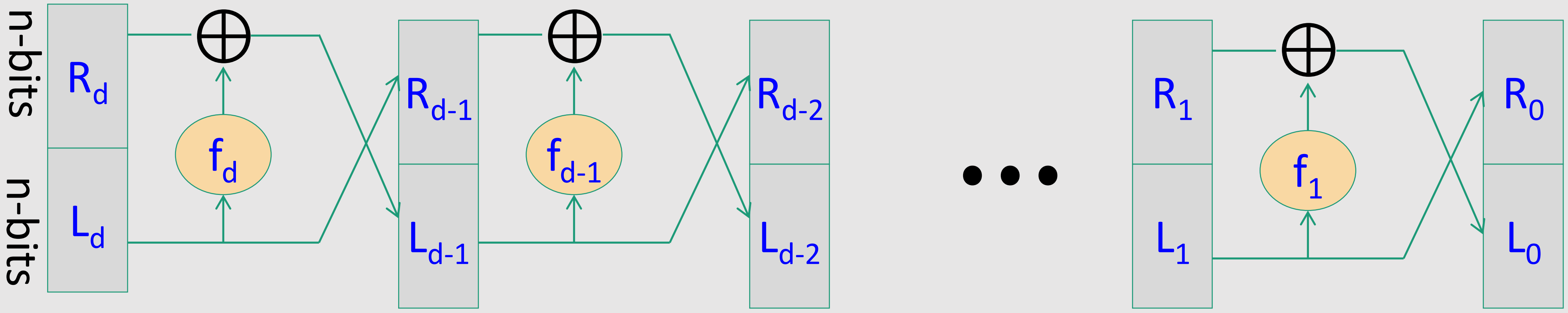
Claim: for all (**arbitrary**) $f_1, \ldots, f_d$: $\{0,1\}^n \longrightarrow \{0,1\}^n$

Feistel network $F: \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ is **invertible**

Proof: construct inverse



Decryption circuit



- Inversion is basically the same circuit, with $f_1, \ldots, f_d$ applied in reverse order

- General method for building invertible functions (block ciphers) from arbitrary functions.

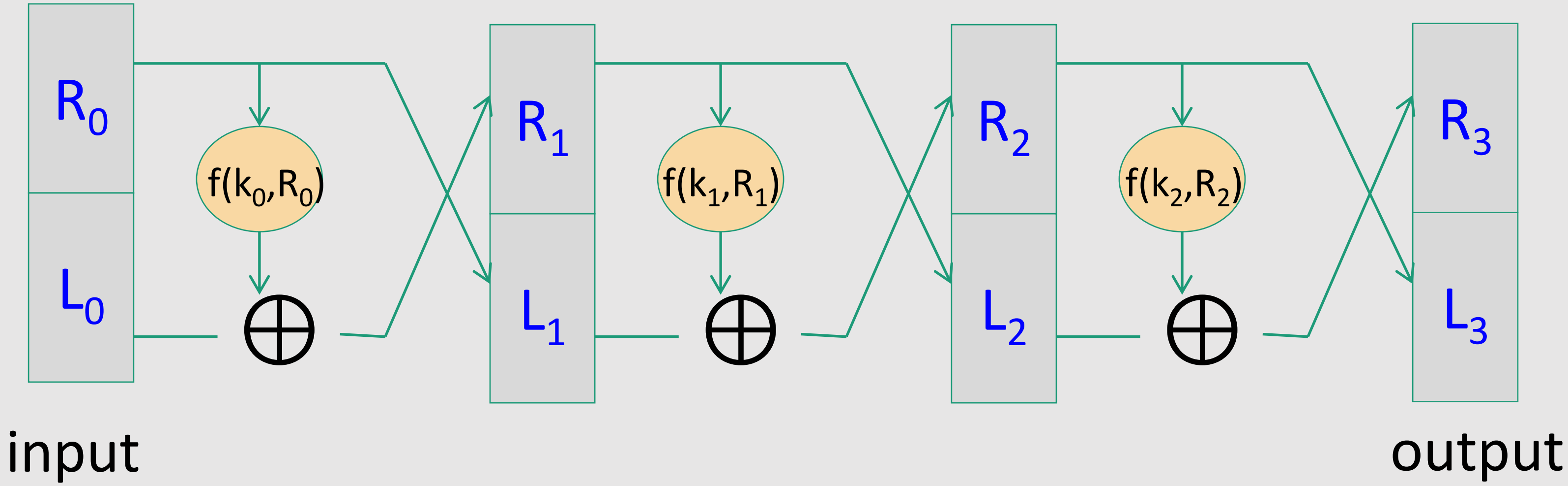
- Used in many block ciphers … but not AES

Theorem (Luby-Rackoff '85):

f: $K \times \{0,1\}^n \longrightarrow \{0,1\}^n$ a **secure PRF**

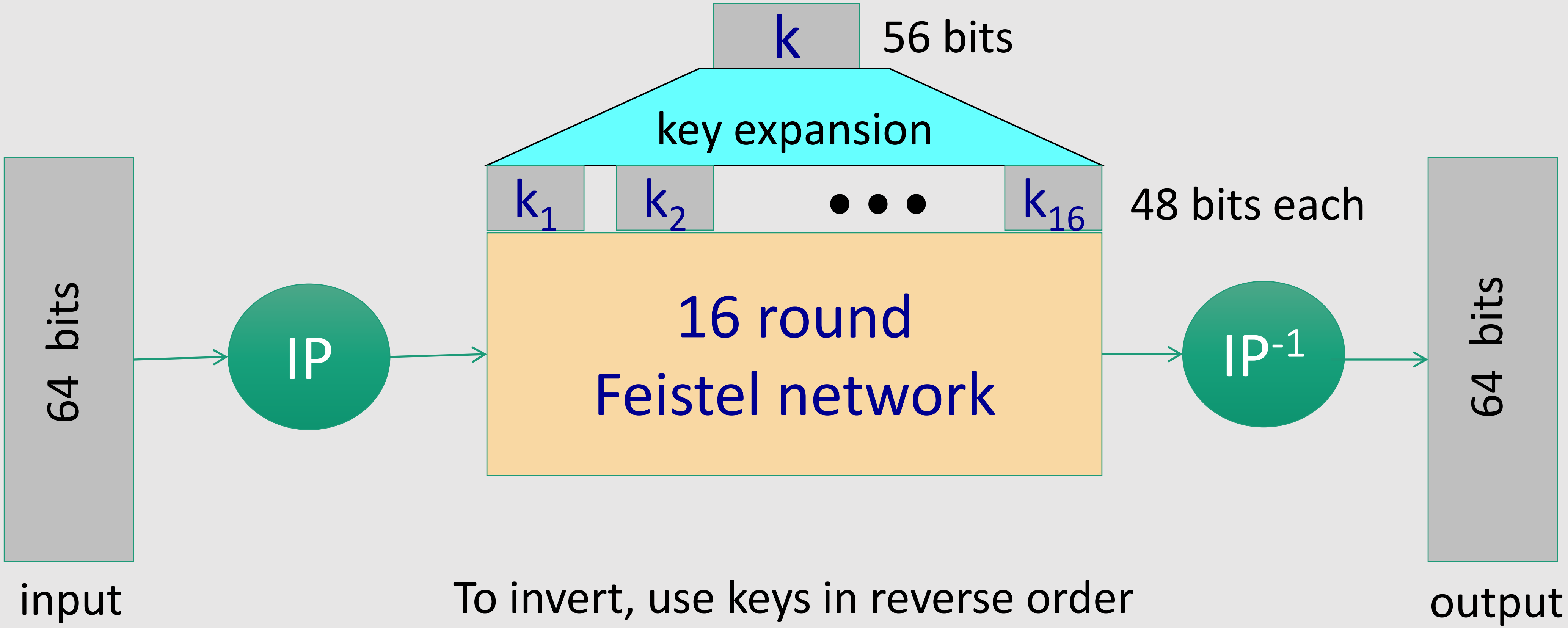
$\Rightarrow$ 3-round Feistel F: $K^3 \times \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ is a **secure PRP**

(k_0, k_1, k_2 three **independent** keys)



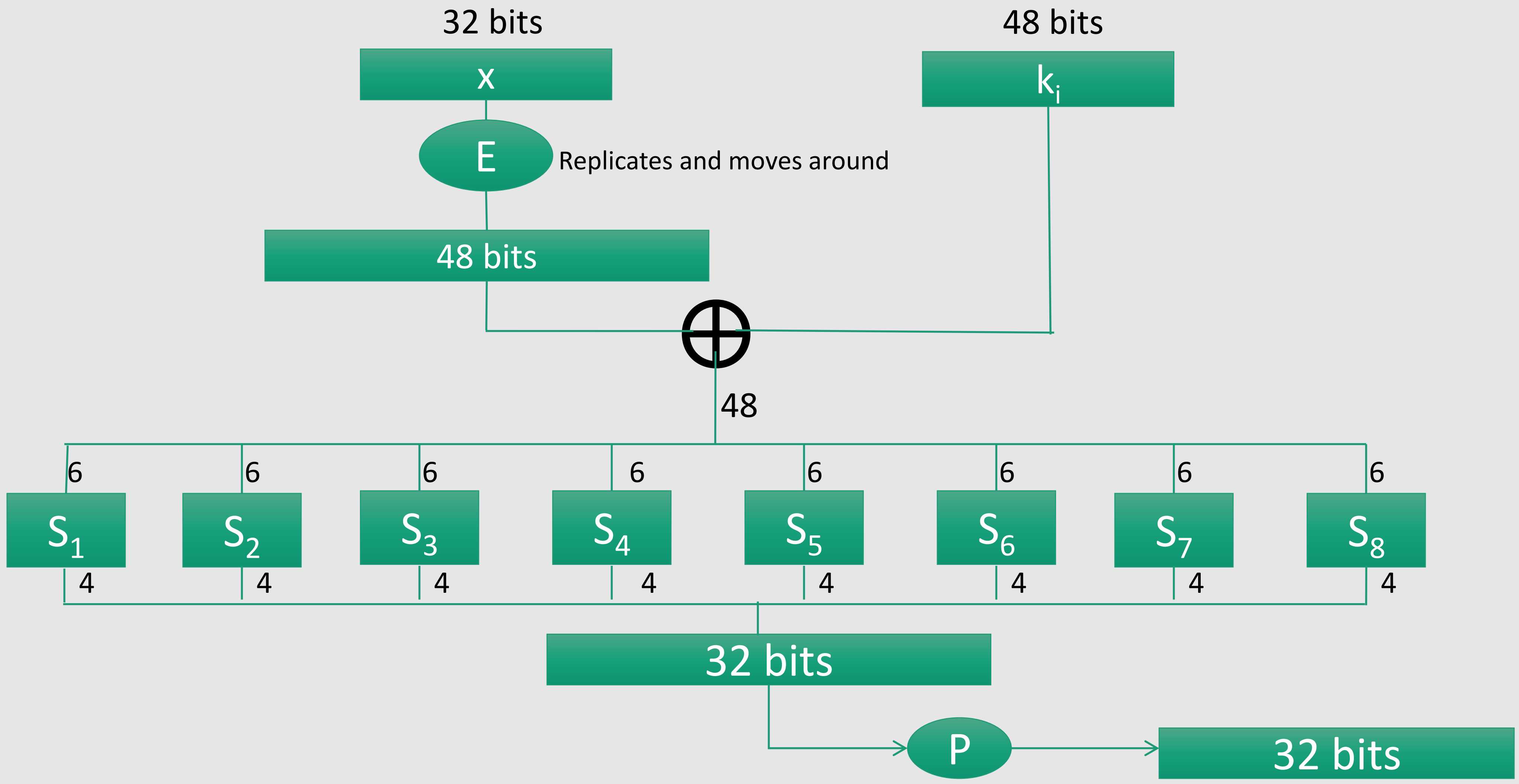
DES: 16 round Feistel network

$$f_1, \ldots, f_{16}: \{0,1\}^{32} \longrightarrow \{0,1\}^{32} \quad , \quad f_i(x) = F(k_i, x)$$



To invert, use keys in reverse order

The function $F(k_i, x)$



S-box: function $\{0,1\}^6 \longrightarrow \{0,1\}^4$, implemented as look-up table.

The S-boxes (substitution boxes)

$$S_i\colon \{0,1\}^6 \longrightarrow \{0,1\}^4$$

S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

$$S_5(011011) \longrightarrow 1001$$

Choosing the S-boxes and P-box

- Choosing the S-boxes and P-box at random would result in an insecure block cipher (key recovery after $\approx 2^{24}$ outputs)
- Several rules used in choice of S and P boxes:
  - No output bit should be close to a linear func. of the input bits
  - S-boxes are 4-to-1 maps (4 pre-images for each output)
  - …

Exhaustive Search for block cipher key

Goal: given a few input output pairs $\big(m_i, c_i = E(k, m_i)\big)$ i=1,..,3
find key k.

Exhaustive Search for block cipher key

Goal: given a few input output pairs $\big(m_i,\ c_i = E(k, m_i)\big)$ i=1,..,3
find key k.

Lemma: Suppose DES is an ***ideal cipher***
(2^{56} random invertible functions $\Pi_1, \ldots, \Pi_{2^{\wedge}56} : \{0,1\}^{64} \to \{0,1\}^{64}$)

Then $\forall$ m, c there is at most **<u>one</u>** key k s.t. c = DES(k, m)

with prob. $\geq 1 - 1/256 \approx 99.5\%$

Proof:

$$\Pr[\exists k' \neq k:\ c=DES(k,m)=DES(k',m)] \leq \sum_{k' \in \{0,1\}^{56}} \Pr[DES(k,m) = DES(k',m)] \leq 2^{56} \times 1/(2^{64}) =$$
$$= 1/(2^{8}) = 1/256$$

Exhaustive Search for block cipher key

For two DES pairs $\big(m_1, c_1{=}\mathrm{DES}(k, m_1)\big)$, $\big(m_2, c_2{=}\mathrm{DES}(k, m_2)\big)$

unicity prob. $\approx 1 - 1/2^{71}$

For AES-128: given two inp/out pairs, unicity prob. $\approx 1 - 1/2^{128}$

$\Rightarrow$ two input/output pairs are enough for exhaustive key search.

Exhaustive Search Attacks

DES challenge

8 byte blocks, encrypted with the same key

	m_1	m_2	m_3		
msg =	"The unkn	own mess	ages is:	XXXX	… "
CT =	c_1	c_2	c_3	c_4	

Goal: find $k \in \{0,1\}^{56}$ s.t. $DES(k, m_i) = c_i$ for $i=1,2,3$ and decrypt $c_4, c_5 \ldots$

1997: Internet search -- **3 months**

1998: EFF machine (deep crack) -- **3 days** (250K $)

1999: combined search -- **22 hours**

2006: COPACOBANA (120 FPGAs) **--** **7 days** (10K $)

⇒ 56-bit ciphers should not be used !!

Strengthening DES against exhaustive search

- Method 1: **Triple-DES**
- Method 2: **DESX**
- General construction that can be applied to other block ciphers as well.

Triple DES

- Consider a **block cipher**

  $\mathbf{E} : K \times M \longrightarrow M$

  $\mathbf{D} : K \times M \longrightarrow M$

- Define $\mathbf{3E: K^3 \times M \longrightarrow M}$ as

$$\mathbf{3E\ (k_1,k_2,k_3,\ m) = E(k_1,\ D(k_2,\ E(k_3,m)))}$$

- For **3DES** (or **Triple DES**)
  - **key lenght** = 3×56 = **168 bits**.
  - **3×slower** than DES.
  - $\mathbf{k_1=k_2=k_3 \Rightarrow}$ **single DES**
  - **simple attack in time** $\approx \mathbf{2^{118}}$ (more on this later …)

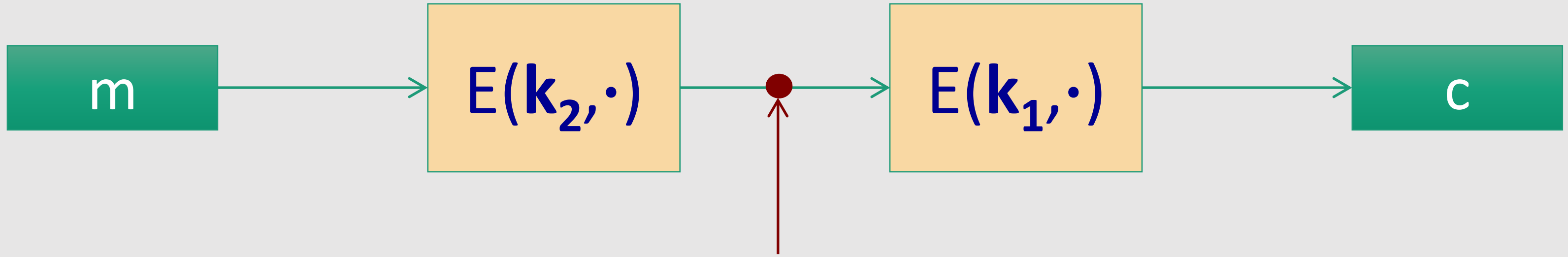
Why not double DES?

- Given a block cipher **E**, define $\mathbf{2E(\ k_1,\ k_2,\ m)\ \ =\ \ E(k_1\ ,\ E(k_2\ ,\ m))}$
- **Double DES:** $\mathbf{2DES(\ k_1,\ k_2,\ m)\ \ =\ \ E(k_1\ ,\ E(k_2\ ,\ m))}$

  key-length = 112 bits for 2DES
- **Attack:** Given **m** and **c** the goal is to

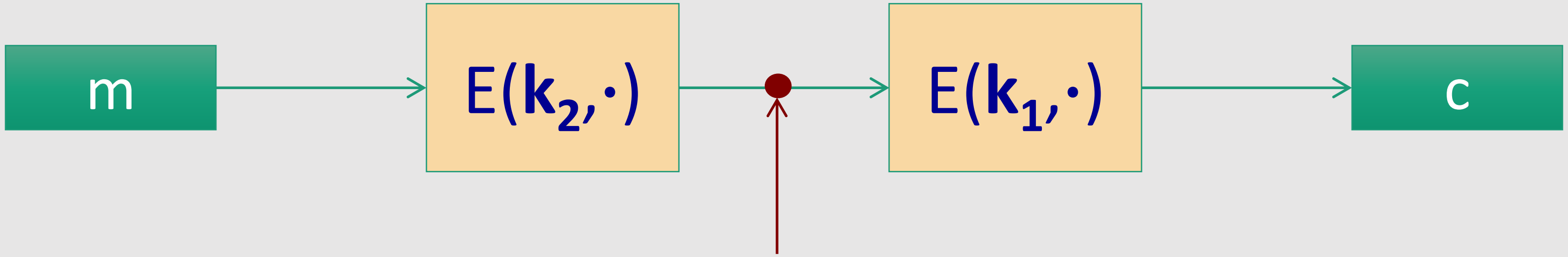
  find (k_1,k_2) s.t. $\mathbf{E(k_1,\ E(k_2,m)) = c}$ **or equivalently**

  find (k_1,k_2) s.t. $\mathbf{E(k_2,m) = D(k_1,c)}$



Meet in the middle attack

- **Attack:** Given **m** and **c** the goal is to
  find (k_1,k_2) s.t. $\mathbf{E(k_1, E(k_2,m)) = c}$ **or equivalently**
  find (k_1,k_2) s.t. $\mathbf{E(k_2,m) = D(k_1,c)}$



- **Attack involves TWO STEPS**

Meet in the middle attack

Step 1:
- build table.
- sort on 2nd column

$k^0 = 00\ldots00$	$E(k^0, m)$
$k^1 = 00\ldots01$	$E(k^1, m)$
$k^2 = 00\ldots10$	$E(k^2, m)$
$\vdots$	$\vdots$
$k^N = 11\ldots11$	$E(k^N, m)$

2^{56} entries

Meet in the middle attack

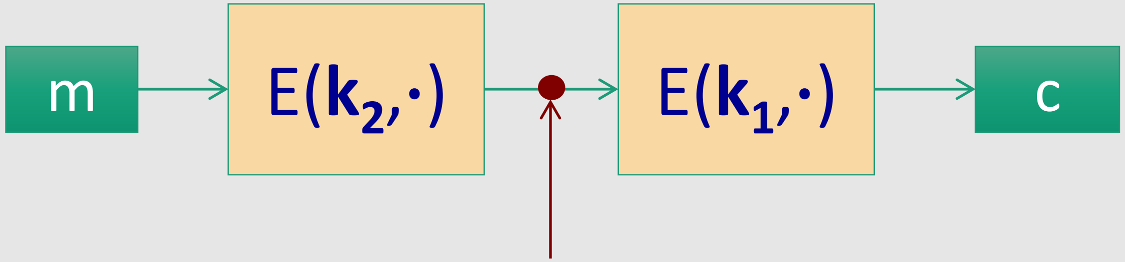
Step 2:

- for each $k \in \{0,1\}^{56}$ do:

  test if $D(k, c)$ is in the 2^{nd} column of the table

  If so, then $\mathbf{E(k^i,m) = D(k,c)} \;\; \Rightarrow \;\; \mathbf{(k^i,k) = (k_2,k_1)}$

$k^0 = 00\ldots00$	$E(k^0, m)$
$k^1 = 00\ldots01$	$E(k^1, m)$
$\vdots$	$\vdots$
$k^i = 00\ldots\ldots.$	$E(k^i, m)$
$\vdots$	$\vdots$
$k^N = 11\ldots11$	$E(k^N, m)$



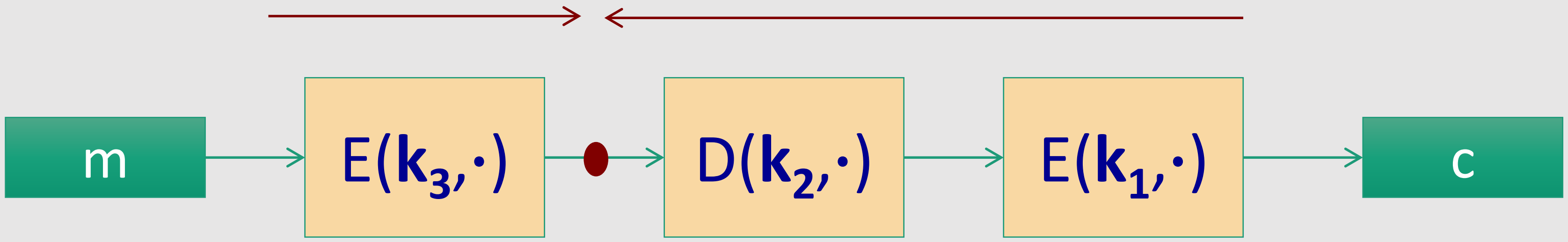
Meet in the middle attack

$$\text{Time} = \underbrace{2^{56}\log(2^{56})}_{\text{build + sort table}} + \underbrace{2^{56}\log(2^{56})}_{\text{search in table}} < 2^{63} \ll 2^{112} \;,$$

$$\text{Space} \approx 2^{56}$$

Meet in the middle attack

Same attack on 3DES:



Time = 2^{118} ,   space ≈ 2^{56}

$$\text{Time} = \underbrace{2^{56}\log(2^{56})}_{\text{build + sort table}} + \underbrace{2^{112}\log(2^{56})}_{\text{search in table}} < 2^{118}$$

DESX

- Consider a **block cipher**

  $\mathbf{E} : K \times M \longrightarrow M$

  $\mathbf{D} : K \times M \longrightarrow M$

- Define **EX** as

$$\mathbf{EX(\, k_1, k_2, k_3, m) \;=\; k_1 \oplus E(k_2,\; m \oplus k_3\,)}$$

- For **DESX**
  - key-len = **64**+**56**+**64** = 184 bits $\qquad \mathbf{k_1 \oplus E(k_2,\; m \oplus k_3\,)}$
  - … but easy attack in time $2^{64+56} = 2^{120}$

- Note: $k_1 \oplus E(k_2, m)$ and $E(k_2, m \oplus k_1)$ insecure !!

  (XOR outside) or (XOR inside) ⇒ As weak as E w.r.t. exhaustive search

Few others attacks on block ciphers

Linear attacks on DES

A tiny bit of linearly in S_5 lead to a 2^{43} time attack.

Total attack time $\approx 2^{43}$ ($<< 2^{56}$) with 2^{42} random inp/out pairs

Quantum attacks

Generic search problem:

Let $f: X \longrightarrow \{0,1\}$ be a function.

Goal: find $x^* \in X$ s.t. $f(x^*)=1$.

Classical computer: best generic algorithm **time = $O(\,|X|\,)$**

Quantum computer [Grover '96] : **time = $O(\,|X|^{1/2}\,)$**

Quantum exhaustive search

Given **m** and **c = E(k,m)** define

$$\text{For } k \in K,\ \ f(k) = \begin{cases} 1 & \text{if } E(k,m) = c \\ 0 & \text{otherwise} \end{cases}$$

Grover $\Rightarrow$ quantum computer can find k in time $O(\ |K|^{1/2}\ )$

DES: time $\approx 2^{28}$, AES-128: time $\approx 2^{64}$

Quantum computer $\Rightarrow$ 256-bits key ciphers (e.g., AES-256)

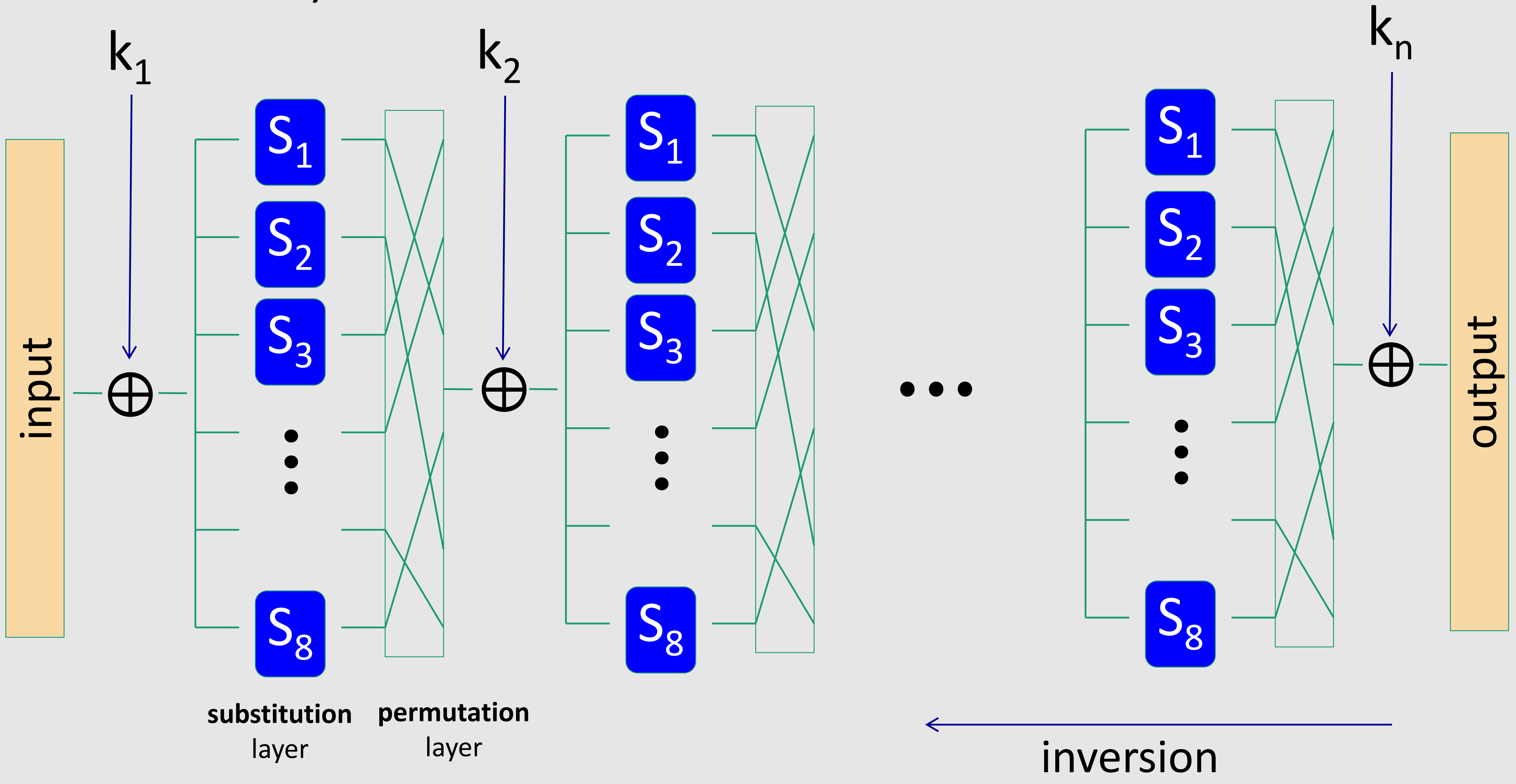
Advanced Encryption Standard (AES)

The AES process

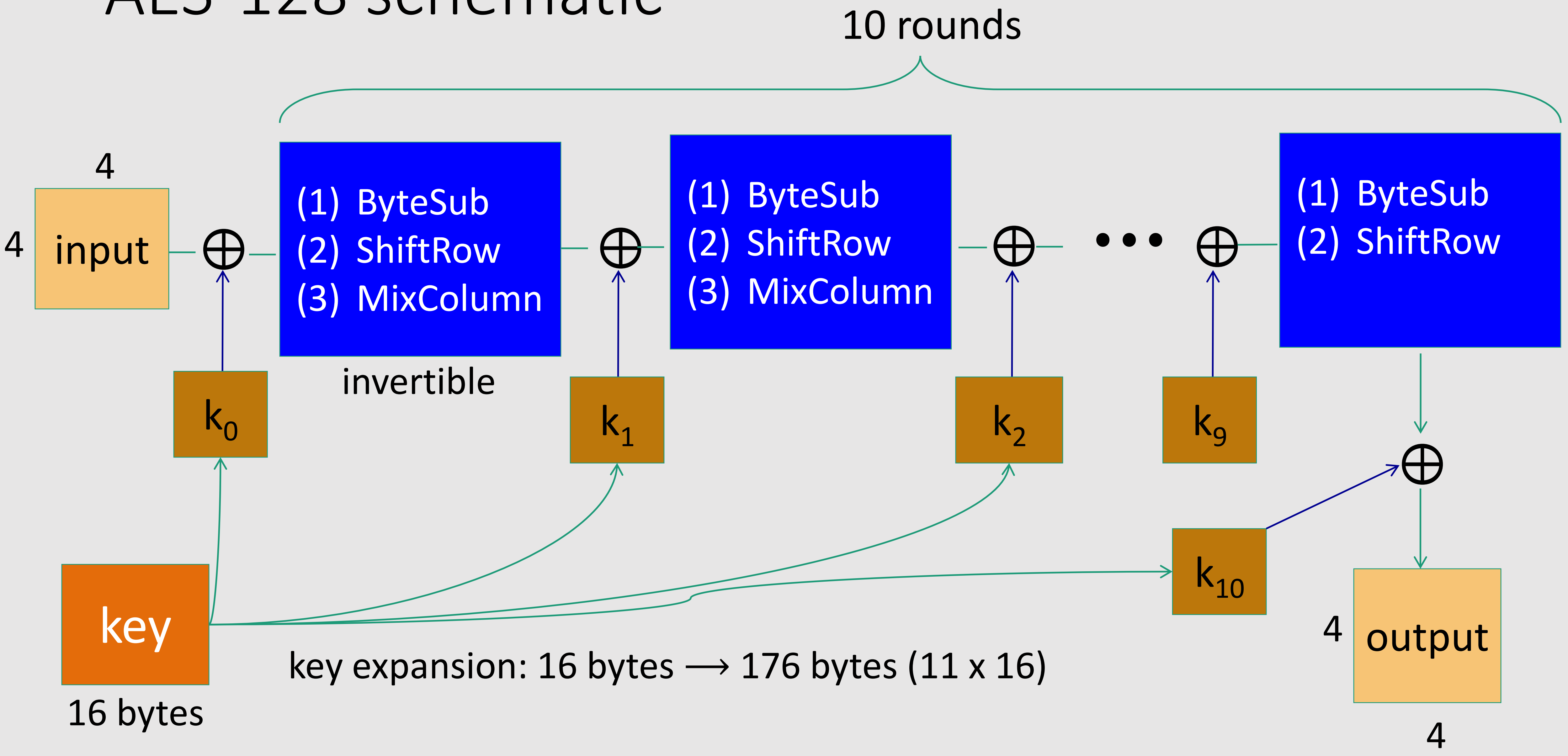
- 1997: NIST publishes request for proposal
- 1998: 15 submissions. Five claimed attacks.
- 1999: NIST chooses 5 finalists
- 2000: NIST chooses Rijndael as AES (designed in Belgium)

Key sizes: 128, 192, 256 bits. Block size: 128 bits

AES is a Substitution–permutation Network (not Feistel)



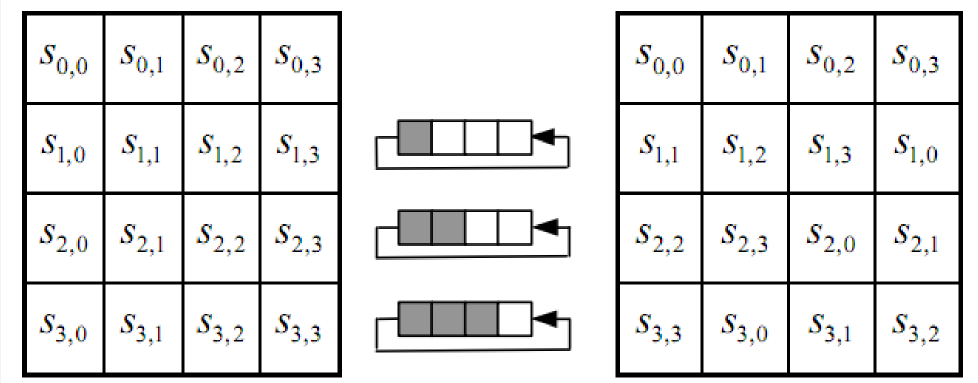
AES-128 schematic



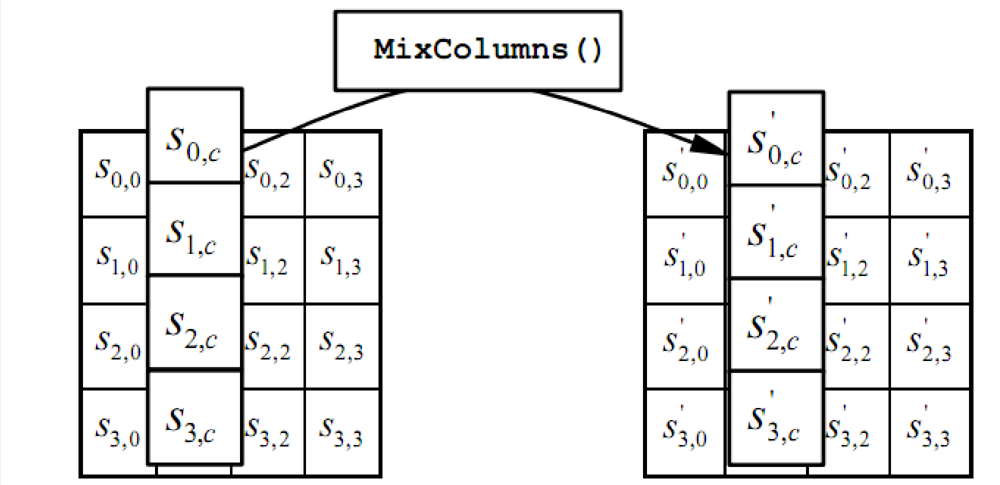
The round function

- **ByteSub**: a 1 byte S-box. 256 byte table (easily computable)
  - Apply S-box to each byte of the 4x4 input A, i.e., A[i,j] = S[A[i,j]], for 1 ≤i,j≤4

- **ShiftRows**:



- **MixColumns**:



AES in hardware

AES instructions in Intel Westmere:

- **aesenc, aesenclast**: do one round of AES
  128-bit registers: xmm1=state, xmm2=round key
  aesenc xmm1, xmm2 ; puts result in xmm1
- **aeskeygenassist**: performs AES key expansion
- Claim 14 x speed-up over OpenSSL on same hardware

Similar instructions on AMD Bulldozer

Attacks

- Best key recovery attack:

  four times better than ex. search [BKR'11]

- Related key attack on AES-256: [BK'09]

  Given 2^{99} inp/out pairs from **four related keys** in AES-256

  can recover keys in time $\approx 2^{99}$

$$\text{PRF} \Rightarrow \text{PRG}$$
$$\text{PRG} \Rightarrow \text{PRF}$$

An easy application: PRF $\Rightarrow$ PRG (counter mode)

- Let **F: K $\times$ $\{0,1\}^n \to \{0,1\}^n$** be a **PRF.**

- We define the **PRG** **G: K $\to \{0,1\}^{nt}$** as follows:

  (**t** is a parameter that we can choose)

$$G(k) = F(k, \langle 0 \rangle n) \;||\; F(k, \langle 1 \rangle n) \;||\; \cdots \;||\; F(k, \langle t\text{-}1 \rangle n)$$

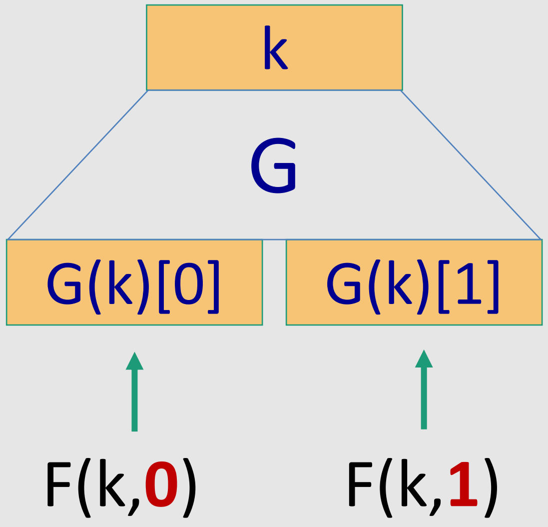
- **Properties:**
  - **Theorem:** If **F** is a **secure PRF** then **G** is a **secure PRG**
  - Key property: **parallelizable**

Can we build a PRF from a PRG?

Let $G: K \longrightarrow K^2$ be a PRG

Define a 1-bit PRF $F: K \times \{0,1\} \longrightarrow K$ as

$$F(k, x\in\{0,1\}) = G(k)[x]$$



Theorem. If **G** is a **secure PRG** then **F** is a **secure PRF**

Can we build a PRF with a larger domain? (e.g., 128 bits)

Extending a PRG

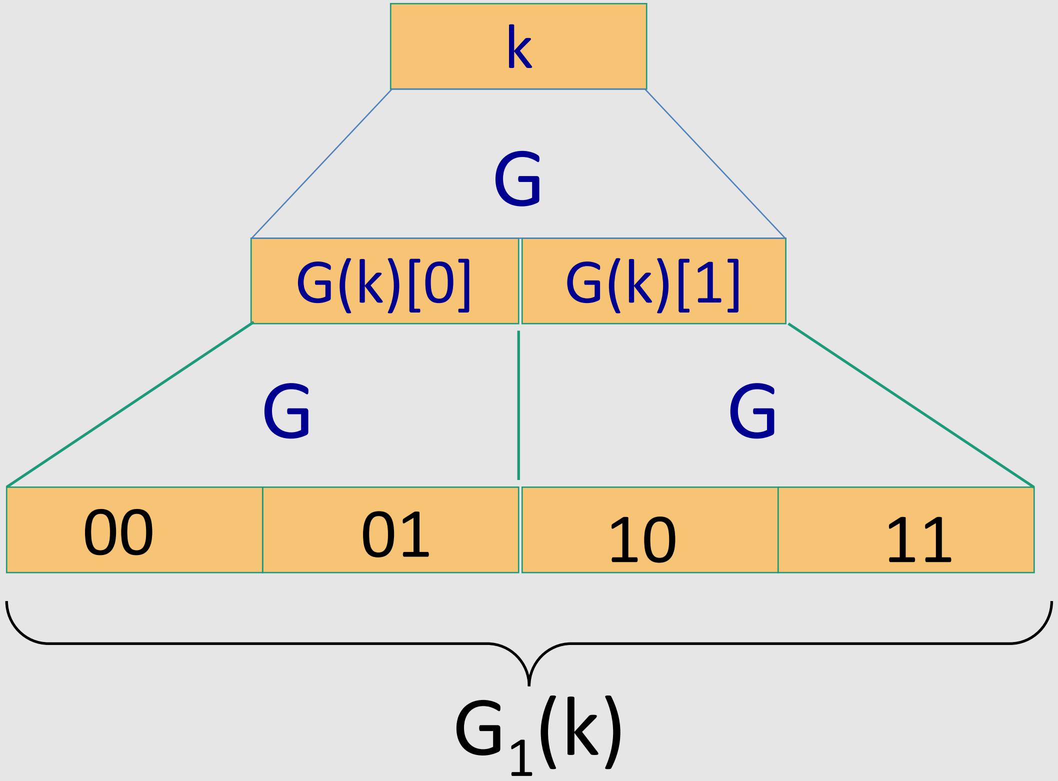
Let $\qquad G:\ K \longrightarrow K^2$ be a PRG

Define $\qquad G_1:\ K \longrightarrow K^4$ as

$$G_1(k) = G(G(k)[0]) \parallel G(G(k)[1])$$

Then define a 2-bit PRF $F: K \times \{0,1\}^2 \longrightarrow K$ as

$$F(k, x\in\{0,1\}^2) = G_1(k)[x]$$



Extending more

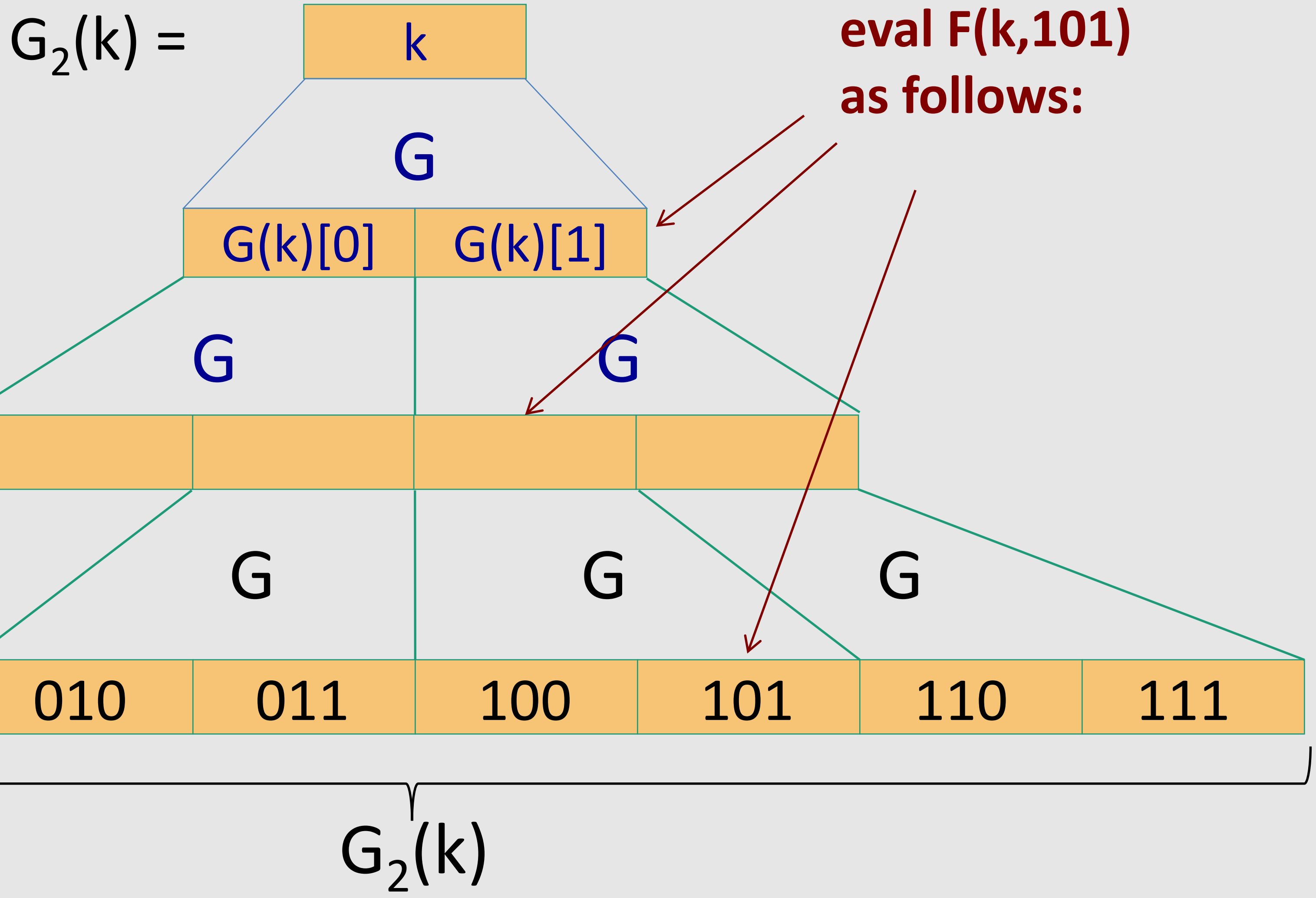
Let $\quad G: K \longrightarrow K^2$.

Define $G_2: K \longrightarrow K^8$ as

Then define a 3-bit PRF

$F: K \times \{0,1\}^3 \longrightarrow K$ as

$F(k, x \in \{0,1\}^3) = G_2(k)[x]$



Extending even more: the GGM PRF

Let $G: K \longrightarrow K^2$. define PRF $F: K \times \{0,1\}^n \longrightarrow K$ as

For input $x = x_0\, x_1 \ldots x_{n-1} \in \{0,1\}^n$ do:

$$k \xrightarrow{\mathbf{G(k)[x_0]}} k_1 \xrightarrow{\mathbf{G(k_1)[x_1]}} k_2 \xrightarrow{\mathbf{G(k_2)[x_2]}} k_3 \quad \bullet\bullet\bullet \quad \xrightarrow{\mathbf{G(k_{n-1})[x_{n-1}]}} k_n$$

Security: **G** a **secure PRG** $\Rightarrow$ **F** is a **secure PRF** on $\{0,1\}^n$.

Not used in practice due to slow performance.

Secure block cipher from a PRG?

Can we build a secure PRP from a secure PRG?

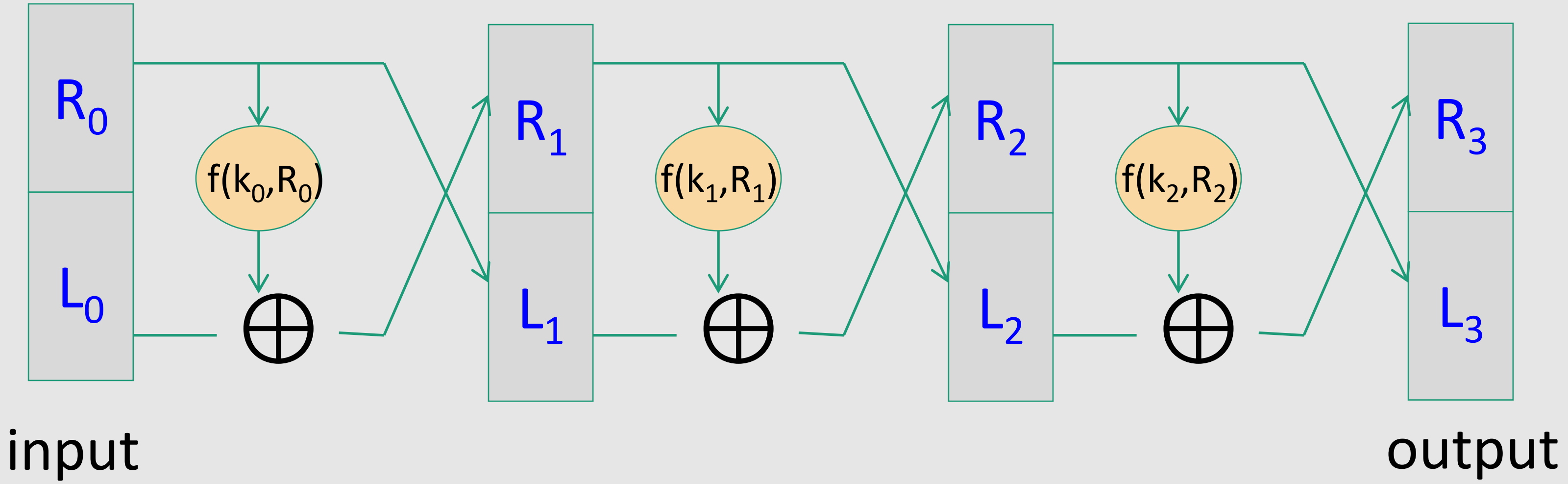
- No, it cannot be done
- Yes, just plug the GGM PRF into the Luby-Rackoff theorem
- It depends on the underlying PRG

Theorem (Luby-Rackoff '85):

f: $K \times \{0,1\}^n \longrightarrow \{0,1\}^n$ a **secure PRF**

$\Rightarrow$ 3-round Feistel F: $K^3 \times \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ is a **secure PRP**

(k_0, k_1, k_2 three **independent** keys)



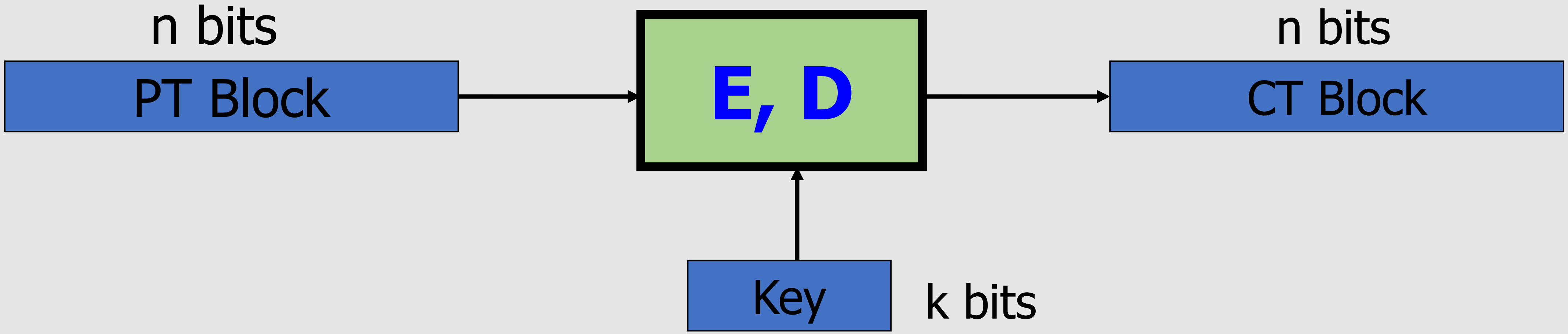
Modes of Operation (using block ciphers)

Outline

- One-Time Key
  - Semantic Security
  - Electronic Code Book (ECB)
  - Deterministic Counter Mode (DETCTR)
- Many-Time Key
  - Semantic Security for Many-Time Key:
    Semantic Security under Chosen-Plaintext Attack (CPA)
  - Cipher Block Chaining (CBC)
    - Randomized
    - Nonce-based

Review: PRPs and PRFs

Block Ciphers



Canonical examples:

- **DES**: n= 64 bits, k = 56 bits
- **3DES**: n= 64 bits, k = 168 bits
- **AES**: n=128 bits, k = 128, 192, 256 bits

Abstractly: PRPs and PRFs

- **Pseudo Random Function** (**PRF**) defined over (K,X,Y):

$$F:\ K \times X \ \rightarrow\ Y$$

such that there exists "efficient" algorithm to evaluate F(k,x)

- **Pseudo Random Permutation** (**PRP**) defined over (K,X):

$$E:\ K \times X \ \rightarrow\ X$$

such that:

1. There exists "efficient" <u>deterministic</u> algorithm to evaluate E(k,x)
2. The function $E(k, \cdot)$ is one-to-one, for every k
3. There exists "efficient" inversion algorithm D(k,y)

Using block ciphers

- Don’t think about the **inner-workings** of AES and 3DES.

- We assume both are **secure PRPs** and will see how to use them

Modes of Operation

How to use a **block cipher** on **messages consisting of more than one block**

- **One-Time Key**
  - Electronic Code Book
  - Deterministic Counter Mode

- **Many-Time Key**
  - Cipher Block Chaining
  - Counter Mode

Modes of Operation
One-Time Key

(example: encrypted email, new key for every message)

Using PRPs and PRFs

Goal: build "secure" encryption from a secure PRP (e.g., AES).

This segment: **one-time key**

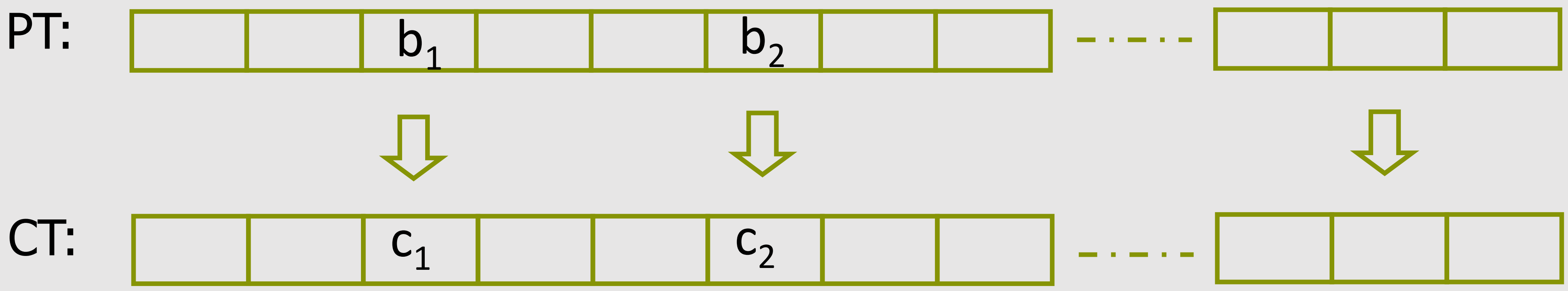
1. **Adversary's power:** Adversary sees only one ciphertext (one-time key)

2. **Adversary's goal:** Learn info about PT from CT (semantic security)

Next segment: many-time keys (a.k.a. *chosen-plaintext security*)

Incorrect use of a PRP

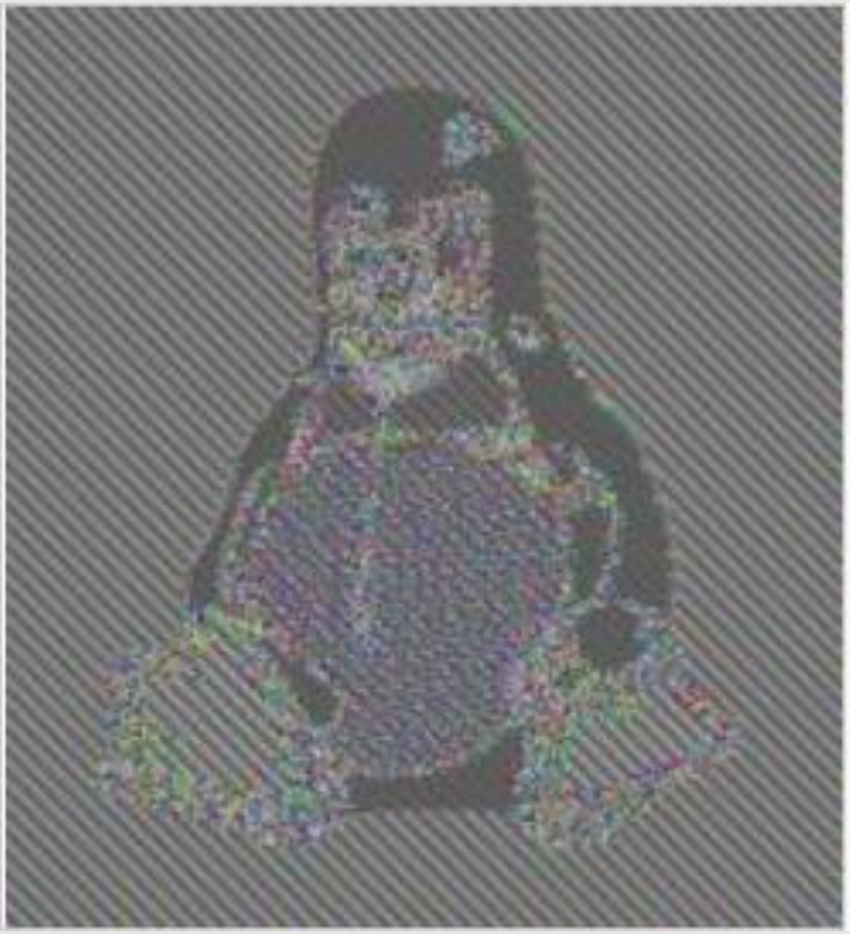
Electronic Code Book (ECB):



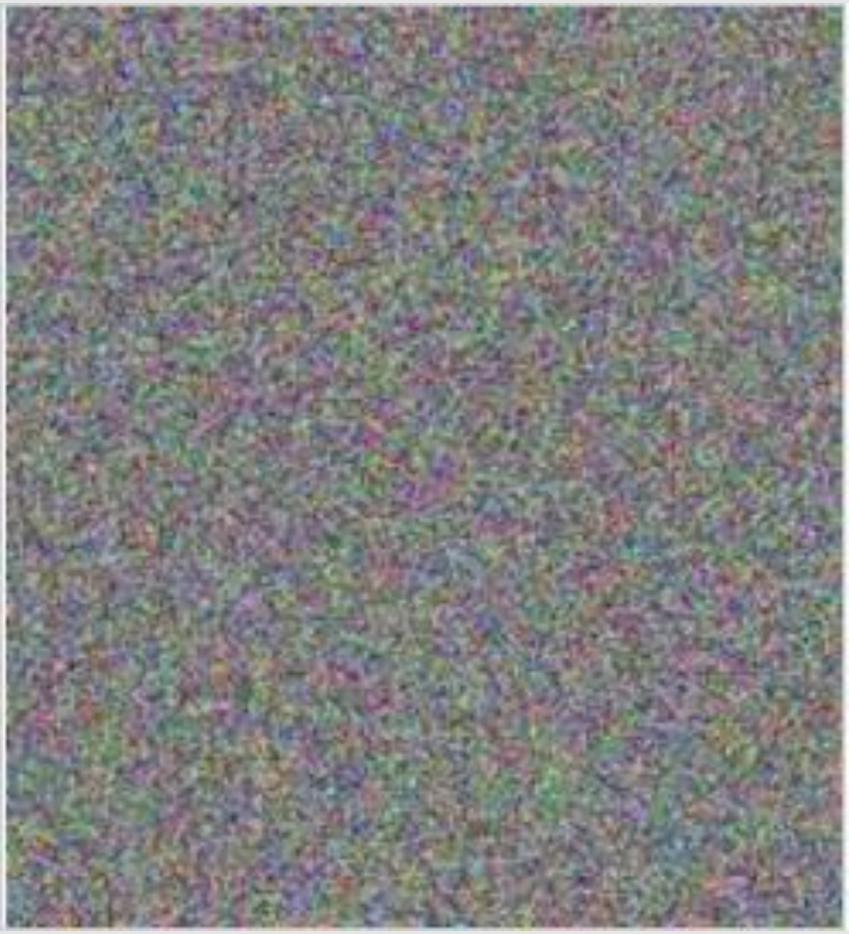
Problem: if $b_1 = b_2$ then $c_1 = c_2$

In pictures

Plain text



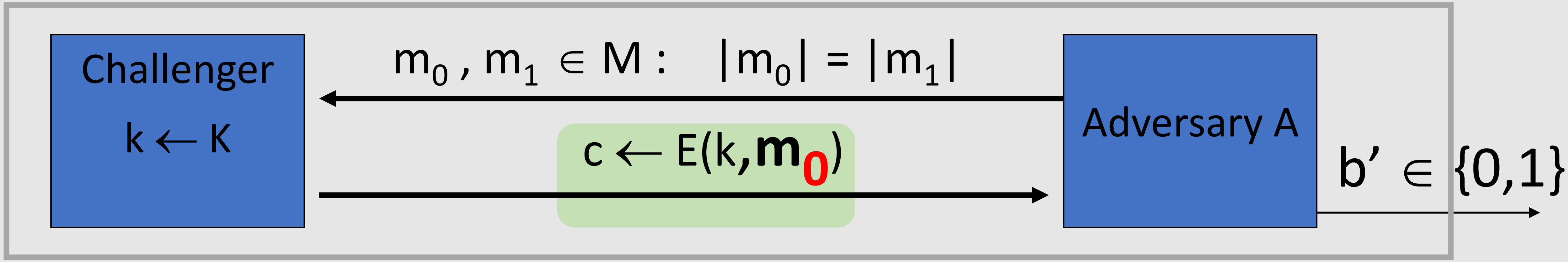
Cipher text with **ECB**



Cipher text with
other modes of operation

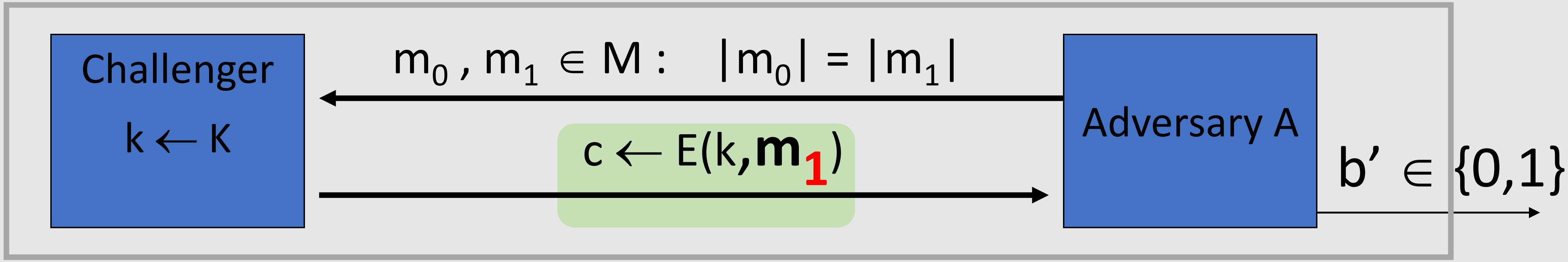
Semantic Security (one-time key)

EXP(0):



one time key $\Rightarrow$ adversary sees only one ciphertext

EXP(1):



$\mathrm{Adv}_{SS}[A,\mathrm{Cipher}] = \left| \Pr[\ \mathbf{EXP(0)}=1\ ] - \Pr[\ \mathbf{EXP(1)}=1\ ] \right|$ should be “negligible” for all “efficient” A

ECB is not Semantically Secure

ECB is not semantically secure for messages that contain **more than one block.** (known-plaintext attack)

$b \in \{0,1\}$

Challenger

$k \leftarrow K$

Two blocks

m_0 = `"Hello World"`

m_1 = `"Hello Hello"`

Adversary A

$c = (c_1, c_2) \leftarrow E(k, \mathbf{m_b})$

If $c_1 = c_2$ output 1, else output 0

Then $Adv_{SS}[A, ECB] =$

Deterministic Counter Mode (Secure Construction)

- **PRF** $F : K \times \{0,1\}^n \rightarrow \{0,1\}^n$ (e.g., n=128 with AES)
- $\mathbf{E_{DETCTR}}$ **(k, m) =** (Encryption)

	m[0]	m[1]	…	m[L]
$\oplus$	**F(k,0)**	**F(k,1)**	…	**F(k,L)**
	c[0]	c[1]	…	c[L]

$\Rightarrow$ Stream cipher built from a PRF (e.g., AES, 3DES)

Deterministic Counter Mode (Secure Construction)

- **PRF** $F : K \times \{0,1\}^n \to \{0,1\}^n$ (e.g., n=128 with AES)
- **D_{DETCTR} (k, c) =** (Decryption)

c[0]	c[1]	…	c[L]

$\oplus$

F(k,0)	**F(k,1)**	…	**F(k,L)**

m[0]	m[1]	…	m[L]

No need to **invert** F when decrypting

Deterministic Counter Mode Security

Theorem: For any L>0,

If **F** is a **secure PRF** over (K,X,X) then

DETCTR is **semantically secure** over (K,X^L,X^L).

In particular, for every efficient adversary **A attacking DETCTR**

there exists an efficient adversary **B attacking F** s.t.:

$$\mathrm{Adv}_{SS}[A, \mathrm{DETCTR}] = 2 \cdot \mathrm{Adv}_{PRF}[B, F]$$

$\mathrm{Adv}_{PRF}[B, F]$ is negligible (since F is a secure PRF)

Hence, $\mathrm{Adv}_{SS}[A, \mathrm{DETCTR}]$ must be negligible.

Modes of Operation
Many-Time Key

Examples:

- File systems: Same AES key used to encrypt many files.
- IPsec: Same AES key used to encrypt many packets.

Semantic Security for Many-Time Key

Key used **more than once** ⇒ adversary sees many CTs with same key (i.e., used for **multiple messages**)

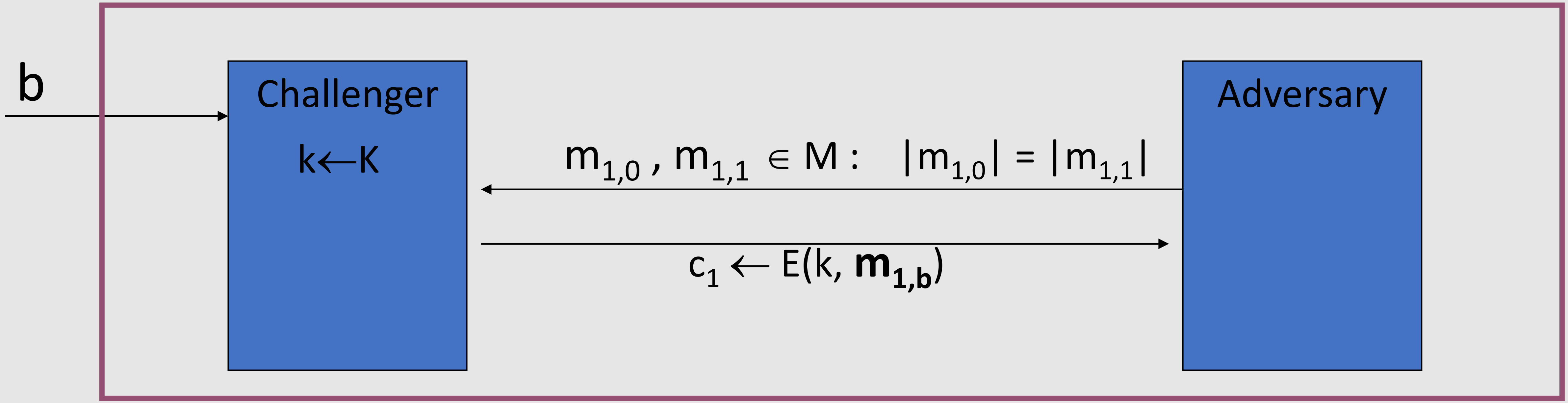
Adversary's power: **Chosen-Plaintext Attack (CPA)**

- Adversary can obtain the encryption of arbitrary messages of his choice (conservative modeling of real life)

Adversary's goal: Break semantic security

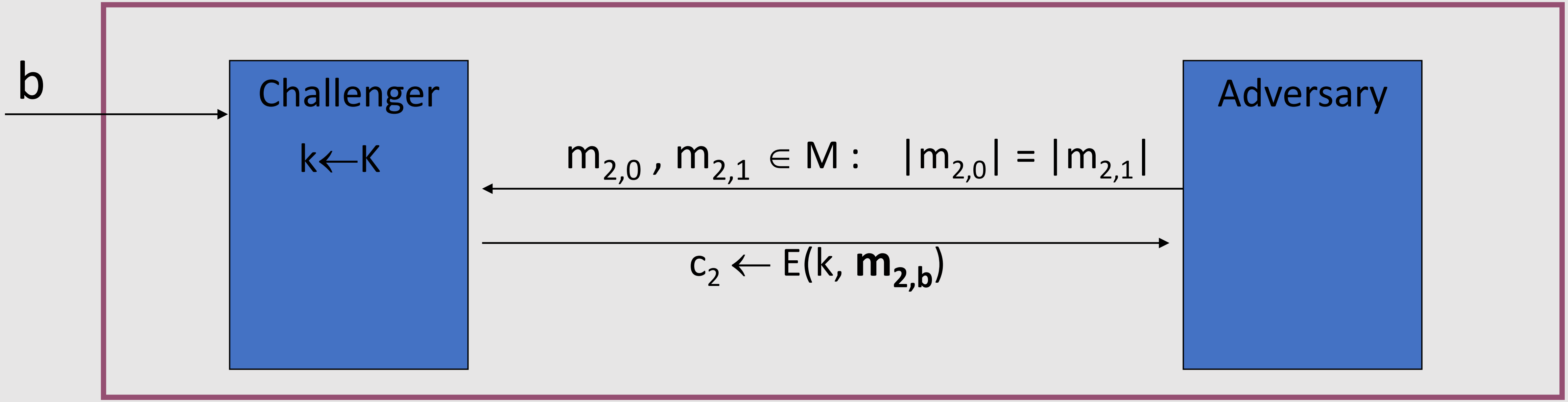
Semantic Security for Many-Time Key (CPA Security)

Q = (E,D) a cipher defined over (K,M,C). For b=0,1 define EXP(b) as:



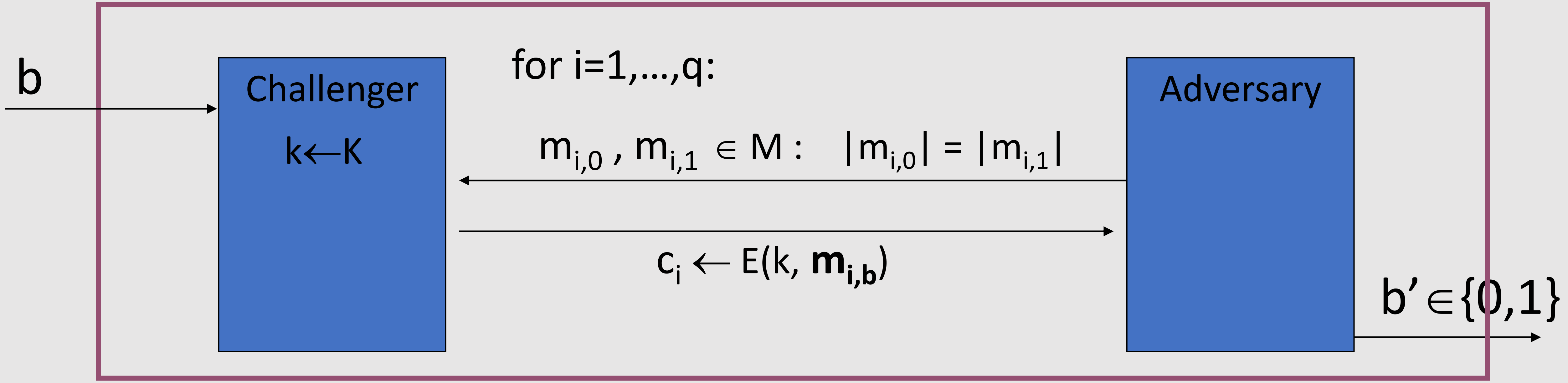
Semantic Security for Many-Time Key (CPA Security)

Q = (E,D) a cipher defined over (K,M,C). For b=0,1 define EXP(b) as:



Semantic Security for Many-Time Key (CPA Security)

$Q = (E,D)$ a cipher defined over (K,M,C). For $b=0,1$ define $EXP(b)$ as:



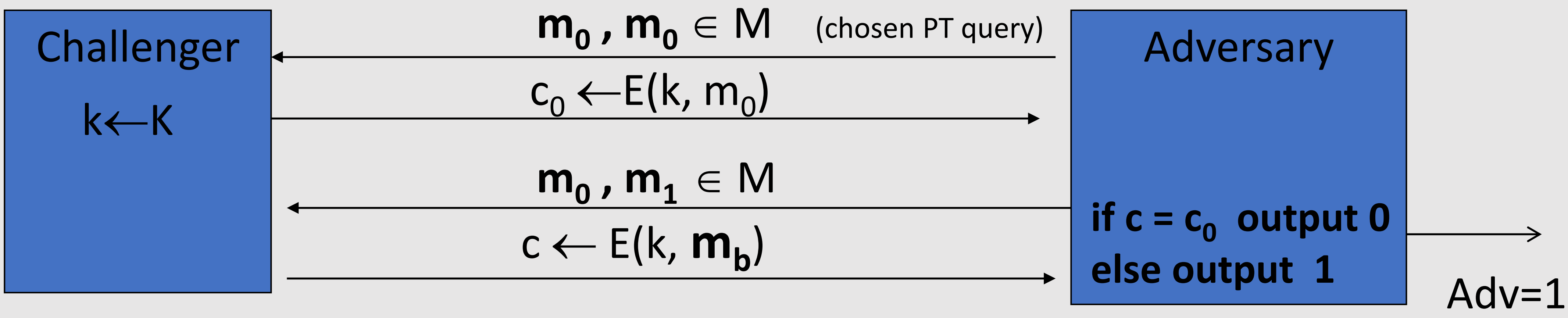
CPA ⇒ if adversary wants $c = E(k, m)$ it queries with $m_{j,0} = m_{j,1} = m$

Definition: Q is **semantically secure under CPA** if for all "efficient" adversary A:

$$\mathbf{Adv_{CPA}\,[A,Q] = \left|\, Pr[EXP(0)=1] - Pr[EXP(1)=1] \,\right|}$$ is "negligible".

Ciphers Insecure under CPA

Suppose E(k,m) **always outputs same ciphertext for msg m and key k**. Then:

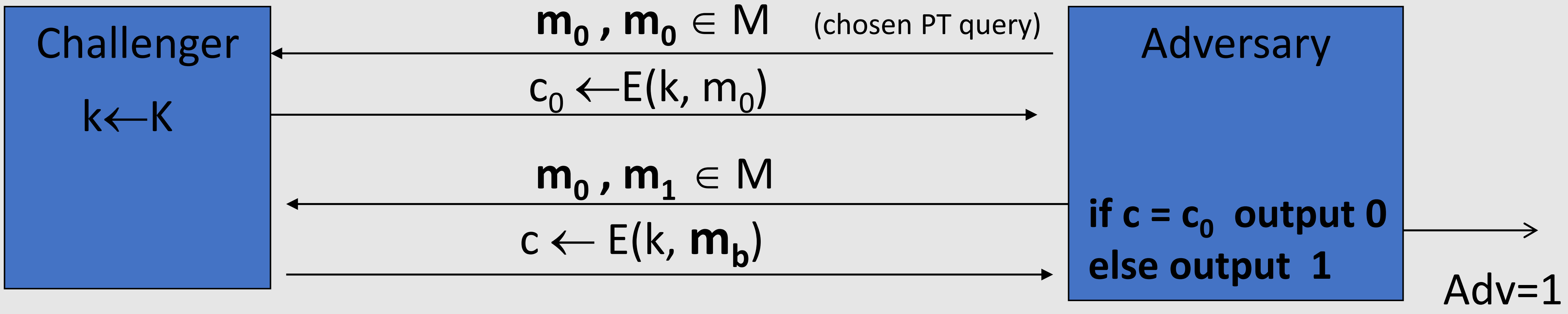


So what? an attacker can learn that two encrypted files are the same, two encrypted packets are the same, etc.

- Leads to significant attacks when the message space M is small

Ciphers Insecure under CPA

Suppose E(k,m) **always outputs same ciphertext for msg m and key k**. Then:

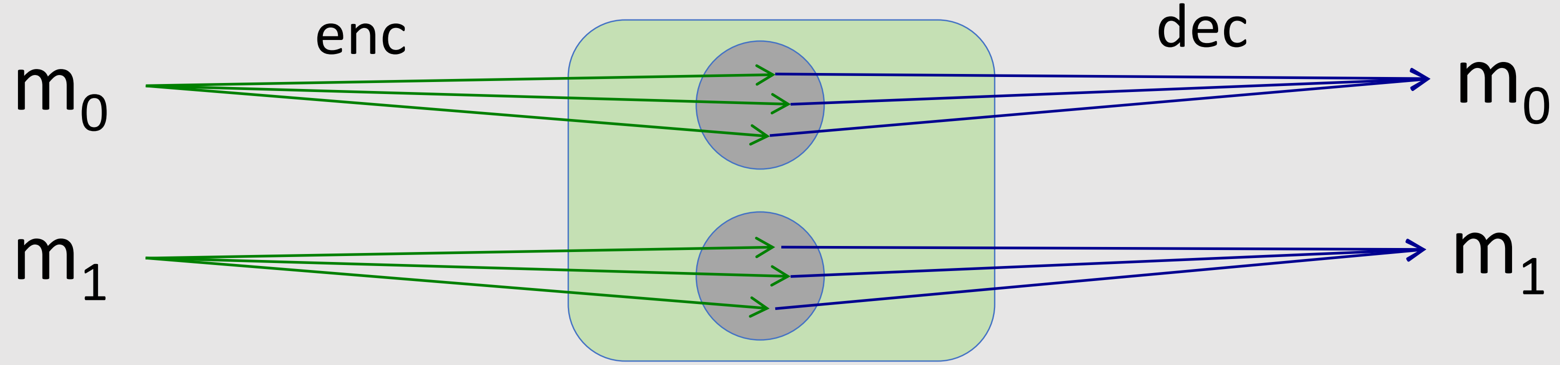


If secret key is to be used multiple times $\Rightarrow$

given **the same plaintext message twice**,
encryption must produce different outputs.

Solution 1: Randomized Encryption

- E(k,m) is a randomized algorithm:

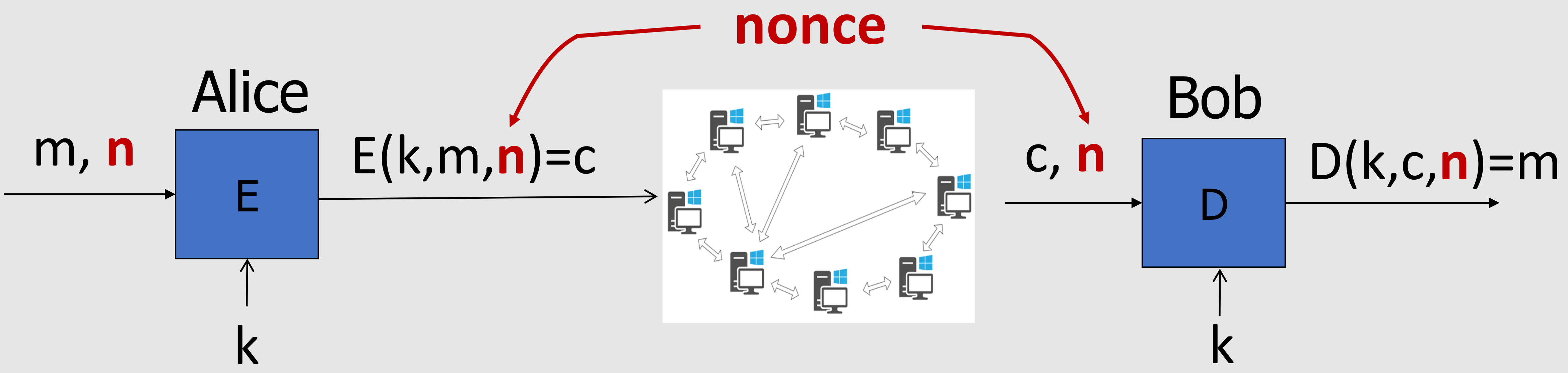


⇒ encrypting same msg twice gives different ciphertexts (w.h.p.)

⇒ ciphertext must be longer than plaintext

Roughly speaking: CT-size = PT-size + “# random bits”

Solution 2: Nonce-based Encryption



Nonce n:

- a value that changes from msg to msg
- (k,n) pair **<u>never</u> used more than once**
- n does **not** need to be **secret** and does **not** need to be **random**

Solution 2: Nonce-based Encryption

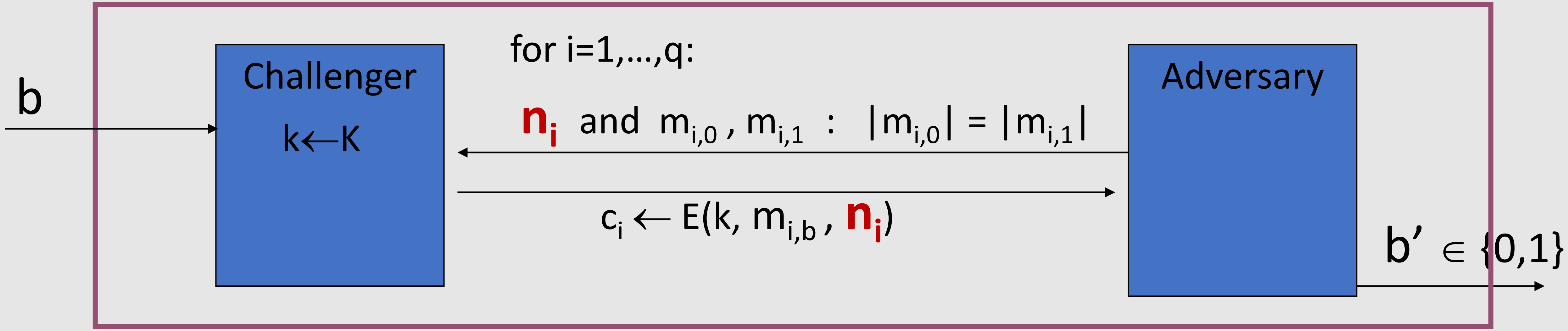
Nonce

- **Method 1:** nonce is a **counter** (e.g., packet counter)
  - used when encryptor keeps state from msg to msg
  - if decryptor has same state, need not send nonce with CT

- **Method 2:** encryptor chooses a **random nonce**, $n \leftarrow \mathcal{N}$
  (It's like randomized encryption)
  (ex. Multiple devices encrypting with the same key)
  - $\mathcal{N}$ must be large enough to ensure that the same nonce is not chosen twice with high probability

CPA Security for Nonce-based Encryption

System should be secure when **nonces are chosen adversarially.**



All nonces $\{n_1, \dots, n_q\}$ must be distinct.

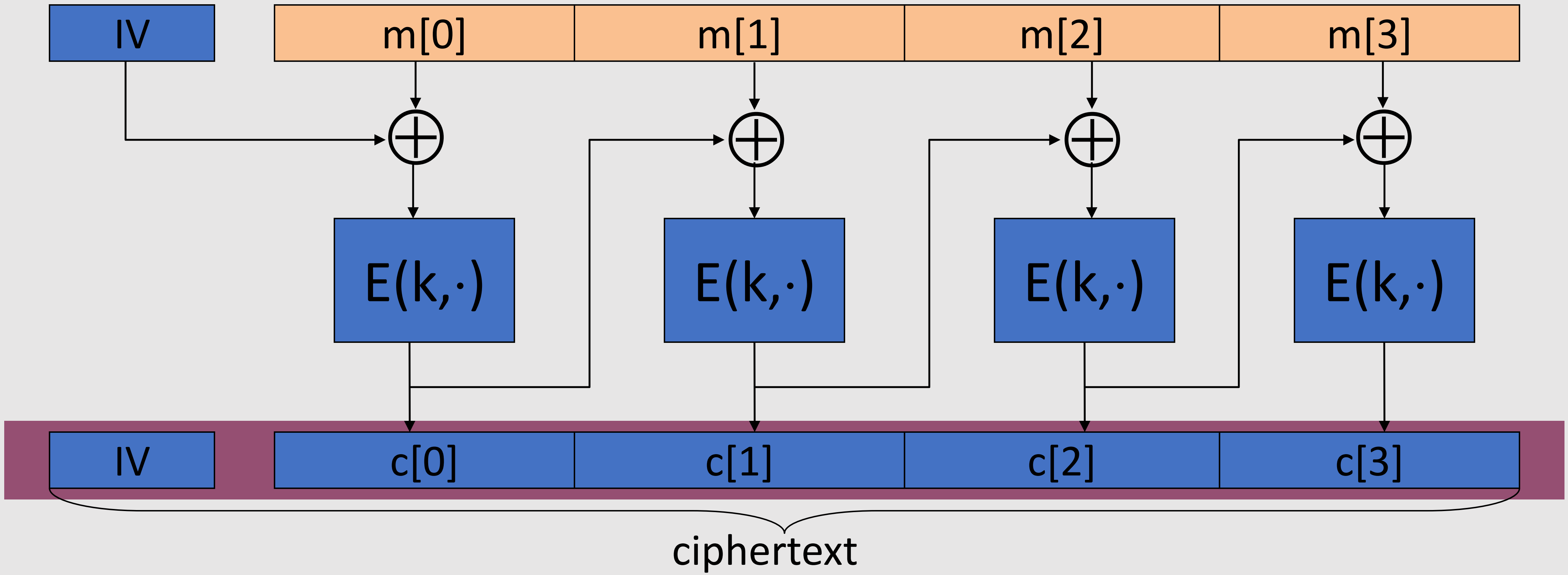
Definition. Nonce-based **Q** is **semantically secure under CPA** if for all "efficient" adversary A:

$$\mathbf{Adv_{nCPA}[A,Q] = |Pr[EXP(0)=1] - Pr[EXP(1)=1]|} \text{ is "negligible".}$$

Many-time Key Mode of Operation:
Cipher Block Chaining (CBC)

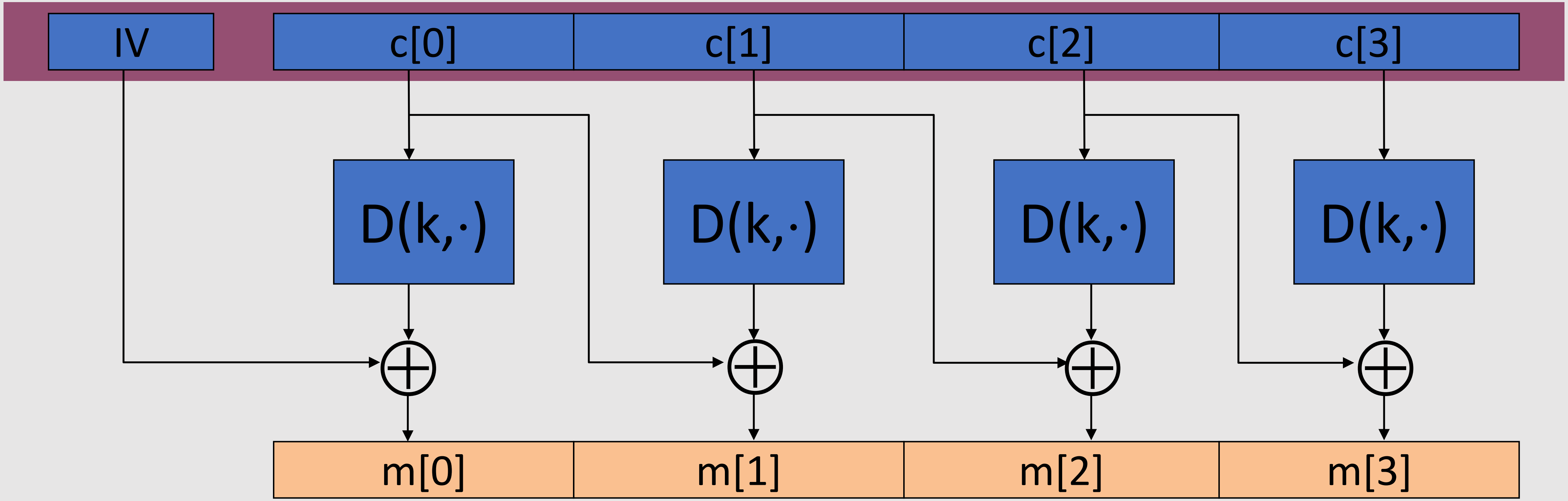
Construction 1: CBC with random IV

- **PRP** $E : K \times \{0,1\}^n \to \{0,1\}^n$
- (Encryption) **$E_{CBC}(k,m)$:** choose **random** $IV \in \{0,1\}^n$ and do:



Construction 1: CBC with random IV

- $D : K \times \{0,1\}^n \to \{0,1\}^n$ **inversion algorithm** of E
- (Decryption) $\mathbf{D_{CBC}(k,c)}$:



(Randomized) CBC Security

Theorem: For any $L>0$ (length of the message we are encrypting),
If **E** is a **secure PRP** over (K,X) then
CBC is **semantically secure under CPA** over (K, X^L, X^{L+1}).

In particular, for every efficient q-query adversary **A** **attacking CBC** there exists an efficient PRP adversary **B** **attacking E** s.t.

$$\mathrm{Adv}_{\mathrm{CPA}}[A, \mathrm{CBC}] \leq 2 \cdot \mathrm{Adv}_{\mathrm{PRP}}[B, E] + \mathbf{2\, q^2\, L^2 / |X|}$$

Note: CBC is only secure as long as $q^2 L^2 \ll |X|$

(the error term should be negligible)

An example

$$\mathrm{Adv}_{\mathrm{CPA}}\,[A, \mathrm{CBC}] \leq 2\cdot \mathrm{Adv}_{\mathrm{PRP}}[B, E] + \mathbf{2\, q^2 L^2 / |X|}$$

q = # messages encrypted with k , L = length of max message

Suppose we want $\mathrm{Adv}_{\mathrm{CPA}}\,[A, \mathrm{CBC}] \leq 1/2^{32}$ $\Leftarrow$ $q^2 L^2 / |X| < 1/ 2^{32}$

- AES: $|X| = 2^{128}$ $\Rightarrow$ $q\,L < 2^{48}$

  So, after 2^{48} AES blocks, must change key

- 3DES: $|X| = 2^{64}$ $\Rightarrow$ $q\,L < 2^{16}$

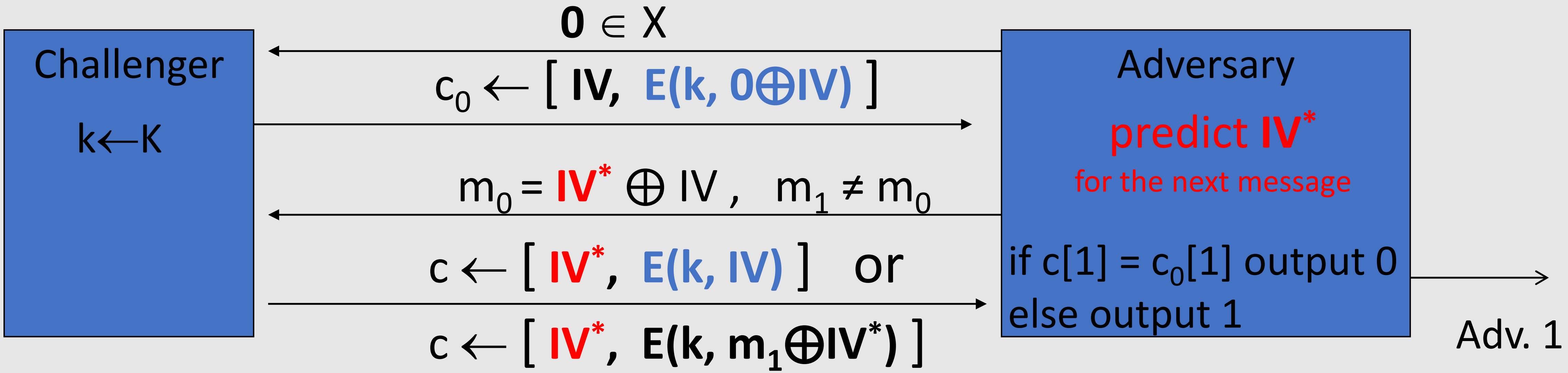
  So, after 2^{16} DES blocks, must change key

  $\Rightarrow$ after 2^{16} blocks (each of 8 bytes) need to change key $\Rightarrow$ $2^{16} \times 8 = ½$ MB !!!

Warning: an attack on CBC with rand. IV

CBC where adversary can **predict** the IV is not CPA-secure !!

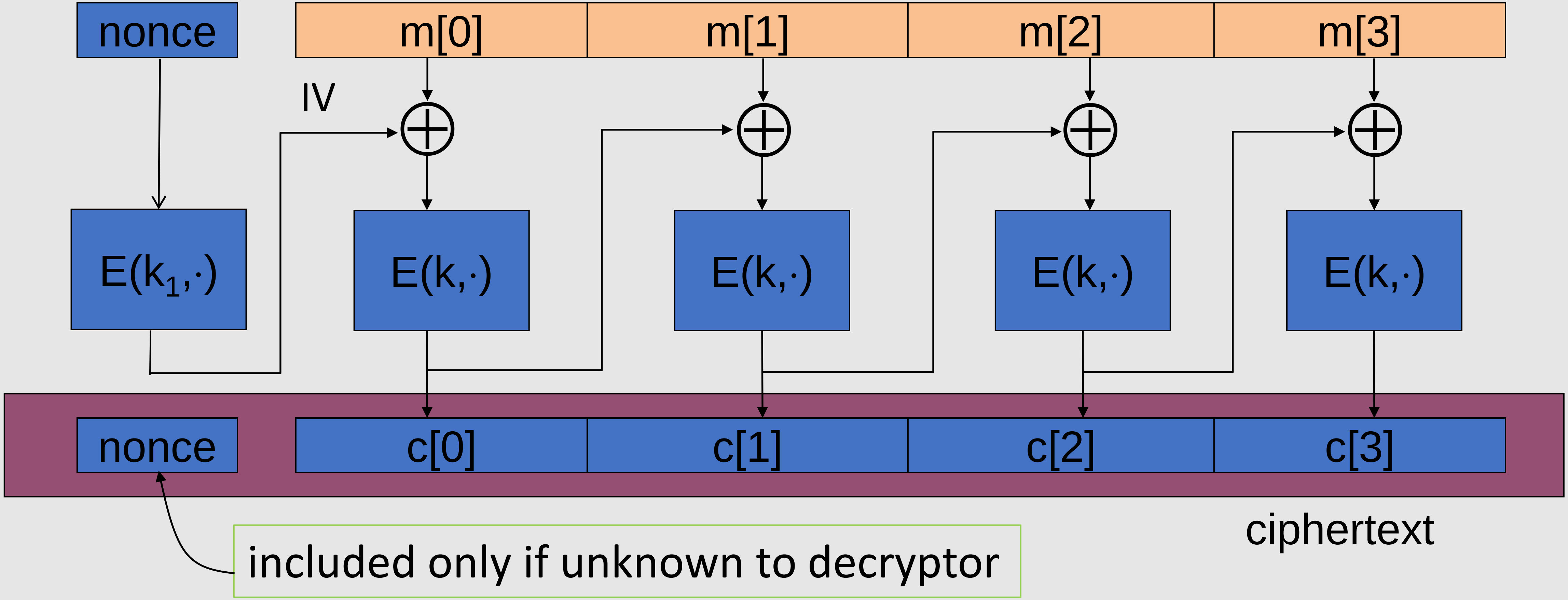
Suppose given $c \longleftarrow E_{CBC}(k,m)$ adversary can predict IV for next message



Bug in SSL/TLS 1.0: IV for record #i is last CT block of record #(i-1)

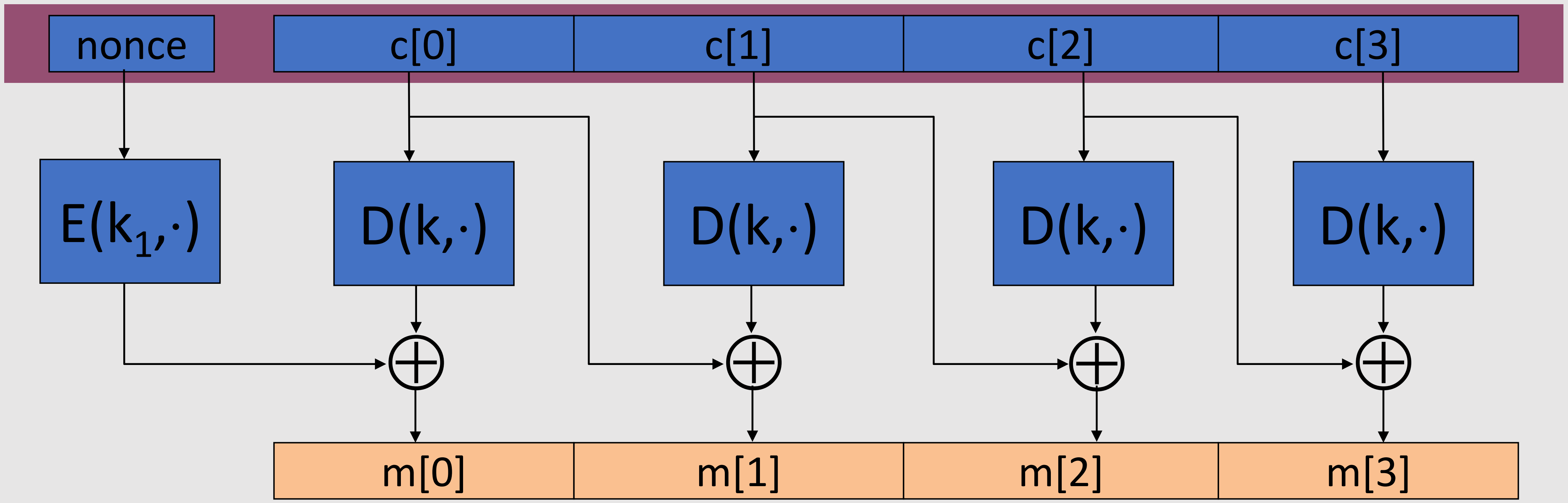
Construction 2: Nonce-based CBC

- key = (k, k_1)
- (key, nonce) pair is used for only one message
- **Encryption:**



Construction 2: Nonce-based CBC

- **Decryption:**

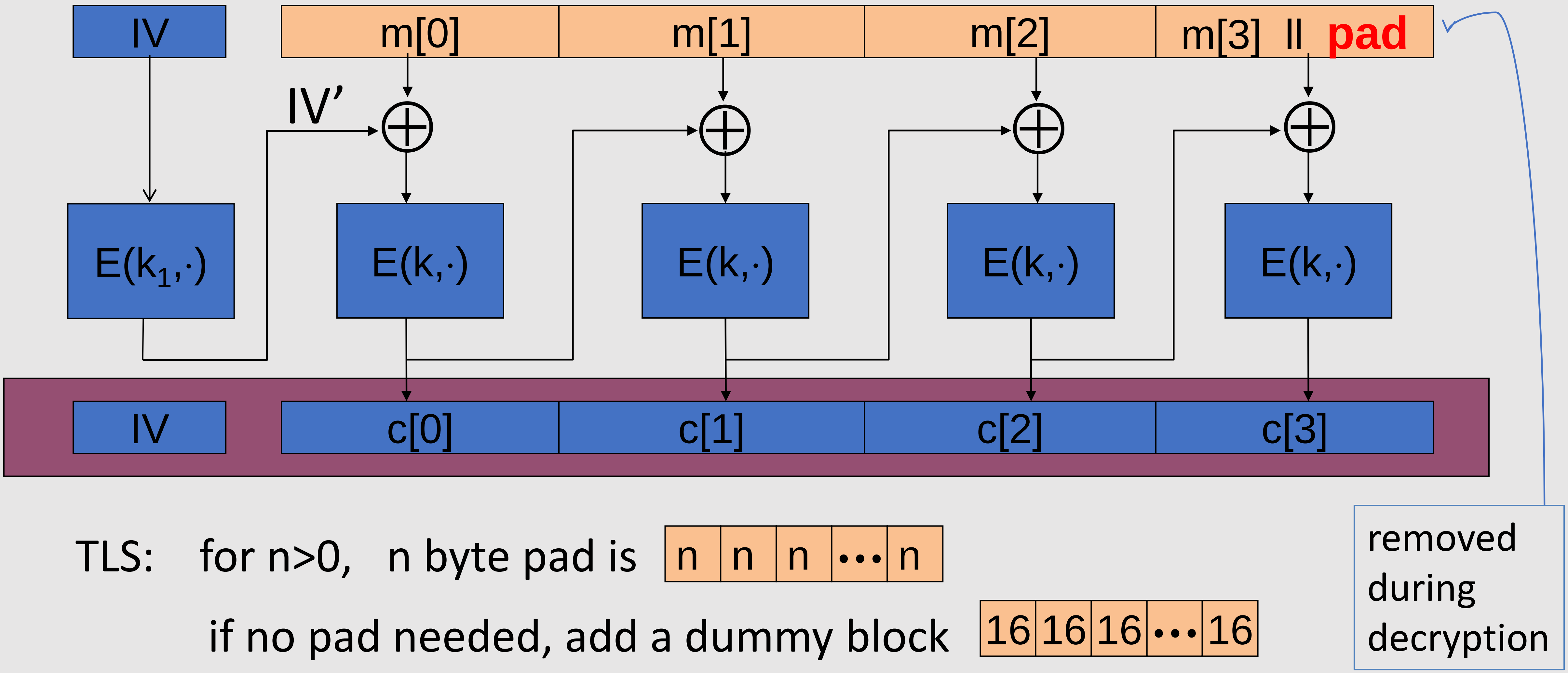


An example Crypto API (OpenSSL)

```
void AES_cbc_encrypt(
        const unsigned char *in,
        unsigned char *out,
        size_t length,
        const AES_KEY *key,
        unsigned char *ivec,          ⟵  user supplies IV
        AES_ENCRYPT or AES_DECRYPT);
```

When it is non-random need to encrypt it before use
(Otherwise, no CPA security!!)

A CBC technicality: padding



TLS: for n>0, n byte pad is | n | n | n | ••• | n |

if no pad needed, add a dummy block | 16 | 16 | 16 | ••• | 16 |

Key Exchange

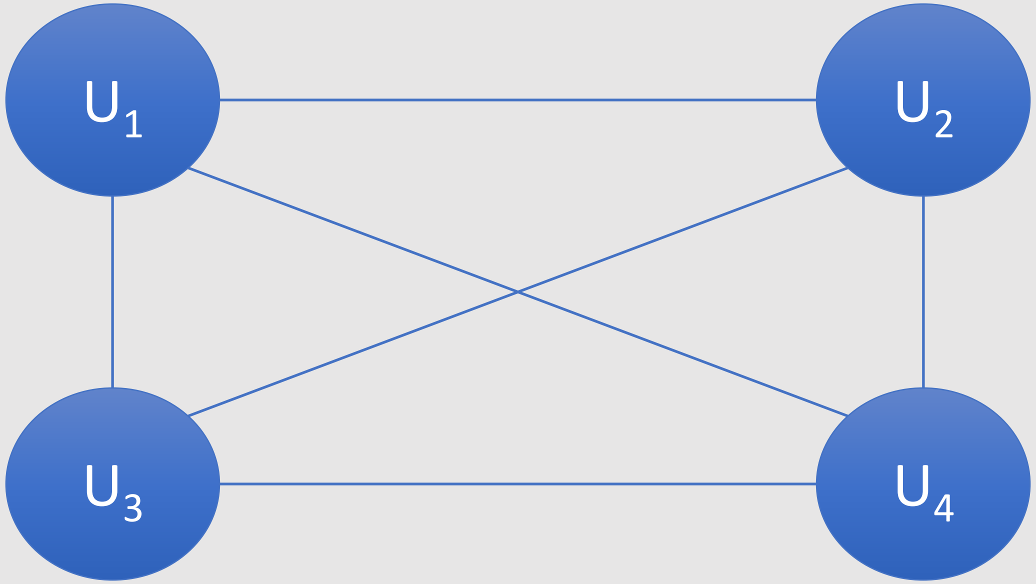
Outline

- Trusted 3rd Parties
- Merkle Puzzles
- The Diffie-Hellman Protocol

Trusted 3rd Parties

Key Management

Problem: **n** users. Storing mutual secret keys is difficult

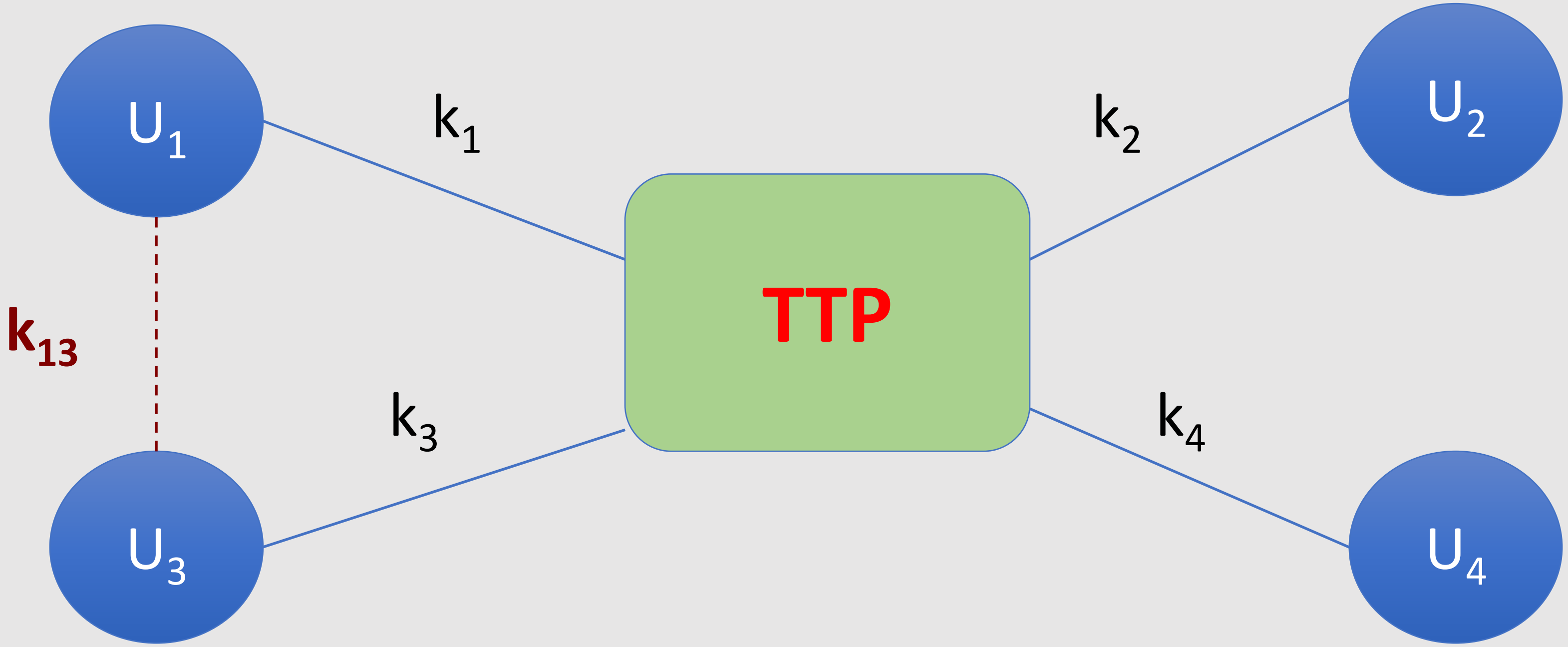


O(n) keys per user

O(n^2) keys in total

A Better Solution

Online **Trusted 3rd Party** (TTP)



Every user only remembers **ONE key**

Generating keys: A toy protocol

Alice wants a shared key with Bob. **Eavesdropping** security **only**.

<u>Bob</u> **(k_B)**

<u>Alice</u> **(k_A)**

<u>TTP</u>

"Alice wants key with Bob" $\rightarrow$

choose random k_{AB}

$E(k_A, \text{"Alice, Bob"} \parallel k_{AB})$;

$\leftarrow$

ticket = $E(k_B, \text{"Alice, Bob"} \parallel k_{AB})$

$\leftarrow$ ticket

$\mathbf{k_{AB}}$

$\mathbf{k_{AB}}$

(E,D) a CPA-secure cipher

Generating keys: A toy protocol

Alice wants a shared key with Bob.   Eavesdropping security only.

Eavesdropper sees: $E(k_A,$ "A, B" $\|$ k_{AB})   ; $E(k_B,$ "A, B" $\|$ k_{AB})

(E,D) is CPA-secure $\Rightarrow$ eavesdropper learns nothing about k_{AB}

Note: **TTP needed for every key exchange, knows all session keys.**

(basis of Kerberos system)

Key Question

Can we generate shared keys **without** an **online** trusted 3rd party?

Answer: **yes!**

Starting point of public-key cryptography:

- Merkle (1974),
- Diffie-Hellman (1976),
- RSA (1977)
- …

Merkle Puzzles

Key exchange without an online TTP?

- Goal: Alice and Bob want a shared key, unknown to eavesdropper
- Security against **eavesdropping only** (**no tampering**)

- **Can this be done using generic symmetric crypto?**

Merkle Puzzles (1974)

Answer: yes, but **very inefficient**

Main tool: **"puzzles"**

- Puzzles: Problems that can be solved with "some effort"
- Example:
  - $E(k,m)$ a symmetric cipher with $k \in \{0,1\}^{128}$
  - **puzzle = E(P, "message")** where $\mathbf{P = 0^{96} \,\|\, b_1 \ldots b_{32}}$
  - To "solve" a puzzle, find **P** by trying all $\mathbf{2^{32}}$ possibilities

Merkle Puzzles

Alice:

- Prepare **2^{32} puzzles:**
  - For i = 1, …, 2^{32} choose random $P_i \in \{0,1\}^{32}$ and random x_i, $k_i \in \{0,1\}^{128}$ $x_i \neq x_j$

    Set **puzzle$_i$** $\longleftarrow$ **E(0^{96} ll P_i, "Puzzle # " ll x_i ll k_i)**
  - Send **puzzle$_1$, … , puzzle$_{2^{32}}$** to Bob.

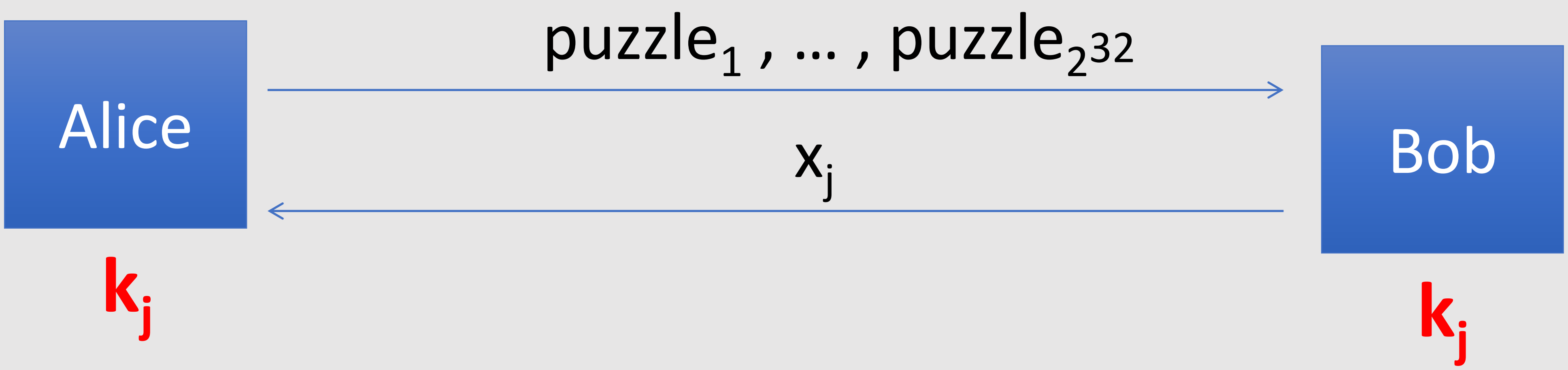
Bob:

- Choose a **random puzzle$_j$** and **solve** it. Obtain **(x_j, k_j)** and use **k_j** as shared secret.
- Send **x_j** to Alice.

Alice:

- Lookup puzzle with number **x_j** .
- Use **k_j** as shared secret.

In a figure



Alice's work: **O(2^{32})** (prepare 2^{32} puzzles) in general **O(n)**

Bob's work: **O(2^{32})** (solve **one** puzzle) in general **O(n)**

Eavesdropper's work: **O(2^{64})** (solve **2^{32}** puzzles) in general **O(n^2)**

Impossibility Result

Can we achieve a better gap using a general symmetric cipher?

Answer: **unknown**

The Diffie-Hellman Protocol

Key exchange without an online TTP?

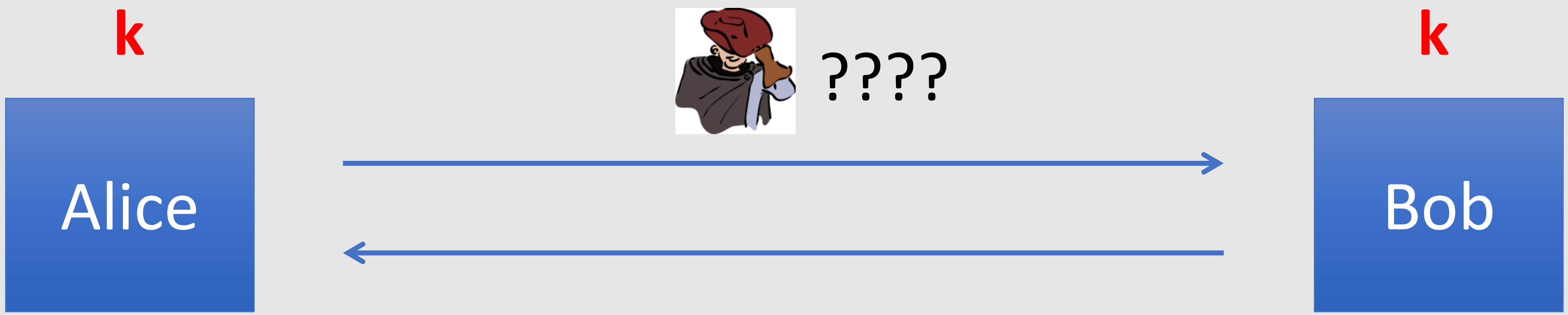
- Goal: Alice and Bob want a shared key, unknown to eavesdropper
- Security against **eavesdropping only** (**no tampering**)

- **Can this be done with an exponential gap?**

The Diffie-Hellman Protocol

High-level idea:

- Alice and Bob **do NOT share any secret information beforehand**
- Alice and Bob exchange messages
- After that, Alice and Bob have agreed on a shared secret key **k**
- **k** unknown to eavesdropper



The Diffie-Hellman Protocol

(Security) Based on the **Discrete Logarithm** Problem:

Given

- **g**
- **p**
- **$g^k \bmod p$**

Find k

The Diffie-Hellman Protocol

Fix a large prime **p** (e.g., 600 digits)

Fix an integer **g** in {2, …, p-2}

Alice | **Bob**

Choose random **a** in {1,…,p-2} | Choose random **b** in {1,…,p-2}

$\mathbf{g^a \ (mod\ p)}$ →

← $\mathbf{g^b \ (mod\ p)}$

Alice computes | Bob computes

$$(g^b)^a \pmod p \quad = \quad g^{ab} \pmod p \quad = \quad (g^a)^b \pmod p$$

SECRET KEY

Security

Eavesdropper sees: **p**, **g**, $\mathbf{g^a}$ **(mod p)**, and $\mathbf{g^b}$ **(mod p)**

Can she compute $\mathbf{g^{ab}}$ **(mod p)** ??

How hard is the DH function mod p?

Suppose prime **p** is **n** bits long.

Best known algorithm (GNFS): run time $\exp(\tilde{O}(\sqrt[3]{n}))$

Insecure against man-in-the-middle

As described, the protocol is insecure against **active** attacks

Alice | **MiTM** | **Bob**

$A = g^a \pmod p$ → MiTM (a') → $A' = g^{a'} \pmod p$ → Bob

Alice ← $B' = g^{b'} \pmod p$ ← MiTM (b') ← $B = g^b \pmod p$

$g^{ab'} \pmod p$ | $g^{ab'}, g^{a'b} \pmod p$ | $g^{a'b} \pmod p$

Introduction Number Theory

Background

We will use a bit of number theory to construct:

- Key exchange protocols
- Digital signatures
- Public-key encryption

Notation

From here on:

- N denotes a positive integer.
- p denote a prime.

Notation: $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$

Can do addition and multiplication modulo N

Modular arithmetic

Examples: let $N = 12$

$9 + 8 = 5$ in $\mathbb{Z}_{12}$

$5 \times 7 = \square$ in $\mathbb{Z}_{12}$

$5 - 7 = \square$ in $\mathbb{Z}_{12}$

Arithmetic in $\mathbb{Z}_N$ works as you expect, e.g $x \cdot (y+z) = x \cdot y + x \cdot z$ in $\mathbb{Z}_N$

Modular arithmetic

Examples: let $N = 12$

$$9 + 8 = 5 \quad \text{in } \mathbb{Z}_{12}$$

$$5 \times 7 = 11 \quad \text{in } \mathbb{Z}_{12}$$

$$5 - 7 = 10 \quad \text{in } \mathbb{Z}_{12}$$

Arithmetic in $\mathbb{Z}_N$ works as you expect, e.g $x \cdot (y+z) = x \cdot y + x \cdot z$ in $\mathbb{Z}_N$

Greatest common divisor

Def: For ints. x,y: **gcd(x, y)** is the greatest common divisor of x,y

Example: gcd(12, 18) = 6

Fact: for all ints. x,y there exist ints. a,b such that

$$a \cdot x + b \cdot y = \gcd(x,y)$$

a,b can be found efficiently using the extended Euclid alg.

If gcd(x,y)=1 we say that x and y are **relatively prime**

Example: 2 x 12 -1 x 18 = 6

Modular inversion

Over the rationals, inverse w.r.t. the moltiplication of 2 is ½ .
What about $\mathbb{Z}_N$?

<u>Def</u>: The **inverse** of x in $\mathbb{Z}_N$ is an element y in $\mathbb{Z}_N$ s.t. $x \cdot y = 1$

y is denoted x^{-1} .

Example: let N be an odd integer.

The inverse of 2 in $\mathbb{Z}_N$ is $\frac{N+1}{2}$ since $2 \cdot \frac{N+1}{2} = N + 1 = 1$

Modular inversion

Which elements have an inverse in $\mathbb{Z}_N$?

<u>Lemma</u>: x in $\mathbb{Z}_N$ has an inverse if and only if gcd(x,N) = 1

Proof:

gcd(x,N)=1 $\Rightarrow$ $\exists$ a,b: $a \cdot x + b \cdot N = 1 \Rightarrow a \cdot x = 1$ in $\mathbb{Z}_N$

$\Rightarrow x^{-1} = a$ in $\mathbb{Z}_N$

gcd(x,N) > 1 $\Rightarrow$ $\forall$a: gcd($a \cdot x$, N) > 1 $\Rightarrow$ $a \cdot x \neq 1$ in $\mathbb{Z}_N$

More notation

<u>Def:</u> $\mathbb{Z}_N^*$ = (set of invertible elements in $\mathbb{Z}_N$) =

= { x∈ $\mathbb{Z}_N$: gcd(x,N) = 1 }

Examples:

1. for prime p, $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\} = \{1, 2, \ldots, p-1\}$
2. $\mathbb{Z}_{12}^*$ =

For x in $\mathbb{Z}_N^*$, can find x^{-1} using extended Euclid algorithm.

More notation

Def: $\mathbb{Z}_N^*$ = (set of invertible elements in $\mathbb{Z}_N$) =

= { x∈ $\mathbb{Z}_N$: gcd(x,N) = 1 }

Examples:

1. for prime p, $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\} = \{1, 2, \ldots, p-1\}$
2. $\mathbb{Z}_{12}^*$ = { 1, 5, 7, 11}

For x in $\mathbb{Z}_N^*$, can find x^{-1} using extended Euclid algorithm.

Solving modular linear equations

Solve: $a \cdot x + b = 0$ in $\mathbb{Z}_N$

Solution: $x = -b \cdot a^{-1}$ in $\mathbb{Z}_N$

Find a^{-1} in $\mathbb{Z}_N$ using extended Euclid. Run time: $O(\log^2 N)$

What about modular quadratic equations?

next segments

Fermat's theorem (1640)

<u>Thm</u>: Let p be a prime

$$\forall\, x \in (Z_p)^* : \quad \color{red}{x^{p-1} = 1 \text{ in } Z_p}$$

Example: p=5. $3^4 = 81 = 1$ in Z_5

Example of application:

So: $x \in (Z_p)^* \Rightarrow x \cdot x^{p-2} = 1 \Rightarrow x^{-1} = x^{p-2}$ in Z_p

another way to compute inverses, but less efficient than Euclid

Application: generating random primes

Suppose we want to generate a large random prime

say, prime p of length 1024 bits (i.e. $p \approx 2^{1024}$)

Step 1: choose a random integer $p \in [\ 2^{1024}\ ,\ 2^{1025}-1\ ]$

Step 2: test if $2^{p-1} = 1$ in Z_p

If so, output p and stop. If not, goto step 1 .

Simple algorithm (not the best). **Pr[p not prime] < 2^{-60}**

The structure of $(Z_p)^*$

<u>Thm</u> (Euler): $(Z_p)^*$ is a **cyclic group**, that is

$\exists\, g \in (Z_p)^*$ such that $\{1, g, g^2, g^3, \ldots, g^{p-2}\} = (Z_p)^*$

g is called a **<u>generator</u>** of $(Z_p)^*$

Example: $p=7$. $\{1, 3, 3^2, 3^3, 3^4, 3^5\} = \{1, 3, 2, 6, 4, 5\} = (Z_7)^*$

Not every elem. is a generator: $\{1, 2, 2^2, 2^3, 2^4, 2^5\} = \{1, 2, 4\}$

Order

For $g \in (Z_p)^*$ the set $\{1, g, g^2, g^3, \dots\}$ is called

the **group generated by g**, denoted $<g>$

Def: the **order** of $g \in (Z_p)^*$ is the size of $<g>$

$$\mathbf{ord_p(g) \quad = \quad |<g>| \quad = \quad (\text{smallest } a>0 \text{ s.t. } g^a = 1 \text{ in } Z_p)}$$

Examples: $ord_7(3) = 6$; $ord_7(2) = 3$; $ord_7(1) = 1$

Thm (Lagrange): $\forall g \in (Z_p)^*$: **$ord_p(g)$** divides $p-1$

Euler's generalization of Fermat (1736)

Def: For an integer N define $\varphi(N) = |(Z_N)^*|$ (Euler's φ func.)

Examples: $\varphi(12) = |\{1,5,7,11\}| = 4$; $\varphi(p) = p-1$

For $N=p\cdot q$: $\varphi(N) = N-p-q+1 = (p-1)(q-1)$

Thm (Euler): $\forall x \in (Z_N)^*$: $x^{\varphi(N)} = 1$ in Z_N

Example: $5^{\varphi(12)} = 5^4 = 625 = 1$ in Z_{12}

Generalization of Fermat. Basis of the RSA cryptosystem

Modular e'th roots

We know how to solve modular **linear** equations:

$\mathbf{a \cdot x + b = 0}$ in Z_N Solution: $\mathbf{x = -b \cdot a^{-1}}$ in Z_N

What about higher degree polynomials?

Example: let p be a prime and $c \in Z_p$. Can we solve:

$$x^2 - c = 0 \quad , \quad y^3 - c = 0 \quad , \quad z^{37} - c = 0 \quad \text{in } Z_p$$

Modular e'th roots

Let p be a prime and $c \in Z_p$.

Def: $x \in Z_p$ s.t. $x^e = c$ in Z_p is called an **e'th root** of c .

$6^3 = 216 = 7$ in $\mathbb{Z}_{11}$

Examples: $7^{1/3} = 6$ in $\mathbb{Z}_{11}$

$3^{1/2} = 5$ in $\mathbb{Z}_{11}$

$1^{1/3} = 1$ in $\mathbb{Z}_{11}$

$2^{1/2}$ does not exist in $\mathbb{Z}_{11}$

The easy case

When does **$c^{1/e}$ in Z_p** exist? Can we compute it efficiently?

<u>The easy case</u>: suppose $\gcd(e, p-1) = 1$

Then for all c in $(Z_p)^*$: **$c^{1/e}$** exists in Z_p and is easy to find.

The case e=2: square roots

If p is an odd prime then gcd(2, p-1) ≠ 1

$$\begin{tikzcd} x \arrow[dr] & & -x \arrow[dl] \\ & x^2 & \end{tikzcd}$$

<u>Fact</u>: in $\mathbb{Z}_p^*$, $x \longrightarrow x^2$ is a 2-to-1 function

<u>Example</u>: in $\mathbb{Z}_{11}^*$:

$$\begin{tikzcd}[column sep=small] 1 \arrow[dr] & & 10 \arrow[dl] & 2 \arrow[dr] & & 9 \arrow[dl] & 3 \arrow[dr] & & 8 \arrow[dl] & 4 \arrow[dr] & & 7 \arrow[dl] & 5 \arrow[dr] & & 6 \arrow[dl] \\ & 1 & & & 4 & & & 9 & & & 5 & & & 3 & \end{tikzcd}$$

<u>Def</u>: x in $\mathbb{Z}_p$ is a **quadratic residue** (Q.R.) if it has a square root in $\mathbb{Z}_p$

p odd prime $\Rightarrow$ the # of Q.R. in $\mathbb{Z}_p$ is (p-1)/2 + 1

Euler's theorem

Thm: x in $(Z_p)^*$ is a Q.R. $\iff$ $x^{(p-1)/2} = 1$ in Z_p (p odd prime)

Example:

in $\mathbb{Z}_{11}$:		1^5,	2^5,	3^5,	4^5,	5^5,	6^5,	7^5,	8^5,	9^5,	10^5
	=	1	-1	1	1	1,	-1,	-1,	-1,	1,	-1

Note: $x \neq 0 \Rightarrow x^{(p-1)/2} = \left(x^{p-1}\right)^{1/2} = 1^{1/2} \in \{1, -1\}$ in Z_p

Def: $x^{(p-1)/2}$ is called the **Legendre Symbol** of x over p (1798)

Computing square roots mod p

Suppose $p = 3 \pmod 4$

<u>Lemma</u>: if $c \in (Z_p)^*$ is Q.R. then $\sqrt{c} = c^{(p+1)/4}$ in Z_p

Solving quadratic equations mod p

Solve: $a \cdot x^2 + b \cdot x + c = 0$ in Z_p

Solution: $x = \left(-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}\right) / 2a$ in Z_p

- Find $(2a)^{-1}$ in Z_p using extended Euclid.

- Find square root of $b^2 - 4 \cdot a \cdot c$ in Z_p (if one exists)

  using a square root algorithm

Computing e'th roots mod N ??

Let N be a composite number and e>1

When does $\mathbf{c^{1/e}}$ **in** $\mathbf{Z_N}$ exist? Can we compute it efficiently?

Answering these questions requires the factorization of N

(as far as we know)

Easy problems

- Given composite N and x in Z_N find x^{-1} in Z_N

- Given prime p and polynomial $f(x)$ in $Z_p[x]$

  find x in Z_p s.t. $f(x) = 0$ in Z_p (if one exists)

  Running time is linear in deg(f) .

… but many problems are difficult

Intractable problems with primes

Fix a prime p>2 and g in $(Z_p)^*$ of order q.

Consider the function: $x \mapsto g^x$ in Z_p

Now, consider the inverse function:

$\mathbf{Dlog_g(g^x) = x}$ **where x in {0, …, q-2}**

Example:

in $\mathbb{Z}_{11}$:	1,	2,	3,	4,	5,	6,	7,	8,	9,	10
$\mathrm{Dlog}_2(\cdot)$:	0,	1,	8,	2,	4,	9,	7,	3,	6,	5

Intractable problems with composites

Consider the set of integers: (e.g. for n=1024)

$$\mathbb{Z}_{(2)}(n) \;:=\; \{\, N = p\cdot q \;\text{ where } p,q \text{ are n-bit primes} \,\}$$

Problem 1: Factor a random N in $\mathbb{Z}_{(2)}(n)$ (e.g. for n=1024)

Problem 2: Given a polynomial **f(x)** where degree(f) > 1

and a random N in $\mathbb{Z}_{(2)}(n)$

find x in $\mathbb{Z}_N$ s.t. f(x) = 0 in $\mathbb{Z}_N$

The factoring problem

Gauss (1805): *"The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic."*

Best known alg. (NFS): run time exp($\tilde{O}(\sqrt[3]{n})$) for n-bit integer

Current world record: **RSA-768** (232 digits)

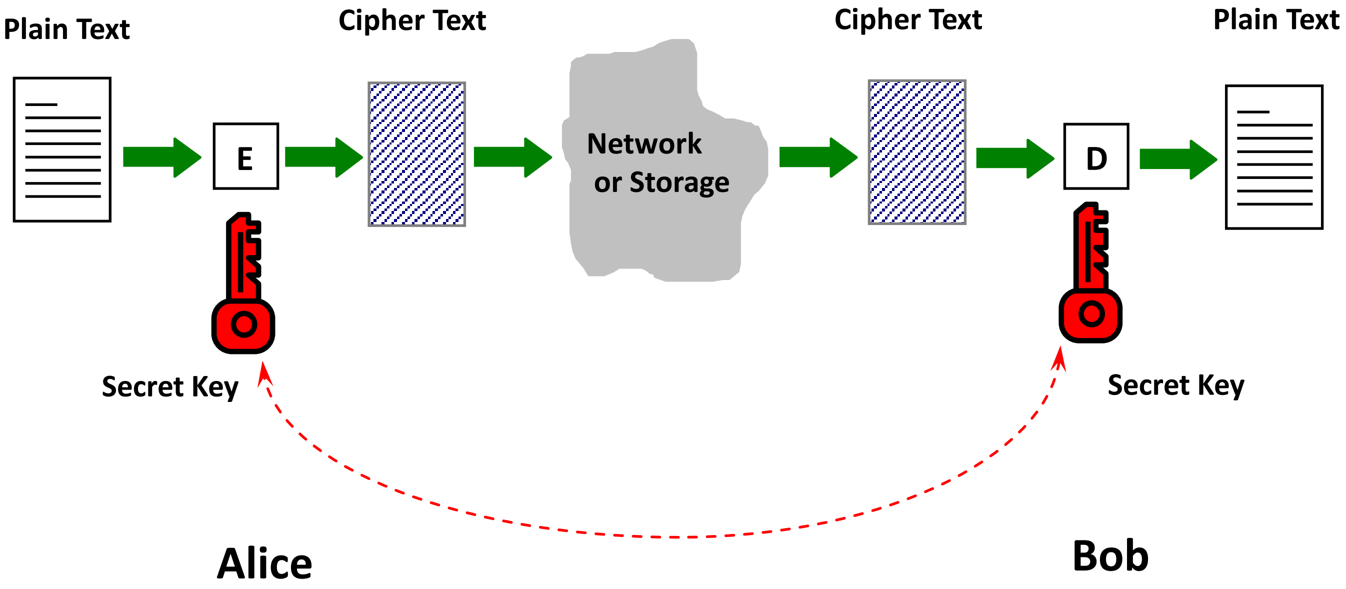
- Work: two years on hundreds of machines
- Factoring a 1024-bit integer: about 1000 times harder

$\Rightarrow$ likely possible this decade

Asymmetric Cryptography

Public key encryption:
definitions and security

Symmetric Cipher



Problems with Symmetric Ciphers

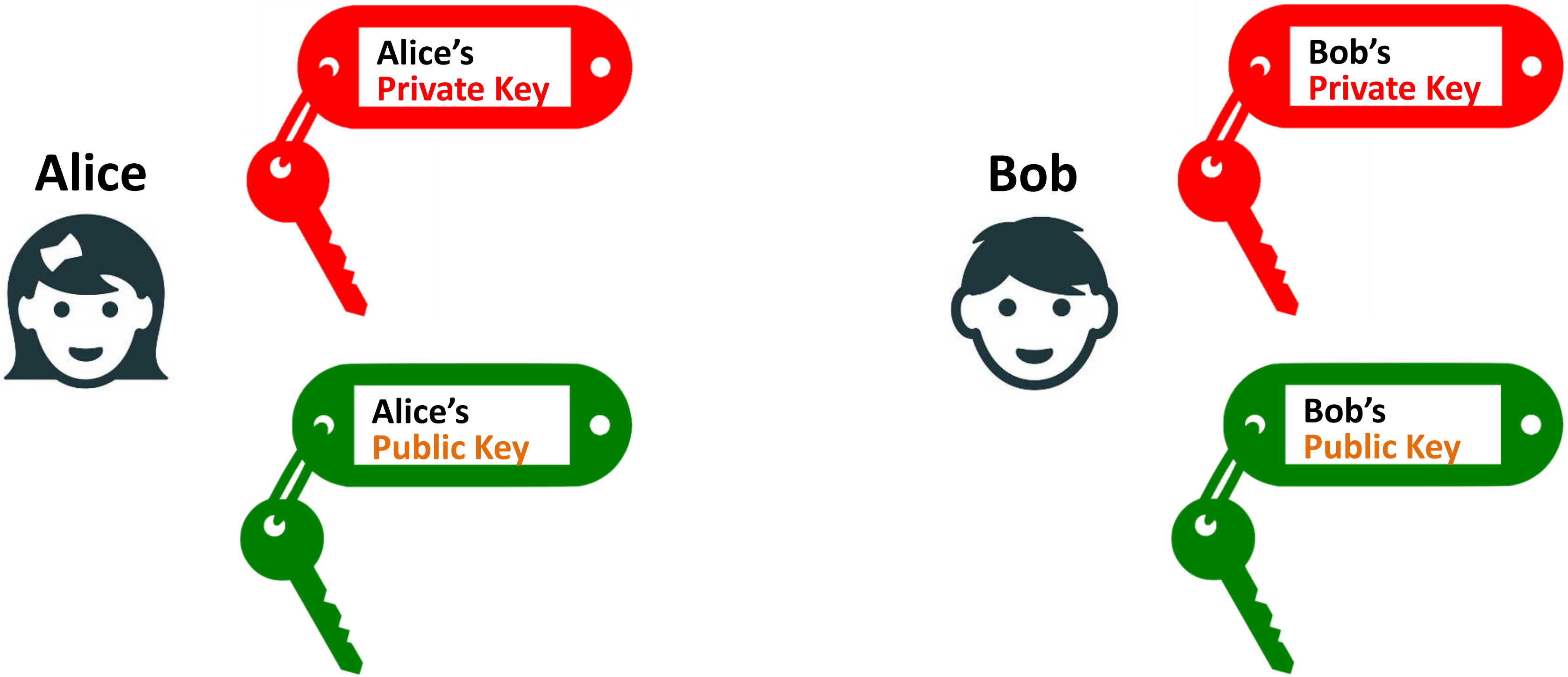
- In order for Alice & Bob to be able to communicate securely using a symmetric cipher, such as AES, they have to have a **shared key** in the first place.
  - What if they have never met before?
- Alice needs to keep 100 different keys if she wishes to communicate with 100 different people

Motivation of Asymmetric Cryptography

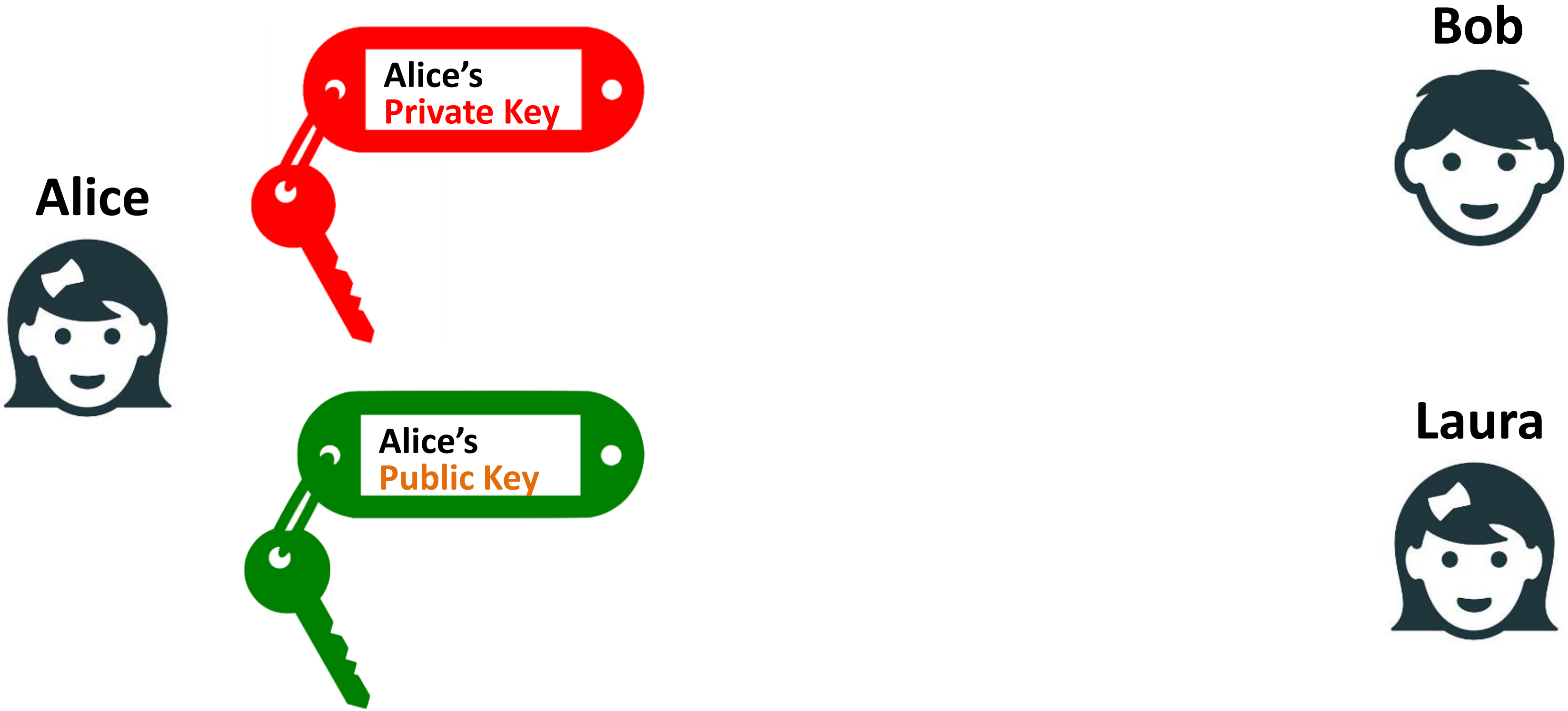
- Is it possible for Alice & Bob, who have no shared secret key, to communicate securely?

- This led to **Asymmetric Cryptography**

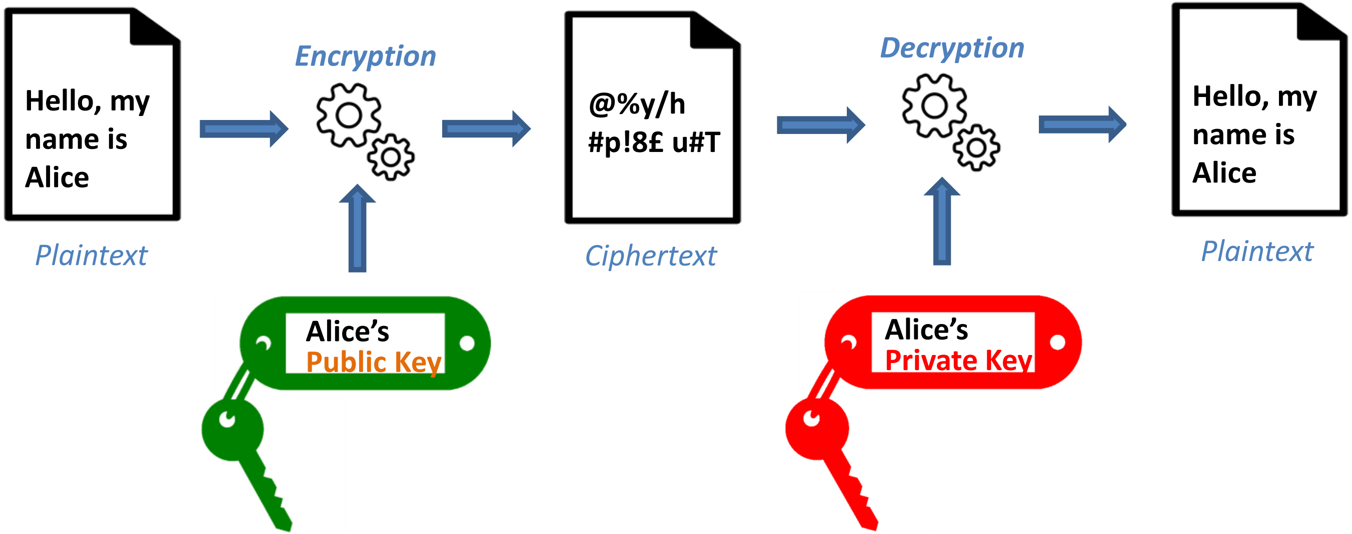
Asymmetric Cryptography



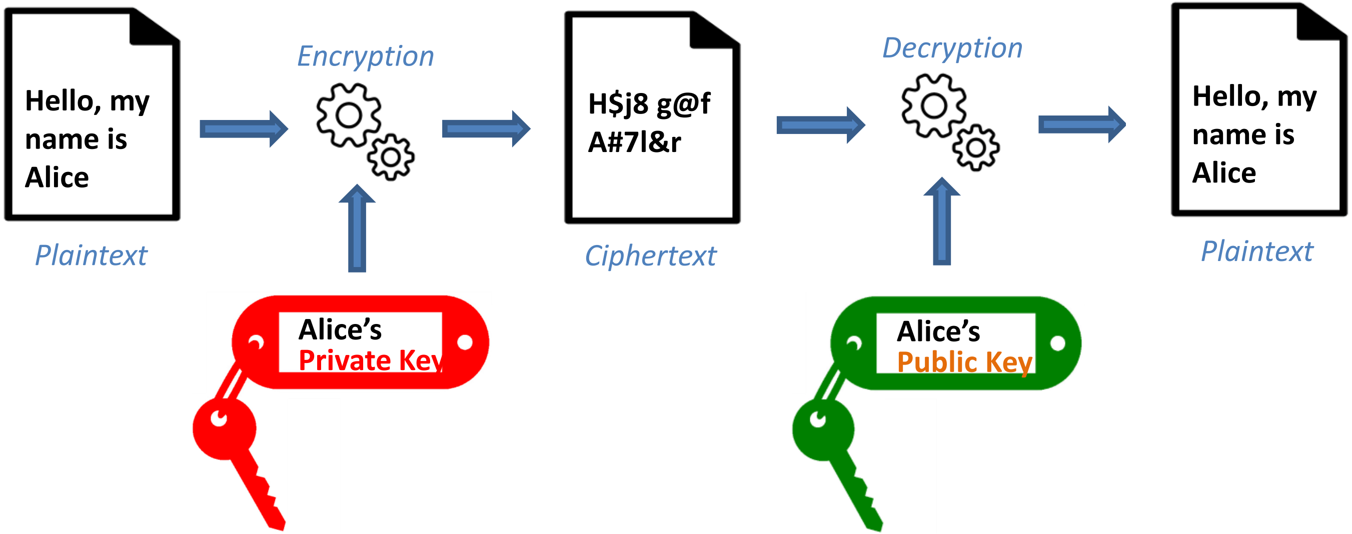
Asymmetric Cryptography



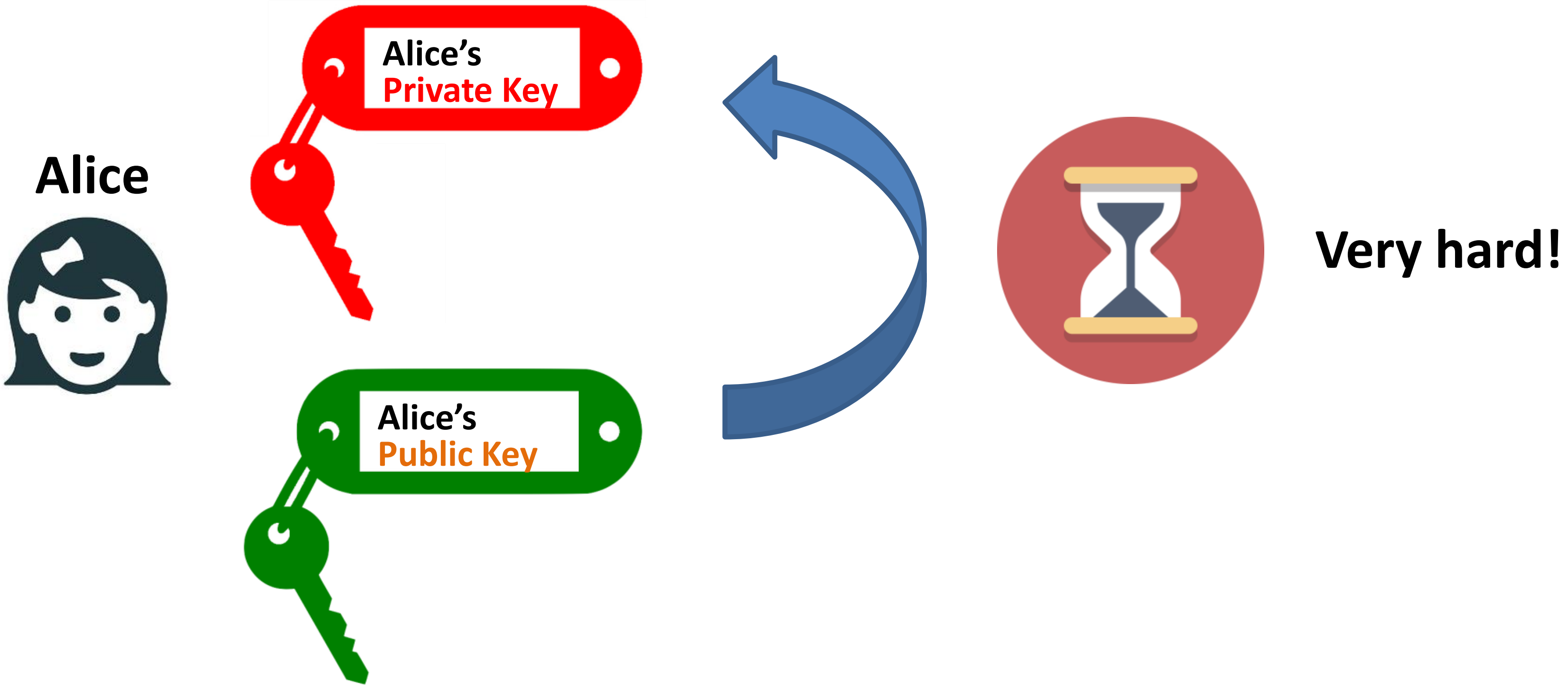
Public and private keys



Public and private keys



Public and private keys



Asymmetric Cryptography

- **Public key**
- **Private key**

- **E(private-key$_{Alice,}$ m) = c**
- **D(public-key$_{Alice,}$ c) = m**

- **E(public-key$_{Alice,}$ m) = c**
- **D(private-key$_{Alice,}$ c) = m**

Main ideas

- Bob:
  - **publishes**, say in Yellow/White pages, his **public key**, and
  - **keeps** to himself the **matching private key**.

Main ideas (Confidentiality)

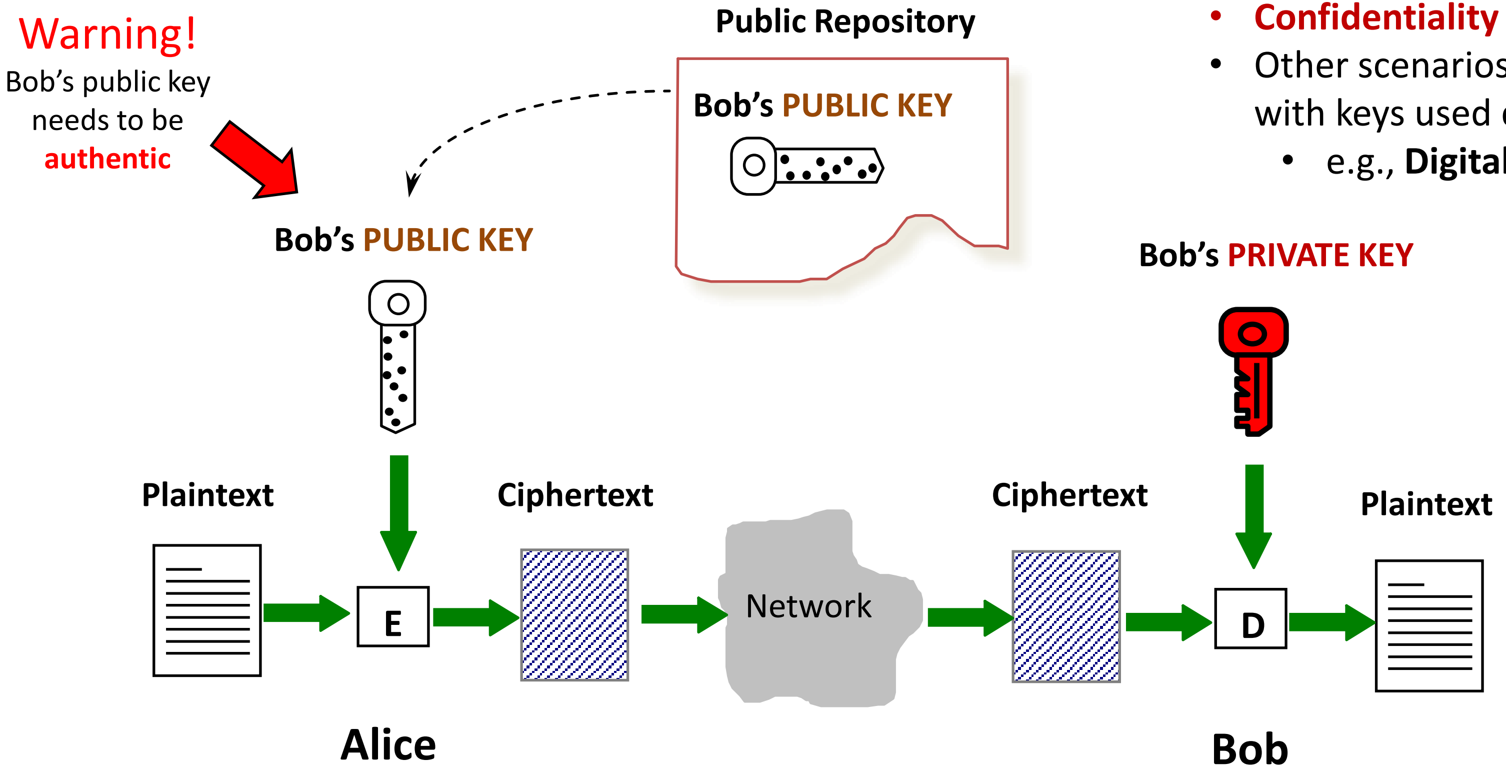
- Alice:
  - Looks up the phone book, and **finds out Bob's public key**
  - **Encrypts a message using Bob's public key** and the encryption algorithm.
  - **Sends the ciphertext** to Bob.

Main ideas (Confidentiality)

- Bob:
  - **Receives the ciphertext** from Alice.
  - **Decrypts** the ciphertext **using his private key**, together with the decryption algorithm

Asymmetric Encryption

- **Confidentiality scenario**
- Other scenarios are possible, with keys used differently…
  - e.g., **Digital signatures**



Main differences with Symmetric Crypto

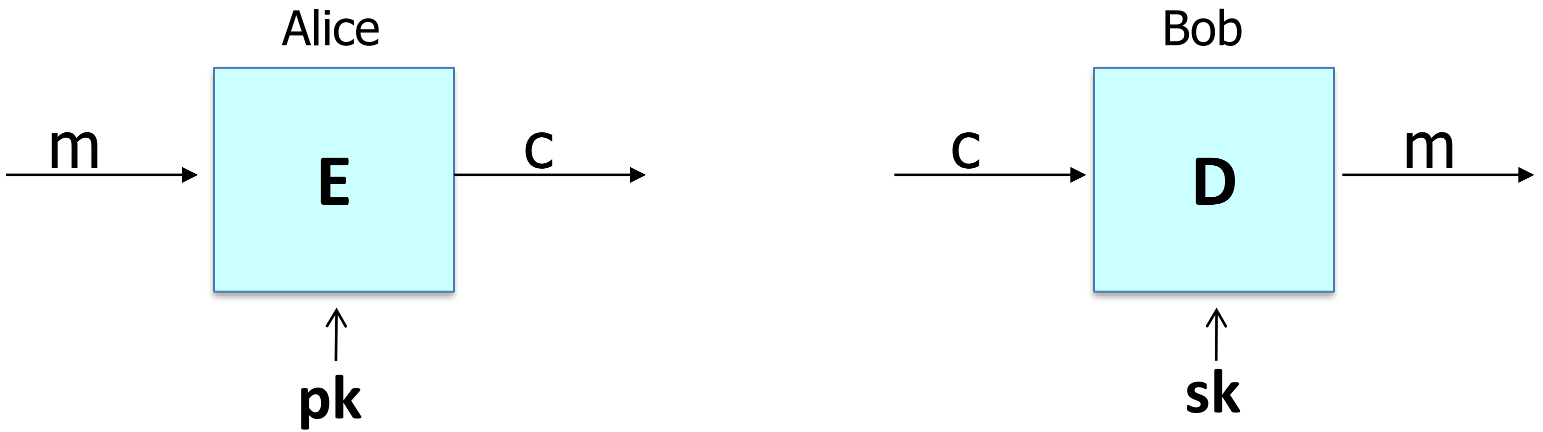
- The public key is different from the private key.
- Infeasible for an attacker to find out the private key from the public key.
- No need for Alice & Bob to distribute a shared secret key beforehand!
- Only one pair of public and private keys is required for each user!

Let’s start seriously

- define what is public key encryption
- what it means for public key encryption to be secure

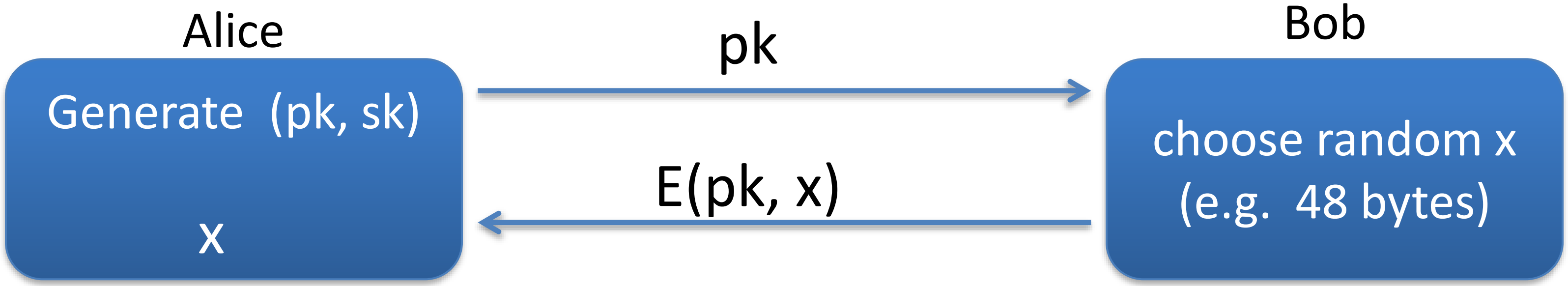
Public key encryption

Bob: generates (PK, SK) and gives PK to Alice



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)

Public key encryption

Def: a public-key encryption system is a triple of algs. (G, E, D)

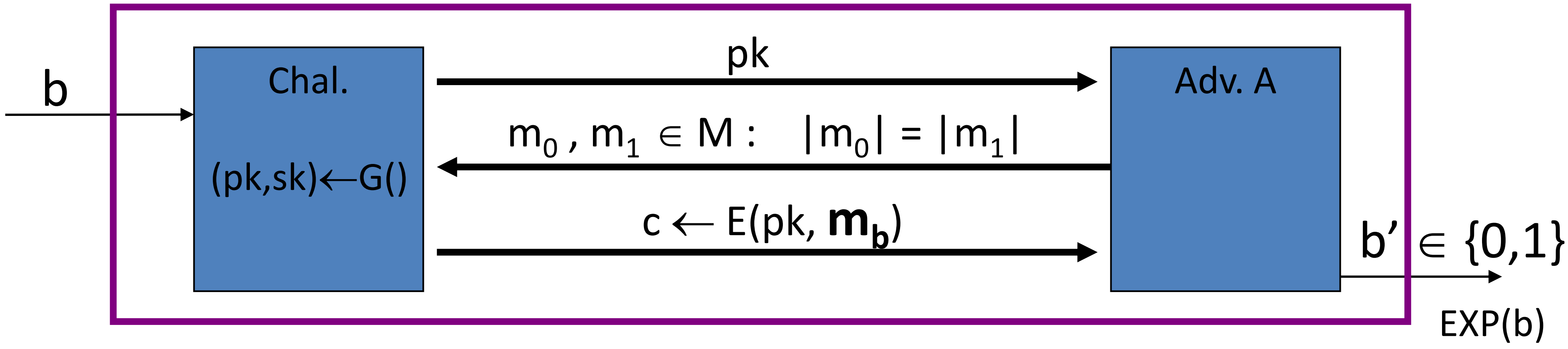
- G(): randomized alg. outputs a key pair (pk, sk)
- E(pk, m): randomized alg. that takes $m \in M$ and outputs $c \in C$
- D(sk,c): det. alg. that takes $c \in C$ and outputs $m \in M$ or $\perp$

Consistency: $\forall$(pk, sk) output by G :

$$\forall m \in M: \quad D(sk, \; E(pk, m) \,) = m$$

Security: eavesdropping

For b=0,1 define experiments EXP(0) and EXP(1) as:



Def: $\mathbb{E}$ =(G,E,D) is sem. secure (a.k.a IND-CPA) if for all efficient A:

$$\text{Adv}_{SS}[A,\mathbb{E}] = \Big| \Pr[\text{EXP}(0)=1] - \Pr[\text{EXP}(1)=1] \Big| < \text{negligible}$$

Relation to symmetric cipher security

Recall: for symmetric ciphers we had two security notions:

- One-time security and many-time security (CPA)
- We showed that one-time security $\not\Rightarrow$ many-time security

For public key encryption:

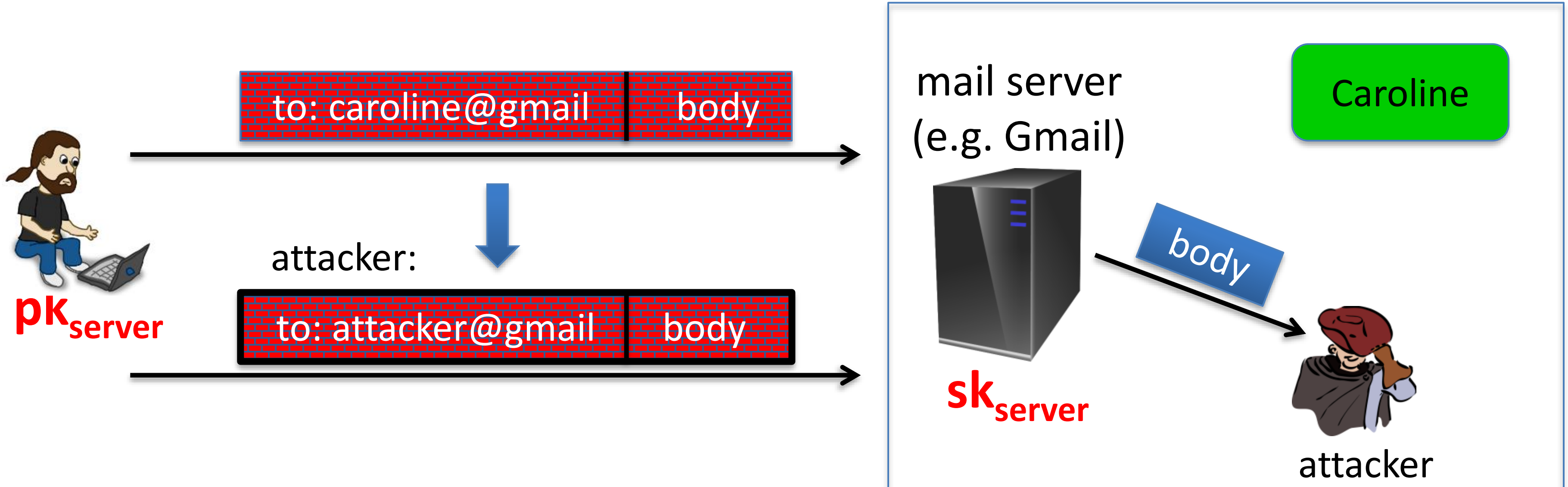
- One-time security $\Rightarrow$ many-time security (CPA)

  (follows from the fact that attacker can encrypt by himself)

- Public key encryption **must** be randomized

Security against active attacks

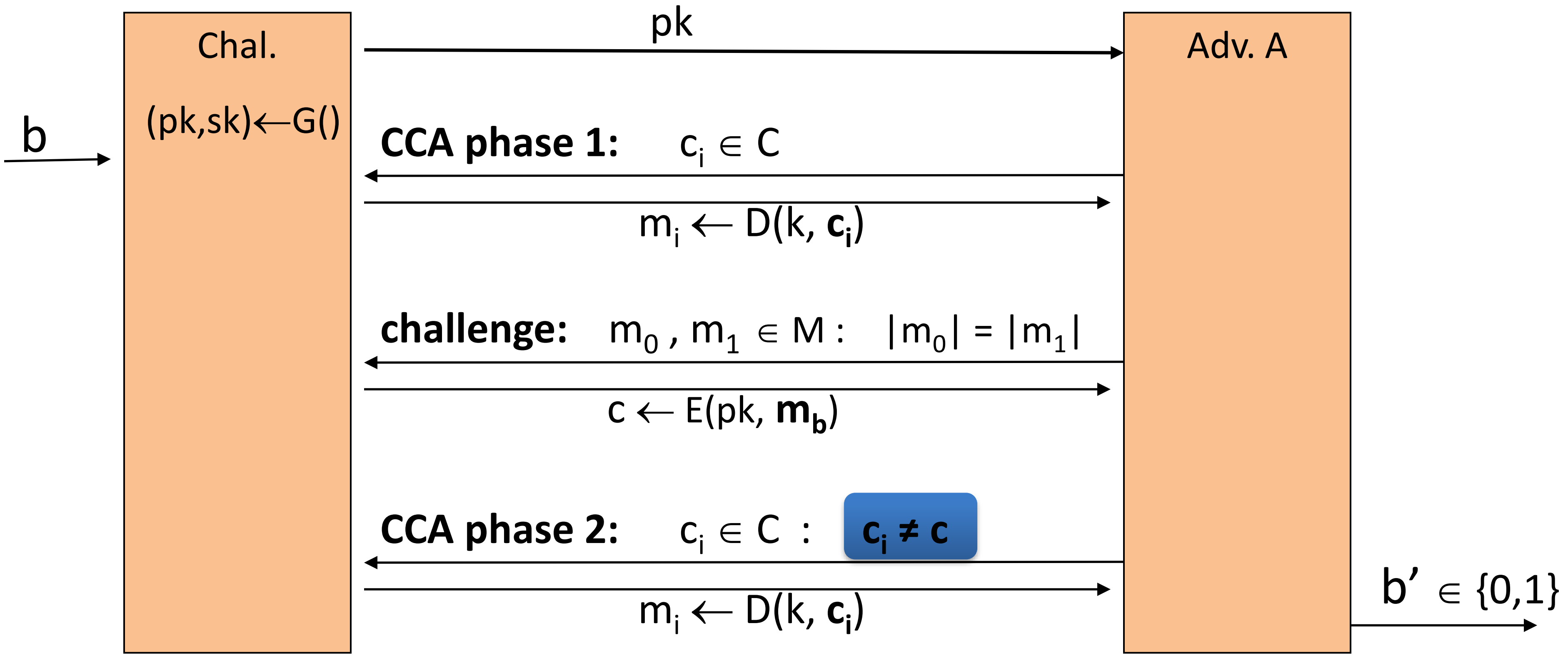
What if attacker can tamper with ciphertext?



Attacker is given decryption of msgs that start with **"to: attacker"**

(pub-key) Chosen Ciphertext Security: definition

$\mathbb{E}$ = (G,E,D) public-key enc. over (M,C). For b=0,1 define EXP(b):

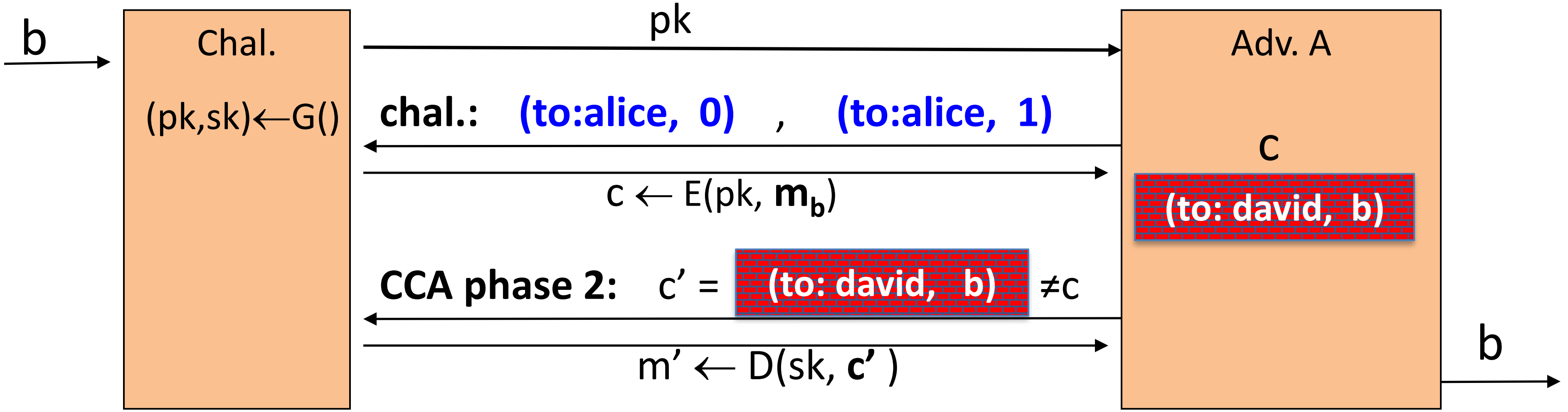


Chosen ciphertext security: definition

Def: $\mathbb{E}$ is CCA secure (a.k.a IND-CCA) if for all efficient A:

$$Adv_{CCA}[A,\mathbb{E}] = \left| \Pr[EXP(0)=1] - \Pr[EXP(1)=1] \right|$$ is negligible.

Example: Suppose **(to: alice, body)** ⟶ **(to: david, body)**



Active attacks: symmetric vs. pub-key

Recall: secure symmetric cipher provides **authenticated encryption**

[chosen plaintext security & ciphertext integrity **]**

- Roughly speaking: **attacker cannot create new ciphertexts**
- Implies security against chosen ciphertext attacks

In public-key settings:

- Attacker **can** create new ciphertexts using pk !!
- So instead: we directly require chosen ciphertext security

Trapdoor Permutations

Trapdoor functions (TDF)

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

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

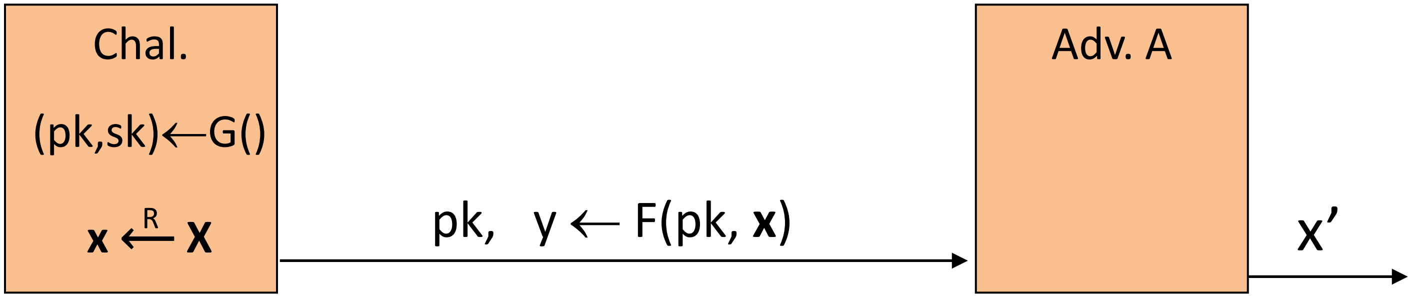
More precisely: $\forall (pk, sk)$ output by G

$$\forall x \in X: \quad F^{-1}(sk, F(pk, x)) = x$$

Secure Trapdoor Functions (TDFs)

(G, F, F^{-1}) is secure if $F(pk, \cdot)$ is a "one-way" function:

can be evaluated, but cannot be inverted without sk

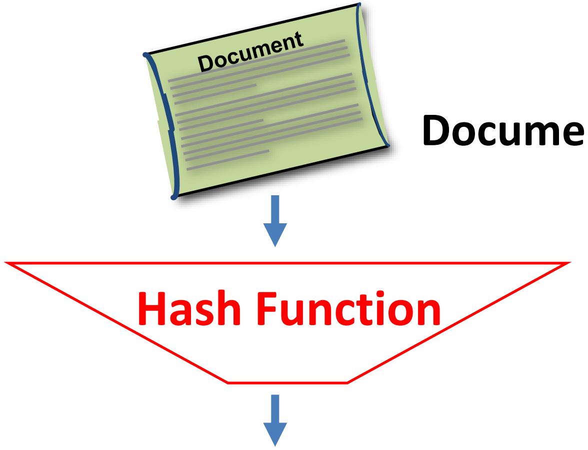


<u>Def</u>: (G, F, F^{-1}) is a secure TDF if for all efficient A:

$$\mathrm{Adv}_{\mathrm{OW}}[A,F] = \mathbf{\Pr[\, x = x' \,]} < \text{negligible}$$

Hash Functions

- **Hash functions:**
  - **Input:** arbitrary length
  - **Output:** fixed length (generally much shortern than the input)



One-Way Hash Algorithm

- A one-way hash algorithm hashes an input document into a condensed short output (say of 256 bits)
  - Denoting a one-way hash algorithm by H(.), we have:
    - Input: m - a binary string of any length
    - Output: H(m) - a binary string of L bits, called the "hash of m under H".
    - The output length parameter L is fixed for a given one-way hash function H,
    - Examples:
      - The one-way hash function "MD5" has L = 128 bits
      - The one-way hash function "SHA-1" has L = 160 bits

Properties of One-Way Hash Algorithm

- A good one-way hash algorithm H needs to have these properties:
  - 1. **Easy to Evaluate**:
    - The hashing algorithm should be fast
  - 2. **Hard to Reverse**:
    - There is no feasible algorithm to "**reverse**" a hash value,
    - That is, given any hash value **h**, it is computationally infeasible to find any document **m** such that **H(m) = h**.
  - 3. **Hard to find Collisions**:
    - There is no feasible algorithm to find two or more input documents which are hashed into the same condensed output,
    - That is, it is computationally infeasible to find any two documents **m1**, **m2** such that **H(m1)= H(m2)**.
  - 4. **A small change** to a message **should change the hash value so extensively** that the new hash value appears uncorrelated with the old hash value

Public-key encryption from TDFs

- (G, F, F^{-1}): secure TDF $X \longrightarrow Y$
- (E_s, D_s) : symmetric auth. encryption defined over (K,M,C)
- $H: X \longrightarrow K$ 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) : symmetric auth. encryption defined over (K,M,C)
- $H: X \longrightarrow K$ 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)

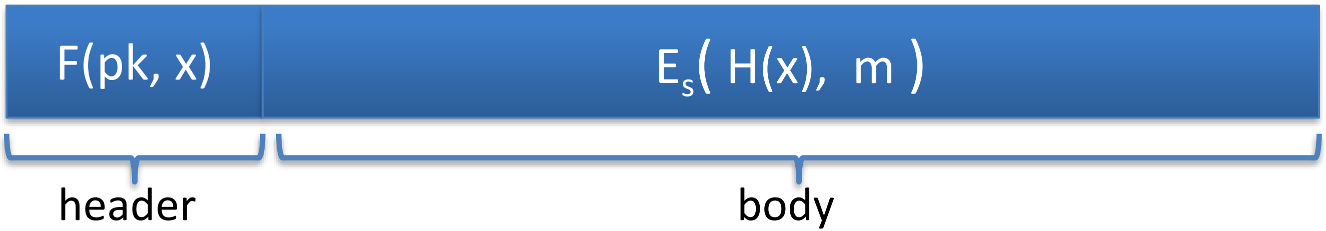
D(sk, (y,c)) :

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

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

output m

In pictures:



<u>Security Theorem</u>:

If **(G, F, F^{-1})** is a secure TDF, **(E$_s$, D$_s$)** provides auth. enc.

and **H:** X ⟶ K is a "random oracle"

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

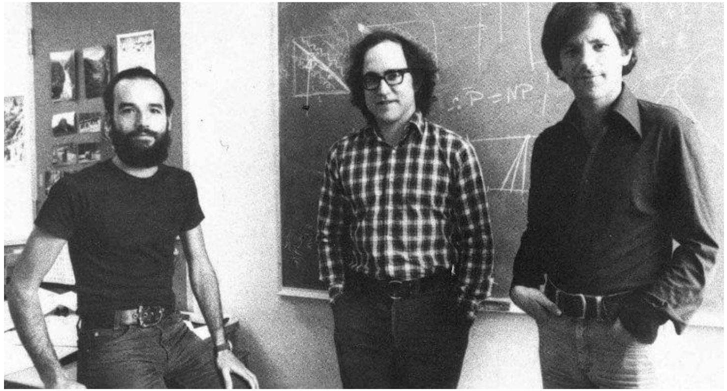
Incorrect use of a Trapdoor Function (TDF)

Never encrypt by applying F directly to plaintext:

E(pk, m) :	**D(sk, c) :**
output $c \longleftarrow F(pk, m)$	output $F^{-1}(sk, c)$

Problems:

- Deterministic: cannot be semantically secure !!
- Many attacks exist (next segment)



The RSA trapdoor permutation

- One of the first practical responses to the challenge posed by Diffie-Hellman was developed by ***R****on Rivest*, *Adi* ***S****hamir*, and *Len* ***A****dleman* of MIT in 1977
- Resulting algorithm is known as *RSA*
- Based on properties of *prime numbers* and results from *number theory*

Review: trapdoor permutations

Three algorithms: (G, F, F^{-1})

- G: outputs pk, sk. pk defines a function $F(pk, \cdot): X \to X$
- $F(pk, x)$: evaluates the function at x
- $F^{-1}(sk, y)$: inverts the function at y using sk

Secure trapdoor permutation:

The function $F(pk, \cdot)$ is one-way without the trapdoor sk

Review: arithmetic mod composites

Let $N = p \cdot q$ where p,q are prime where $p,q \approx N^{1/2}$

$Z_N = \{0,1,2,\ldots,N-1\}$; $(Z_N)^* = \{\text{invertible elements in } Z_N\}$

<u>Facts</u>: $x \in Z_N$ is invertible $\Leftrightarrow$ $\gcd(x,N) = 1$

– Number of elements in $(Z_N)^*$ is $\varphi(N) = (p-1)(q-1) = N-p-q+1$

<u>Euler's thm</u>: $\boxed{\forall\, x \in (Z_N)^* \; : \; x^{\varphi(N)} = 1}$

The RSA trapdoor permutation

First published: Scientific American, Aug. 1977.

Very widely used:

- SSL/TLS: certificates and key-exchange
- Secure e-mail and file systems

  … many others

The RSA trapdoor permutation

G(): choose random primes p,q $\approx$1024 bits. Set **N=pq**.

choose integers **e , d** s.t. **e·d = 1 (mod φ(N))**

output pk = (N, e) , sk = (N, d)

F(pk, x): $\mathbb{Z}_N^* \to \mathbb{Z}_N^*$; **RSA(x) = x^e^** (in Z_N)

$\mathbf{F^{-1}}$**(sk, y)** = y^d ; $y^d = \mathbf{RSA(x)^d} = x^{ed} = x^{k\varphi(N)+1} = \left(x^{\varphi(N)}\right)^k \cdot \mathbf{x} = x$

RSA - small example

- Bob (**keys generation**):
  - chooses 2 primes: **p=5, q=11**
  - multiplies p and q: $\mathbf{n = p \times q = 55}$
  - chooses a number **e=3** s.t. **gcd(e, 40) = 1**
  - compute **d=27** that satisfy $\mathbf{(3 \times d) \bmod 40 = 1}$

  - **Bob's public key: (3, 55)**
  - **Bob's private key: 27**

RSA - small example

- Alice (**encryption**):
  - has a message **m=13** to be sent to Bob
  - finds out **Bob's public encryption key (3, 55)**
  - calculates **c** as follows:

$$\begin{aligned} c &= m^e \bmod n \\ &= 13^3 \bmod 55 \\ &= 2197 \bmod 55 \\ &= 52 \end{aligned}$$

  - sends the ciphertext **c=52** to Bob

RSA - small example

- Bob (**decryption**):
  - receives the ciphertext **c=52** from Alice
  - uses his matching private decryption key **27** to calculate **m**:

$$m = 52^{27} \bmod 55$$
$$= 13 \text{ (Alice's message)}$$

The RSA assumption

RSA assumption: RSA is one-way permutation

For all efficient algs. A:

$$\Pr\left[\ A(N,e,y) = y^{1/e}\ \right] < \text{negligible}$$

where $p,q \xleftarrow{R}$ n-bit primes, $N \leftarrow pq$, $y \xleftarrow{R} Z_N^*$

Review: RSA pub-key encryption (ISO std)

(E_s, D_s): symmetric enc. scheme providing auth. encryption.

H: $Z_N \to K$ where K is key space of (E_s, D_s)

- **G**(): generate RSA params: pk = (N,e), sk = (N,d)
- **E**(pk, m): (1) choose random x in Z_N

  (2) $y \leftarrow RSA(x) = x^e$, $k \leftarrow H(x)$

  (3) output $(y, E_s(k,m))$
- **D**(sk, (y, c)): output $D_s\big(H\big(RSA^{-1}(y)\big), c\big)$ -> m

Textbook RSA is insecure

Textbook RSA encryption:

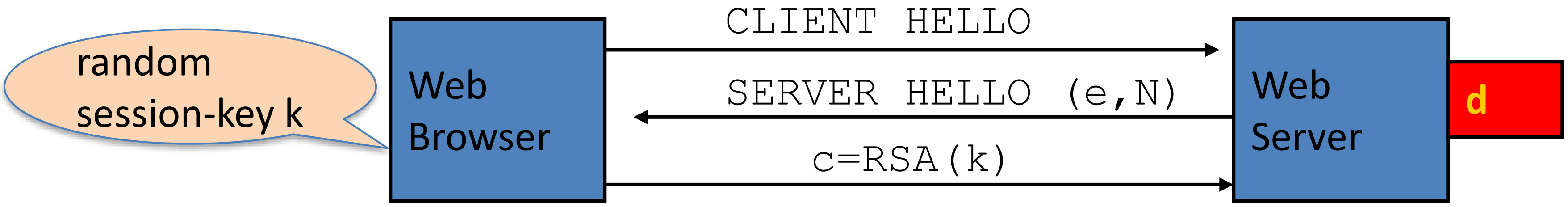
- public key: **(N,e)**   Encrypt: $\mathbf{c \longleftarrow m^e}$ (in Z_N)
- secret key: **(N,d)**   Decrypt: $\mathbf{c^d \longrightarrow m}$

Insecure cryptosystem !!

- Is not semantically secure and many attacks exist

⇒ The RSA trapdoor permutation is not an encryption scheme !

A simple attack on textbook RSA



Suppose k is 64 bits: $k \in \{0,\ldots,2^{64}\}$. Eve sees: $c = k^e$ in Z_N

If $\mathbf{k = k_1 \cdot k_2}$ where $k_1, k_2 < 2^{34}$ (prob. ≈20%) then $\mathbf{c/k_1^e = k_2^e}$ in Z_N

Meet-in-the-middle attack:

Step 1: build table: $c/1^e, c/2^e, c/3^e, \ldots, c/2^{34e}$. time: 2^{34}

Step 2: for $k_2 = 0,\ldots, 2^{34}$ test if k_2^e is in table. time: 2^{34}

Output matching (k_1, k_2). Total attack time: $\approx 2^{40} << 2^{64}$

Is RSA a one-way function?

Is it really hard to invert RSA without knowing the trapdoor?

Is RSA a one-way permutation?

To invert the RSA one-way func. (without d) attacker must compute:

$$x \quad \text{from} \quad c = x^e \pmod{N}.$$

How hard is computing e'th roots modulo N ??

Best known algorithm:

- Step 1: factor N (hard)
- Step 2: compute e'th roots modulo p and q (easy)

Shortcuts?

Must one factor N in order to compute e'th roots?

To prove no shortcut exists show a reduction:

- Efficient algorithm for e'th roots mod N

  $\Rightarrow$ efficient algorithm for factoring N.

- Oldest problem in public key cryptography.

Some evidence no reduction exists: (BV'98)

- "Algebraic" reduction $\Rightarrow$ factoring is easy.

How **not** to improve RSA's performance

To speed up RSA decryption use small private key d ($d \approx 2^{128}$)

$$\boxed{c^d = m \pmod{N}}$$

Wiener'87: if $d < N^{0.25}$ then RSA is insecure.

BD'98: if $d < N^{0.292}$ then RSA is insecure (open: $d < N^{0.5}$)

<u>Insecure:</u> priv. key d can be found from (N,e)

Wiener's attack

(N,e) => d and d < $N^{0.25}/3$

Recall: $e \cdot d = 1 \pmod{\varphi(N)} \Rightarrow \exists\, k \in Z: \quad e \cdot d = k \cdot \varphi(N) + 1$

$$\left|\frac{e}{\psi(N)} - \frac{k}{d}\right| = \frac{1}{d \cdot \varphi(N)} \leq \frac{1}{\sqrt{N}}$$

$\varphi(N) = N-p-q+1 \Rightarrow |N - \varphi(N)| \leq p+q \leq 3\sqrt{N}$

$d \leq N^{0.25}/3 \Rightarrow \frac{1}{2d^2} - \frac{1}{\sqrt{N}} \geq \frac{3}{\sqrt{N}}$

$$\left|\frac{\mathrm{e}}{N} - \frac{k}{d}\right| \leq \left|\frac{\mathrm{e}}{N} - \frac{\mathrm{e}}{\varphi(N)}\right| + \left|\frac{\mathrm{e}}{\varphi(N)} - \frac{k}{d}\right| \leq \frac{1}{2d^2}$$

Continued fraction expansion of e/N gives k/d.

$e \cdot d = 1 \pmod{k} \Rightarrow \gcd(d,k)=1 \Rightarrow$ can find d from k/d

RSA in Practice

RSA With Low public exponent

To speed up RSA encryption use a small e: $c = m^e \pmod N$

- Minimum value: **$e=3$** ($\gcd(e, \varphi(N)) = 1$) (Q: why not 2?)
- Recommended value: **$e=65537=2^{16}+1$**

Encryption: 17 multiplications

<u>Asymmetry of RSA:</u> fast enc. / slow dec.

– ElGamal (next week): approx. same time for both.

Key lengths

Security of public key system should be comparable to security of symmetric cipher:

Cipher key-size	RSA Modulus size
80 bits	1024 bits
128 bits	3072 bits
256 bits (AES)	**15360** bits

Implementation attacks

Timing attack: [Kocher et al. 1997] , [BB'04]

The time it takes to compute $c^d \pmod N$ can expose d

Power attack: [Kocher et al. 1999)

The power consumption of a smartcard while it is computing $c^d \pmod N$ can expose d.

Faults attack: [BDL'97]

A computer error during $c^d \pmod N$ can expose d.

A common defense: check output. 10% slowdown.

An Example Fault Attack on RSA (CRT)

A common implementation of RSA decryption: $x = c^d$ in Z_N

$$\left.\begin{array}{ll} \text{decrypt mod } p\text{:} & x_p = c^d \ \text{ in } Z_p \\[1em] \text{decrypt mod } q\text{:} & x_q = c^d \ \text{ in } Z_q \end{array}\right\} \text{ combine to get } x = c^d \text{ in } Z_N$$

Suppose error occurs when computing x_q, but no error in x_p

Then: output is x' where $x' = c^d$ in Z_p but $x' \neq c^d$ in Z_q

$\Rightarrow$ $(x')^e = c$ in Z_p but $(x')^e \neq c$ in Z_q $\Rightarrow$ $\gcd\big((x')^e - c, N\big) =$

RSA Key Generation Trouble [Heninger et al./Lenstra et al.]

OpenSSL RSA key generation (abstract):

```
prng.seed(seed)
p = prng.generate_random_prime()
prng.add_randomness(bits)
q = prng.generate_random_prime()
N = p*q
```

Suppose poor entropy at startup:

- Same p will be generated by multiple devices, but different q
- N_1 , N_2 : RSA keys from different devices $\Rightarrow$ $\gcd(N_1,N_2) = p$

RSA Key Generation Trouble [Heninger et al./Lenstra et al.]

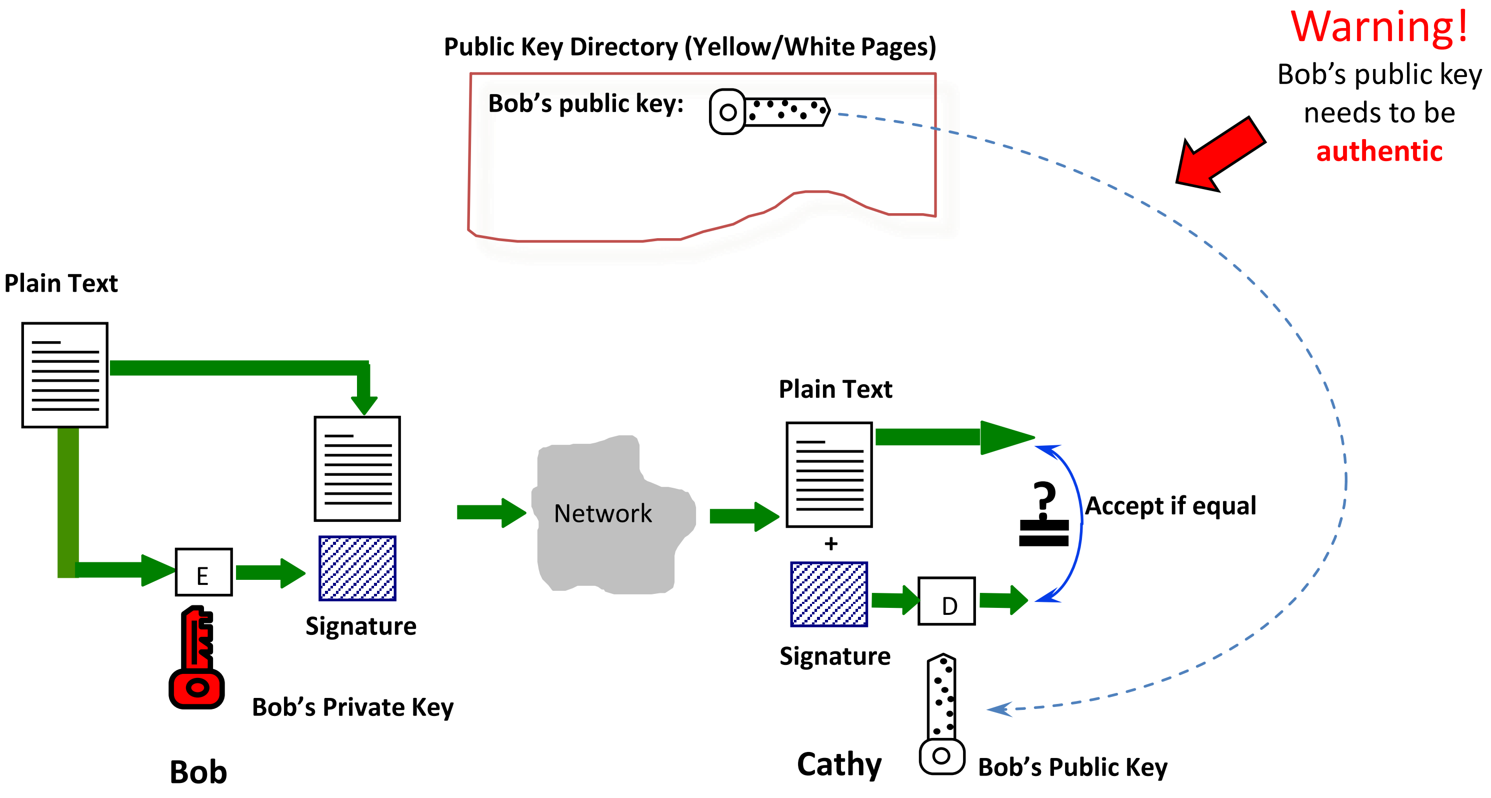
Experiment: factors 0.4% of public HTTPS keys !!

Lesson:

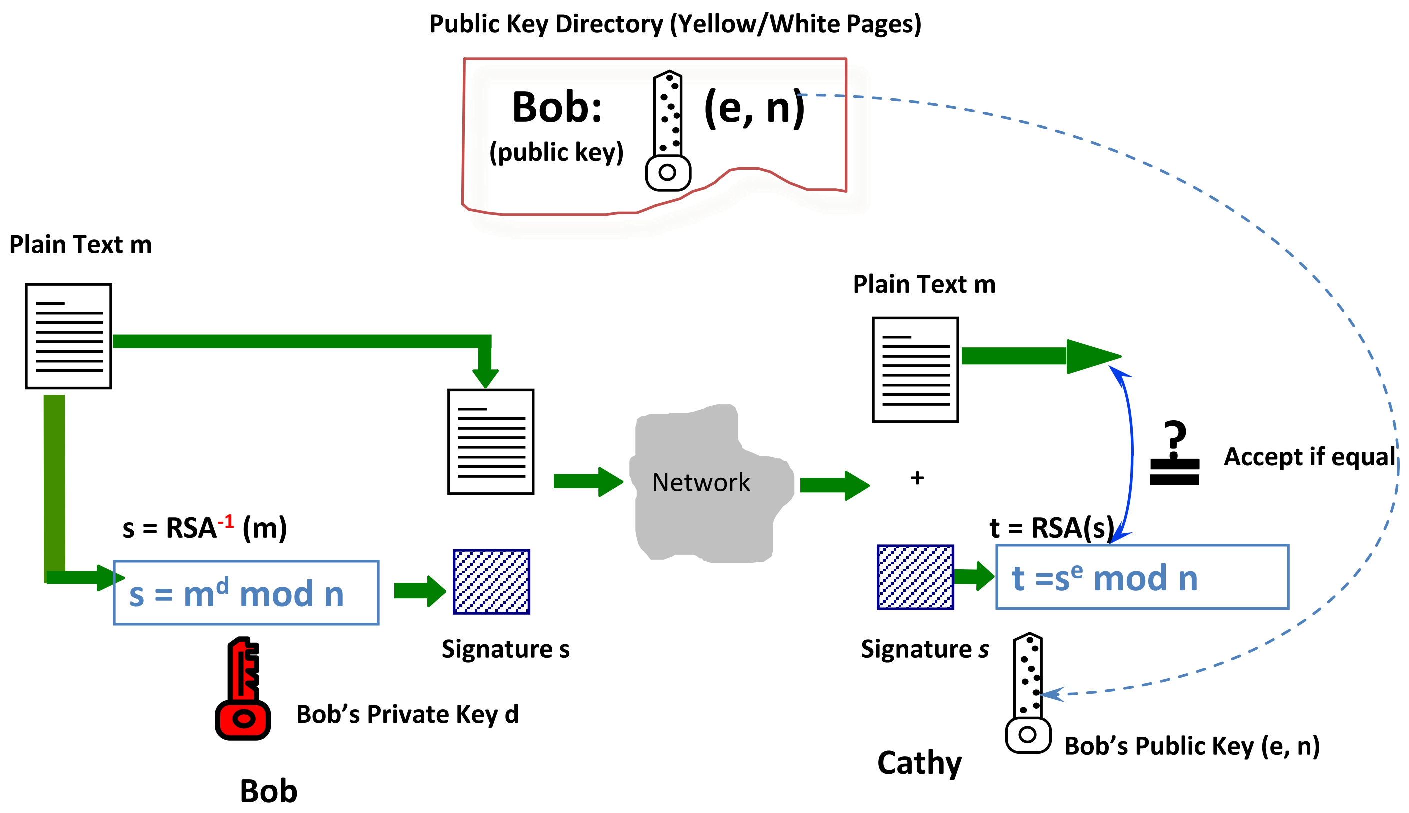
- Make sure random number generator is properly seeded when generating keys

Digital Signatures

Digital Signature



Digital Signature (based on RSA)



RSA Signature - small example

- Bob (**keys generation**):
  - chooses 2 primes: **p=5, q=11**
  - multiplies p and q: **n = p × q = 55**
  - chooses a number **e=3** s.t. **gcd(e, 40) = 1**
  - compute **d=27** that satisfy **(3 × d) mod 40 = 1**

  - **Bob's public key: (3, 55)**
  - **Bob's private key: 27**

RSA Signature - small example

- Bob:
  - has a document **m=19** to sign:
  - uses **his private key d=27** to **calculate the digital signature** of **m=19**:

$$
\begin{aligned}
s &= m^d \bmod n \\
&= 19^{27} \bmod 55 \\
&= 24
\end{aligned}
$$

  - **appends 24 to 19**.

    Now **(m, s) = (19, 24**) indicates that the **doc is 19**, and **Bob's signature on the doc is 24**.

RSA Signature - small example

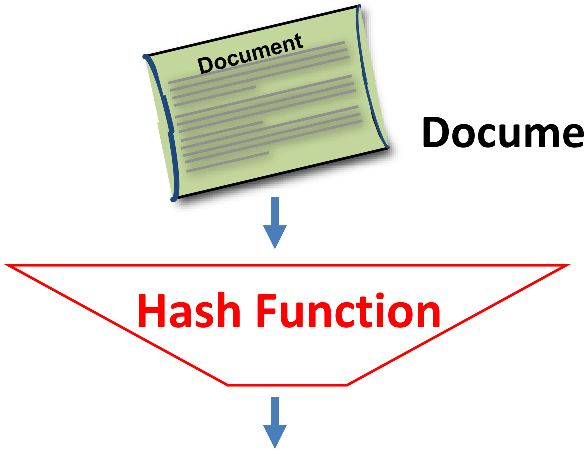
- Cathy, a verifier:
  - **receives** a pair **(m,s)=(19, 24)**
  - looks up the phone book and **finds out Bob's public key** (e, n)=(3, 55)
  - **calculates** $\mathbf{t} = s^e \bmod n$
    $= 24^3 \bmod 55$
    $= 19$
  - **checks whether t=m**
  - **confirms that (19,24) is a genuinely signed document of Bob if t=m.**

How about Long Documents ?

- In the previous example, a document has to be an integer in [0,...,n)
- To sign a very long document, we need a so called one-way **hash algorithm**
- **Instead of signing directly on a doc**,
  - we **hash the doc first**,
  - and **sign the hashed data** which is normally short.

Hash Functions

- **Hash functions:**
  - **Input:** arbitrary length
  - **Output:** fixed length (generally much shortern than the input)

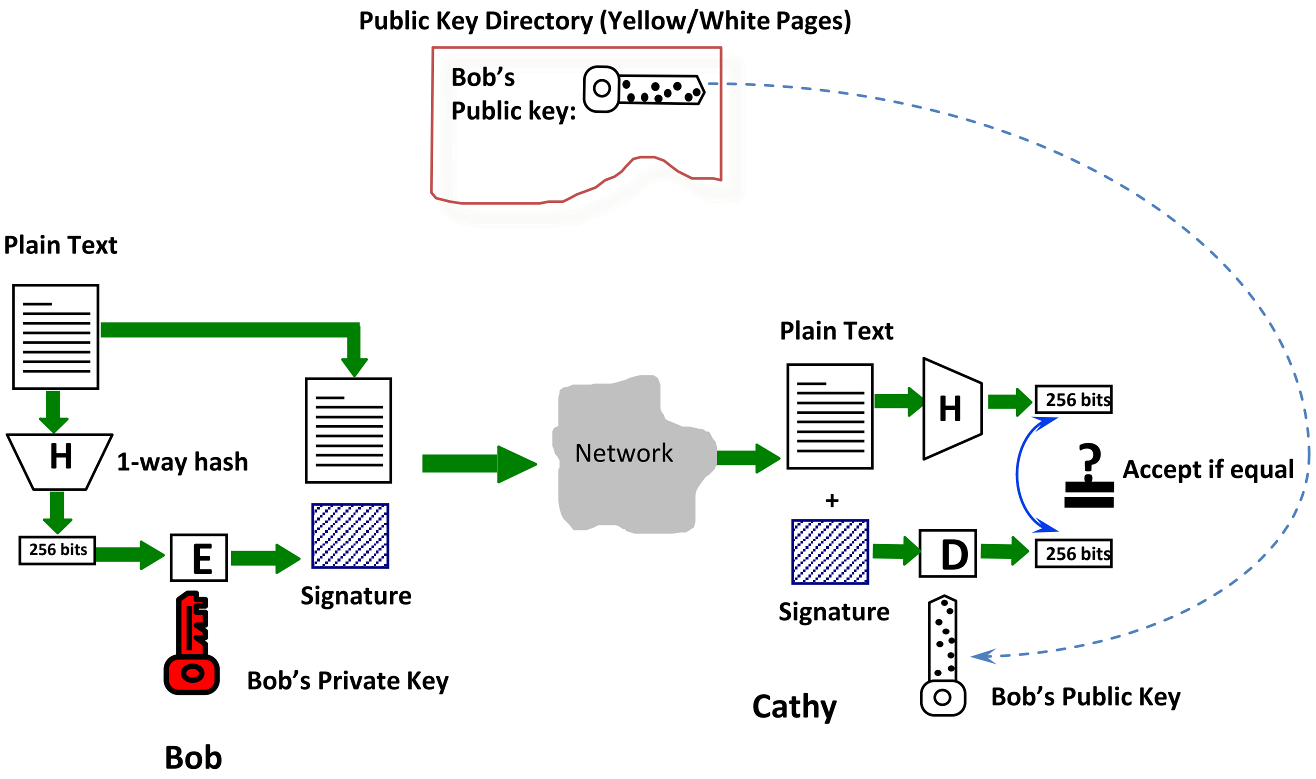


Document with arbitrary length

Hash value for the document
(fixed length, e.g. 256 bit)

Rather than signing the original document, we **sign its hash value**

Digital Signature (for long docs)



Why Digital Signature ?

- Unforgeable
  - takes 1 billion years to forge !
- Un-deniable by the signatory
- Universally verifiable
- Differs from doc to doc

Digital Signature - summary

- Three (3) steps are involved in digital signature
  - Setting up public and private keys
  - Signing a document
  - Verifying a signature

Setting up Public & Private Keys

- Bob does the following
  - prepares a pair of public and private keys
  - Publishes his public key in the public key file (such as an on-line phone book)
  - Keeps the private key to himself
- Note:
  - Setting up needs only to be done **once** !

Signing a Document

- Once setting up is completed, Bob can sign a document (such as a contract, a cheque, a certificate, ...) using the private key
- The pair of document & signature is a proof that Bob has signed the document.

Verifying a Signature

- Any party, say Cathy, can verify the pair of document and signature, by using Bob's public key in the public key file.
- Important !
  - Cathy does NOT have to have public or private key !

(Other) Asymmetric Cryptosystems

ElGamal Cryptosystem

Encryption schemes built from the Diffie-Hellman protocol

- **Key Generation** (for Bob)
  - chooses a prime **p** and a number **g** *primitive root modulo p*
    - i.e., for every integer **a** coprime to **p**, there is an interger **k** such that $\mathbf{g^k = a \bmod p}$
      - Two integers are coprime if their gcd is 1
  - chooses a random exponent **a** in [0, p-2]
  - computes $\mathbf{A = g^a \bmod p}$
  - **public key** (published in the phone book): **(p,g,A)**
  - **private key**: **a**

ElGamal Cryptosystem

- **Encryption:** Alice has a message **m** (0<=m<n) to be sent to Bob:
  - finds out **Bob's public key (p,g,A)**.
  - chooses a random exponent **b** in [0,p-2]
  - computes **$B = g^b \bmod p$**
  - computes **$c = A^b m \bmod p$**.
  - The complete ciphertex is **(B,c)**
  - sends the ciphertext **(B,c)** to Bob.

ElGamal Cryptosystem

- **Decryption**: Bob
  - receives the ciphertext **(B,c)** from Alice.
  - uses his matching private decryption key a to calculate **m** as follows.
    - Compute $\mathbf{x = p\text{-}1\text{-}a}$
    - Compute $\mathbf{m = B^x\, c \bmod p}$

ElGamal Cryptosystem

- Randomized cryptosystem
- Based on the Diffie–Hellman key exchange
- Efficiency
  - The ciphertext is twice as long as the plaintext. This is called message expansion and is a disadvantage of this cryptosystem.
- Security
  - Its security depends upon the difficulty of a certain problem related to computing discrete logarithms.

Rabin Cryptosystem

Key Generation (for Bob)

- generates 2 large random and distinct primes **p, q** s.t.

  $p \pmod 4 = q \pmod 4 = 3$ (other options are possible, this makes decryption more efficient)

- multiplies p and q: $\mathbf{n = p \times q}$
- **public key** (published in the phone book): **n**
- **private key**: **(p, q)**

Rabin Cryptosystem

- **Encryption:** Alice has a message **m** (0<=m<n) to be sent to Bob:
  - finds out **Bob's public key n**.
  - calculates the ciphertext **$c= m^2$ mod n**.
  - sends the ciphertext **c** to Bob.

Rabin Cryptosystem

- **Decryption:** Bob
  - receives the ciphertext $\mathbf{c}$ from Alice.
  - uses his matching private decryption key $\mathbf{(p,q)}$ to calculate $\mathbf{m}$ as follows.
    - Compute $m_p = c^{(p+1)/4} \bmod p$
    - Compute $m_q = c^{(q+1)/4} \bmod q$
    - Find y_p and y_q such that $y_p\, p + y_q\, q = 1$ (Euclidean algorithm)
    - Compute $r = (y_p\, p\, m_q + y_q\, q\, m_p) \bmod n$
    - Compute $s = (y_p\, p\, m_q - y_q\, q\, m_p) \bmod n$
    - One of $r, -r, s, -s$ must be the original message $\mathbf{m}$

Rabin Cryptosystem

- Efficiency
  - Encryption more efficient than RSA encryption

- Security
  - The Rabin cryptosystem has the advantage that the problem on which it relies has been proved to be as hard as integer factorization
    - Recovering the plaintext ***m*** from the ciphertext ***c*** and the public key ***n*** is computationally equivalent to factoring
    - Not currently known to be true for the RSA problem.