



# Introduction to Cryptography

Alessandro Barenghi

Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)  
Politecnico di Milano

alessandro -dot- barenghi - at - polimi -dot- it



# Word of Warning

- This is a short, simplified introduction to cryptography
- We will only introduce what is needed for systems security discussions

# What is cryptography (alt. cryptology)?

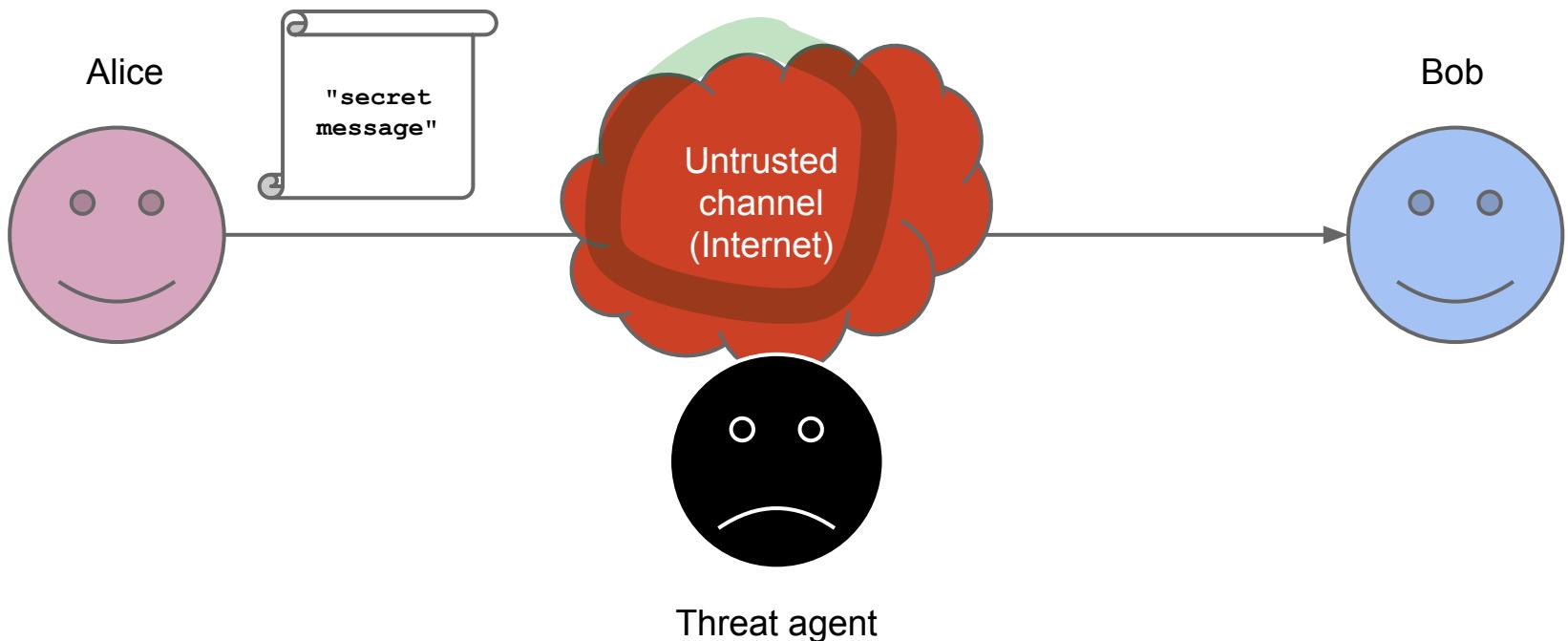
## Definition

- The study of techniques to allow secure communication and data storage in presence of attackers

## Features provided

- **Confidentiality:** data can be accessed only by chosen entities
- **Integrity/freshness:** detect/prevent tampering or replays
- **Authenticity:** data and their origin are certified
- **Non-repudiation:** data creator cannot repudiate created data
- **Advanced features:** proofs of knowledge/computation

# The Problem to Solve: Confidentiality and Integrity



# A Brief History of Cryptography

- From Greek: *kryptos*, hidden, and *graphein*, to write (i.e., “*art of secret writing*”)
- Ancient history: writing itself was already a “secret technique”.
- Cryptography born in Greek society, when writing became more common, and hidden writing became a need.

# Cryptographic prehistory

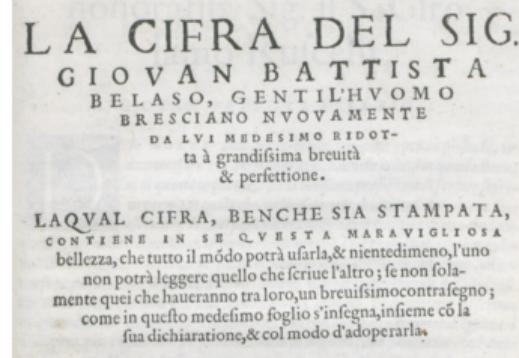
As old as written communication

- Born for commercial (recipe for lacquer on clay tablets) or military (Spartans) uses
- Designed by humans, for human computers
- Algorithms computed by hand, with pen and paper



## Original approach

- A battle of wits between
  - cryptographers: ideate a secret method to obfuscate a text
  - cryptanalysts: figure out the method, break the “cipher”
- Bellaso (1553) [1] separates the encryption method from the key



# A Brief History of Cryptography

- Medieval and renaissance studies
  - **Gabriele de Lavinde**, who wrote a manual in 1379, copy available at the Vatican archives.
  - The mirror writing of **Leonardo da Vinci**.
- Mostly a *military interest*
  - Italian Army General **Luigi Sacco** wrote a famous “*Nozioni di crittografia*” book in 1925, one of the last “non-formalized” exercises in cryptography.

# Cryptographic history

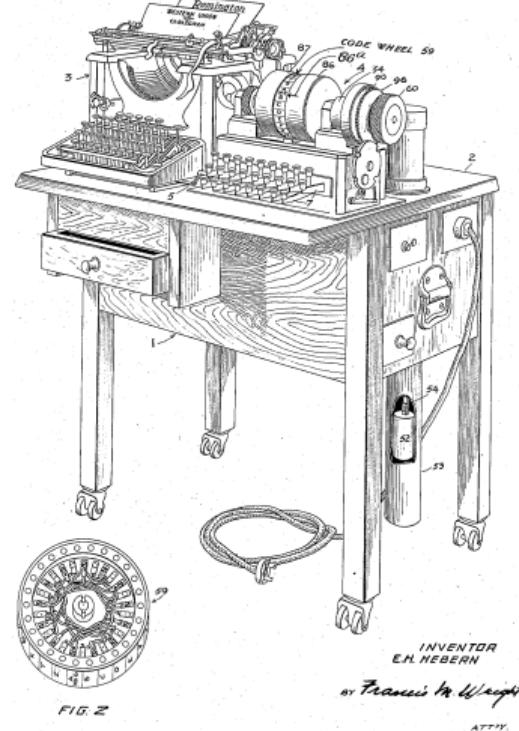
1883 - Kerchoff's six principles for a good cipher (apparatus)

- ① It must be practically, if not mathematically, unbreakable
- ② It should be possible to make it public, even to the enemy
- ③ The key must be communicable without written notes and changeable whenever the correspondants want
- ④ It must be applicable to ~~telegraphic~~ communication
- ⑤ It must be portable, and should be operable by a single person
- ⑥ Finally, given the operating environment, it should be easy to use, it shouldn't impose excessive mental load, nor require a large set of rules to be known

# Cryptographic modern history

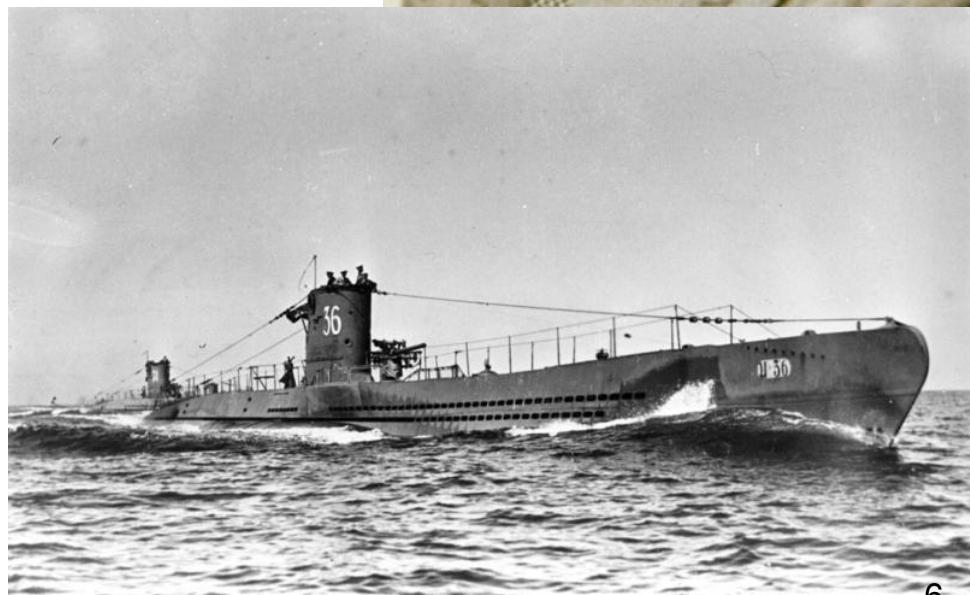
## The advent of the machines

- Mechanical computation changes cryptography
  - First rotor machine in 1917 by Ed Hebern
  - Design “popularized” in WWII by German Enigma
- Cryptanalyst at Bletchley park (Turing among them) credited for a decisive effort in winning the war by Eisenhower



# When Math Won a War

- During WWII, **Alan Turing** worked at Bletchley Park to **break** Axis ciphers, in particular the **Enigma cipher**.

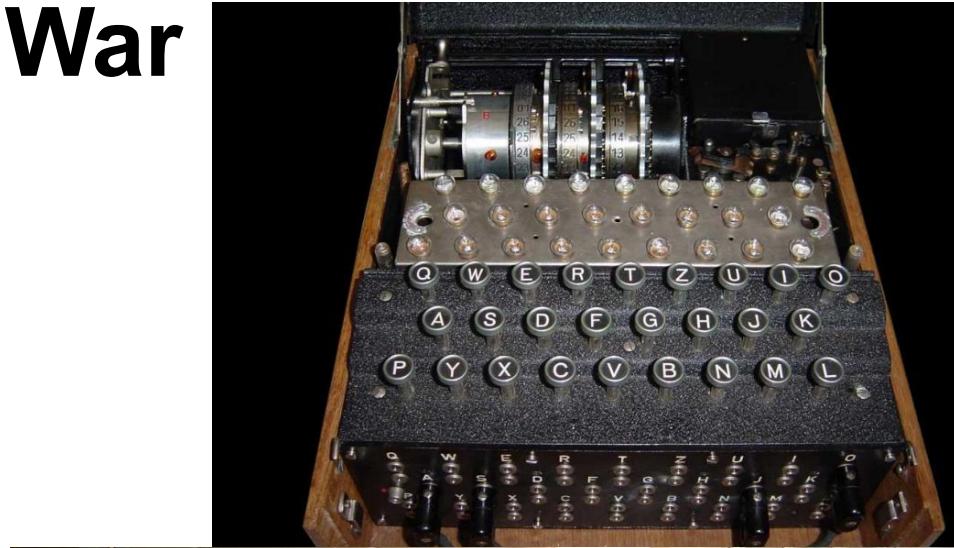


Bundesarchiv, DVM 10 Bild-23-63-65  
Foto: o. Ang. | 1936/1939 ca.



# When Math Won a War

- During WWII, Alan Turing worked at Bletchley Park to **break** Axis ciphers, in particular the **Enigma cipher**.
- Birth of the first universal computers was stimulated by this effort.



THE BOMBE (replica)

## An end to the battle of wits

- Shannon (1949) [4] - Proves that a mathematically unbreakable cipher exists
- Nash (1955) [2] - Argues that **computationally secure** ciphers are ok
  - Considers a cipher with a finite,  $\lambda$  bit long, key
  - Conjecture: if "*parts of the key interact complexly [...] in the determination of their effects on the ciphertext*", the attacker effort to break the cipher would be  $\mathcal{O}(2^\lambda)$
  - The owner of the key takes  $\mathcal{O}(\lambda^2)$  to compute the cipher
  - The computational gap is unsurmountable for large  $\lambda$

# Key Concepts in Cryptography

- First formalized by Claude Shannon in his 1949 paper “*Communication theory of secrecy systems*”.
- **Cryptosystem**: a system that takes in input a message (known as **plaintext**) and transforms it into a **ciphertext** with a **reversible function** that usually takes a **key** as a further input.
- The use of “text” is historical, and today we mean “*string of bits*”.

# Kerckhoffs' Principle

- *The security of a cryptosystem relies only on the secrecy of the key, and never on the secrecy of the algorithm.*
  - Auguste Kerckhoffs, “La cryptographie militaire”, 1883
- This means that:
  - In a secure cryptosystem we cannot retrieve the plaintext from the ciphertext without the key.
  - *Also, we cannot retrieve the key from analyzing ciphertext-plaintext pairs.*
  - Algorithms must always be assumed known to the attacker, no secret sauce!

# Outline of the topics

## In this course

- Definitions of ciphers as components with functionalities
- How to obtain confidentiality, integrity, data/origin authentication
- An overview of protocols (combinations of ciphers)
- Goal: be able to use cryptographic components properly

## In Cryptography and Architectures for Computer Security

- Design and cryptanalysis techniques for ciphers
- Cryptographic protocols and their inner workings
- Efficient, side channel attack resistant implementations
- Goal: be able to engineer cryptographic components

## Before we start...

```
int getRandomNumber()
{
    return 4; // chosen by fair dice roll.
               // guaranteed to be random.
}
```

<https://xkcd.com/221/>

### A word on randomness

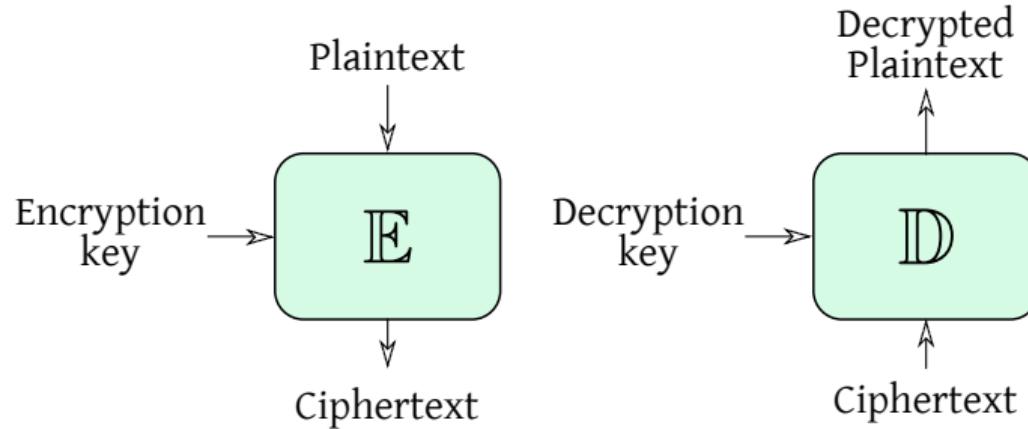
- Randomness (in this course) characterizes a generative process
- Stating: “00101 is a random string” actually makes little sense

## Data

- Plaintext space **P**: set of possible messages  $\text{ptx} \in \mathbf{P}$ 
  - Old times: words in some human-readable alphabet, modern times  $\{0, 1\}^l$
- Ciphertext space **C**: set of possible ciphertext  $\text{ctx} \in \mathbf{C}$ 
  - Usually  $\{0, 1\}^{l'}$ , not necessarily  $l = l'$  (ciphertexts may be larger)
- Key space **K**: set of possible keys
  - $\{0, 1\}^\lambda$ , keys with special formats are derived from bitstrings

## Functions

- Encryption function  $\mathbb{E} : \mathbf{P} \times \mathbf{K} \rightarrow \mathbf{C}$
- Decryption function  $\mathbb{D} : \mathbf{C} \times \mathbf{K} \rightarrow \mathbf{P}$ 
  - Correctness: for all  $\text{ptx} \in \mathbf{P}$ , we need  $k, k' \in \mathbf{K}$  s.t.  $\mathbb{D}(\mathbb{E}(\text{ptx}, k), k') = \text{ptx}$



## Goal

- Prevent anyone not authorized from being able to understand data

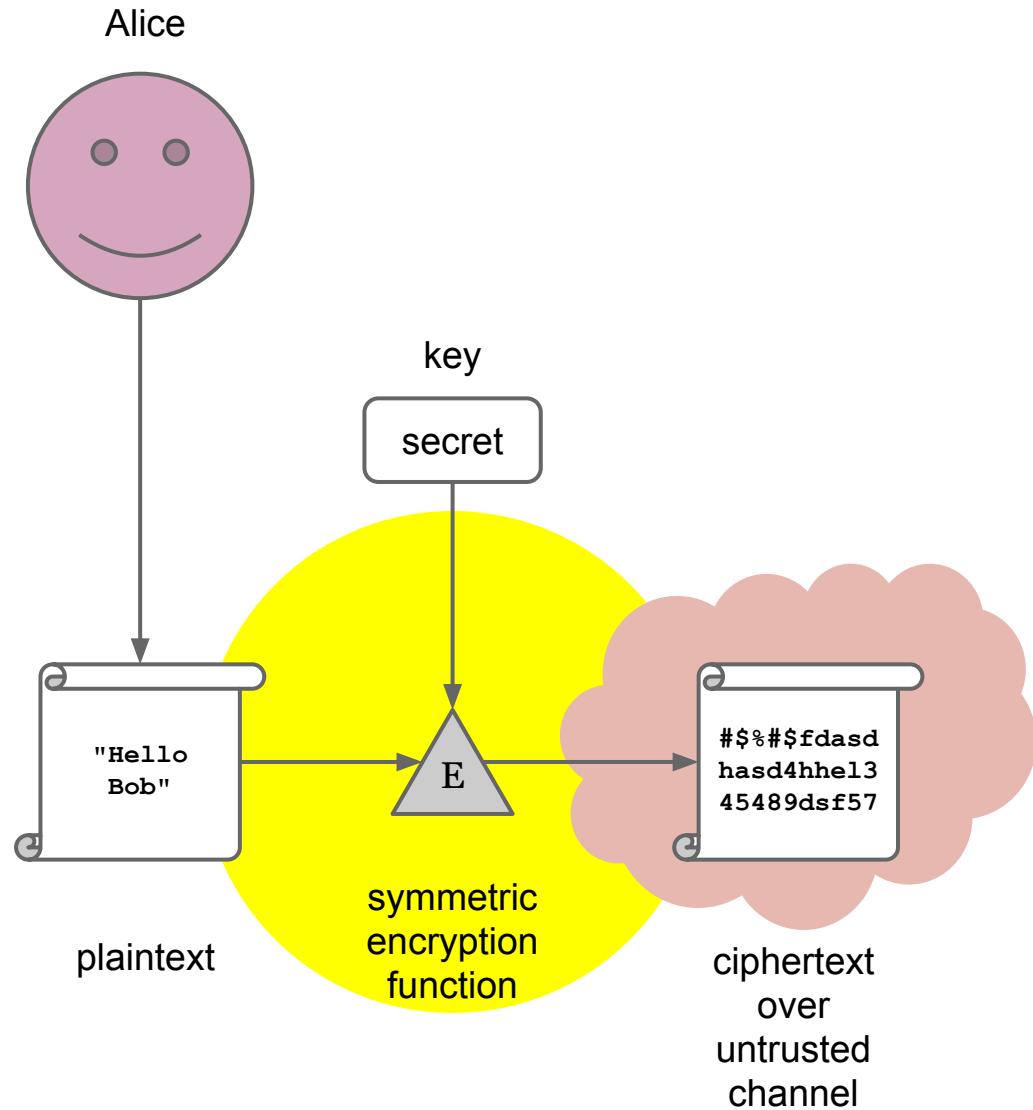
## Possible attacker models

- The attacker simply eavesdrops (ciphertext only attack)
- The attacker knows a set of possible plaintexts
  - Limit case: the attacker chooses the set of plaintexts
- The attacker may tamper with the data and observe the reactions of a decryption-capable entity
  - Limit case: the attacker sees the actual decrypted value

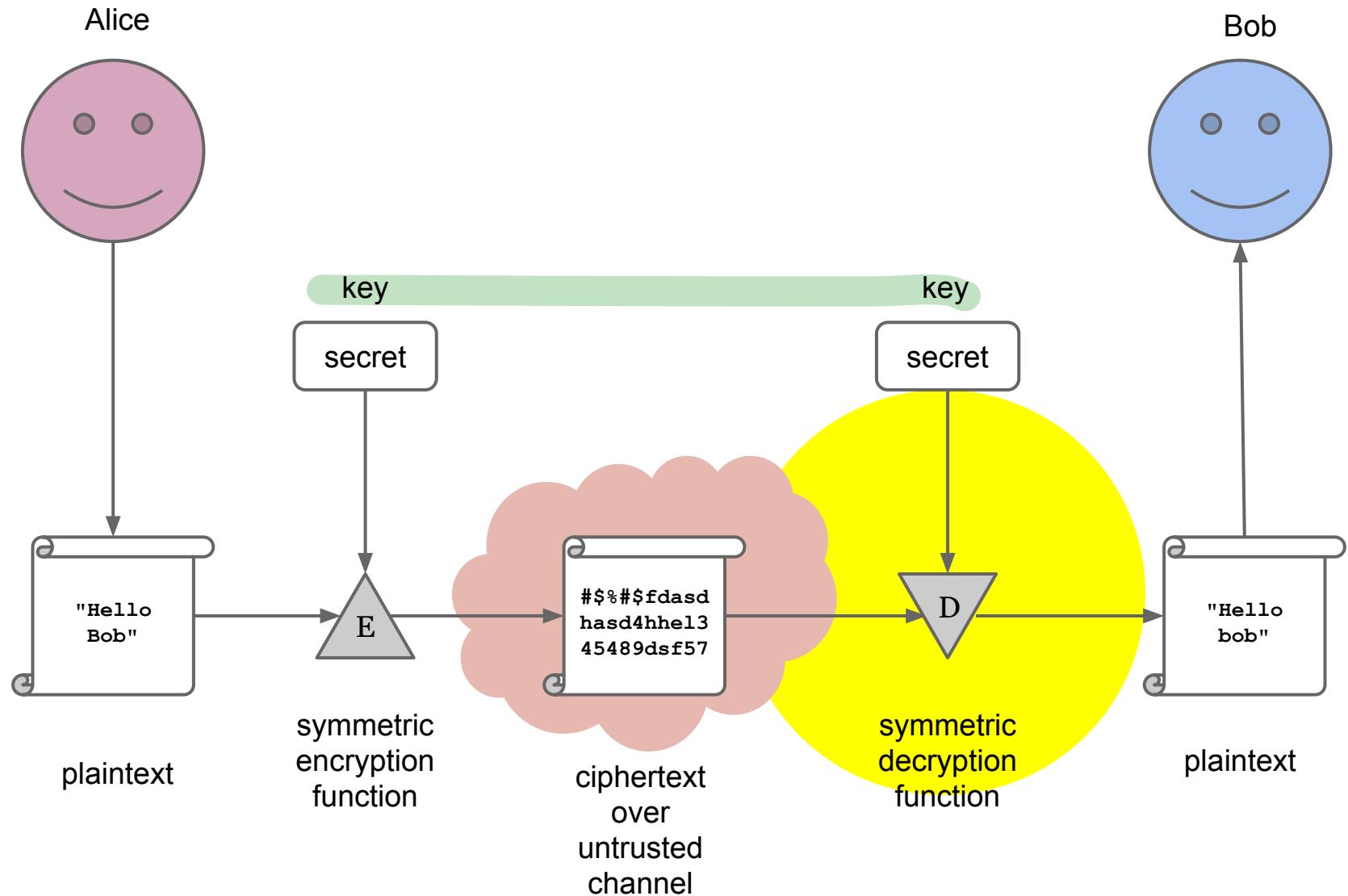
# Symmetric Encryption

Confidentiality

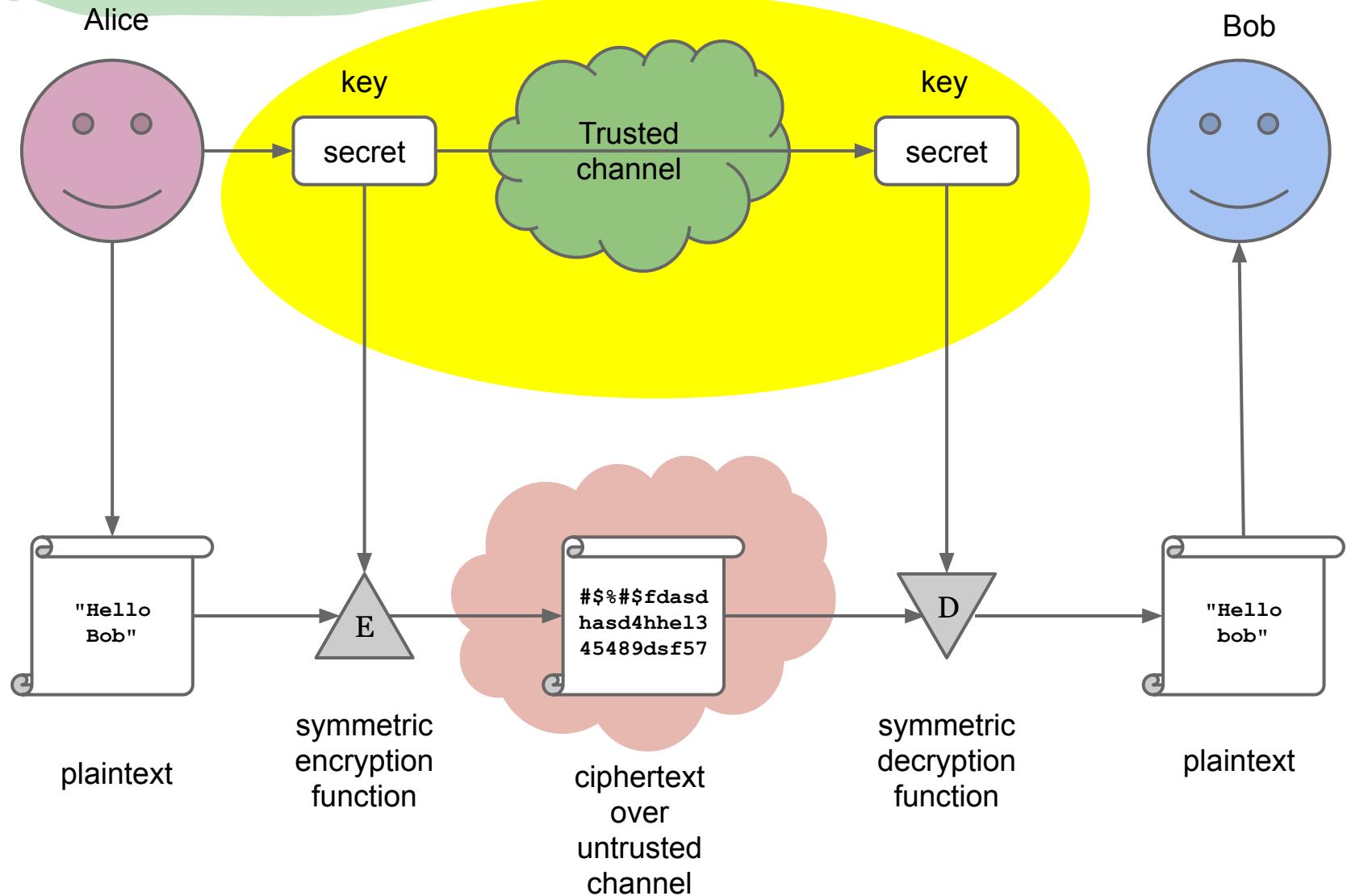
# Symmetric Encryption



# Symmetric Encryption



# Symmetric Encryption



# Symmetric Encryption

- The basic idea of encryption
  - Use key  $K$  to **encrypt** plaintext in ciphertext
  - Use same key  $K$  to **decrypt** ciphertext in plaintext
- Synonyms: shared key encryption, secret key encryption
- **Issue:** how do we agree on the key?
  - Cannot send key on same channel as message!
  - Off-band transmission mechanism needed
- **Issue:** scalability
- A symmetric algorithm is a cocktail...

# First ingredient: substitution

**Substitution:** “replacing **each byte** with another”

Toy example (Caesar cipher)

- replace each letter in a sentence with the one following it by  $K$  positions in the alphabet
- Example: “**SECURE**” becomes “**VHFXUH**” with  $K = 3$

● **Many issues** (it’s a toy example!)

# First ingredient: substitution

**Substitution:** “replacing each byte with another”

Toy example (Caesar cipher)

- replace each letter in a sentence with the one following it by  $K$  positions in the alphabet
- Example: “**SECURE**” becomes “**VHFXUH**” with  $K = 3$

**Many issues** (it’s a toy example!):

- if cipher known, with 25 attempts at most, 13 on average, we have the key: **keyspace too small**

# First ingredient: substitution

**Substitution:** “replacing each byte with another”

Toy example (Caesar cipher)

- replace each letter in a sentence with the one following it by  $K$  positions in the alphabet
- Example: “**SECURE**” becomes “**VHFXUH**” with  $K = 3$

**Many issues** (it’s a toy example!):

- if cipher known, with 25 attempts at most, 13 on average, we have the key: **keyspace too small**.
- **repetitions** and **structure** “visible” in ciphertext: monoalphabetic ciphers are weak (frequency analysis - CPTX only attack) [https://en.wikipedia.org/wiki/Letter\\_frequency](https://en.wikipedia.org/wiki/Letter_frequency)

# Polyalphabetic ciphers (Vigenere Cipher)

m=SECURE

k=SECRET

c=?

--PLAINTEXT--																										
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

# Polyalphabetic ciphers (Vigenere Cipher)

$m = \text{SECURE}$

$k = \text{SECRET}$

$c = K$

--PLAINTEXT--																										
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

# Polyalphabetic ciphers (Vigenere Cipher)

m=SECURE  
k=SECRET  
c=KIELVX

--PLAINTEXT--																										
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

# Second ingredient: transposition

Transposition (or diffusion) means “swapping the values of given bits”

Toy example (matrix):

- Write by rows, read by columns
- Key:  $K = (R, C)$  with  $R * C \sim \text{len}(\text{msg})$

plaintext					ciphertext
H	A	L	L	O	
	E	V	E	R	
Y	O	N	E	!	

**Many issues** (it's a toy example!)

# Example - Diffusion

H	A	L	L	O
	E	V	E	R
Y	O	N	E	!

m= HALLO EVERYONE!

k=(3,5)

c=H YAEOLVNLEEOR!

# Example - Diffusion

m= HALLO

k=(3,5)

c=H A L L O

H	A	L	L	O

R \* C >> len(msq)

15 >> 4

# Second ingredient: transposition

Transposition (or diffusion) means “swapping the values of given bits”

Toy example (matrix):

- Write by rows, read by columns
- Key:  $K = (R, C)$  with  $R * C \sim \text{len}(\text{msg})$

Many issues (it's a toy example!):

- Keyspace still relatively small

plaintext				
H	A	L	L	O
	E	V	E	R
Y	O	N	E	

ciphertext

# Second ingredient: transposition

Transposition (or diffusion) means “swapping the values of given bits”

Toy example (matrix):

- Write by rows, read by columns
- Key:  $K = (R, C)$  with  $R * C \sim \text{len}(\text{msg})$

Many issues (it's a toy example!):

- Keyspace still relatively small

But **repetitions** and **structure** gone

- We now really need to test all possible structures

plaintext				
H	A	L	L	O
	E	V	E	R
Y	O	N	E	

# Perfectly secure cipher

## Definition

- In a *perfect cipher*, for all  $\text{ptx} \in \mathbf{P}$  and  $\text{ctx} \in \mathbf{C}$ ,  
 $\Pr(\text{ptx sent} = \text{ptx}) = \Pr(\text{ptx sent} = \text{ptx} \mid \text{ctx sent} = \text{ctx})$
- In other words: seeing a ciphertext  $c \in \mathbf{C}$  gives us *no information* on what the plaintext corresponding to  $c$  could be

## Question

- The definition is not constructive! Does a perfect cipher exist?
  - If yes, what does it look like?

# Perfectly secure cipher

## Theorem (Shannon 1949)

Any symmetric cipher  $\langle \mathbf{P}, \mathbf{K}, \mathbf{C}, \mathbb{E}, \mathbb{D} \rangle$  with  $|\mathbf{P}| = |\mathbf{K}| = |\mathbf{C}|$  is perfectly secure if and only if

- every key is used with probability  $\frac{1}{|\mathbf{K}|}$
- a unique key maps a given plaintext into a given ciphertext:  
$$\forall (\text{ptx}, \text{ctx}) \in \mathbf{P} \times \mathbf{C}, \exists! k \in \mathbf{K} \text{ s.t. } \mathbb{E}(\text{ptx}, k) = \text{ctx}$$

# Perfectly secure cipher

## Theorem (Shannon 1949)

Any symmetric cipher  $\langle \mathbf{P}, \mathbf{K}, \mathbf{C}, \mathbb{E}, \mathbb{D} \rangle$  with  $|\mathbf{P}| = |\mathbf{K}| = |\mathbf{C}|$  is perfectly secure if and only if

- every key is used with probability  $\frac{1}{|\mathbf{K}|}$
- a unique key maps a given plaintext into a given ciphertext:  
 $\forall (\text{ptx}, \text{ctx}) \in \mathbf{P} \times \mathbf{C}, \exists! k \in \mathbf{K} \text{ s.t. } \mathbb{E}(\text{ptx}, k) = \text{ctx}$

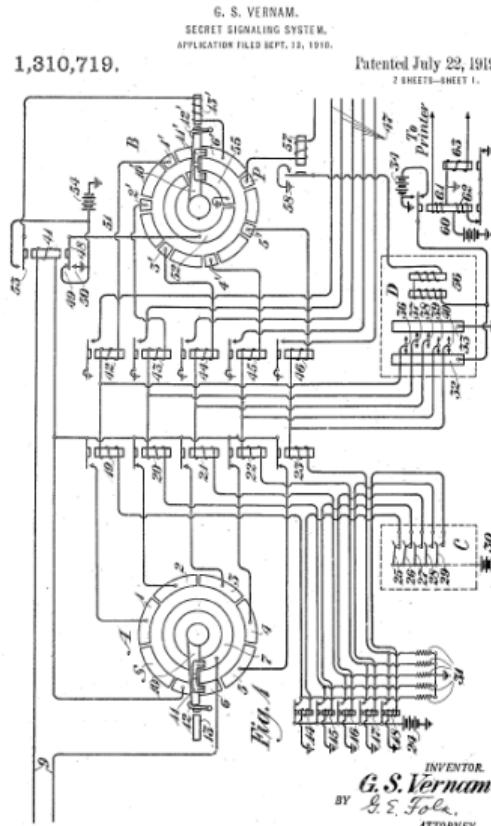
## A simple working example

- Assume  $\mathbf{P}, \mathbf{K}, \mathbf{C}$  to be set of binary strings. The encryption function draws a *uniformly random, fresh* key  $k$  out of  $\mathbf{K}$  each time it is called and computes  
 $\text{ctx} = \text{ptx} \oplus k$

# Anticausal implementations

## A concrete apparatus

- Gilbert Vernam actually patented a telegraphic machine implementing  $ptx \oplus k$  on Baudot code in 1919
- Joseph Mauborgne suggested the use of a random tape containing  $k$
- Using Vernam's encrypting machine with Mauborgne's suggestion implements a perfect cipher



# The One Time Pad: Perfect Cipher

- XOR of a message  $m$  and a random key  $k$  of the same size of  $m$ :  $\text{len}(k) = \text{len}(m)$ 
  - The key is pre-shared and consumed while writing.  
Can never be re-used again!
- The OTP is a **minimal perfect cipher**
  - Minimal because  $|K| = |M|$

Plain text: THIS IS SECRET
OTP-Key : XVHE UW NOPGDZ

---

Ciphertext: Q C P W C O F S R X H S

In groups : QCPWC OFSRX HS

## Key storage/management

- storing key material and changing keys is a nightmare
- perfect cipher broken in practice due to key theft/reuse
- generating random keys was also an issue (and caused breaks)



Photo courtesy of [Cryptomuseum.com](http://Cryptomuseum.com)

# Imperfections and Brute Force

- Real-world algorithms are not perfect ( $|K| < |M|$ ), and so can be broken
  - each ciphertext-plaintext pair leaks a small amount of information (because the key is re-used)
- Only thing unknown is the key (Kerckhoffs)
  - Remember: the algorithm itself is known!
- Brute forcing is possible for any real world cipher
  - Try all possible keys, until **one** produces an output that “makes sense”.
- Perfect ciphers (one time pads) are not vulnerable to Brute Force
  - because trying all the (random) keys will yield all the (possible) plaintexts, which are all equally likely (= no clue)

# Non Perfect Cipher Example

M = UGO (21 7 15)

K=+3

C = XJR (24 10 18)

Bruteforcing: k=-1-2-3.....-26 For each k, he/she shifts all letters...

# Non Perfect Cipher Example

M = UGO (21 7 15)

K=+3

C = XJR (24 10 18)

Bruteforcing: k=-1-2-3.....-26 For each k, he/she shifts all letters...

K=-1 ... M1 = WIQ

# Non Perfect Cipher Example

M = UGO (21 7 15)

K=+3

C = XJR (24 10 18)

Bruteforcing: k=-1-2-3.....-26 For each k, he/she shifts all letters...

K=-1 ... M1 = WIQ

K=-2 ... M2 = VHP

# Non Perfect Cipher Example

M = UGO (21 7 15)

K=+3

C = XJR (24 10 18)

Bruteforcing: k=-1-2-3.....-26 For each k, he/she shifts all letters...

K=-1 ... M1 = WIQ

K=-2 ... M2 = VHP

K=-3 ... M3 = UGO

# “Toy” Perfect Cipher Example

M = UGO (21 7 15)

K=+1+2+3

C = VIR (22 9 18)

Bruteforcing...For each letter try all k=-1-1-1,  
-1-1-2,.....-26-26-26

# “Toy” Perfect Cipher Example

M = UGO (21 7 15)

K=+1+2+3

C = VIR (22 9 18)

Bruteforcing... For each letter try all k=-1-1-1,  
-1-1-2,.....-26-26-26

M1=ADA

M2=XKR

M3=ELM

M4=UGO



# Cryptanalysis: Breaking Ciphers

A real (non perfect) cryptosystem is **broken** if there is a way to break it that is **faster** than brute forcing.

Types of attacks:

- **Ciphertext attack**: analyst has only ciphertexts
- **Known plaintext attack**: analyst has a set of pairs plain-ciphertext
- **Chosen plaintext attack**: analyst can choose plaintexts and obtain their respective ciphertexts

# Example: can you break this?

# The Zip Example

## **Algorithm:** $C = K \text{ xor } M$

- $K(\text{hex}) = \text{AA } \text{BB } \text{CC } \text{DD } \dots \dots \dots \dots \dots \dots \dots \dots$  (repeat the key)
  - $M(\text{hex}) = \text{50 } \text{4B } \text{03 } \text{04 } \text{BA } \text{DA } \text{55 } \text{55 } \dots \dots \dots$  (and so on)

XOR

  - $C(\text{hex}) = \text{FA } \text{F0 } \text{CF } \text{D9 } \text{10 } \text{61 } \text{99 } \text{88 } \dots \dots \dots \dots \dots \dots \dots \dots$

# *PERFECT OR NOT ?*

# The Zip Example

**Algorithm:**  $C = K \text{ xor } M$

- $K(\text{hex}) = \text{AA } \text{BB } \text{CC } \text{DD } \dots \dots \dots \dots \dots \dots \dots \dots$  (repeat the key)
  - $M(\text{hex}) = \text{50 } \text{4B } \text{03 } \text{04 } \text{BA } \text{DA } \text{55 } \text{55 } \dots \dots \dots$  (and so on)

XOR

  - $C(\text{hex}) = \text{FA } \text{F0 } \text{CF } \text{D9 } \text{10 } \text{61 } \text{99 } \text{88 } \dots \dots \dots \dots \dots \dots \dots \dots$

*NOT PERFECT* ->  $\text{len}(k) < \text{len}(M)$  ->  $k$  is reused

# The Zip Example: Can you break this?

**Algorithm:**  $C = K \text{ xor } M$

- $K(\text{hex}) = \text{AA } \text{BB } \text{CC } \text{DD } \dots \dots \dots \dots \dots \dots \dots \dots$  (repeat the key)
  - $M(\text{hex}) = \text{50 } \text{4B } \text{03 } \text{04 } \text{BA } \text{DA } \text{55 } \text{55 } \dots \dots \dots$  (and so on)

XOR

  - $C(\text{hex}) = \text{FA } \text{F0 } \text{CF } \text{D9 } \text{10 } \text{61 } \text{99 } \text{88 } \dots \dots \dots \dots \dots \dots \dots \dots$

NOT PERFECT ->  $\text{len}(k) < \text{len}(M)$  ->  $k$  is reused

- $C = K \text{ xor } M$
  - $K = M \text{ xor } C$
  - $K = \text{XXXX...} \text{ xor } FA\ F0\ CF\ D9\ ...$

```
michele@starkiller ~/Desktop
└─➤ xxd test.zip | head
00000000: 504b 0304 1400 0808 0800 bb74 3150 0000 PK.....t1P..
00000010: 0000 0000 0000 0000 0000 1400 0000 6974 .....it
00000020: 2d36 3931 3439 3678 3038 3931 3635 2e70 -691496x089165.p
00000030: 6466 cccb 6554 5ccb ba36 8abb bbbb 5be3 df..eT\..6...[.
00000040: eeee ee6e 8dbb bb6b 20b8 8510 9ce0 ee2e ...n...k .....
00000050: 0182 bb3b c125tb806 bb49 d65e 7baf 7dce ...;%...I.^{.}.
00000060: face ddf7 8cef c71d 73d4 acaa 77be 56d2 .....s...w.V.
00000070: b3bb 9ef1 34a5 b2b8 2423 0b13 3b1c e5b7 ...4...$#..;...
00000080: 9dc9 5938 0e12 6612 4753 1b12 387e 7e38 ..Y8..f.GS..8~~8
00000090: 80ba b713 9004 20e1 e526 a5e6 66e2 0684 ..... .&..f...
```

```
michele@starkiller ~/Desktop
└─➤ xxd test2.zip | head
00000000: 504b 0304 1400 0808 0000 c051 3050 0000 PK.....Q0P..
00000010: 0000 0000 0000 0000 0000 3200 0000 5241 .....2...RA
00000020: 4d53 4553 2046 494e 414c 2052 6576 6965 MSES FINAL Revie
00000030: 7720 5054 2050 6572 7370 6563 7469 7665 w PT Perspective
00000040: 204a 616e 2032 3020 2831 292e 7070 7478 Jan 20 (1).pptx
00000050: 504b 0304 1400 0600 0800 0000 2100 4fac PK.....!.0.
00000060: d3c4 4302 0000 d416 0000 1300 0802 5b43 ..C.....[C
00000070: 6f6e 7465 6e74 5f54 7970 6573 5d2e 786d ontent_Types].xm
00000080: 6c20 a204 0228 a000 0200 0000 0000 0000 l ...(. .....
00000090: 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
```

```
michele@starkiller ~/Desktop
└─➤ xxd test.zip | head
00000000: 504b 0304 1400 0808 0800 bb74 3150 0000 PK.....t1P..
00000010: 0000 0000 0000 0000 0000 1400 0000 6974 .....it
00000020: 2d36 3931 3439 3678 3038 3931 3635 2e70 -691496x089165.p
00000030: 6466 cccb 6554 5ccb ba36 8abb bbbb 5be3 df..eT\..6...[.
00000040: eeee ee6e 8dbb bb6b 20b8 8510 9ce0 ee2e ...n...k .....
00000050: 0182 bb3b c125tb806 bb49 d65e 7baf 7dce ...;%...I.^{.}.
00000060: face ddf7 8cef c71d 73d4 acaa 77be 56d2 .....s...w.V.
00000070: b3bb 9ef1 34a5 b2b8 2423 0b13 3b1c e5b7 ...4...$#..;...
00000080: 9dc9 5938 0e12 6612 4753 1b12 387e 7e38 ..Y8..f.GS..8~~8
00000090: 80ba b713 9004 20e1 e526 a5e6 66e2 0684 ..... .&..f...
```

```
michele@starkiller ~/Desktop
└─➤ xxd test2.zip | head
00000000: 504b 0304 1400 0808 0000 c051 3050 0000 PK.....Q0P..
00000010: 0000 0000 0000 0000 0000 3200 0000 5241 .....2...RA
00000020: 4d53 4553 2046 494e 414c 2052 6576 6965 MSES FINAL Revie
00000030: 7720 5054 2050 6572 7370 6563 7469 7665 w PT Perspective
00000040: 204a 616e 2032 3020 2831 292e 7070 7478 Jan 20 (1).pptx
00000050: 504b 0304 1400 0600 0800 0000 2100 4fac PK.....!.0.
00000060: d3c4 4302 0000 d416 0000 1300 0802 5b43 ..C.....[C
00000070: 6f6e 7465 6e74 5f54 7970 6573 5d2e 786d ontent_Types].xm
00000080: 6c20 a204 0228 a000 0200 0000 0000 0000 l ...(. .....
00000090: 0000 0000 0000 0000 0000 0000 0000 0000 ..... .
```

# The Zip Example

## **Algorithm:** $C = K \text{ xor } M$

- $K(\text{hex}) = \underline{\text{AA } \text{BB } \text{CC } \text{DD}} \dots \dots \dots \dots \dots \dots \dots$  (repeat the key)
  - $M(\text{hex}) = \underline{\text{50 } \text{4B } \text{03 } \text{04}}$  BA DA 55 55 ... ... ... (and so on)  
XOR
  - $C(\text{hex}) = \text{FA } \text{F0 } \text{CF } \text{D9 } \text{10 } \text{61 } \text{99 } \text{88} \dots \dots \dots \dots \dots \dots \dots \dots$
  - $K = M \text{ xor } C$
  - $K = \underline{\text{50 } \text{4B } \text{03 } \text{04}} \text{ xor } \text{FA } \text{F0 } \text{CF } \text{D9} = \underline{\text{AA } \text{BB } \text{CC } \text{DD}} \rightarrow$

# The Zip Example

## **Algorithm:** $C = K \text{ xor } M$

- $K(\text{hex}) = \underline{\text{AA } \text{BB } \text{CC } \text{DD}} \dots \dots \dots \dots \dots \dots \dots \dots$  (repeat the key)
  - $M(\text{hex}) = \underline{\text{50 } \text{4B } \text{03 } \text{04}}$  BA DA 55 55 .. .. .. (and so on)  
XOR
  - $C(\text{hex}) = \text{FA } \text{F0 } \text{CF } \text{D9 } \text{10 } \text{61 } \text{99 } \text{88} \dots \dots \dots \dots \dots \dots \dots \dots$
  - $K = M \text{ xor } C$
  - $K = \underline{\text{50 } \text{4B } \text{03 } \text{04}} \text{ xor } \text{FA } \text{F0 } \text{CF } \text{D9} = \underline{\text{AA } \text{BB } \text{CC } \text{DD}} \rightarrow$

- KNOWN PLAINTEXT ATTACK

# Computationally secure cryptography

## A more practical assumption

- Build a cipher so that a successful attack is also able to solve a hard computational problem efficiently
  - Solve a generic nonlinear Boolean simultaneous equation set
  - Factor large integers / find discrete logarithms
  - Decode a random code/find shortest lattice vector

## Can we avoid assumptions?

- Open question: prove an exponential lower bound for the time taken to solve a hard problem, which has efficiently verifiable solution
  - it would shift from a computational security assumption to a theorem
  - ... and prove  $P \neq NP$  as a corollary

# Factor Large Integers (hints)

If  $p$  and  $q$  are two *large primes*:

- computing  $n = p * q$  is **easy**
- but given  $n$  it is painfully **slow** to get  $p$  and  $q$ 
  - quadratic sieve field, basically “try all primes until you get to the smaller between  $p$  and  $q$ ”

Here “slow/difficult” means “computationally very intensive”, for all practical purposes the problem requires bruteforce over all possible values of  $x$

# Discrete logarithm (hints)

- If  $y = a^x$  then  $x = \log_a y$  (Math 101)
- given  $x, a, p,$ 
  - it is **easy** to compute  $y = a^x \bmod p,$
  - but knowing  $y$ , it is **difficult** to compute  $x$

Different problem than factorization, but it can be shown that they are related

## Outline of the method

- 
- ① Define the ideal attacker behaviour
  - ② Assume a given computational problem is hard
  - ③ Prove that any non ideal attacker solves the hard problem

## How to represent attacker and properties?

- Attacker represented as a program able to call given libraries
- Libraries implement the cipher at hand
- Define the security property as answering to a given question
- The attacker wins the game if it breaks the security property more often than what is possible through a random guess



## Motivation and assumption

- We want to use a finite-length key and a Vernam cipher
  - We somehow need to “expand” the key
- We assume that the attacker can only perform  $\text{poly}(\lambda)$  computations

## Definition

A CSPRNG is a deterministic function  $\text{PRNG}: \{0, 1\}^\lambda \rightarrow \{0, 1\}^{\lambda+I}$  whose output cannot be distinguished from an uniform random sampling of  $\{0, 1\}^{\lambda+I}$  in  $\mathcal{O}(\text{poly}(\lambda))$ .  $I$  is the CSPRNG **stretch**.



## Existence

- In practice, we have only candidate CSPRNGs
  - We have no **proof** that a function PRNG exists
  - Proving that a CSPRNG exists implies directly  $P \neq NP$

## Practical constructions

- Building a CSPRNG “from scratch” is possible, but it is not the way they are commonly built (not efficient)
- Practically built with another building block: PseudoRandom Permutations (PRPs)
  - defined starting from PseudoRandom Functions (PRFs)

# Random Functions (RFs)

## Randomly drawing a function

- Consider the set  $\mathbf{F} = \{f : \{0, 1\}^{in} \rightarrow \{0, 1\}^{out}; in, out \in \mathbb{N}\}$
- A uniformly randomly sampled  $f \xleftarrow{\$} \mathbf{F}$  can be encoded by a  $2^{in}$  entries table, each entry  $out$  bit wide.  $|\mathbf{F}| = (2^{out})^{2^{in}}$

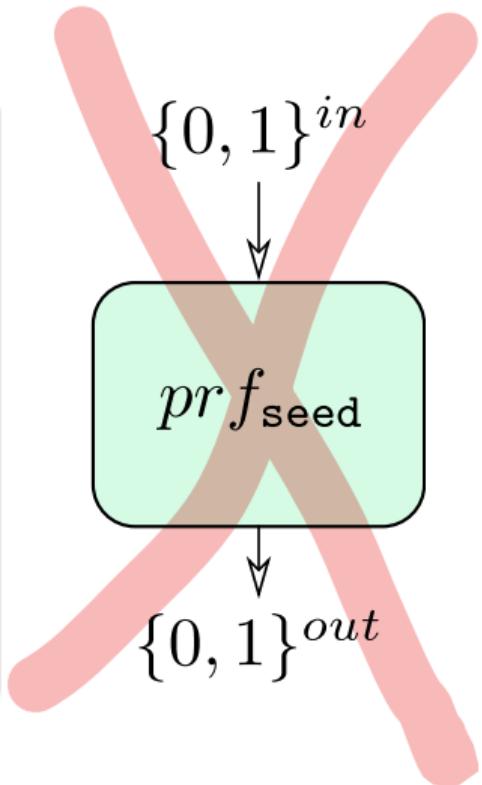
## Toy example $in = 2, out = 1$

- $\mathbf{F} = \{f : \{0, 1\}^2 \rightarrow \{0, 1\}^1\}$  is the set of the 16 Boolean functions w/ two inputs
- Each one is represented by a 4-entry truth table
- Intuitively, the functions are 16 as there are  $2^4 = 16 = (2^1)^{2^2}$  tables

# Pseudorandom Functions (PRFs)

## Definition

- A function  $\text{prf}_{\text{seed}} : \{0, 1\}^{\text{in}} \rightarrow \{0, 1\}^{\text{out}}$  taking an input and a  $\lambda$  bits seed.
- The entire  $\text{prf}_{\text{seed}}$  is described by the value of the seed
- It cannot be told apart from a random  $f \in \{f : \{0, 1\}^{\text{in}} \rightarrow \{0, 1\}^{\text{out}}\}$  in  $\text{poly}(\lambda)$
- That is, if they give you  $a \in \{f : \{0, 1\}^{\text{in}} \rightarrow \{0, 1\}^{\text{out}}\}$ , you can't tell which one of the following is true
  - $a = \text{prf}_{\text{seed}}(\cdot)$  with seed  $\xleftarrow{\$} \{0, 1\}^\lambda$
  - $b \xleftarrow{\$} \mathbf{F}$ , where  $\mathbf{F} = \{f : \{0, 1\}^{\text{in}} \rightarrow \{0, 1\}^{\text{out}}\}$



# Pseudorandom Permutations (PRPs)

## Pseudorandom Permutation definition

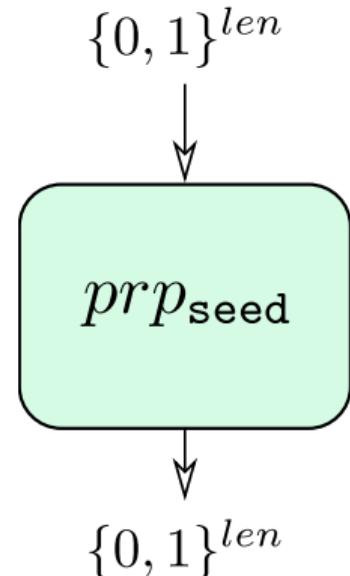
- A **bijective** PRF:  $prf_{seed} : \{0, 1\}^{len} \rightarrow \{0, 1\}^{len}$

## Wrapping your mind around it

- It is uniquely identified by the value of the seed
- It is not possible to tell apart in  $\text{poly}(\lambda)$  from a RF
- It's a permutation of all the possible  $\{0, 1\}^{len}$  strings

## Operatively speaking

- acts on a **block** of bits outputs another one of the same size
- the output “looks unrelated” to the input
- its action is fully identified by the seed
  - Useful to think of the seed as a key



## The issue

- No formally proven PRP exists, yet
  - again, its existence would imply  $P \neq NP$

## Typical construction

- ① Compute a small bijective Boolean function  $f$  of input and key
- ② Compute  $f$  again between the previous output and the key
- ③ Repeat 2 until you're satisfied

# In the real world...

## Practical solution: public scrutiny

- Modern PRPs are the outcome of public contests
- Cryptanalytic techniques provide ways ( $=\text{poly}(\lambda)$  tests) to detect biases in their outputs: good designs are immune

## PRPs a.k.a. Block ciphers

- Concrete PRPs go by the historical name of **block ciphers**
- Considered broken if, with less than  $2^\lambda$  operations, they can be told apart from a PRP, e.g., via:
  - Deriving the input corresponding to an output without the key
  - Deriving the key identifying the PRP, or reducing the amount of plausible ones
  - Identifying non-uniformities in their outputs
- The key length  $\lambda$  is chosen to be large enough so that computing  $2^\lambda$  guesses is not **practically feasible**

# Keyspace and Brute Forcing

Keyspace generally measured in bits

- Attack time exponential on the number of bits (i.e., 33 bits need twice the time of 32)

Basic Solution ?

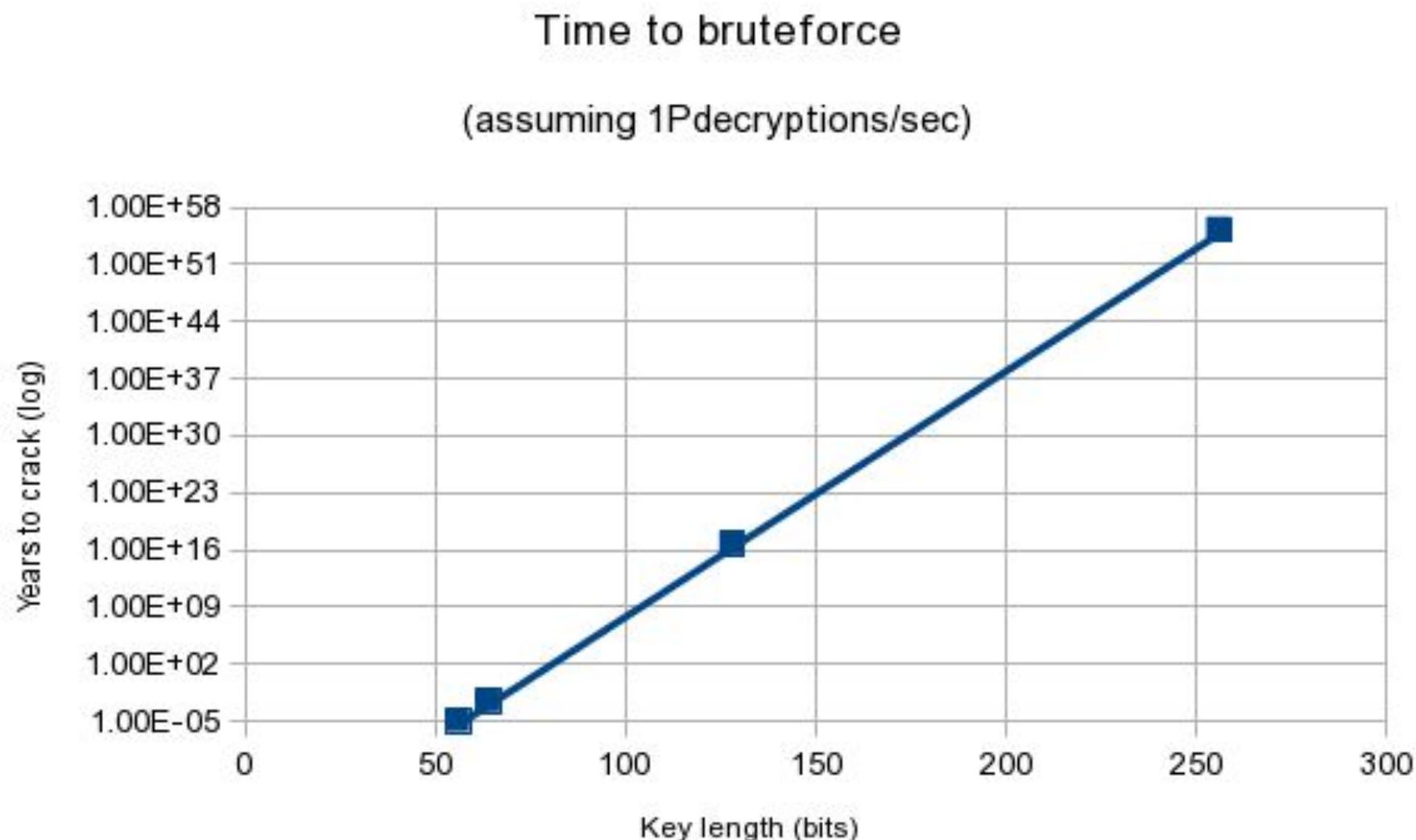


# Keyspace and Brute Forcing

Keyspace generally measured in bits

- Attack time exponential on the number of bits (i.e., 33 bits need twice the time of 32)
- Need to balance computational power vs key length.

# Keyspace vs. Time for Brute Forcing



# Quantifying computational unfeasibility

Boolean operations vs. energy to bring from 20 °C → 100 °C

- $2^{65}$  op.s ≈ an Olympic swimming pool
- $2^{80}$  op.s ≈ the annual rainfall on the Netherlands
- $2^{114}$  op.s ≈ all water on Earth

Practically acceptable unfeasibility

- Legacy level security: at least  $2^{80}$  Boolean operations
- 5 to 10 years security: at least  $2^{128}$  Boolean operations
- Long term security: at least  $2^{256}$  Boolean operations

# Widespread block ciphers

## C Advanced Encryption Standard (AES)

- 128 bit block, three key lengths: 128, 192 and 256 bits
- Selected after a 3 years public contest in 2000-10-2 by NIST out of 15 candidates, re-standardized by ISO/IEC
- ARMv8 and AMD64 include dedicated instructions accelerating its computation (hitting 3+ GB/s)

## D Data Encryption Algorithm (DEA, a.k.a. DES)

- Legacy standard by NIST (1977), the key is too short (56b)
- Patch via triple encryption,  $\lambda = 112$  equivalent security
- Still found in some legacy systems, officially deprecated

# Case Study: DES

Originally designed by IBM (1973-1974)

Its core function is an S-box (a key-dependent substitution)

It uses a 56 bit key ( $2^{56}$  keyspace)

# Case Study: DES vs. NSA

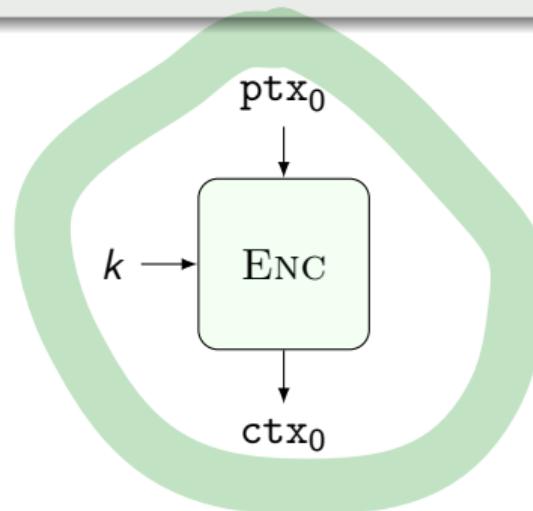
In 1976 it becomes a US standard; its S-boxes are "redesigned" by the NSA

- **Late 1980s:** *differential cryptanalysis* discovered
- **1993:** shown that the original S-boxes would have made DES vulnerable to the differential cryptanalysis, whereas the NSA-designed S-boxes were specifically immune to that.
- **Wait!** Wasn't differential cryptanalysis unknown until late 1980s? *Mmmmmmmaybe* the NSA knew about differential crypto in the 70s.

# An Electronic CodeBook (ECB)

A first attempt to encryption with PRPs

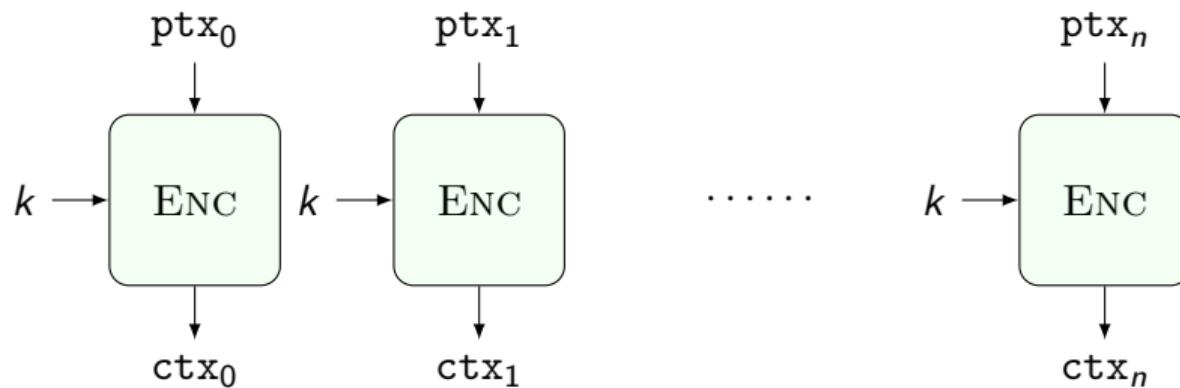
- It's ok to encrypt a plaintext  $\leq$  block size with a block cipher



# An Electronic CodeBook (ECB)

A first attempt to encryption with PRPs

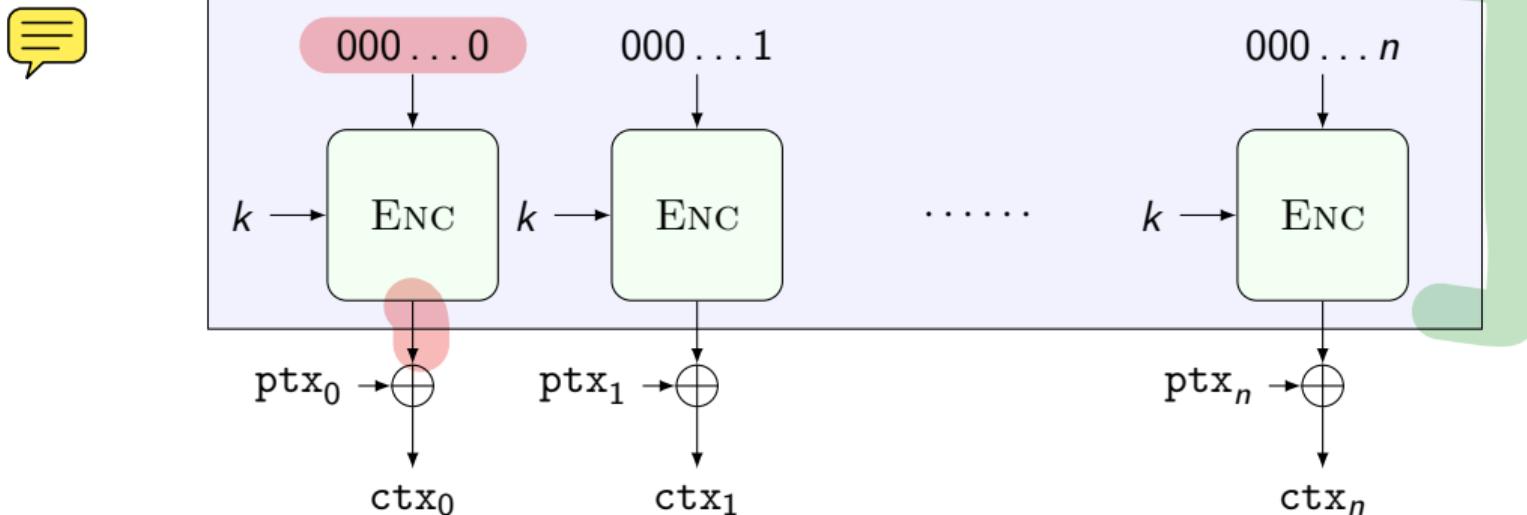
- It's ok to encrypt a plaintext  $\leq$  block size with a block cipher
- An extension to multiple blocks could be split-and-encrypt
  - Is it good (equivalent to Vernam fed with a CSPRNG)?



# Counter (CTR) mode

## Getting it right

- The boxed construction is provably a PRNG if Enc is a PRP
- There is nothing special in the starting point of the counters





## Confidentiality achieved ... for CoA

- Up to now, the attacker knew only ciphertext material

## Confidentiality against Chosen Plaintext Attacks (CPAs)

- Our attacker knows a set of plaintexts which can be encrypted
- He wants to understand **which** one is being encrypted
- Ideal attacker: cannot tell which plaintext was encrypted out of two *he chose* (having the same length)
- Feels strange, but it happens with:
  - management data packets in network protocols (e.g., ICMP)
  - telling apart encrypted commands to a remote host

## No deterministic encryption

- The CTR mode of operation is insecure against CPA
  - The encryption is deterministic: same ptxs → same ctx

## Decryptable nondeterministic encryption

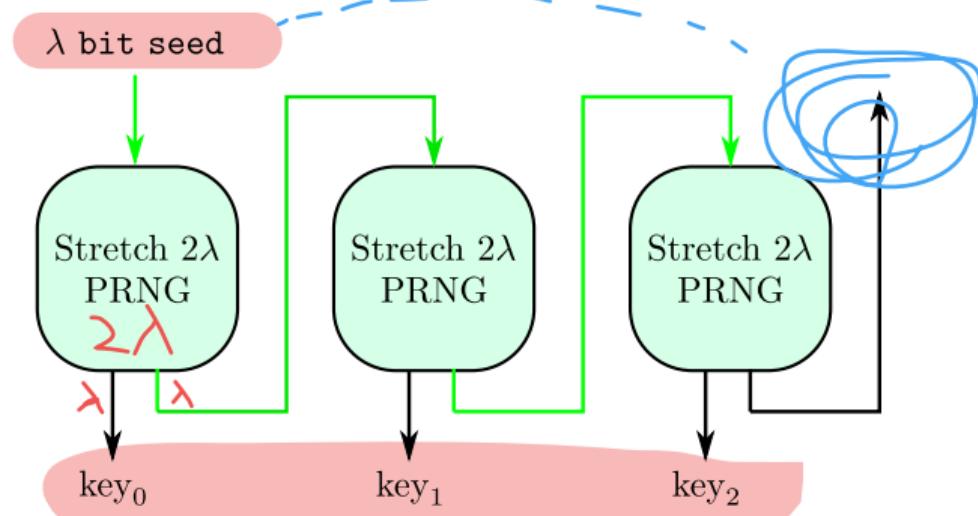
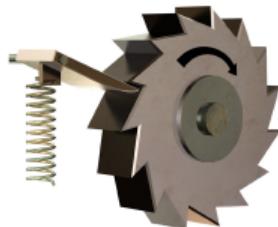
- ① ~~Rekeying: change the key for each block with a ratchet~~
- ② ~~Randomize the encryption: add (removable) randomness to the encryption  
(change mode of employing PRP)~~
- ③ Numbers used ONCE (NONCEs): in the CTR case, pick a NONCE as the counter starting point. NONCE is public



# Symmetric ratcheting

## Getting it right

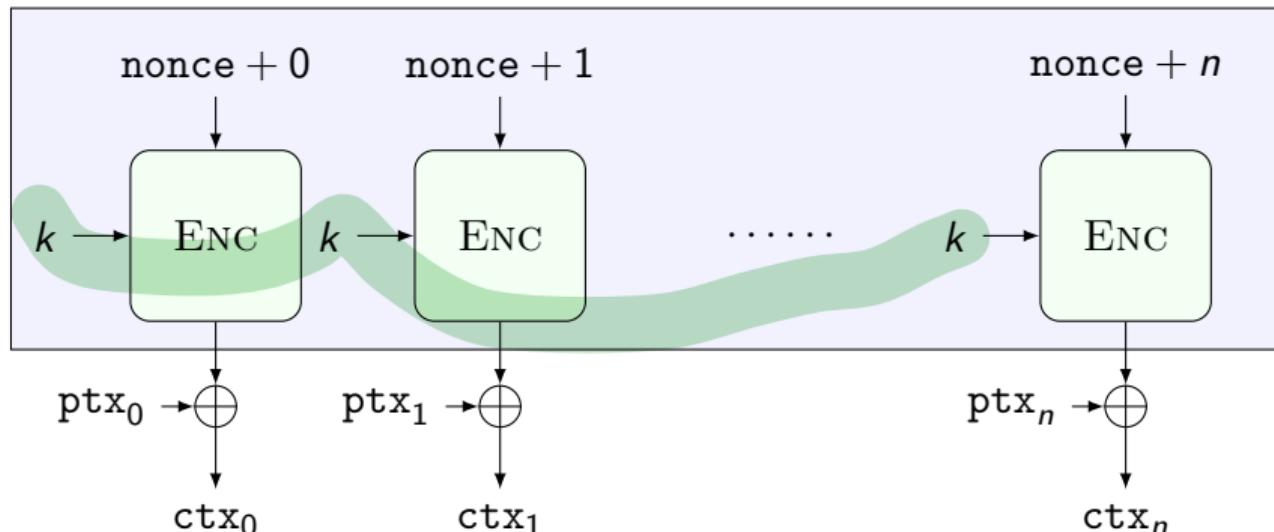
- The construction takes the name from the mechanical component: it is not possible to roll-back the procedure once you delete the value carried by green arrows



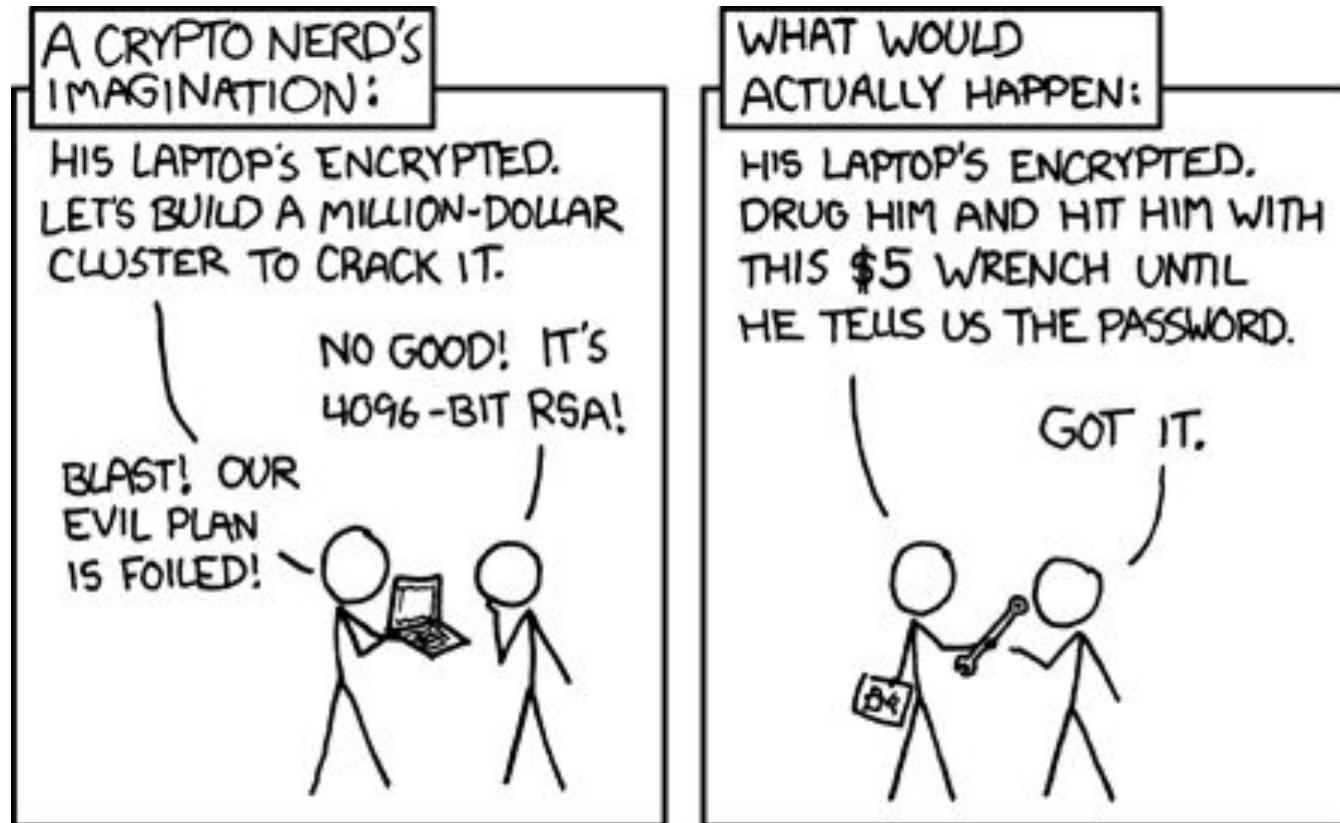
# CPA-Secure Counter (CTR) mode

## Getting it right

- Picking the counter start as a NONCE generates different bitstreams to be xor-ed with the ptx each time
- The same plaintext encrypted twice is turned into two different, random-looking, ciphertexts



# A good point to remember...



# Malleability and active attackers

## Malleability

- Making changes to the ciphertext (not knowing the key) maps to predictable changes in the plaintext
  - Think about AES-CTR and AES-ECB
- Can be creatively abused to build decryption attacks
- Can be turned into a feature (homomorphic encryption)

## How to avoid malleability

- Design an intrinsically non malleable scheme (non trivial)
- Add a mechanism ensuring data integrity (against attackers)

# Providing data integrity

Confidentiality  $\not\Rightarrow$  Integrity

- Up to now our encryption schemes provide confidentiality
- Changes in the ciphertext are undetected (at best)

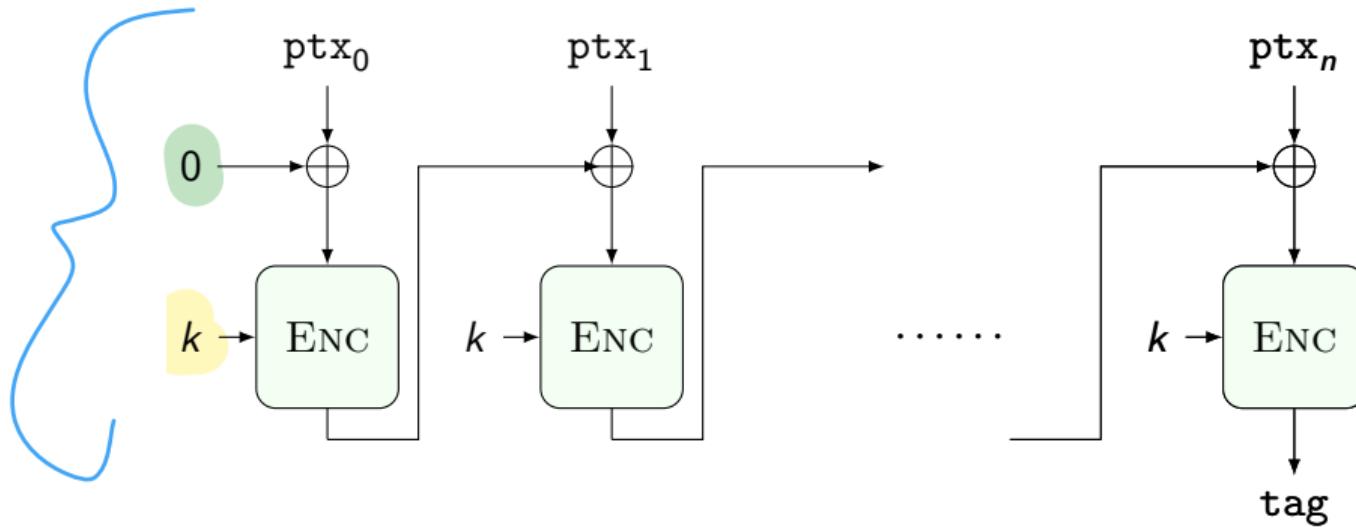
Message Authentication Codes (MAC)

- Add a small piece of information (tag) allowing us to test for the message **integrity** of the encrypted message itself
  - Adding it to the plaintext and then encrypting is not good
- Nomenclature misleads: MACs do not provide data authentication

## Definition

- A MAC is constituted by a pair of functions:
  - COMPUTE\_TAG(string,key): returns the tag for the input string
  - VERIFY\_TAG(string,tag,key): returns true or false
- Ideal attacker model:
  - knows as many message-tag pairs as he wants
  - cannot forge a valid tag for a message for which he does not know it already
  - forgery also includes tag splicing from valid messages
- N.B. the tag creating entity and the verifying entity must both know the same secret key
  - The tag verifier is able to create a valid tag too
  - ... and there goes the non-repudiation property

# How to build a MAC? the CBC-MAC



## Building a MAC with a PRP (block cipher)

- The CBC-MAC is secure for prefix free messages (why?)
- Encrypting the tag once more fixes (provably) the issue

## Browser cookies

- HTTP cookies are a “note to self” for the HTTP server<sup>a</sup>
- The note should not be tampered between server reads
- Solution: server runs  $\text{COMPUTE\_TAG(cookie, } k)$  and stores both the (cookie,tag)

---

<sup>a</sup>You can find a two slides cookies refresher at slides 40-41 of  
[https://polimi365-my.sharepoint.com/:b/r/personal/10032133\\_polimi\\_it/Documents/FCI/2-Livello\\_Applicativo\\_v2020.pdf](https://polimi365-my.sharepoint.com/:b/r/personal/10032133_polimi_it/Documents/FCI/2-Livello_Applicativo_v2020.pdf)

## Later in the course

- Mitigating SYN-based denial of service attacks (SYN Cookies)
- Time-based two-factor authentication mechanisms (TOTP/HOTP)

## Testing integrity

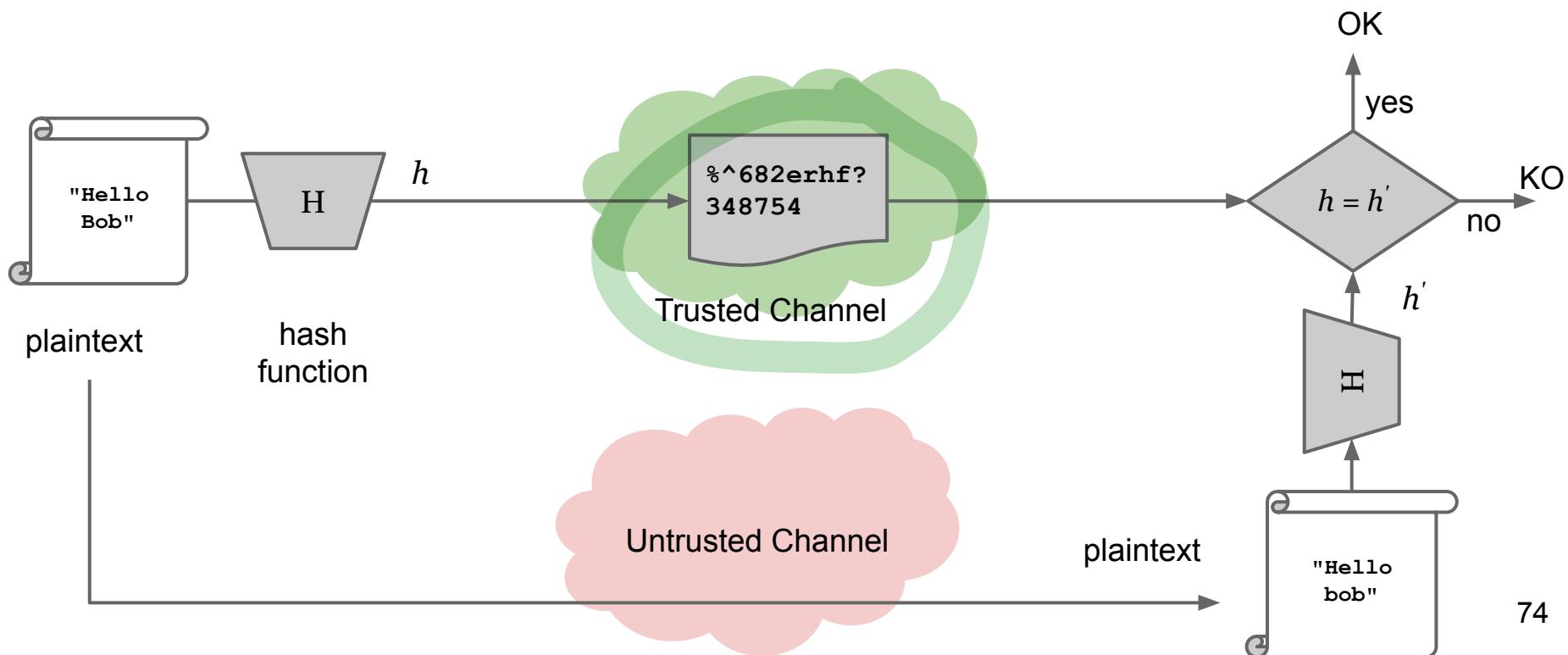
- Testing the integrity of a file requires us to compare it bit by bit with an intact copy or read it entirely to compute a MAC
- It would be **fantastic** to test only short, fixed length strings independently from the file size, representing the file itself
  - Major roadblock: there is a lower bound to the number of bits to encode a given content without information loss
- Can we build something close to the ideal scenario?



# What is a Hash Function

A function  $H()$  that maps arbitrary-length input  $x$  on fixed-length output,  $h$  (*digest*)

- Need to be **Fast**



# What is a Hash Function

A function  $H()$  that maps arbitrary-length input  $x$  on fixed-length output,  $h$

- Need to be **Fast**
- **Collisions:** codomain “smaller” than domain.

**Computationally infeasible to find:**

- input  $x$  such that  $H(x) = h$  (given a specific hash)
  - **preimage attack resistance**
- input  $y$  s.t.  $y \neq x$  and  $H(y) = H(x)$ , with a given  $x$ 
  - **second preimage attack resistance**
- couples of inputs  $\{x, y\}$  s.t.  $H(x) = H(y)$ 
  - **collision resistance**

# Attacks to Hash Functions (1)

Hash functions may be *broken*.

**1. Arbitrary collision or (1st or 2nd) preimage attack:**

Given a specific hash  $h$ , the attacker can find

$x$  such that  $H(x) = h$

or, equivalently, given a specific input  $x$  can find

$y$  such that  $y \neq x$  and  $H(y) = H(x)$

faster than brute forcing.

With a  $n$ -sized hash function, *random collisions* can happen in  $(2^{n-1})$  cases.

# Attacks to Hash Functions (2)

## 2. Simplified collision attack:

The attacker can generate colliding couples

$$\{x, y\} \text{ s.t. } H(x) = H(y)$$

faster than brute forcing.

Random collisions can happen in  $(2^{n/2})$  cases because of the birthday paradox:

- given  $n$  randomly chosen people, some pairs will have same birthday
  - probability = 100% if  $n = 367$
  - probability = 99.9% if  $n = 70$  people
  - probability = 50% if  $n = 23$  people, and so on...
- vs. very low chances that some of you are born on a specific date

## A pseudo-unique labeling function

- A cryptographic hash is a function  $H : \{0, 1\}^* \rightarrow \{0, 1\}^l$  for which the following problems are computationally hard
  - ① given  $d = H(s)$  find  $s$  (1st preimage)
  - ② given  $s, d = H(s)$  find  $r \neq s$  with  $H(r) = d$  (2nd preimage)
  - ③ find  $r, s; r \neq s$ , with  $H(s) = H(r)$  (collision)
- Ideal behaviour of a concrete cryptographic hash:
  - ① finding 1st preimage takes  $\mathcal{O}(2^d)$  hash computations guessing  $s$
  - ② finding 2nd preimage takes  $\mathcal{O}(2^d)$  hash comp.s guessing  $r$
  - ③ finding a collision takes  $\approx \mathcal{O}(2^{\frac{d}{2}})$  hash computations
- The output bitstring of a hash is known as a **digest**

# Concrete hash functions

## What to use

- SHA-2 was privately designed (NSA),  $d \in \{256, 384, 512\}$
- SHA-3 followed a public design contest (similar to AES), selected among  $\approx 60$  candidates,  $d \in \{256, 384, 512\}$
- Both currently unbroken and widely standardized (NIST, ISO)

## What not to use

- SHA-1:  $d = 160$ , collision-broken [6] (obtainable in  $\approx 2^{61}$  op.s)
- MD-5: horribly broken [7]. Collisions in  $2^{11}$ , public tools online [5]
  - In particular, collisions with arbitrary input prefixes in  $\approx 2^{40}$

# Uses for hash functions

## Pseudonymized match

- Store/compare hashes instead of values (e.g., Signal contact discovery)

## MACs

- Building MACs: generate tag hashing together the message and a secret string, verify tag recomputing the same hash
- A field-proven way of combining message and secret is HMAC
  - Standardized (RFC 2104, NIST FIPS 198)
  - Uses a generic hash function as a plug-in, combination denoted as HMAC-hash\_name
  - HMAC-SHA1 (!), HMAC-SHA2 and HMAC-SHA3 are ok

## Forensic use

- Write down only the hash of the disk image you obtained in official documents

# Game changing ideas

## Features we would like to have

- Agreeing on a short secret over a public channel
- Confidentially sending a message over a public authenticated channel without sharing a secret with the recipient
- Actual data authentication

## Solution: asymmetric cryptosystems

- Before 1976: rely on human carriers / physical signatures
- DH key agreement (1976) / Public key encryption (1977)
- Digital signatures (1977)

# Asymmetric Cryptosystems

Confidentiality (plus something more)

# The Diffie-Hellman key agreement

## Goal

- Make two parties share secret value w/ only public messages

## Attacker model

- Can eavesdrop anything, but not tamper
- The Computational Diffie-Hellman assumption should hold

## CDH Assumption

- Let  $(\mathbf{G}, \cdot) \equiv \langle g \rangle$  be a finite cyclic group, and two numbers  $a, b$  sampled unif. from  $\{0, \dots, |\mathbf{G}| - 1\}$  ( $\lambda = \text{len}(a) \approx \log_2 |\mathbf{G}|$ )
- given  $g^a, g^b$  finding  $g^{ab}$  costs more than  $\text{poly}(\log |\mathbf{G}|)$
- Best current attack approach: find either  $b$  or  $a$  (discrete log problem)

## Key agreement between Alice and Bob

- Alice: picks  $a \xleftarrow{\$} \{0, \dots, |\mathbf{G}| - 1\}$ , sends  $g^a$  to Bob
- Bob: picks  $b \xleftarrow{\$} \{0, \dots, |\mathbf{G}| - 1\}$ , sends  $g^b$  to Alice
- Alice: gets  $g^b$  from Bob and computes  $(g^b)^a$
- Bob: gets  $g^a$  from Alice and computes  $(g^a)^b$
- $(\mathbf{G}, \cdot)$  is commutative  $\rightarrow (g^b)^a = (g^a)^b$ , we're done!

## Groups used in practice

- A subgroup  $(\mathbf{G}, \cdot)$  of  $(\mathbb{Z}_n^*, \cdot)$  (integers mod  $n$ ), breaking CDH takes  
$$\min \left( \mathcal{O} \left( e^{k(\log(n))^{\frac{1}{3}} (\log(\log(n)))^{\frac{2}{3}}} \right), \mathcal{O}(2^{\frac{\lambda}{2}}) \right)$$
- EC points w/ dedicated addition, breaking CDH takes  $\mathcal{O}(2^{\frac{\lambda}{2}})$

# Example: Diffie-Hellman Agreement

- Used by Alice and Bob to agree on a **secret** over an **insecure channel**
  - two people talk in the middle of the classroom, everybody hears them, but at the end only those two people know a secret, and nobody else
- One-way trapdoor/hard problem: **discrete logarithm**
  - If  $y = a^x$  then  $x = \log_a y$  (Math 101)
  - given  $x, a, p$ , it is easy to compute  $y = a^x \bmod p$ , but knowing  $y$ , it is difficult to compute  $x$
  - Here “difficult” means “*computationally very intensive*”, for all practical purposes the problem requires bruteforce over all possible values of  $x$

# How does D-H work (1) - Example

Pick  $p$  prime,  $a$  primitive root of  $p$ , public

**Primitive root:** a number  $a$  such that raising it to any number between 1 and  $(p - 1)$ , mod  $p$ , we obtain each number between 1 and  $(p - 1)$

- Example: 3 is a primitive root of 7 because
  - $3^1 \text{ mod } 7 = 3$
  - $3^2 \text{ mod } 7 = 2$
  - $3^3 \text{ mod } 7 = 6$
  - $3^4 \text{ mod } 7 = 4$
  - $3^5 \text{ mod } 7 = 5$
  - $3^6 \text{ mod } 7 = 1$

So let  $a = 3$ ,  $p = 7$  known to everyone in the system

# How does D-H work (2) - Keys

**Secret** number (undisclosed): They pick a number  $X$  in  $[1, 2, \dots, (p-1)]$

Alice  $X_A$   
Bob  $X_B$

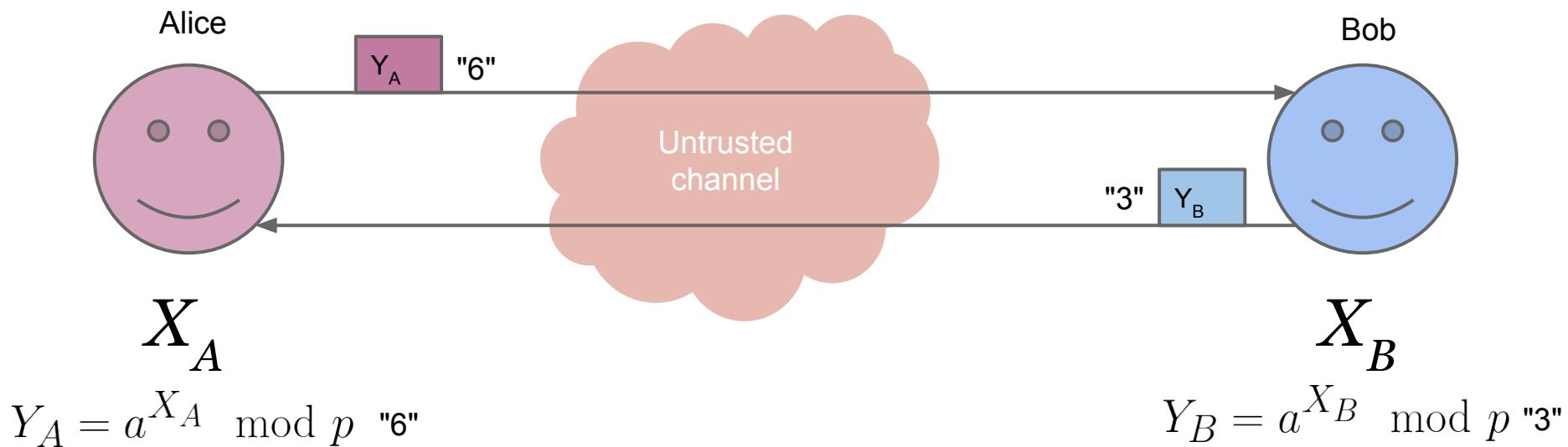
$$X_A = 3$$
$$X_B = 1$$

**Public** number (disclosed to everyone): They compute:

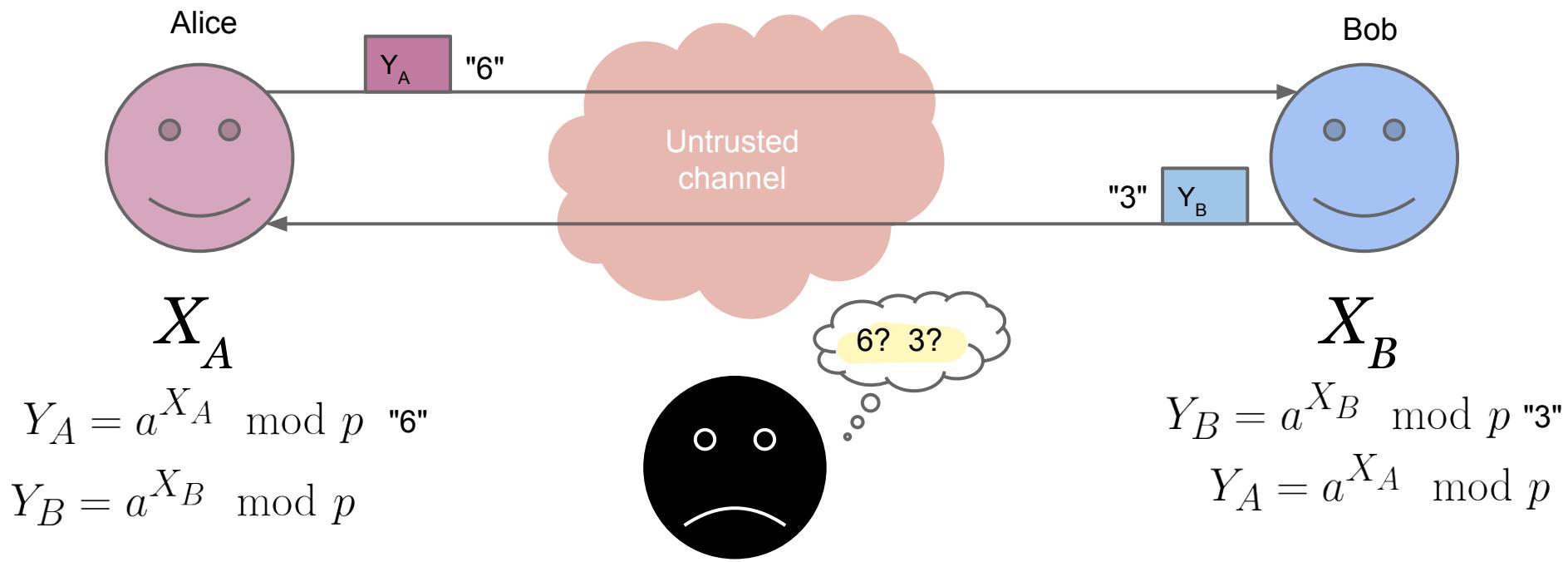
Alice  $Y_A = a^{X_A} \bmod p$   
Bob  $Y_B = a^{X_B} \bmod p$

$$Y_A = 3^3 \bmod 7 = 6$$
$$Y_B = 3^1 \bmod 7 = 3$$

# How does D-H work (3)



# How does D-H work (3)



# How does D-H work (4) - Secret

At this point, they can compute a **secret  $K$**

- Since  $\frac{Y_A^{X_B} \text{ mod } p}{(a^{X_B})^{X_A} \text{ mod } p} = (a^{X_A} \text{ mod } p)^{X_B} \text{ mod } p = (a^{X_A})^{X_B} \text{ mod } p = Y_B^{X_A} \text{ mod } p$
- Alice  $K_A = Y_B^{X_A} \text{ mod } p = 3^3 \text{ mod } 7 = 6$
- Bob  $K_B = Y_A^{X_B} \text{ mod } p = 6^1 \text{ mod } 7 = 6$

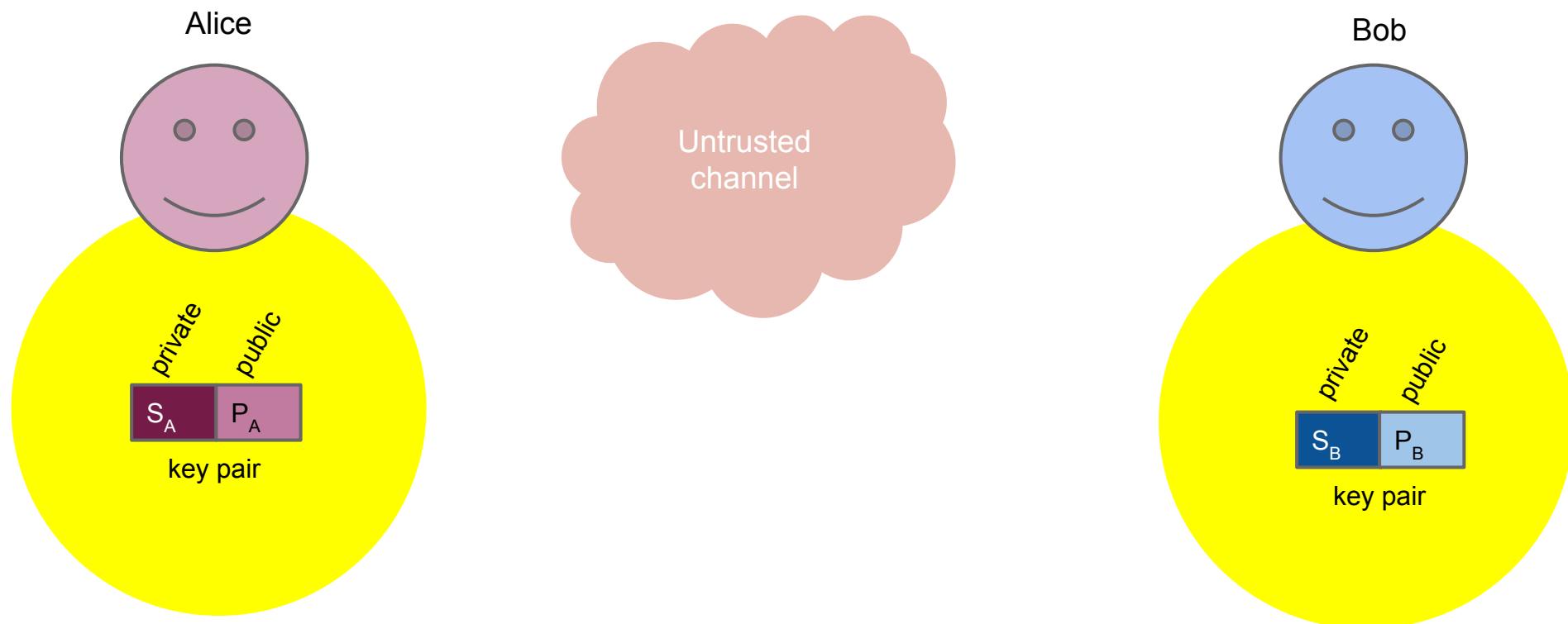
Anybody else can listen, but cannot compute  $K$

- Because they miss the secret.

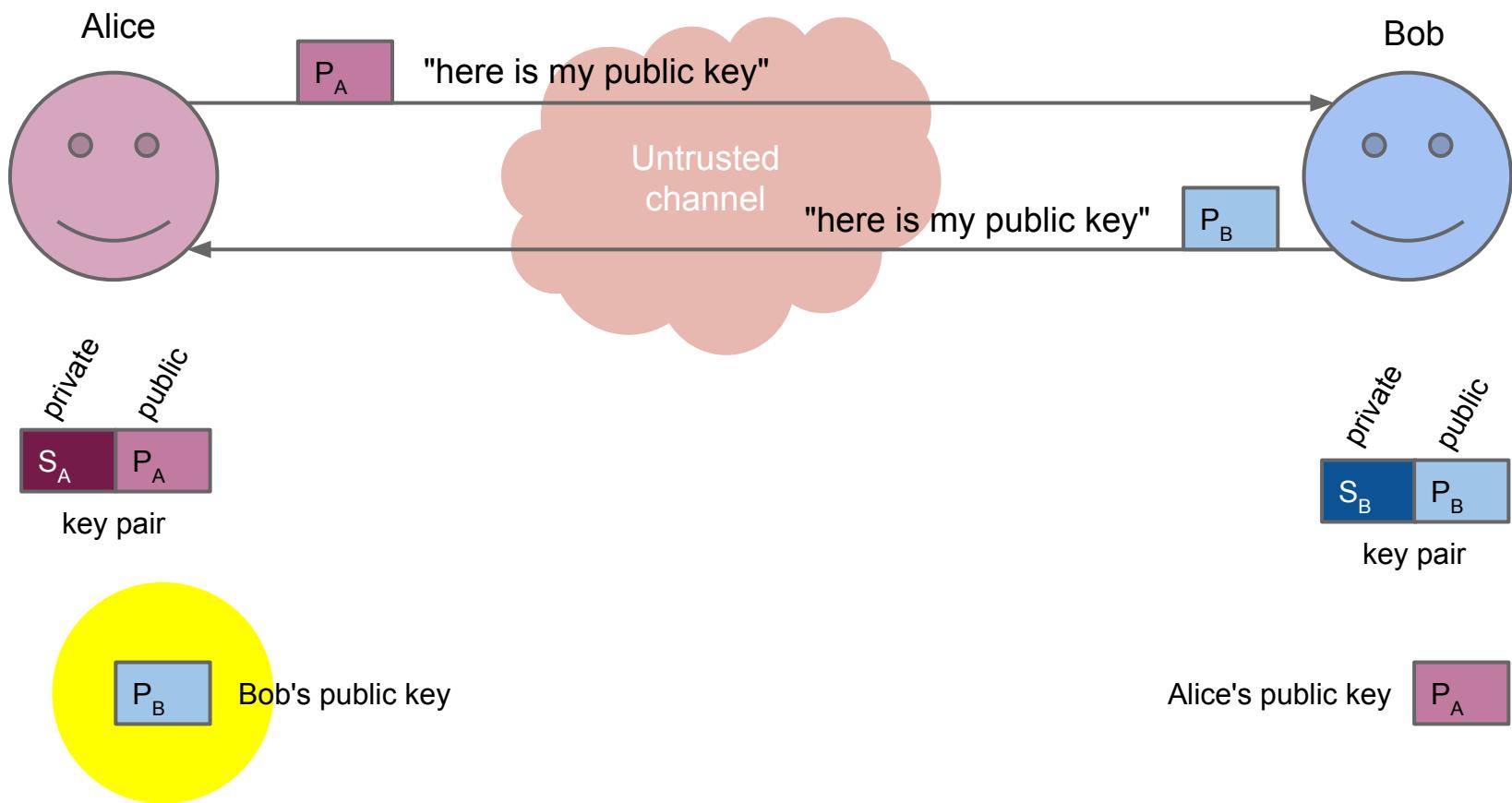
# Public key encryption

- **Concept:** a cipher that uses two keys
  - What is encrypted with **key1** can be decrypted only with **key2** (and not with key1 itself), and viceversa.
- Also called “*public key cryptography*”
  - **Idea:** one of the two keys is kept **private** by the subject, and the other can be **publicly** disclosed.
- It solves the problem of **key exchange**
- We will not describe their maths in depth
  - They use a **one-way function with a trapdoor**
    - The private key "cannot" be retrieved from the public key
    - It should be easy to compute the public from the private
  - They are usually **computational-intensive**

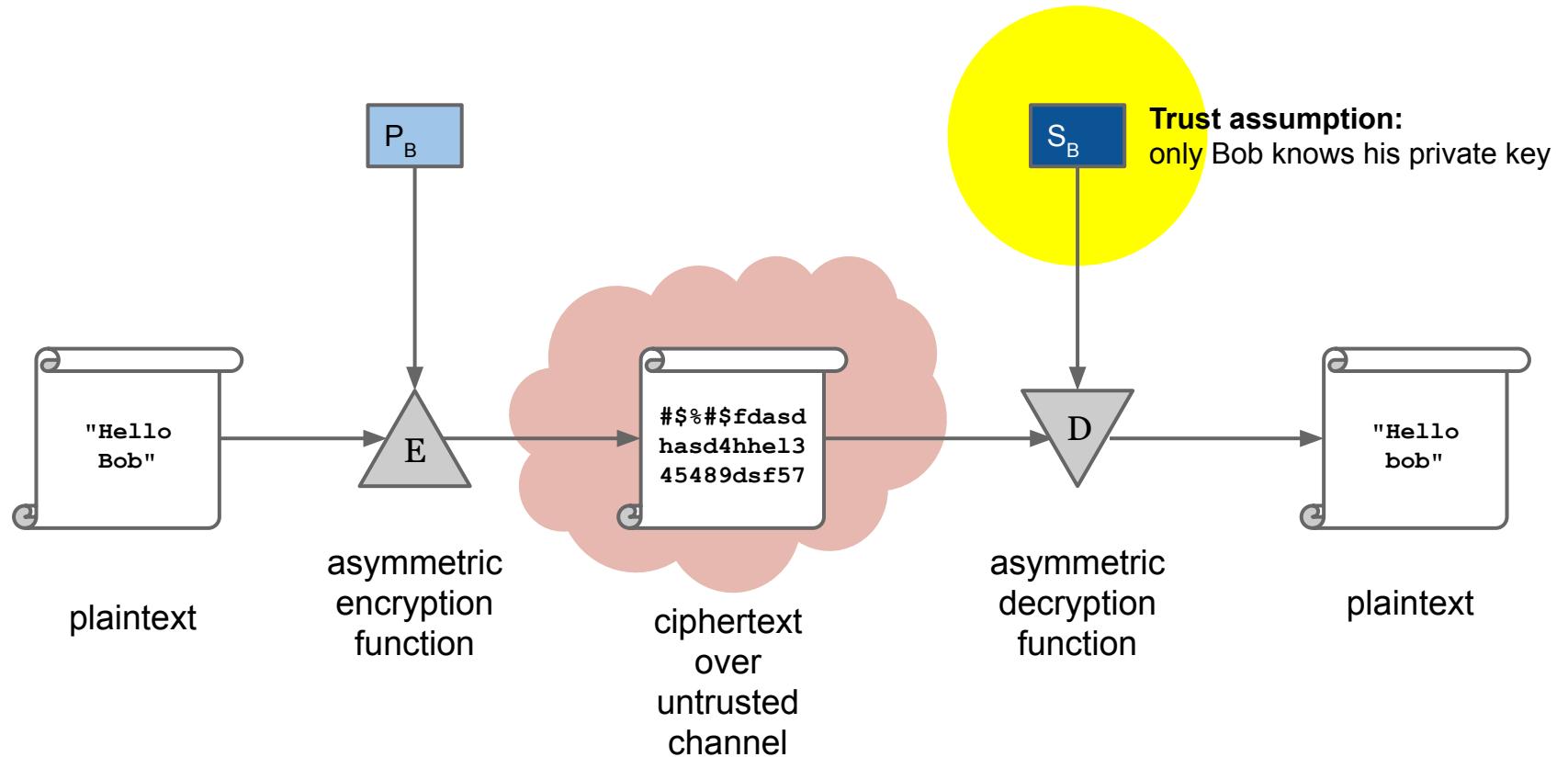
# Public key encryption: Key Exchange



# Public key encryption: Key Exchange



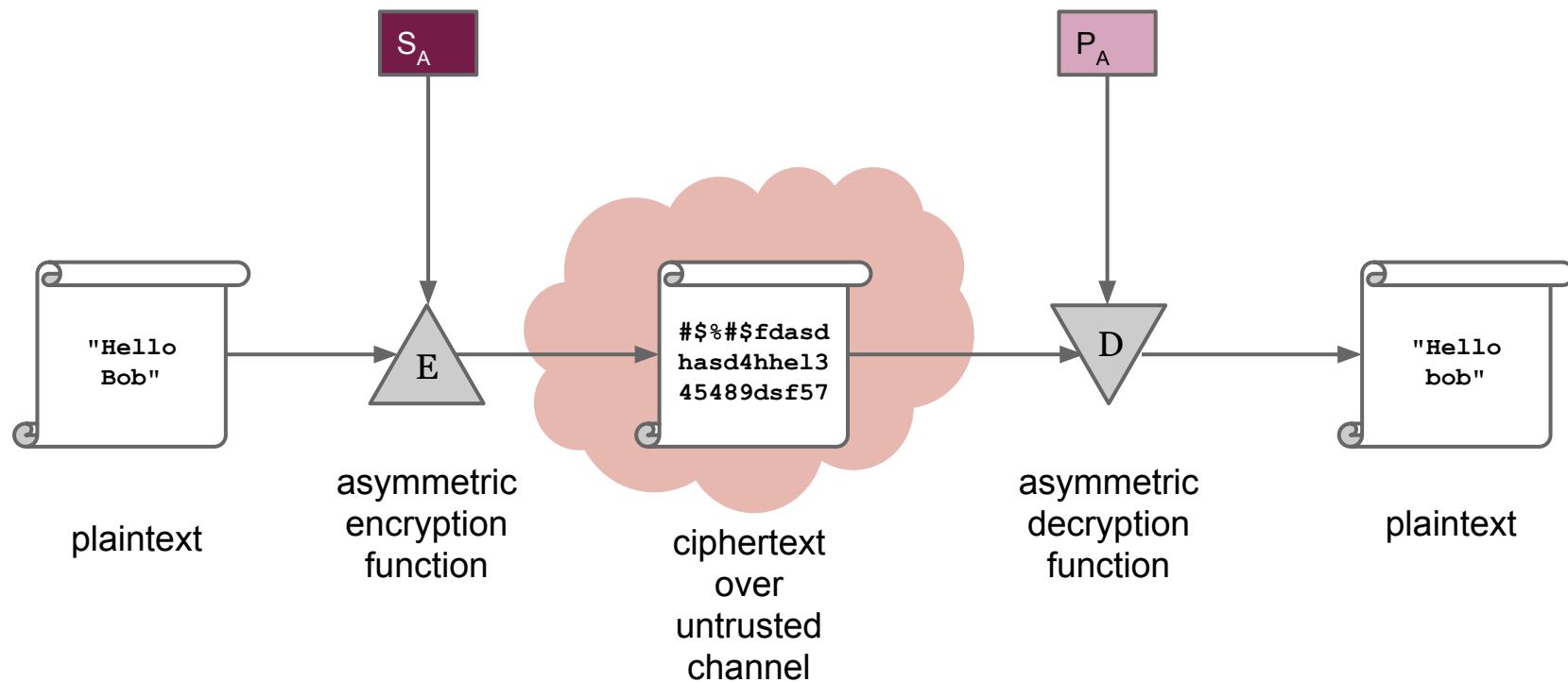
# Public key encryption: Confidentiality



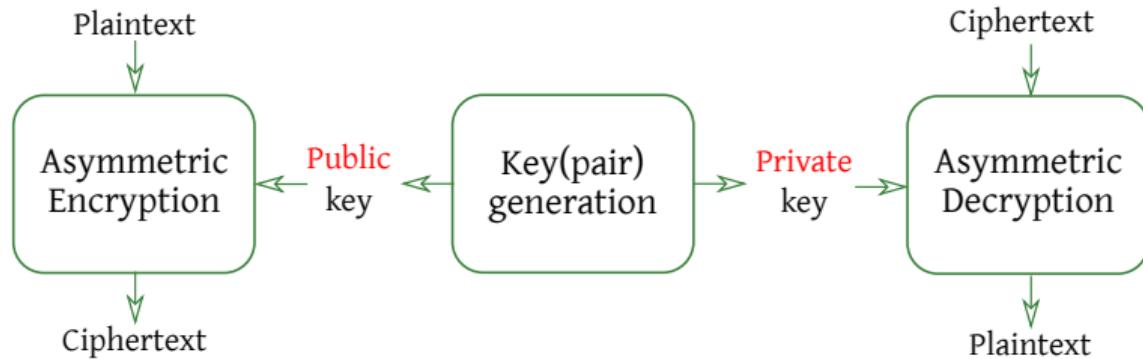
# Exercise: what is this instead?

**Trust assumption:**  
only Alice knows her private key

**Everybody** knows Alice's public key



# Public Key Encryption



## Components

- *Different keys* are employed in encryption/decryption
- It is computationally hard to:
  - Decrypt a ciphertext without the *private key*
  - Compute the *private key* given only the *public key*

# Widespread Asymmetric encryption ciphers

## Rivest, Shamir, Adleman (RSA), 1977

- 2048 to 4096 bit message-and key-sizes
- Patented after the invention, patent now expired
- No ciphertext expansion
- Incidentally, the encryption with a fixed key is a PRP

## Elgamal encryption scheme, 1985

- Either kbit range keys, or 100's of bits keys, depending on the variant
- Not encumbered by patents
- The ciphertext is **twice** the size of the plaintext
- Widely used as an RSA alternative where patents were a concern

# The RSA Algorithm (hints) - 1

Hard problem: **factorization**

If  $p$  and  $q$  are two **large primes**:

- computing  $n = p * q$  is **easy**
- but given  $n$  it is painfully **slow** to get  $p$  and  $q$
- quadratic sieve field, basically “try all primes until you get to the smaller between  $p$  and  $q$ ”
- Different problem than mod-log (D-H), but it can be shown that they are related

# The RSA Algorithm (hints) - 2

- Factoring  $n$  is exponential in the number of bits of  $n$
- Computation time for encryption grows linearly in the number of bits of  $n$ 
  - square-and-multiply algorithm in hardware
- At the moment of writing:
  - 512-bit RSA factored within 4 hours on EC2 for < \$100: <http://seclab.upenn.edu/projects/faas/faas.pdf>
  - No demonstration of practical factoring of anything larger than 700 bits
  - key sizes  $> 1024$  are safe
  - 2048 or 4096 typical choices

## Computational hardness

- Up to now, enumeration of the secret parameter was the best possible attack
- This is ok for modern block ciphers → best attack:  $\mathcal{O}(2^\lambda)$
- Asymmetric cryptosystems rely on hard problems for which bruteforcing the secret parameter is **not** the best attack
  - Factoring a  $\lambda$  bit number takes  $\mathcal{O}\left(e^{k(\lambda)^{\frac{1}{3}}(\log(\lambda))^{\frac{2}{3}}}\right)$
- Comparing bit-sizes of the security parameters instead of actual complexities is **really** wrong
- Concrete bit-sizes for  $\lambda$  depending on the cipher: [www.keylength.org](http://www.keylength.org)

# Key Lengths: caveat

- Key length measured in bits both in **symmetric** and **asymmetric** algorithms
- However, they measure different things
  - Symmetric: number of **decryption attempts**
  - Asymmetric: number of **key-breaking attempts**
- Therefore:
  - You can compare symmetric algorithms based on the key (e.g., CAST-128 bit “weaker” than AES-256)
  - You cannot directly compare asymmetric algorithms based on key length.
  - More importantly, **never compare directly** asymmetric vs. symmetric key lengths!
  - <https://www.keylength.com/en/4/>



## Assumption

- A public channel between Alice and Bob is available
- For the moment, the attacker model is “eavesdrop only”

## Sharing a secret without agreement

- Alice: generates a keypair  $(k_{pri}, k_{pub})$ , sends to Bob
- Bob: gets  $s \in \{0, 1\}^\lambda$ , encrypts it with  $k_{pub}$ , sends ctx to Alice
- Alice: decrypts ctx with  $k_{pri}$ , recovers  $s$
- Note: Bob alone decides the value of the shared secret  $s$ 
  - Repeat the procedure with swapped roles and combine the two secrets to achieve similar guarantees to a key agreement



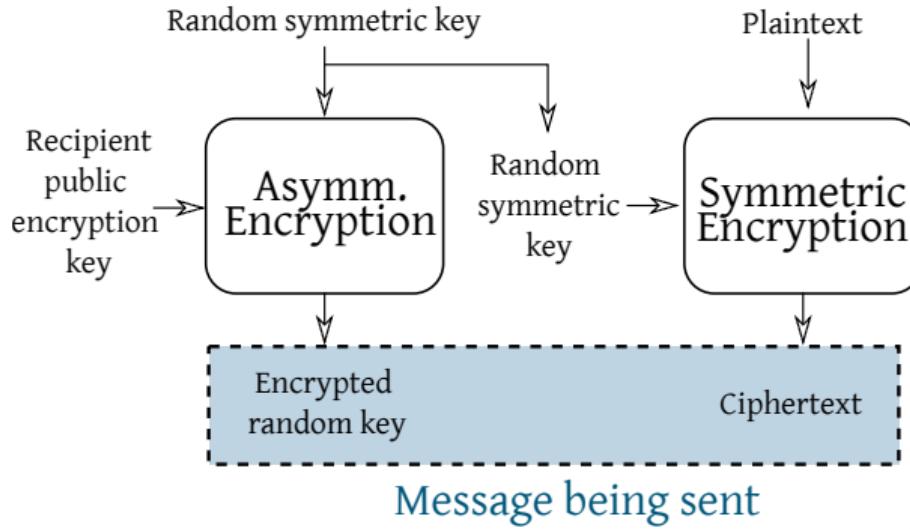
## Can't I skip a step?

- Employing an asymmetric cryptosystem Bob encrypts a text for Alice without the need of sharing a secret beforehand
- In principle, Bob and Alice could employ *only* an asymmetric cryptosystem to communicate

## Can't I skip a step?

- Employing an asymmetric cryptosystem Bob encrypts a text for Alice without the need of sharing a secret beforehand
- In principle, Bob and Alice could employ *only* an asymmetric cryptosystem to communicate
- In practice this approach would be extremely inefficient
  - Asymmetric cryptosystems are from  $10\times$  to  $1000\times$  slower than their symmetric counterparts

# Best of both worlds



## Hybrid encryption schemes

- Asymmetric schemes to provide key transport/agreement
- Symmetric schemes to encrypt the bulk of the data
- All modern secure transport protocols built around this idea

# Authenticating data

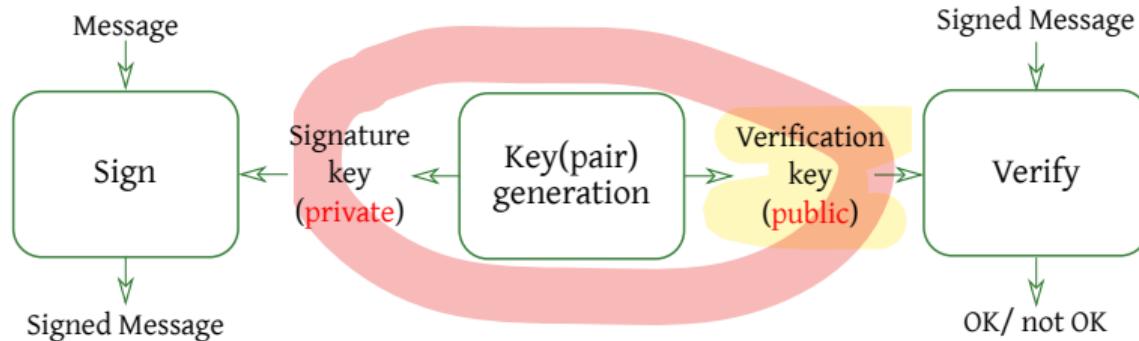
## Motivations

- To build a secure hybrid encryption scheme we need to be sure that the public key the sender uses is the one of the recipient
- We'd like to be able to verify the authenticity of a piece of data without a pre-shared secret

## Digital signatures

- Provide strong evidence that data is bound to a specific user
- No shared secret is needed to check (validate) the signature
- Proper signatures cannot be repudiated by the user
- They are **asymmetric** cryptographic algorithms
  - formally proven that you cannot get non repudiation otherwise

# Digital signatures



## Computationally hard problems

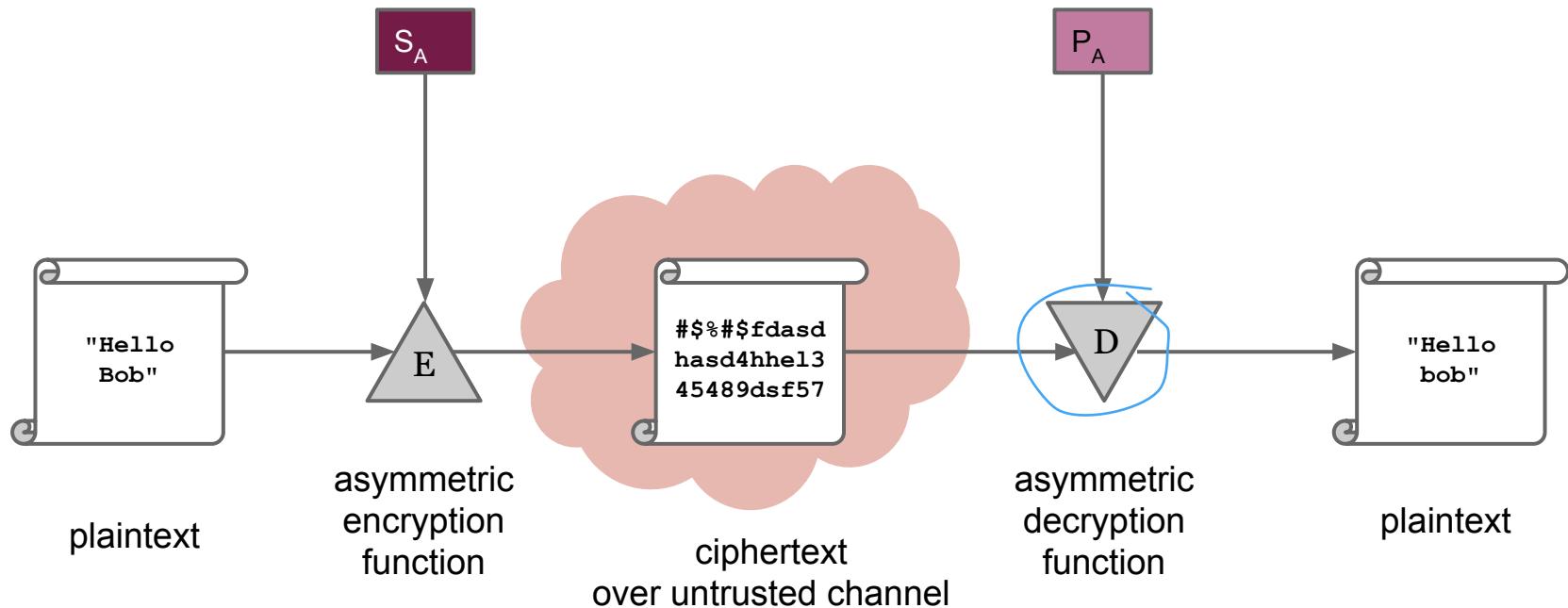
- Sign a message without the **signature** key
  - this includes splicing signatures from other messages
- Compute the **signature** key given only the **verification** key
- Derive the **signature** key from signed messages

# Message Authentication

**Trust assumption:**

only Alice knows her private key

**Everybody** knows Alice's public key

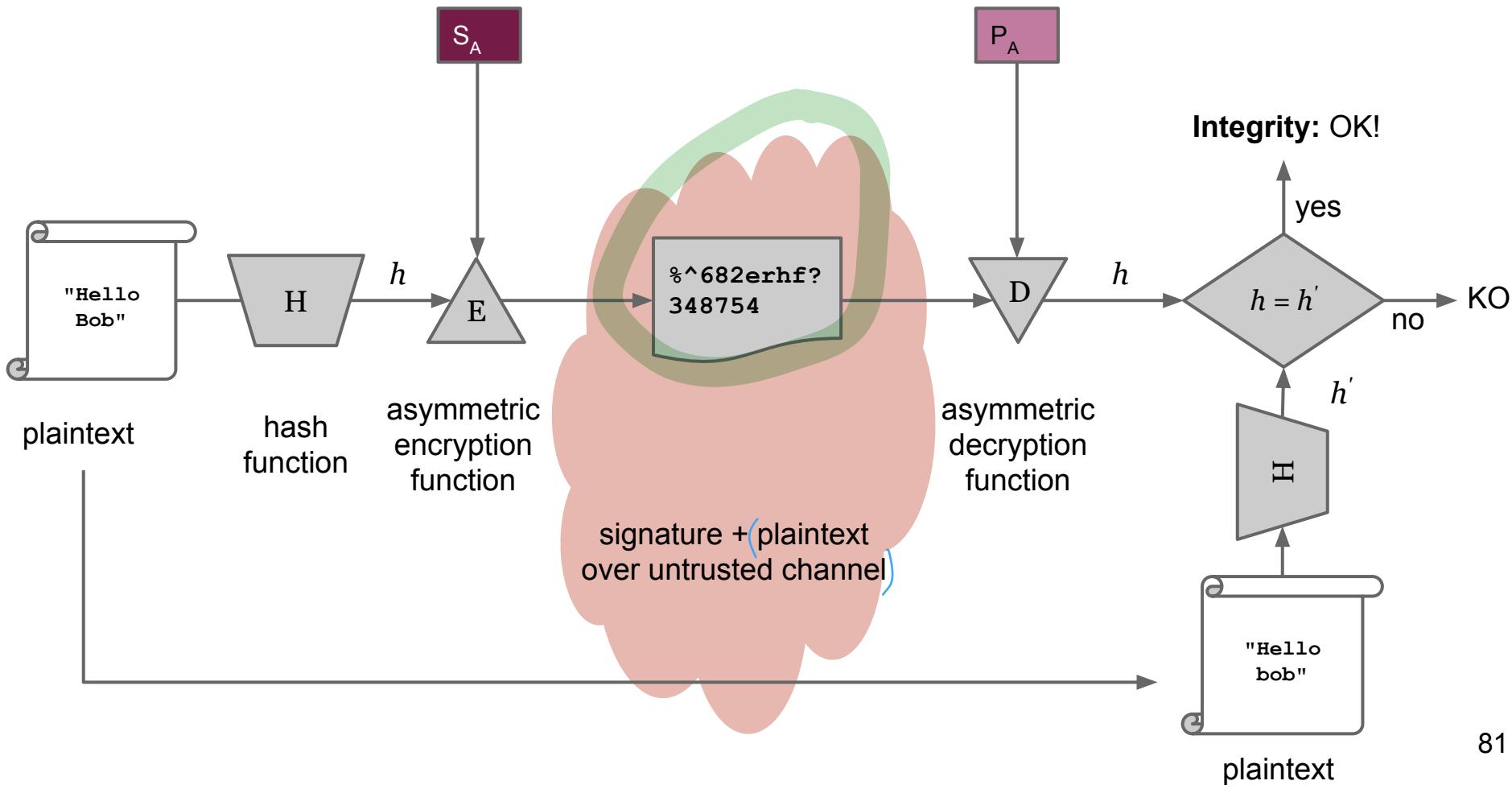


# Digital signature: Authentication and Integrity

**Trust assumption:**

only Alice knows her private key

**Everybody** knows Alice's public key



# Widespread Signature schemes

## Rivest, Shamir, Adleman (RSA), 1977

- Unique case: the same hard-to-invert function to build an asymmetric encryption scheme and a signature (different message processing!)
- Signing definitely slower than verification ( $\approx 300\times$ )
- Standardized in NIST DSS (FIPS-184-4)

## Digital Signature Standard (DSA)

- Derived from tweaking signature schemes by Schnorr and Elgamal
- Also standardized in NIST DSS (FIPS-184-4)
- Signature and verification take roughly the same time

# Digital signature uses

## Authenticating digital documents

- For performance reasons, sign the hash of the document instead of the document
  - Signature properties now guaranteed only if both signature and hash algorithms are not broken

## Authenticating users

- Alternative to password-based login to a system
  - The server has the user's public verification key (e.g. deposited at account creation)
  - The server asks the client to sign a long randomly generated bitstring (challenge)
  - If the client returns a correctly signed challenge, it has proven its identity to the server



## Cautionary note

- Both in asymmetric encryption and digital signatures, the public key must be bound to the correct user identity
- If public keys are not authentic:
  - A MITM attack is possible on asymmetric encryption
  - Anyone can produce a signature on behalf of anyone else
- The public key authenticity is guaranteed with... another signature
  - We need someone to sign the public-key/identity pair
  - We need a format to distribute signed pairs

# A case of identity

- A digital signature ensures that plaintext was authored by someone.

# A case of identity

- A digital signature ensures that plaintext was authored by someone.
- **Not really!** It ensures it was encrypted with a certain key...it says nothing about “who” is using that private key
- Ditto for using public key for encryption!

# A case of identity

- A digital signature ensures that plaintext was authored by someone.
- **Not really!** It ensures it was encrypted with a certain key...it says nothing about “who” is using that private key
- Ditto for using public key for encryption!
- Exchange of public keys **must be secured** (either out of band, or otherwise)
- PKI (Public Key Infrastructure) associates keys with identity on a wide scale

# PKI

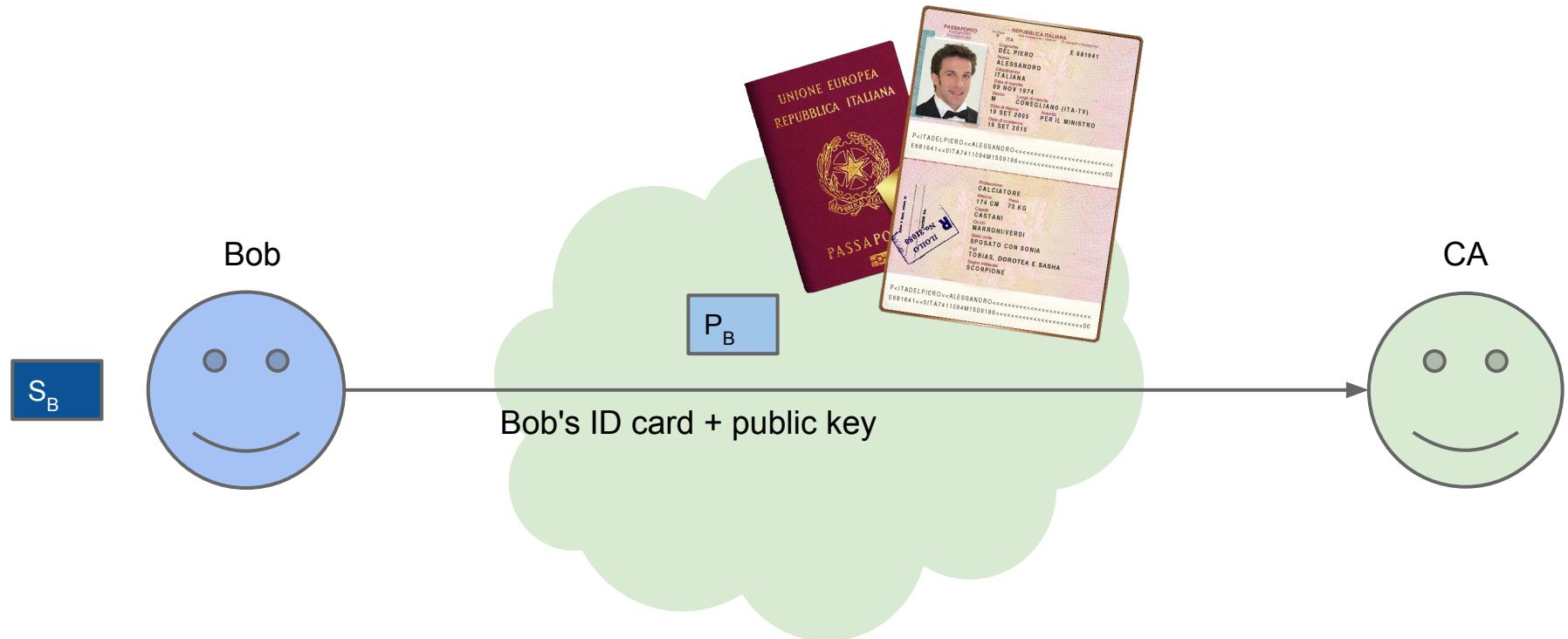
- A PKI uses a trusted third party called a **certification authority (CA)**
- The CA **digitally signs** files called **digital certificates**, which bind an identity to a public key
  - Identity = “Distinguished Name (DN)”
  - As defined in the X.509 standard (most used one)
- Now we can recognize a number of subjects...



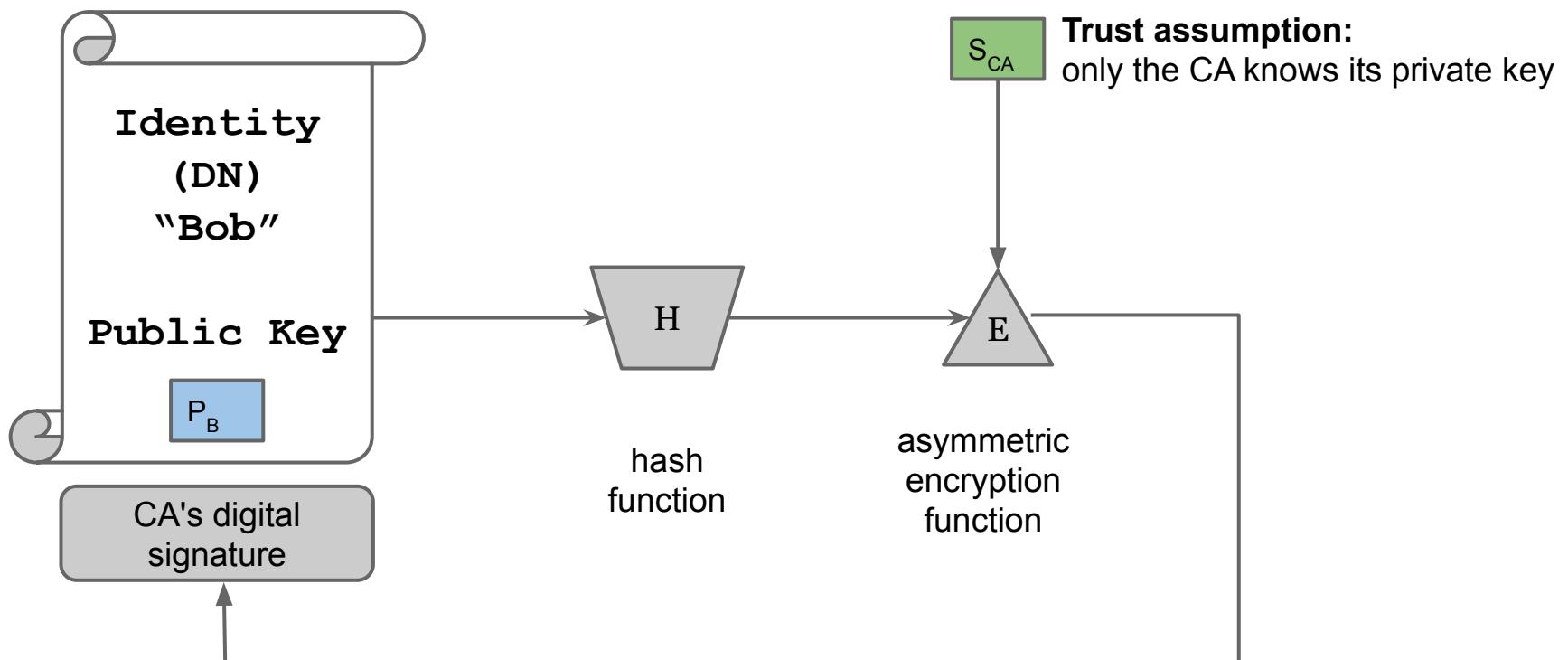
## Digital certificates

- They bind a public key to a given identity, which is:
  - for humans: an ASCII string
  - for machines: either the CNAME or IP address
- They specify the intended use for the public key contained
  - Avoids ambiguities when a key format is ok for both an encryption and a signature algorithm
- They contain a time interval in which they are valid
- Most widely deployed format is described in ITU X.509

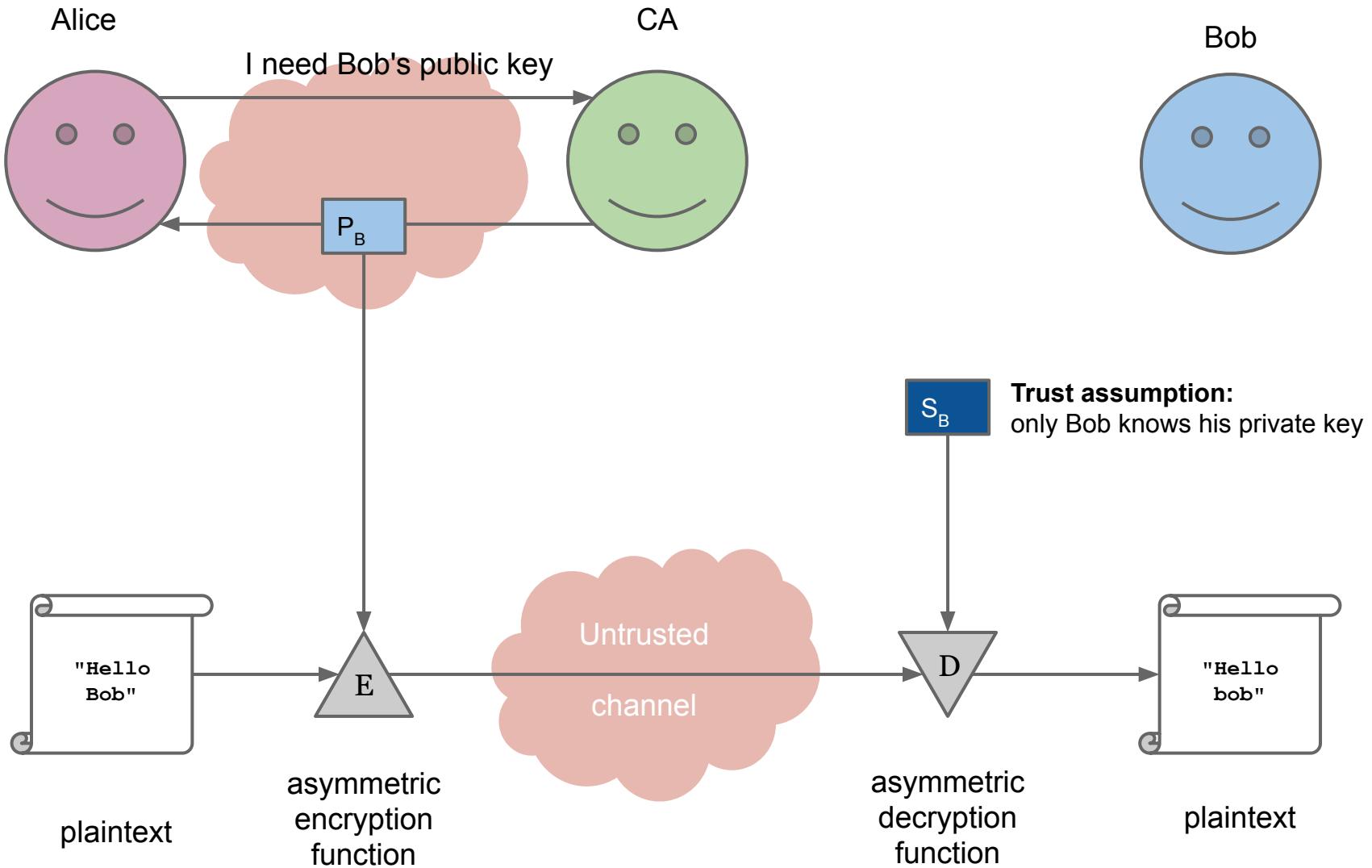
# Bob's Digital Certificate (1)



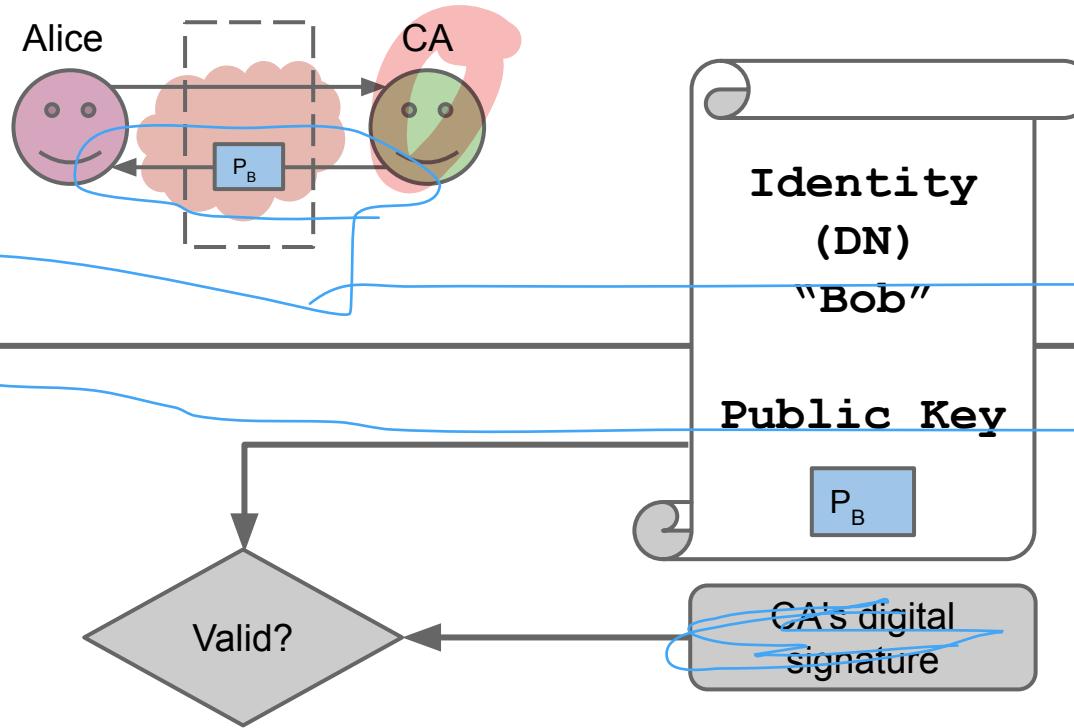
# Bob's Digital Certificate (2)



# Retrieving Bob's Certificate



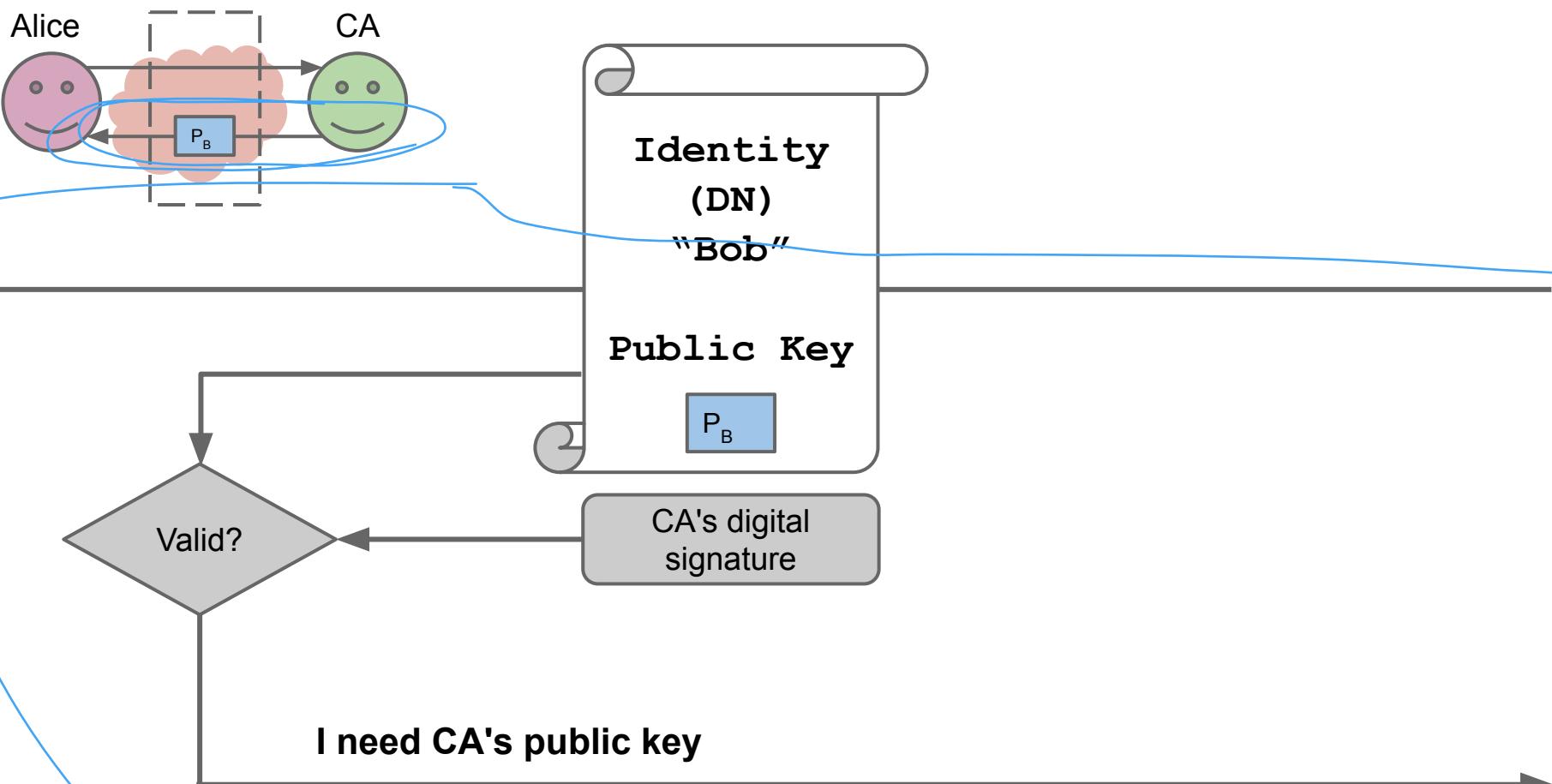
# Zoom in: Is the public key valid?



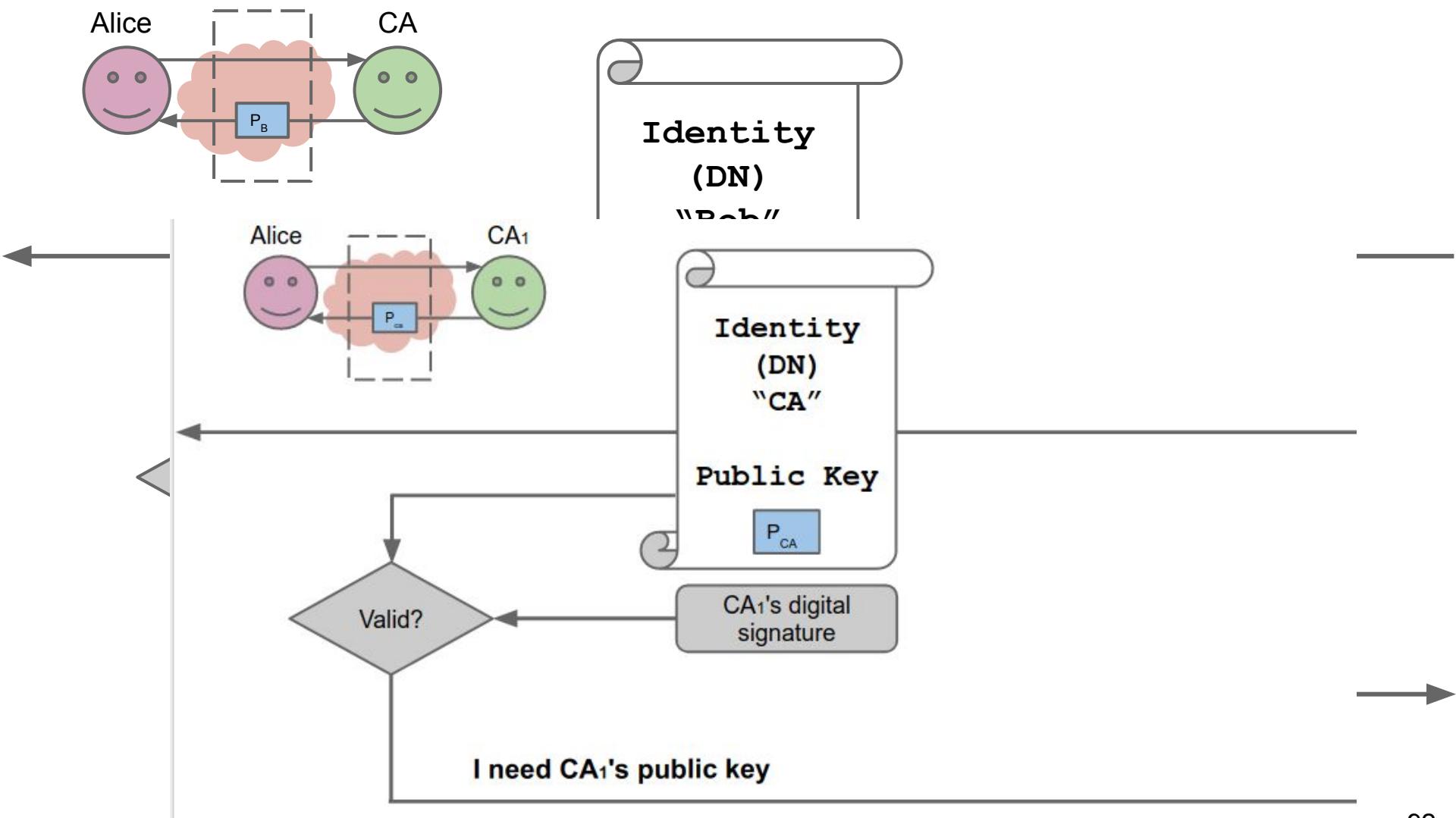
# PKI

- A PKI uses a trusted third party called a **certification authority (CA)**
- The CA **digitally signs** files called **digital certificates**, which bind an identity to a public key
  - Identity = “Distinguished Name (DN)”
  - As defined in the X.509 standard (most used one)
- Now we can recognize a number of subjects...provided that we can obtain the **public key** of the CA

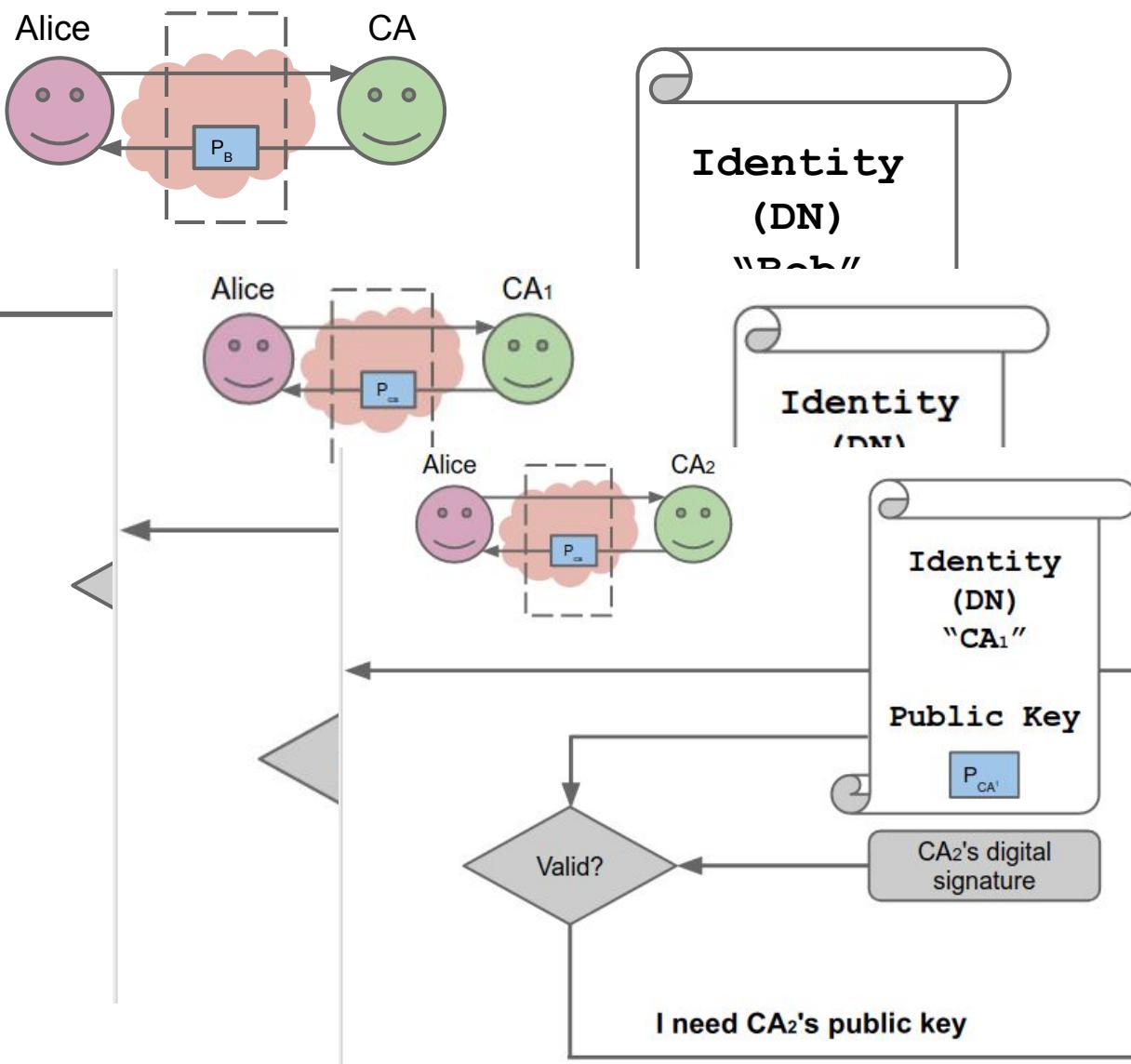
# Zoom in: Is the public key valid?



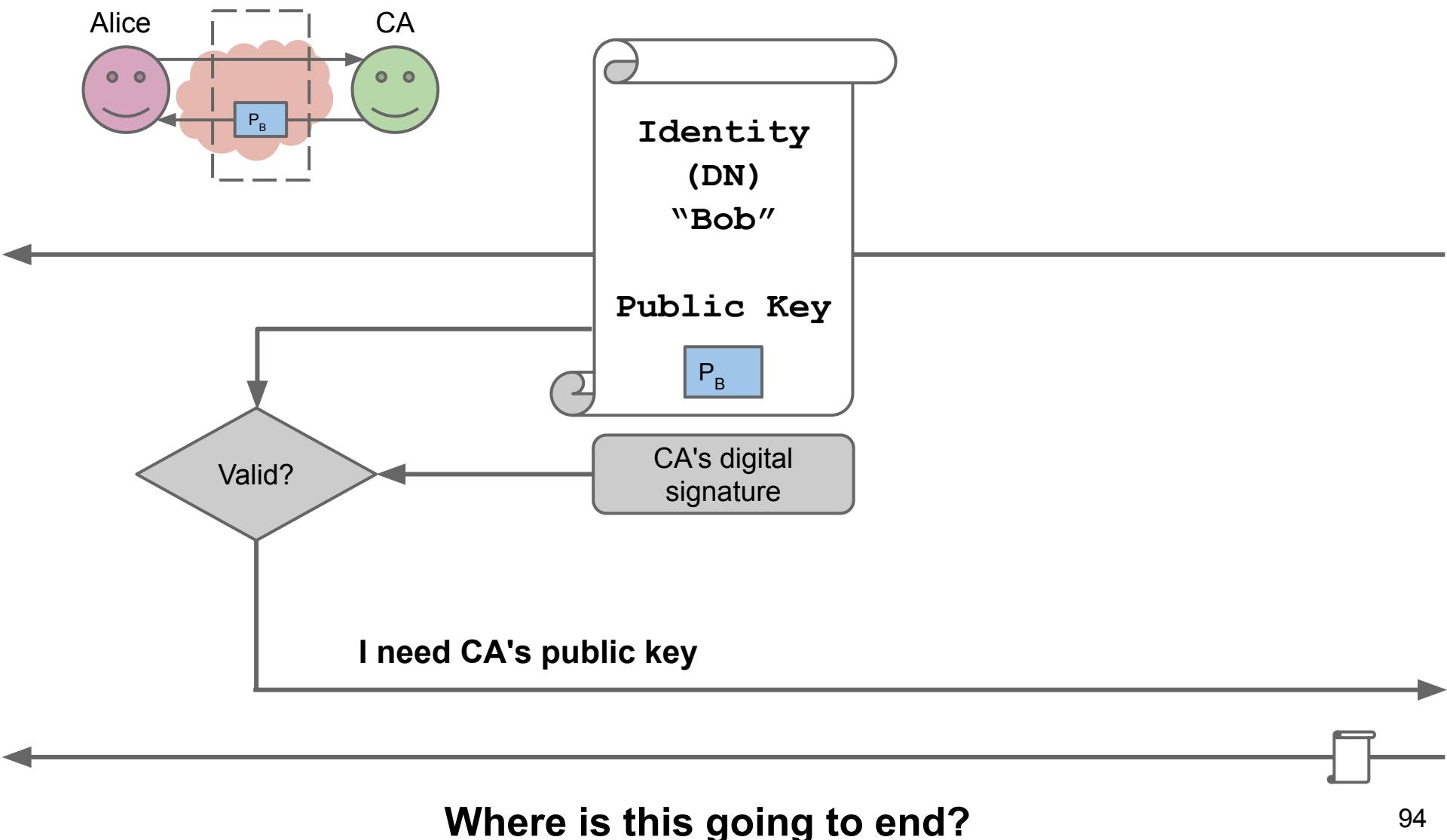
# Zoom in: Is the public key valid?



# Zoom in: Is the public key valid?



# Zoom in: Is the public key valid?



# The Certificate Chain

***"Quis custodiet custodes?"***

The CA needs a *private key* to sign a certificate

- The *public key*...must be in a certificate.

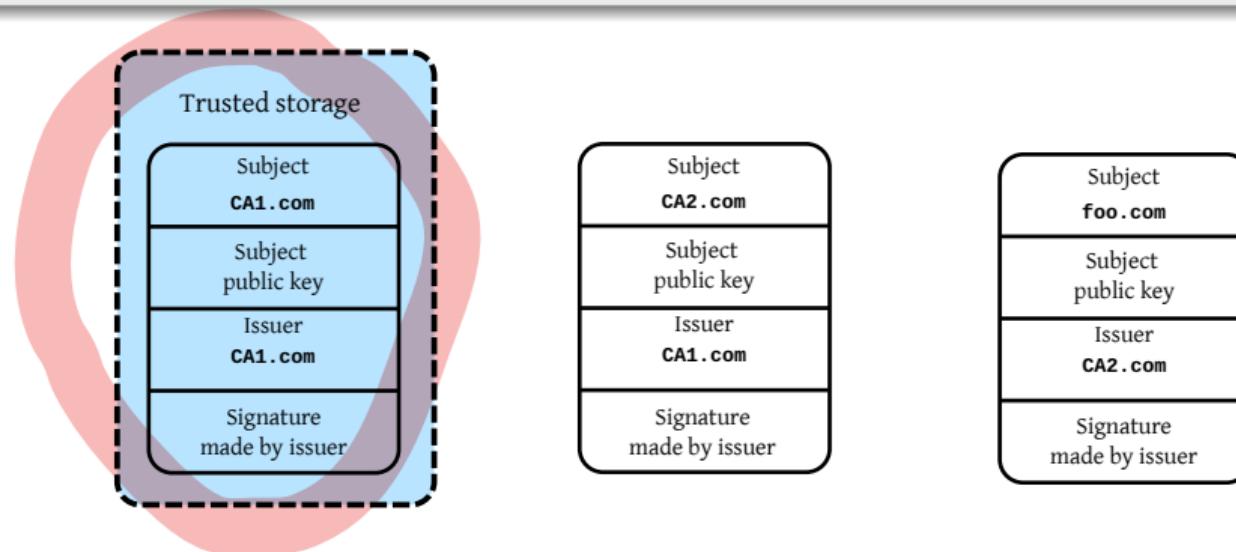
Someone else needs to sign **that** certificate

- And so on...at some point this needs to stop!

# Certification authorities

## Who signs the certificates

- The certificate signer is a trusted third party, the CA
- The CA public key is authenticated... with another certificate
- Up to a self-signed certificate which has to be trusted a priori



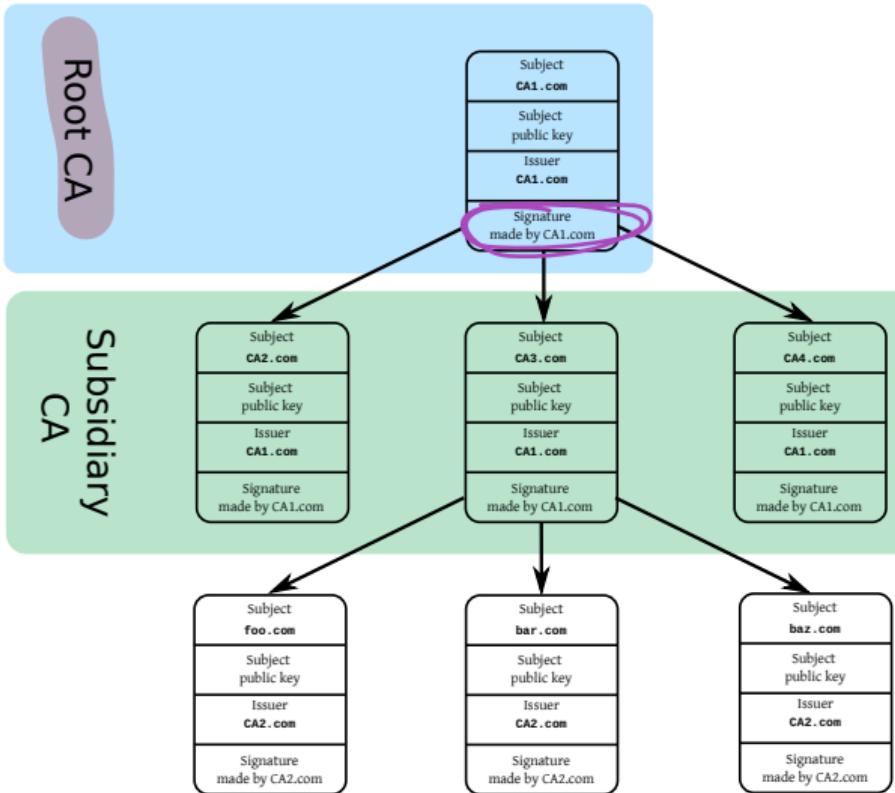
# We Need a Trusted Element: Root of trust

**Top-level CA** (root CA, source CA)

Uses a self-signed certificate

- cannot be verified: it's a **trusted element**
- Basically a document that says “I am myself”

# Certification authorities hierarchy



# How to distribute the trusted element?

An **authority** releases it

- the state
- a regulator
- the organization management

CA already (de facto standard)

Keychain Access

Click to unlock the System Roots keychain.

Keychains

- login
- PrivateEncryptedData
- PrivateEncryptedData
- Local Items
- System
- System Roots**

VeriSign Universal Root Certification Authority  
Root certificate authority  
Expires: Wednesday 2 December 2037 00 h 59 min 59 s Central European Standard Time  
This certificate is valid

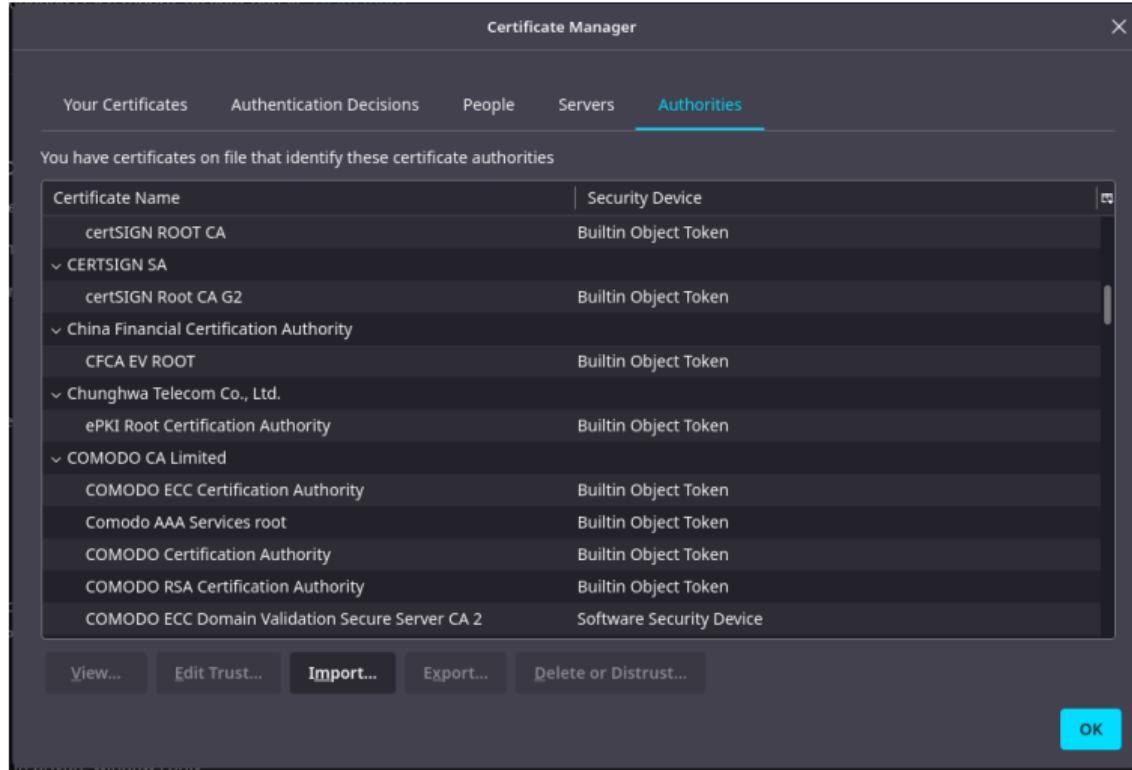
Name	Kind	Expires	Keychain
XRamp Global Certification Authority	certificate	01 Jan 2035 06:37:19	System Roots
WellsSecure...ertificate Authority	certificate	14 Dec 2022 01:07:54	System Roots
VRK Gov. Root CA	certificate	18 Dec 2023 14:51:08	System Roots
Visa Informa...Delivery Root CA	certificate	29 Jun 2025 19:42:42	System Roots
Visa eCommerce Root	certificate	24 Jun 2022 02:16:12	System Roots
<b>VeriSign Universal Certification Authority</b>	<b>certificate</b>	<b>02 Dec 2037 00:59:59</b>	<b>System Roots</b>
VeriSign Class...on Authority - G3	certificate	17 Jul 2036 01:59:59	System Roots
VeriSign Class...on Authority - G5	certificate	17 Jul 2036 01:59:59	System Roots
VeriSign Class...on Authority - G4	certificate	19 Jan 2038 00:59:59	System Roots
VeriSign Class...on Authority - G3	certificate	17 Jul 2036 01:59:59	System Roots
VeriSign Class...on Authority - G3	certificate	17 Jul 2036 01:59:59	System Roots
VeriSign Class...on Authority - G3	certificate	17 Jul 2036 01:59:59	System Roots
VAS Latvijas Pasts SSI(RCA)	certificate	13 Sep 2024 11:27:57	System Roots
UTN-USERFirst-Object	certificate	09 Jul 2019 20:40:36	System Roots
UTN-USERFir...work Applications	certificate	09 Jul 2019 20:57:49	System Roots
UTN-USERFirst-Hardware	certificate	09 Jul 2019 20:19:22	System Roots
UTN-USERFir...cation and Email	certificate	09 Jul 2019 19:36:58	System Roots
UTN - DATACorp SGC	certificate	24 Jun 2019 21:06:30	System Roots
UCA Root	certificate	31 Dec 2029 01:00:00	System Roots
UCA Global Root	certificate	31 Dec 2037 01:00:00	System Roots
TWCA Root C...fication Authority	certificate	31 Dec 2030 16:59:59	System Roots
TWCA Global Root CA	certificate	31 Dec 2030 16:59:59	System Roots
TÜRKTRUST...Hizmet Sağlayıcısı	certificate	22 Mar 2015 11:27:17	System Roots
TÜRKTRUST...Hizmet Sağlayıcısı	certificate	16 Sep 2015 12:07:57	System Roots
TÜRKTRUST...Hizmet Sağlayıcısı	certificate	22 Dec 2017 19:37:19	System Roots
TÜRKTRUST...Hizmet Sağlayıcısı	certificate	21 Aug 2017 12:27:07	System Roots

222 items

+ i Copy

Do you trust your operating system? Do you trust the list of root certificates that ship with it?

# A recent browser certificate storage



# How to distribute the trusted element?

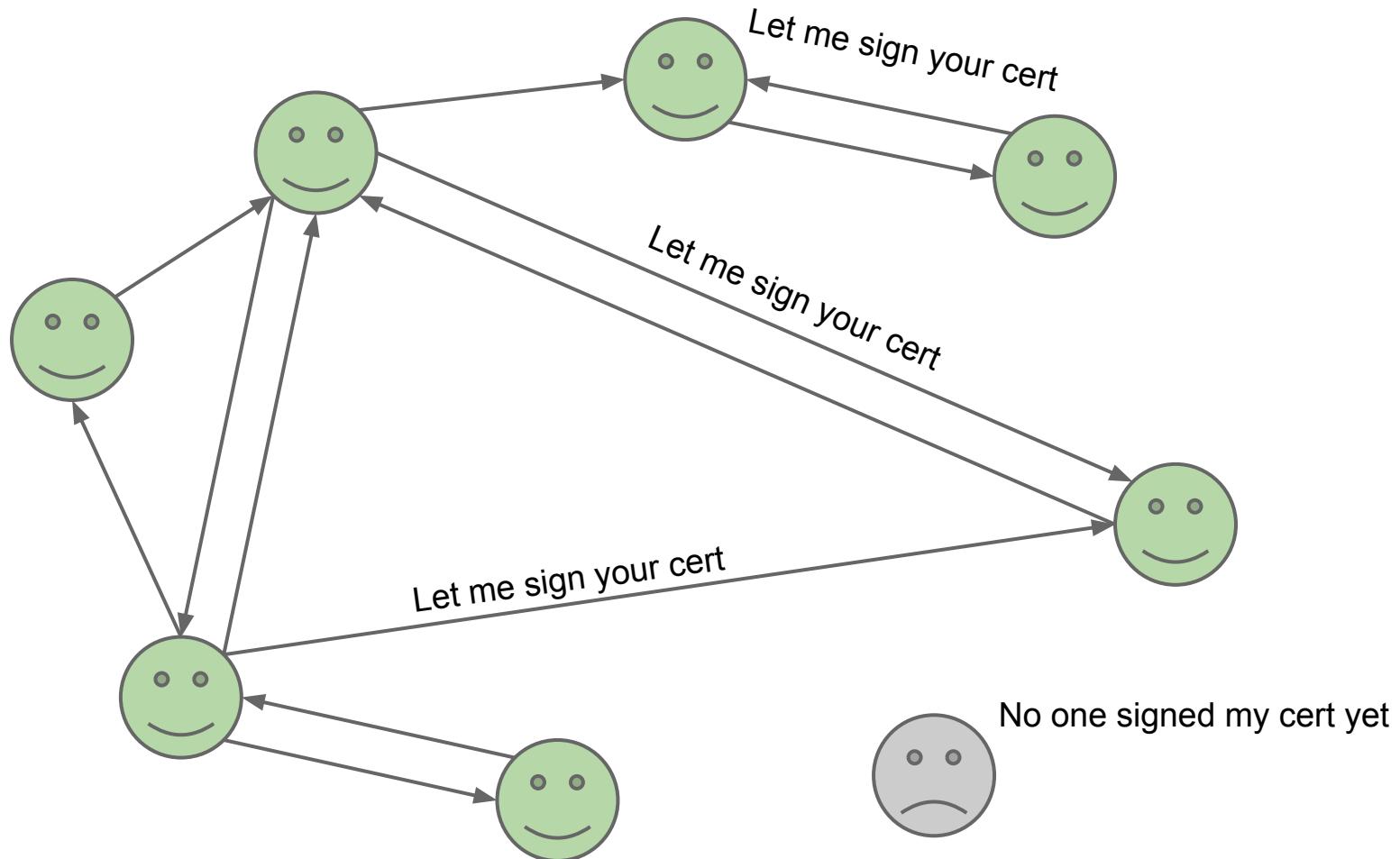
An **authority** releases it

- the state
- a regulator
- the organization management

CA already (de facto standard)

Decentralizing trust (e.g., PGP web-of-trust)

# Decentralizing Trust: Web of Trust



# Certificate Revocation Issues

- Signatures cannot be revoked (destroyed).
- **Certificates** need to be revoked at times.
- Certificate Revocation Lists (CRL)

# Verification Sequence for Certificates

1. Does the **signature** validate the document?
  - Hash verification as we have seen
2. Is the **public key** the one on the certificate?
3. Is the certificate the one of the **subject**?
  - Problems with omonymous subjects, DN
4. Is the **certificate** validated by the CA?
  - Validate the entire certification chain, up to the root
5. Is the **root** certificate trusted?
  - Need to be already in possession of the root cert
6. Is the certificate in a **CRL**?
  - How do we get to the CRL if we are not **online**?

# Verification Sequence for Certificates

1. Does the **signature** validate the document?
  - Hash verification as we have seen
2. Is the **public key** the one on the certificate?
3. Is the certificate the one of the **subject**?
  - Problems with omonymous subjects, DN
4. Is the **certificate** validated by the CA?
  - Validate the entire certification chain, up to the root
5. Is the **root** certificate trusted?
  - Need to be already in possession of the root cert
6. Is the certificate in a **CRL**?
  - How do we get to the CRL if we are not **online**?

Any missing check = vulnerability!

# Case study: Italian “legal” digital signatures framework

Introduced in Italy with D.P.R. 513/97

- many modifications, in particular when implementing EU regulations

Original Italian scheme: a list of “screened” CAs

**Result:** each CA created their own **digital signature application** (i.e., trusted element)

# Attacking Digital Signature Applications

Digital signature stronger than handwritten signature

- **Written** documents can be modified, written signatures can be copied.
- **Digital** signature value **tied to content**, and cannot be forged unless the algorithm is broken
- However, a digital signature is **brittle**: if a fake is forged, it cannot be told from real one

# Crypto: *OK* – Software Design: *KO*

Italian signature standards use **strong, unbroken cryptographic algorithms!**

However, vulnerabilities did emerge

Do you remember the “bank vault door in a tent”?

# Bug 1: Fields of pain

- Bug notified on 9/9/2002
- The software of several CAs (originally DiKe by Infocamere was the subject of scrutiny) allowed users to sign Word documents with **dynamic fields** or **macros** without notice
- A macro does not change the bit sequence of the document, so the **signature does not change** with the visualized content
- Examples and stuff on Prof. Zanero's home:  
<http://home.deib.polimi.it/zanero/bug-firma.html>  
(Italian only, sorry for that!)

# Example

provaDiKe.doc - Microsoft Word

File Edit View Insert Format Tools Table Window Help

Normal Arial 11 B I U

Questo è un documento di prova scritto con MS Word 2000.  
Nelle righe seguenti sono inseriti campi variabili tra asterischi:

Data e ora \*17/09/2002 08.48.53\*  
I dati inseriti sono 17/09/2002 08.48.53

Data e ora \*17/09/2002 08.48.53\* (Questa non dovrebbe modificarsi)  
I dati inseriti sono 17/09/2002 08.48.53

Nome del documento \*provaDiKe.doc\*  
Data inserito provaDiKe.doc

Nome autore \*Manlio Cammarata\*  
Data inserito Manlio Cammarata

Nome utente \*Manlio Cammarata\*  
Data inserito Manlio Cammarata

Ora il file sarà firmato digitalmente con DiKe e dovremo controllare se la verifica va a buon fine anche dopo l'aggiornamento dei campi.

Page 1 Sec 1 1/1 At 2.5cm Ln 1 Col 1 REC TRK EXT OVR Italian (Italy)

DiKe

File Modifica Strumenti Opzioni Guida

Nr. Firmatari: 1  
Firmatario: 1 CAMMARATA MANLIO; Cod. Fiscale: CMMMNLL47P04L424H; Identificativo: 2001111111812  
Stato: IT; Organizzazione: Non Dichiарато; Unità Organizzativa: RA=INFOCAMERE S.C.p.A.  
verifica completata correttamente

Documento: d:\test\provaDiKe18.doc.p7m Dimensioni: 21944 Data: 18/09/2002 14.20.44

Questo è un documento di prova scritto con MS Word 2000.  
Nelle righe seguenti sono inseriti campi variabili tra asterischi:

Data e ora \*18/09/2002 14.28.15\*  
I dati inseriti sono 17/09/2002 08.48.53

Data e ora \*17/09/2002 08.48.53\* (Questa non dovrebbe modificarsi)  
I dati inseriti sono 17/09/2002 08.48.53

Nome del documento \*provaDiKe.doc\*  
Data inserito provaDiKe.doc

Nome autore \*Manlio Cammarata\*  
Data inserito Manlio Cammarata

Nome utente \*Manlio Cammarata\*  
Data inserito Manlio Cammarata

Ora il file sarà firmato digitalmente con DiKe e dovremo controllare se la verifica va a buon fine anche dopo l'aggiornamento dei campi.

# Reactions

- The CAs responded that this was “*intended behavior*” and that it did not violate the law
- However:
  - Microsoft, on 30/1/03, released an Office patch to allow **disabling macros** via API.
  - Nowadays, all software show a big alert when signing an Office document.
  - New legislation explicitly excludes modifiable and scriptable formats (but recommends PDF)
- The issue is actually much deeper
  - Decoders of complex formats should also be validated
  - Research field of “what you see is what you sign”

# Horror story 1 overview

## The importance of being static

- 2002-09-09: several pieces of software performing digital signatures with legal value in Italy allowed to sign MS Word documents *containing macros*
  - Macros allow to dynamically change the displayed text in a document according to, e.g., the current date
- Striking mismatch between what was thought to be signed (the visualized document) and the actual signed object (a program-document blend)
- Current standard for digital signatures on human-intended documents (PAdES,CAdES) target PDF and XML formats
  - n.b.: PDFs may embed Javascript, PDF/A do not

# Bug 2: Firma&Cifra

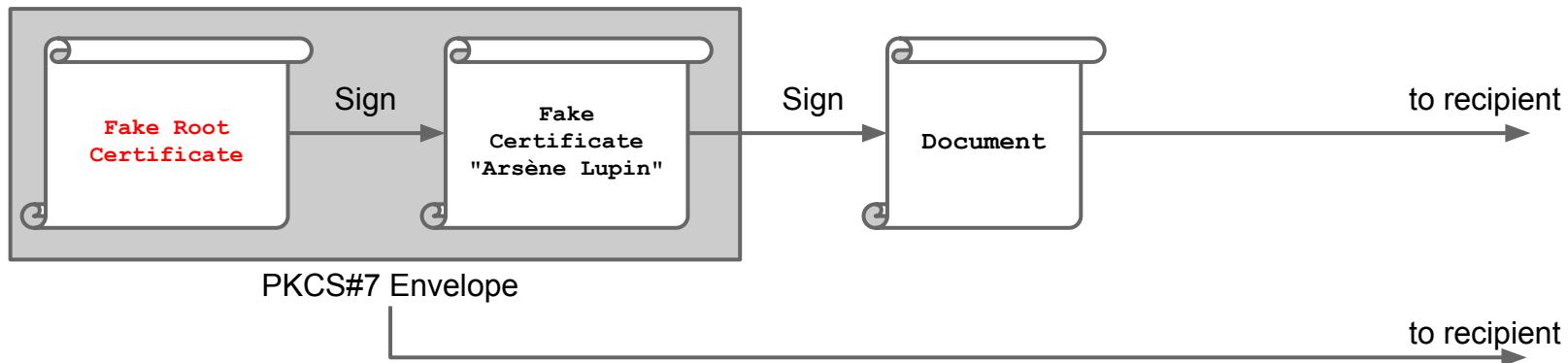
- Firma&Cifra was the digital signature application by PostECom
- Bug found by anonymous on 20/03/2003:  
<http://www.interlex.it/docdigit/sikur159.htm>
- **Result: creation and verification of a signature with a fake certificate**
- Also in this case: no cryptographic algorithm was broken to perform the show

# Vulnerability Description

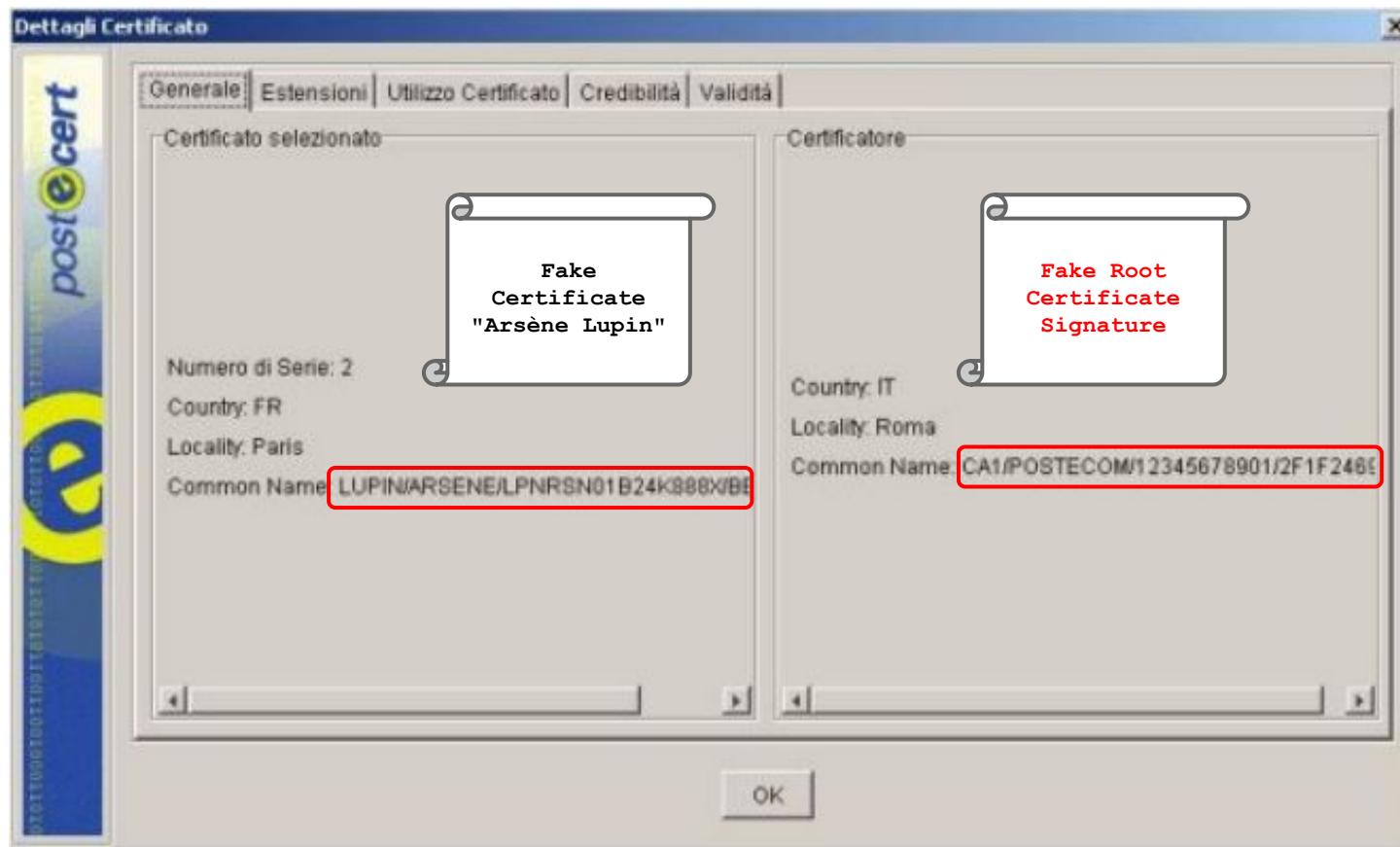
- In order to verify a signature, we need **author certificate** and the **certificate chain**
  - **Theoretically**, all available **online**
  - To allow **offline verification**, everything included with the document, in a PKCS#7 envelope
- Verification of root certificates must use preinstalled ones
  - Most **software** comes with them
  - The **root certificate storage** is a critical point!
- Firma&Cifra trusts the **root certificate** in the **PKCS#7 envelope**, and it even imports it in the secure storage area.

# The Exploit: Arséne Lupin signature

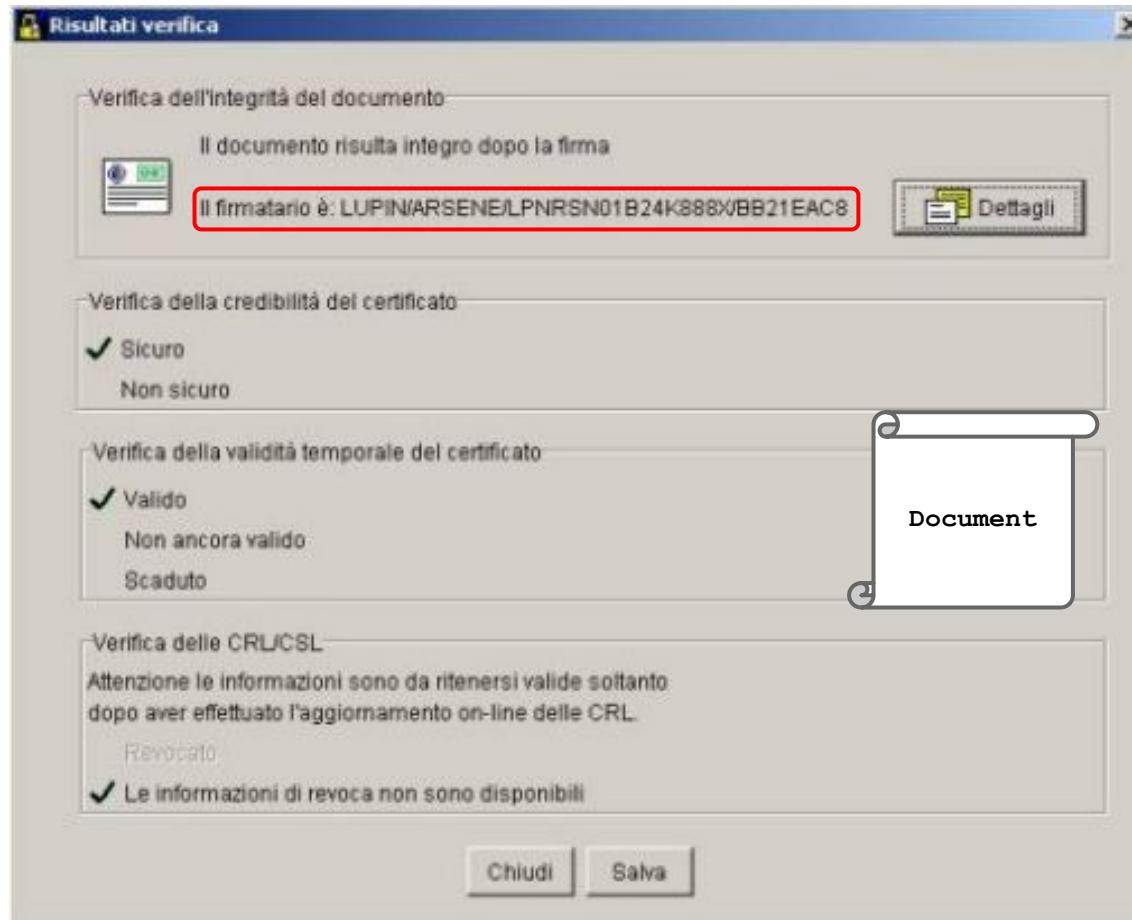
1. Generate a **fake *root* certificate** with the same name as a real one (e.g., PostECom itself)
2. Use this to generate a **fake *user* certificate** (in our example Arsène Lupin)
3. Use **Arsène Lupin's certificate** to sign theft and burglary confessions.
4. Include the fake root cert to the PKCS#7 envelope.



# Les jeux sont faits: Lupin's Certificate



# The Exploit: Arséne Lupin signature



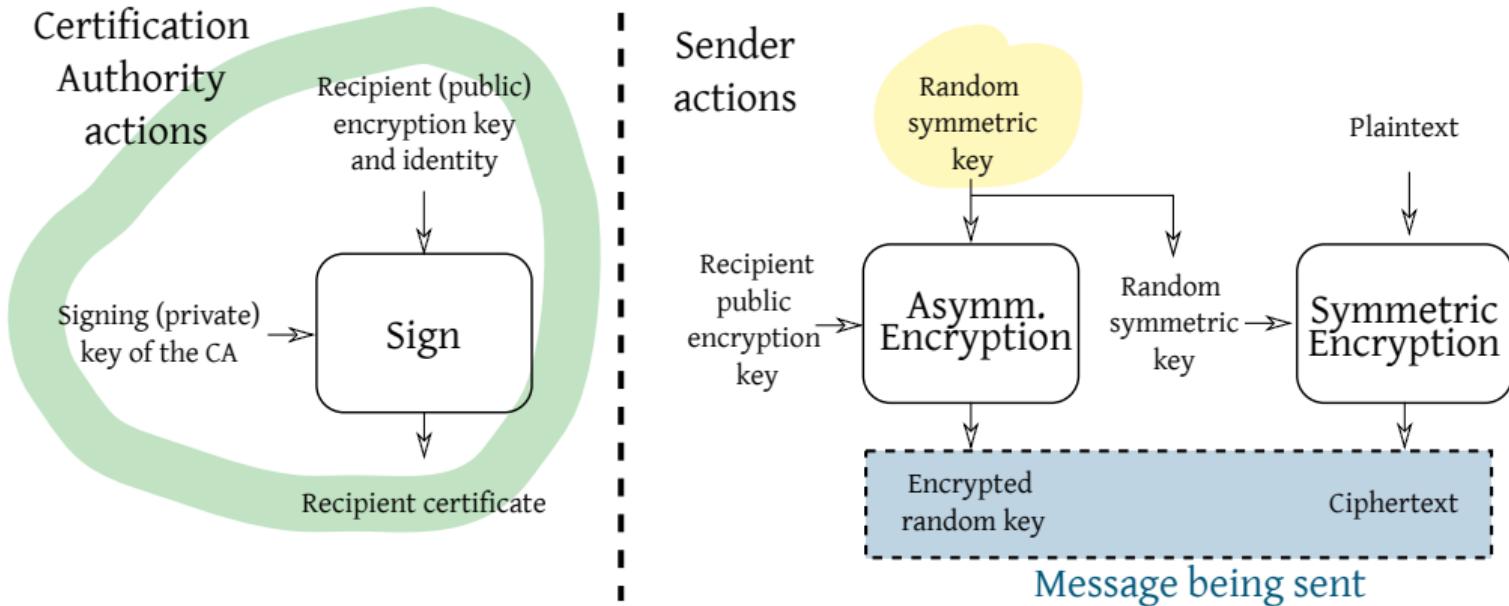
Best comment by Postecom: this is "*by design*"  
(yep: wrong design, but still design!)

# Horror story 2 overview

## Trust with care

- 2003-02-20: Firma&Cifra was the digital signature application by PostECom
- When presented with any certificate bundled with a signed document, it considered the certificate authentic and added it to its trusted storage
- Signature forgery as easy as: 1) create your own CA certificate, 2) sign your target-user certificate, 3) sign the document
- Take away point: which certificates reside in your (applications') trusted storage determine **who you trust**

# Putting it all together



This way of communicating is the mainstay of modern secure comm. protocols  
(TLS, OpenVPN, IPSec)

# Directions in modern cryptography

## Issues to solve, features to realize

- What if we have a quantum computer?

- Some computationally hard problems are no longer hard
- Move away from cryptosystems based on factoring/dlog
- Alternatives available and being standardized (2022-04)



- What if we want to compute on encrypted data?

- Yes, but it's moderately-to-horribly inefficient

- What if the attacker has physical access to the device computing the cipher (or some way of remotely measure it)

- Take into account **side channel** information in the attacker model

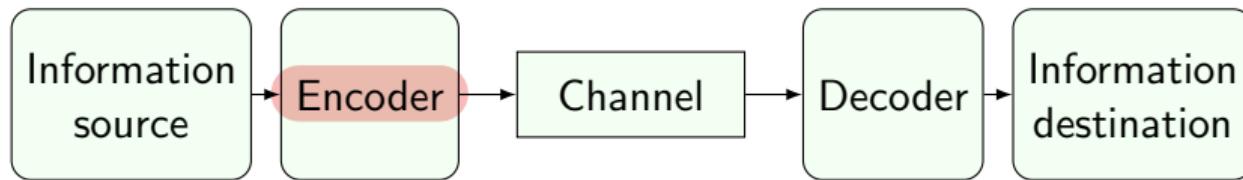
What is Shannon's information theory?

- Shannon's [3] way to mathematically frame communication
- A way to quantify information

What do we need this for (in this course)?

- Quantitatively frame “luck” and “guessing”

# Basic Definitions



## Basics

- A communication takes place between two **endpoints**
  - sender: made of an information source and an encoder
  - receiver: made of an information destination and a decoder
- Information is carried by a channel in the form of a sequence of **symbols** of a finite alphabet

# Transmitting and receiving

Losing uncertainty = Acquiring information

- The receiver gets information only through the channel
  - it will be **uncertain** on what the next symbol is, until the symbol arrives
  - thus we model the sender as a **random variable**
- Acquiring information is modeled as getting to know an outcome of a random variable  $\mathcal{X}$ 
  - the amount of information depends on the distribution of  $\mathcal{X}$
  - intuitively: the closer is  $\mathcal{X}$  to a uniform distribution, the higher the amount of information I get from knowing an outcome
- Encoding maps each outcome as a finite sequence of symbols
  - More symbols should be needed when more information is sent



# Measuring uncertainty: Entropy



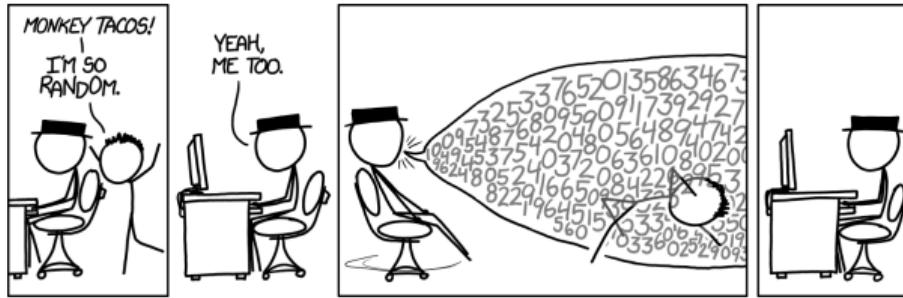
## Desirable properties

- Non negative measure of uncertainty
- “combining uncertainties” should map to adding entropies

## Definition

- Let  $\mathcal{X}$  be a discrete r.v. with  $n$  outcomes in  $\{x_0, \dots, x_{n-1}\}$  with  $\Pr(\mathcal{X} = x_i) = p_i$  for all  $0 \leq i \leq n$
- The entropy of  $\mathcal{X}$  is  $H(\mathcal{X}) = \sum_{i=0}^{n-1} -p_i \log_b(p_i)$
- The measurement unit of entropy depends on the base  $b$  of the logarithm: typical case for  $b = 2$  is bits

# Examples



<https://xkcd.com/1210/>

$\mathcal{X}$  : Uniformly random 6 letters word

- $\mathcal{X}$  is a sequence of 6 unif. random letters ( $6^{26} \approx 3.1 \cdot 10^7$ )
  - $H(\mathcal{X}) \approx \sum_{i=0}^{3.1 \cdot 10^7} -\frac{1}{3.1 \cdot 10^7} \log_b \left( \frac{1}{3.1 \cdot 10^7} \right) \approx 28.2b$
- $\mathcal{X}$  is a uniform pick from 6-letters English words
  - $H(\mathcal{X}) \approx \sum_{i=0}^{6300} -\frac{1}{6300} \log_b \left( \frac{1}{6300} \right) \approx 12.6b$

# Shannon's noiseless coding theorem



## Statement (informal)

It is possible to encode the outcomes  $n$  of i.i.d. random variables, each one with entropy  $H(\mathcal{X})$ , into no less than  $nH(\mathcal{X})$  bits per outcome. If  $< nH(\mathcal{X})$  bits are used, some information will be lost.

## Consequences

- Arbitrarily compression of bitstrings is impossible without loss
  - Cryptographic hashes must discard some information
- Guessing a piece of information (= one outcome of  $\mathcal{X}$ ) is at least as hard as guessing a  $H(\mathcal{X})$  bit long bitstring
  - overlooking for a moment the effort of decoding the guess





## A practical mismatch

- It is possible to have distributions with the same entropies

## Plucking low-hanging fruits

- We define the **min-entropy** of  $\mathcal{X}$  as  $H_{\infty}(\mathcal{X}) = -\log(\max_i p_i)$
- Intuitively: it's the entropy of a r.v. with uniform distribution, where the probability of each outcome is  $(\max_i p_i)$
- Guessing the most common outcome of  $\mathcal{X}$  is at least as hard as guessing a  $H_{\infty}(\mathcal{X})$  bit long bitstring

## Example

A very biased r.v.

Consider  $\mathcal{X}$  : 
$$\begin{cases} \mathcal{X} = 0^{128} & \text{with } \Pr \frac{1}{2} \\ \mathcal{X} = a, a \in 1\{0, 1\}^{127} & \text{with } \Pr \frac{1}{2^{128}} \end{cases}$$

Intuition and quantification

- Predicting an outcome shouldn't be too hard: just say  $0^{128}$
- $H(\mathcal{X}) = \frac{1}{2}(-\log_2(\frac{1}{2})) + 2^{127} \frac{1}{2^{128}} (-\log_2(\frac{1}{2^{128}})) = 64.5\text{b}$
- $H_\infty(\mathcal{X}) = -\log_2(\frac{1}{2}) = 1\text{b}$
- Min-entropy tells us that guessing the most common output is as hard as guessing a single bit string

# The Systems Perspective

“You have probably seen the door to a bank vault...10-inch thick, hardened steel, with large bolts...We often find the digital equivalent of such a vault door installed in a tent. The people standing around it are arguing over how thick the door should be, rather than spending their time looking at the tent.”

(Niels Ferguson & Bruce Schneier, Practical Cryptography)

# Conclusions

## Perfect ciphers vs. real world: brute-forcing

- Broken-unbroken ciphers: need for transparency
- Key lengths matters
- Symmetric, asymmetric algorithms and hash functions
- PKI and CAs and their complexity

We saw several case studies of attacks against crypto applications

- They had everything to do with systems security without even touching the algorithms themselves

# Bibliography I

-  Giovan Battista Bellaso.  
La cifra del sig. giovan battista bellaso, 1553.
-  John Nash.  
Personal communication to the us national security agency, Feb. 1955.
-  Claude E. Shannon.  
A mathematical theory of communication.  
*Bell Syst. Tech. J.*, 27(3 and 4):379–423 and 623–656, 1948.
-  Claude E. Shannon.  
Communication theory of secrecy systems.  
*Bell Syst. Tech. J.*, 28(4):656–715, 1949.
-  Marc Stevens.  
The hashclash project, 2009.
-  Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov.  
The first collision for full SHA-1.  
In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 570–596. Springer, 2017.

# Bibliography II



Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger.

Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate.

In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2009.

# Further reading: a practical attack based on MD5 collisions

- It is known that MD5 allows a **chosen prefix collision** under certain constraints
  - Here the attack is used to create **two valid CA certificates with the same signature**:  
<http://www.win.tue.nl/~bdeweger/CollidingCertificates/>
  - Extended to threaten CAs in 2008:  
<http://www.win.tue.nl/hashclash/rogue-ca/>
- An evolution of the technique was used in **Flame**, a nasty malware used against several middle-Eastern targets
  - <http://trailofbits.files.wordpress.com/2012/06/flame-md5.pdf>