2. A (Quick) Introduction to Cryptography

Computer Security Courses @ POLIMI

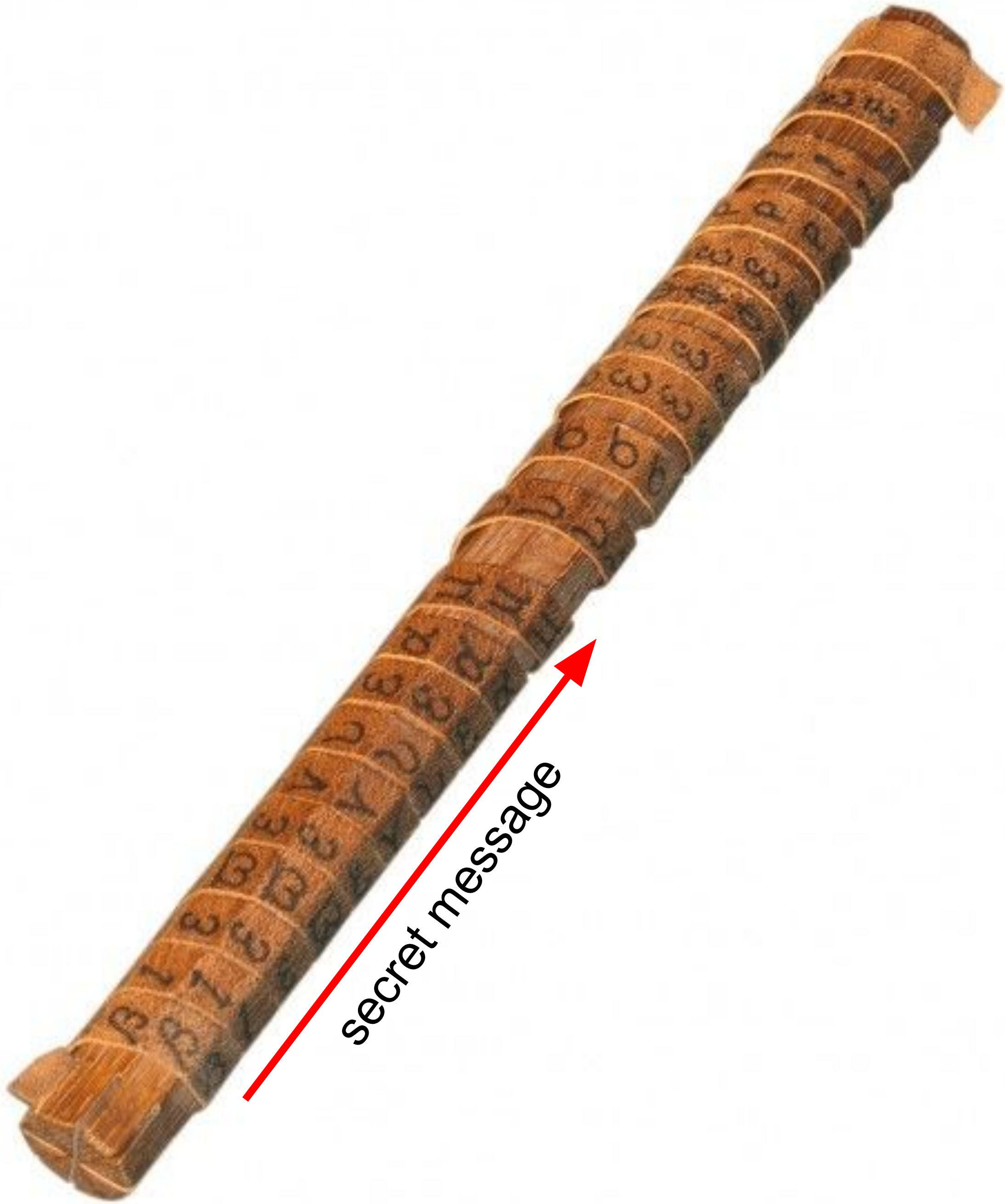
Prof. Carminati & Prof. Zanero

Word of Warning

- This is a short, simplified introduction to cryptography
- We will only introduce what is needed for systems security discussions
- Mostly, we will treat mathematical concepts as black boxes
- A much more in-depth discussion of cryptography takes place in course:
  - 095947 - CRYPTOGRAPHY AND ARCHITECTURES FOR COMPUTER SECURITY (Prof. G. Pelosi)

A Brief History of Cryptography (1)

- 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.
    - E.g. "scytale", the wand of command of the army of Athens. According to Plutarco, in use since the IX century b.C.

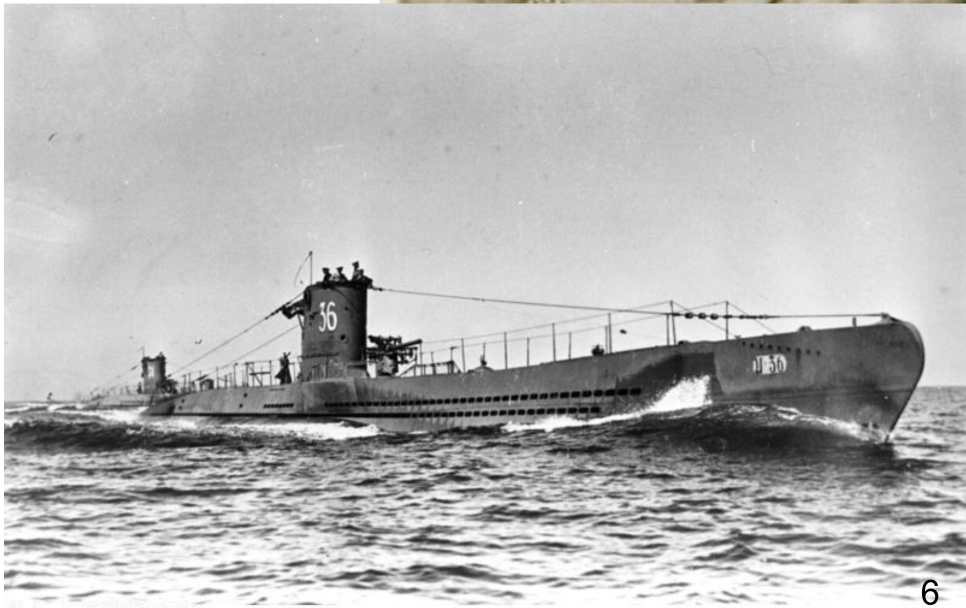

secret message

A Brief History of Cryptography (2)

- 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.
- Also the formalisms of cryptography were born out of wartime needs.

When Math Won a War

- During WW II, **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.



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

When Math Won a War

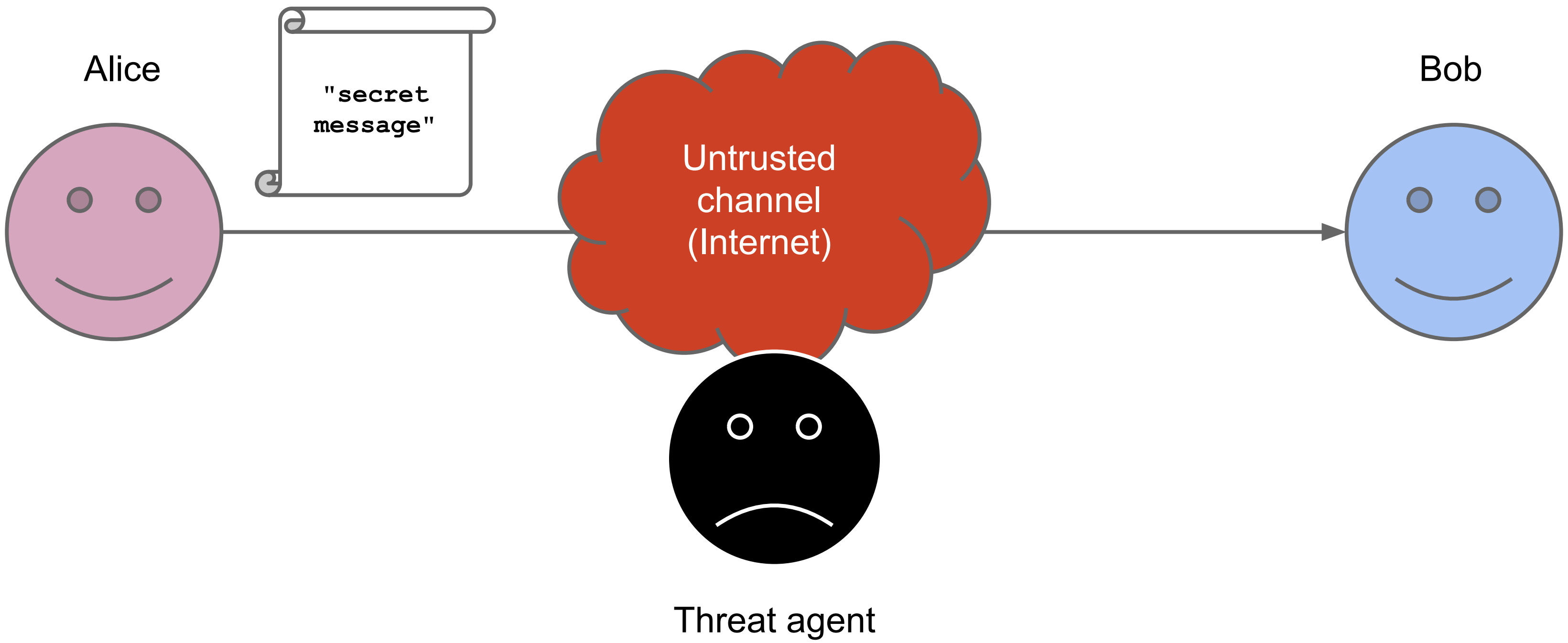
- During WW II, **Alan Turing** worked at Bletchley Park to **break** Axis ciphers, in particular the **Enigma cipher**.
- Birth of the first universal computers was stimulated by this effort.


THE BOMBE (replica)

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"*.

The Problem to Solve: Confidentiality and Integrity



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 criptographie militaire”*, 1883
    - Sometimes called “Kerchoffs/Shannon”
- 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!

Perfection is not of This World

Shannon wondered: *“Is there a perfect cipher?”*

A cipher such that no matter time or strength spent in analysis, does not leak any secret?

- $P(M = m)$ probability of observing message m
- $P(M = m \mid C = c)$ probability that message was m | given that ciphertext c was observed

The cipher is a **perfect cipher** if and only if

$$P(M = m \mid C = c) = P(M = m)$$

Shannon's Theorem

In a *perfect cipher* the number of keys must be greater or equal to the number of possible messages: $|K| \geq |M|$

Sketch of the proof (by reduction to absurdity):

C must be at least "as large" as M: $|C| \geq |M|$.

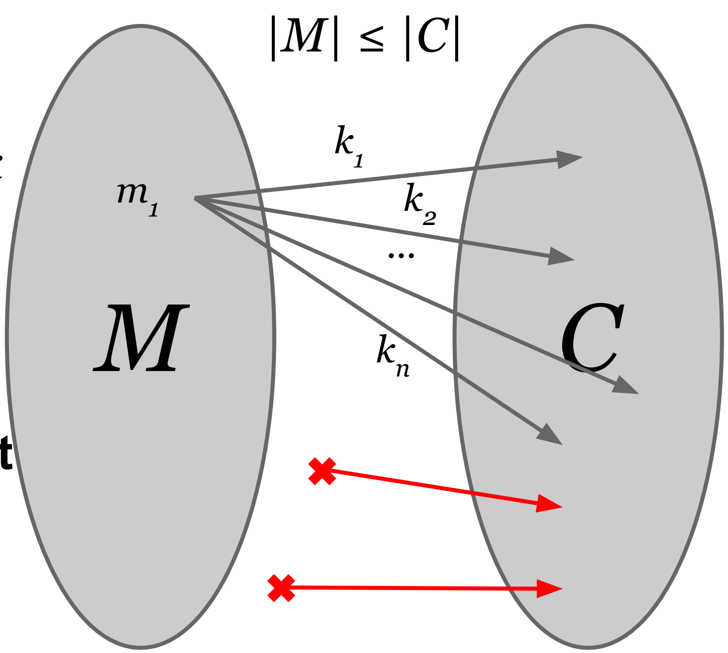
For each *m*, there is one ciphertext *c* for each key k

If $|K| < |M|$, the number of ciphertexts for each message would be < than $|M|$. Thus, given that:

$$|C| \geq |M| > |K|$$

this means that there are values in C that cannot correspond to (some) of the values in M.

This is **absurd by definition** of perfect cipher.



The One Time Pad: Perfect Cipher

- XOR of a message m and a random key k of the same size of m: $len(k) = len(m)$
  - The key is pre-shared and consumed while writing. Can never be re-used again!
- The OTP is a **minimal perfect cipher**
  - Minimal because $|K| = |M|$
- Terribly inconvenient; used only in special settings
- Real-world algorithms are **not perfect**, and so can be broken

Imperfections and Brute Force

- Real-world ciphers are imperfect
  - 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 <u>real world cipher</u>
  - Try all possible keys, until **one** produces an output that *makes sense.*
  - **<u>Perfect ciphers (one time pads) are not vulnerable</u>** because trying all the (random) keys will yield all the (possible) plaintexts, which are all equally likely (= no clue)

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?

- I give you a ZIP-compressed file encrypted with a (secret) 4-bytes key
- I tell you how I encrypted it: algorithm should not be secret (by Kerchoffs):
  - $C = K$ xor M
  - Example:
    - K(hex) = `AA BB CC DD (repeat the key)`
    - M(hex) = `50 4B 03 04 BA DA 55 55 .. .. .. (and so on)`

      XOR
    - C(hex) = `FA F0 CF D9 10 61 99 88 .. .. .. .. .. .. ..`
- I give you a ZIP file encrypted with a key: can you recover the key w/o bruteforcing?

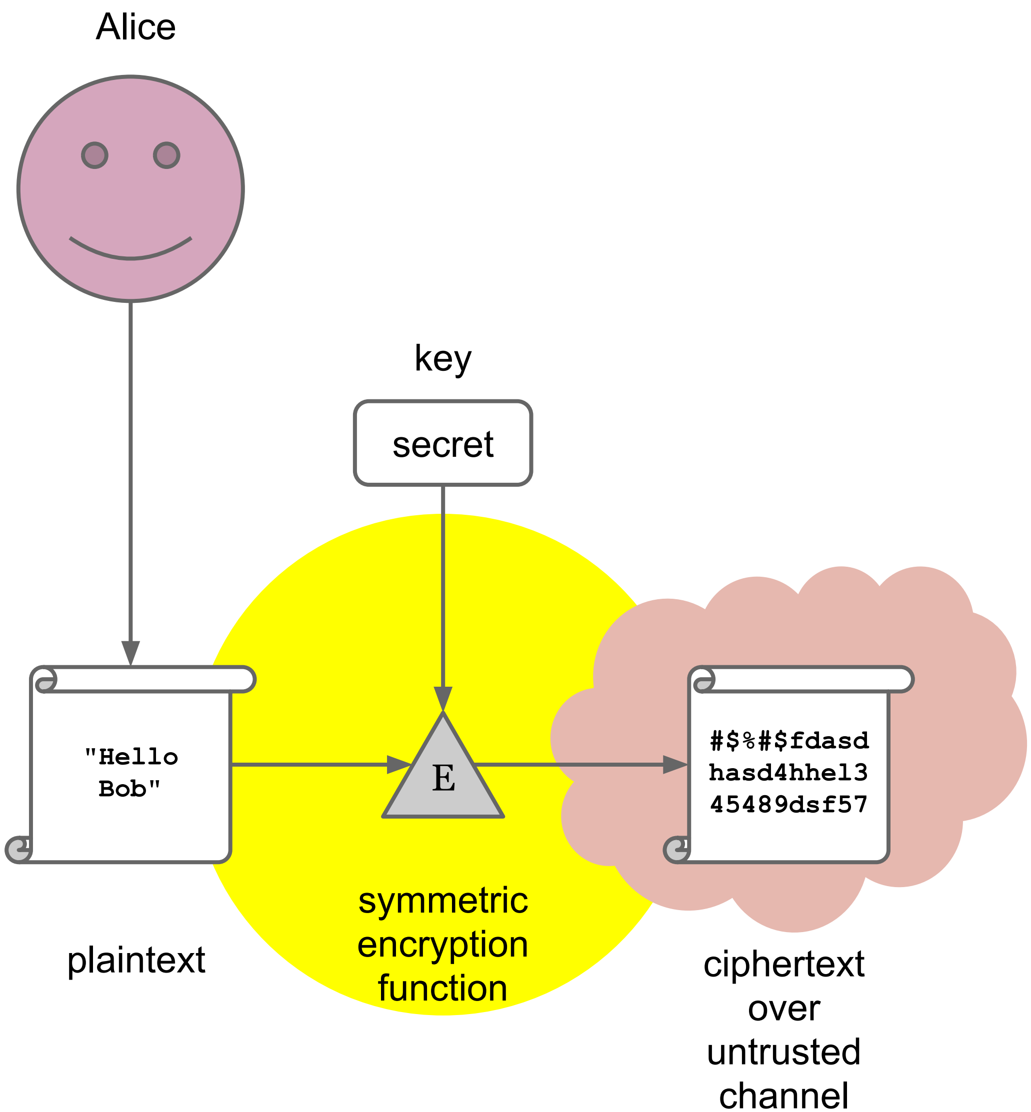
Key Points to Remember

- Security of a cryptosystem is based on the robustness of the *algorithm.*
- No algorithm, save the one-time-pad, is invulnerable.
- An algorithm is *broken* if there is at least one attack faster than brute forcing.
- There is no way to prove robustness of a cipher, save by trying to break it.
- Secret algorithms are insecure. Security is *transparency*.

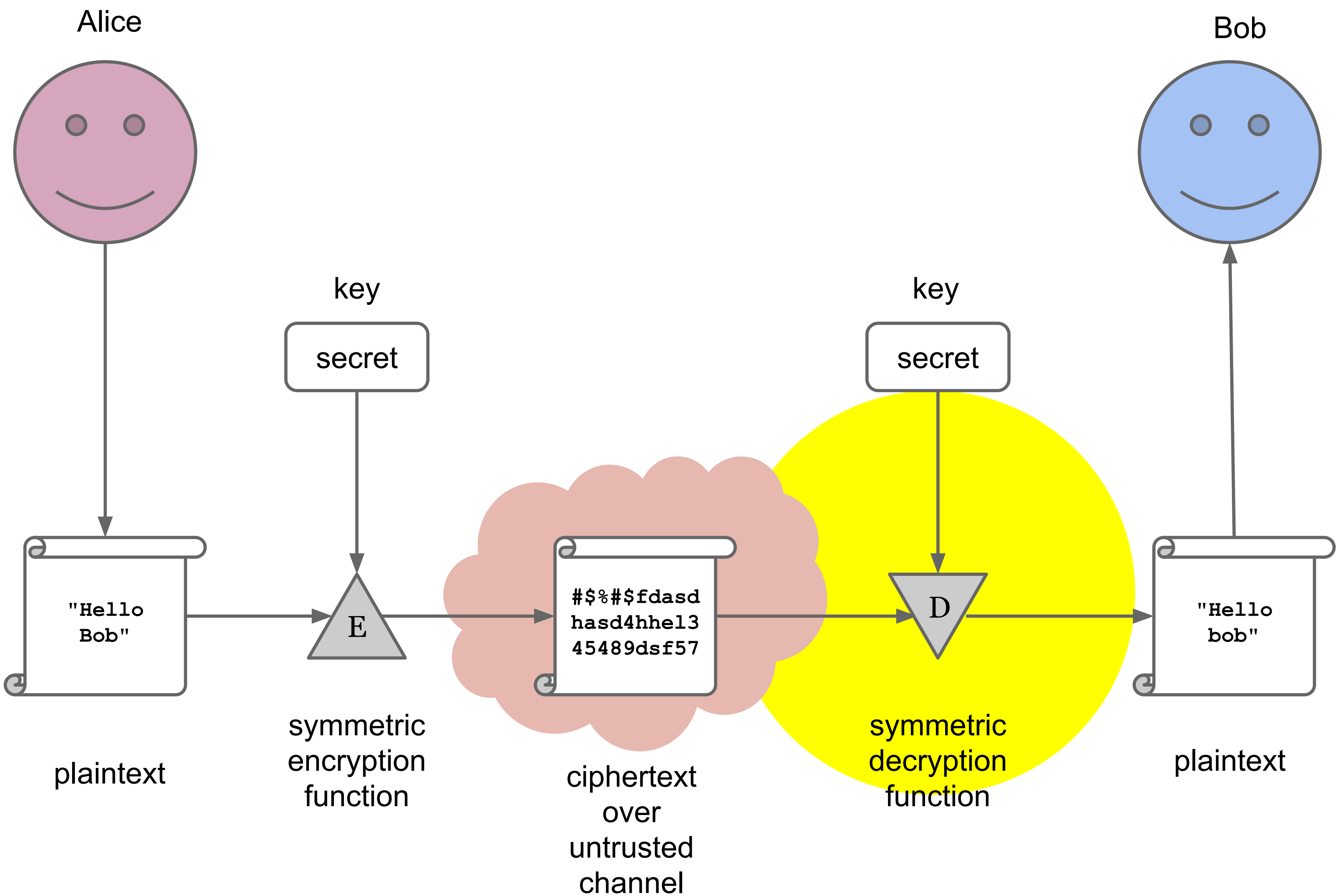
Symmetric Encryption

Confidentiality

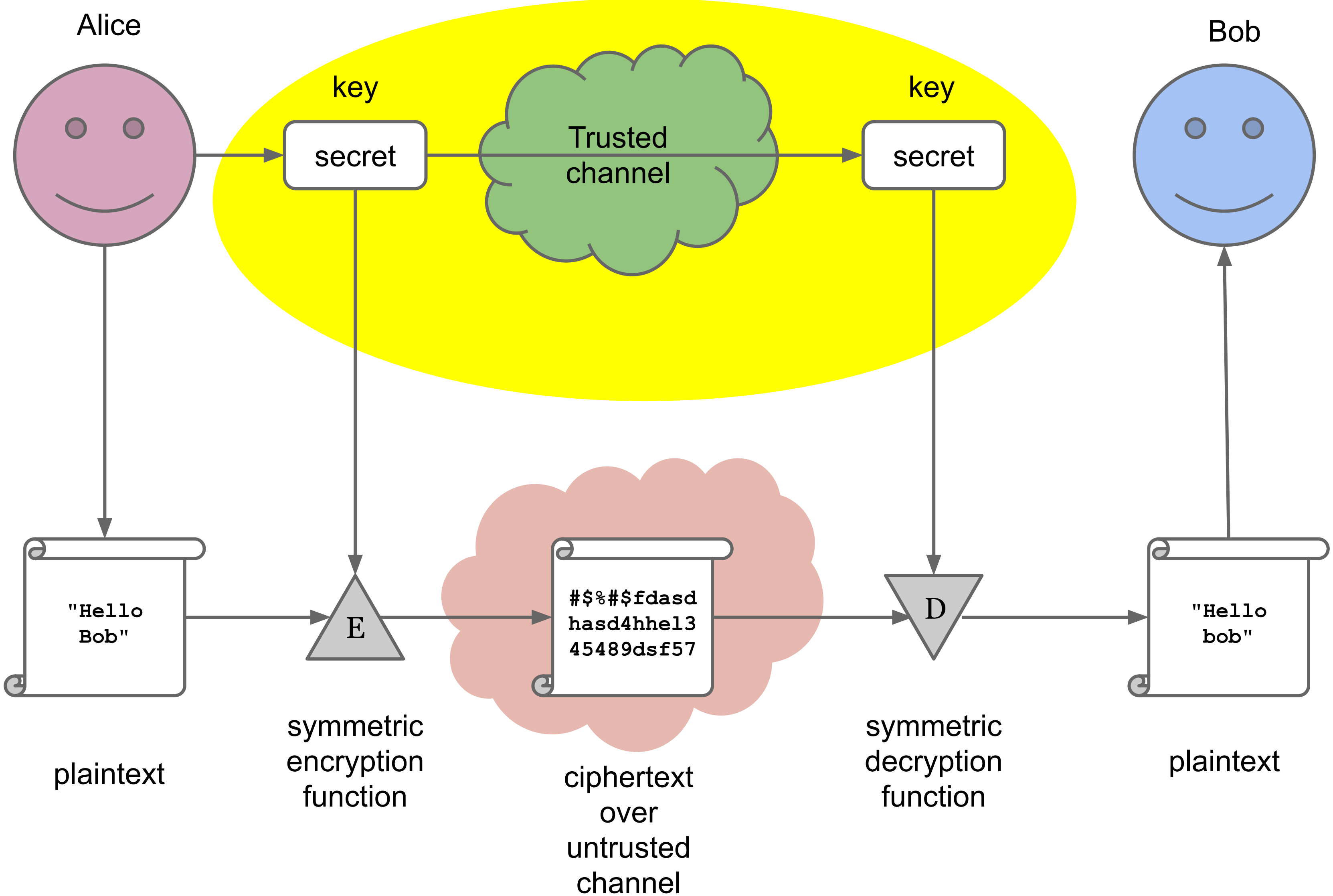
Symmetric Encryption



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!):

- 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.

Second ingredient: transposition

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

plaintext

ciphertext

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

Toy example (matrix):

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

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

Modern Symmetric Ciphers

- Modern ciphers mix diffusion and substitution
- Some well known ciphers (just so you recognize the names)
    - DES (Data Encryption Standard, 1977), and its evolution 3DES
    - IDEA (1991)
    - BlowFish (1993)
    - RC5 (1994)
    - CAST-128 (1997)
    - Rijndael (since 2000 it is the AES, Advanced Encryption Standard)

Case Study: DES

Originally designed by IBM (1973-1974)

Its core function is an S-box

- Normally, they use fixed tables, as in DES
- For some ciphers (e.g., Blowfish and Twofish), tables are generated dynamically from the key.

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

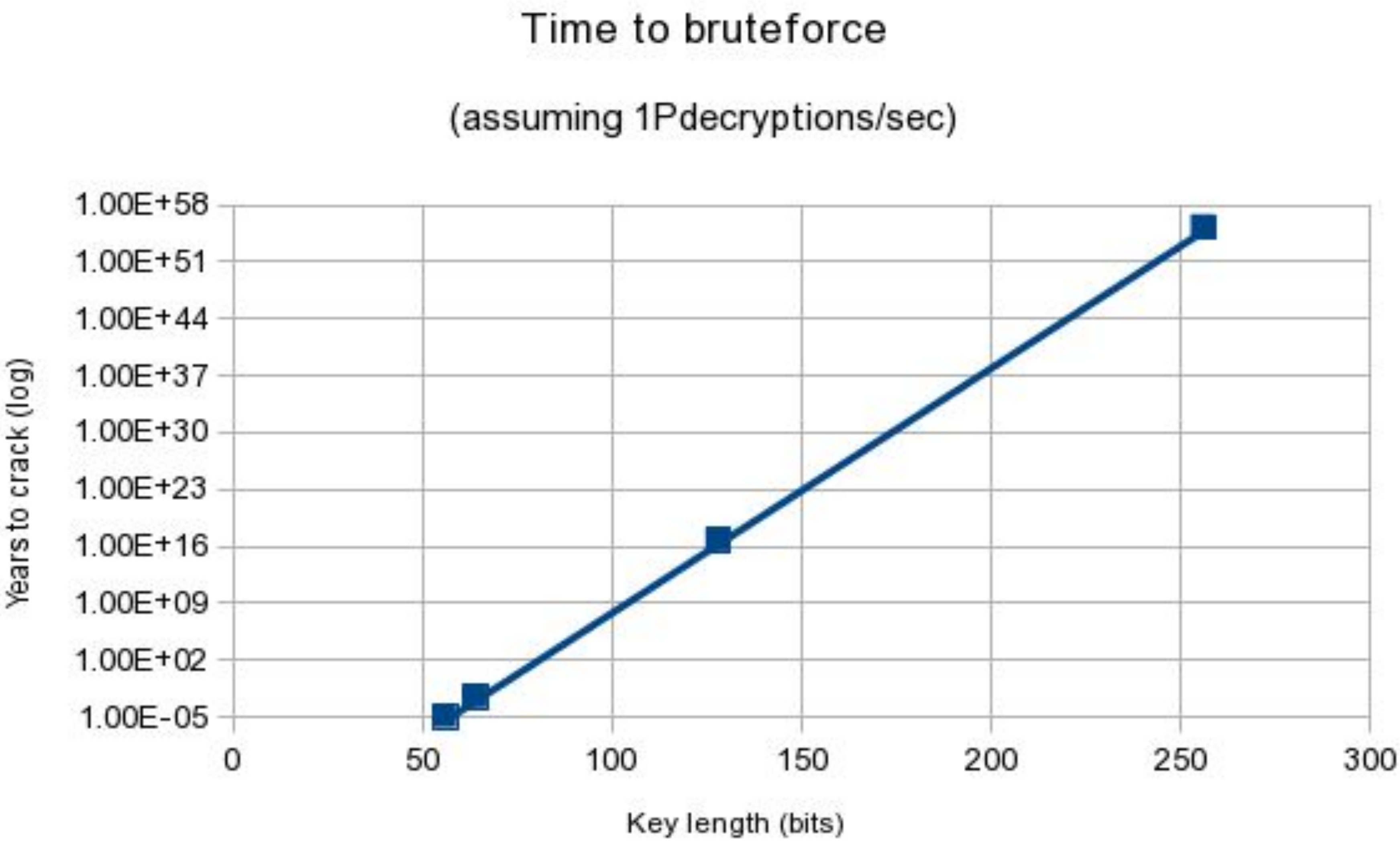
- **Late 1980s:** *differential cryptanalysis* discovered
- **1993**: shown that the original S-boxes would have made DES vulnerable to the differential cryptanalysis, whereas the NSA-designed S-boxes were specifically immune to that.
- **Wait!** Wasn't differential cryptanalysis unknown until late 1980s?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

Keyspace and Brute Forcing

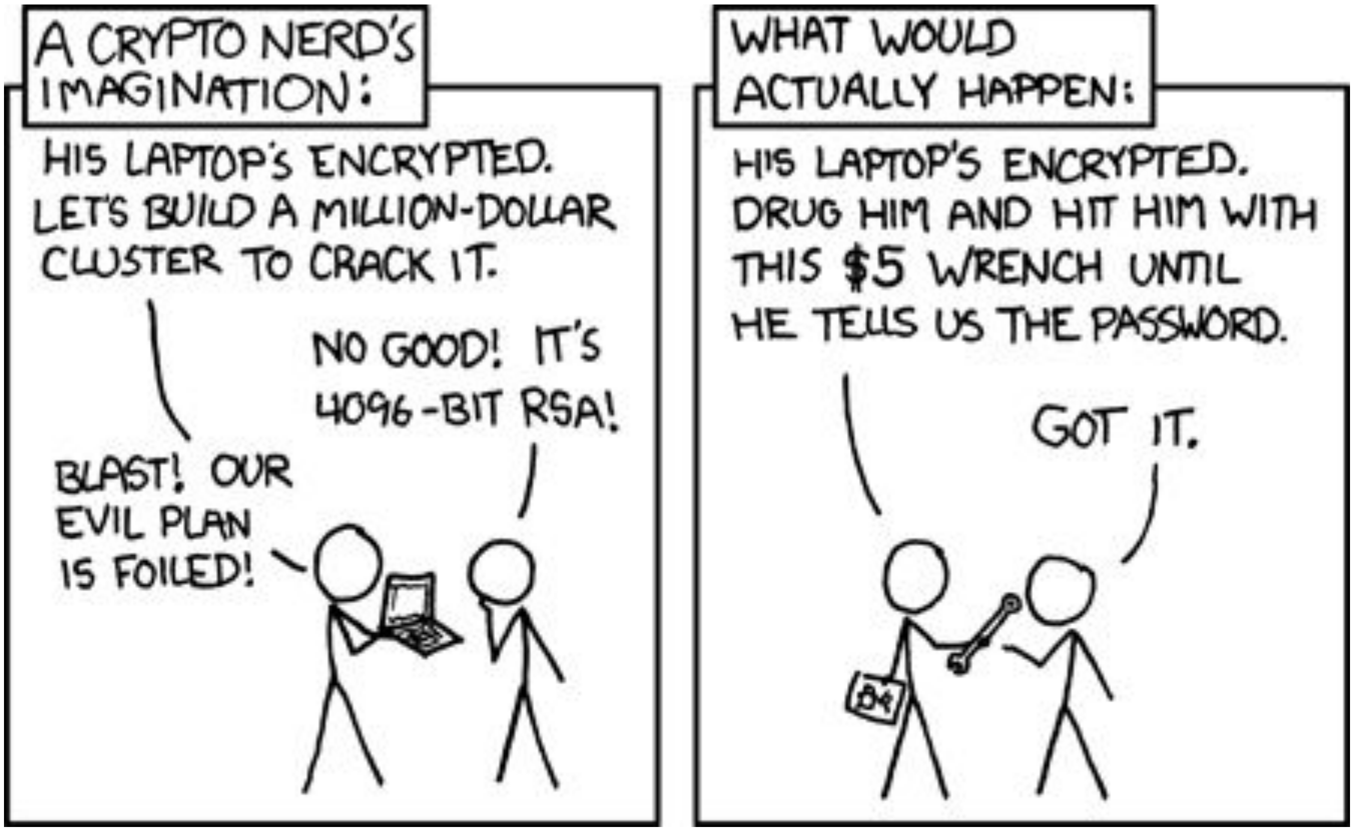
Keyspace generally measured in bits

- 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



A good point to remember...



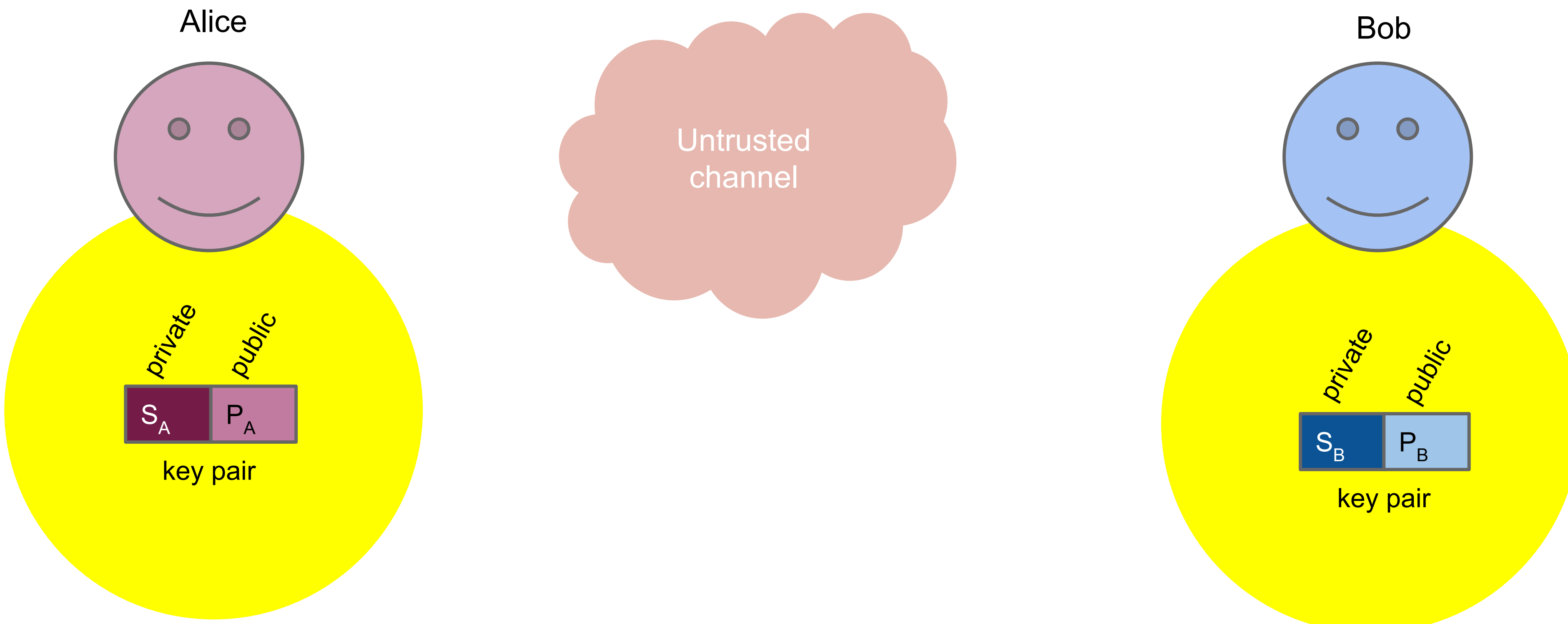
Asymmetric Encryption

Confidentiality (plus something more)

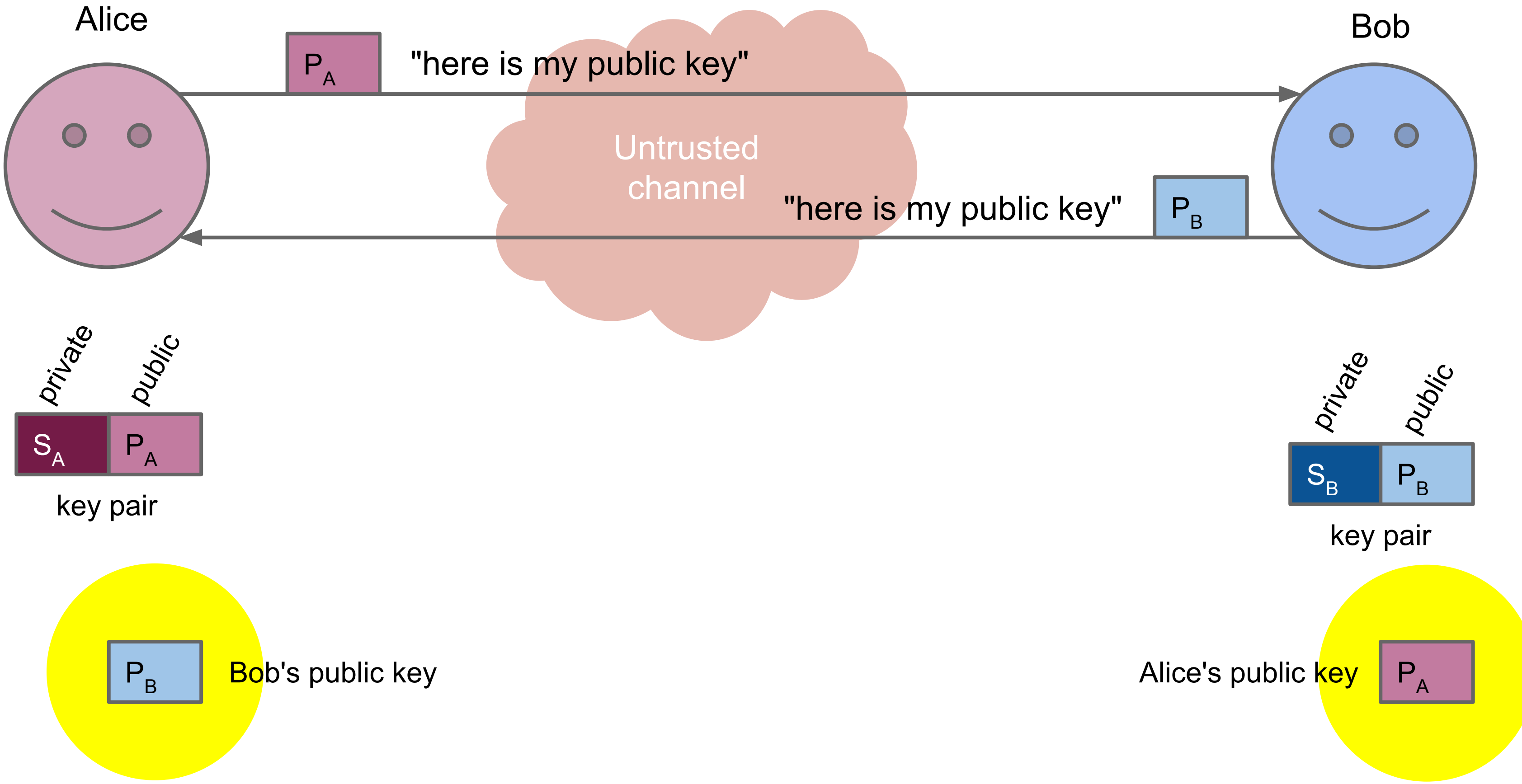
Asymmetric Encryption

- **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.
  - The keys cannot be retrieved from each other
- Introduced in 1976 (W. Diffie & M. Hellman)
- Also called *"public key cryptography"*
  - **Idea:** one of the two keys is kept **private** by the subject, and the other can be **publicly** disclosed.
  - This solves radically the problem of **key exchange**
- We will not describe their maths in depth
  - They use a <u>one-way function with a trapdoor</u>
  - They are usually computation-intensive

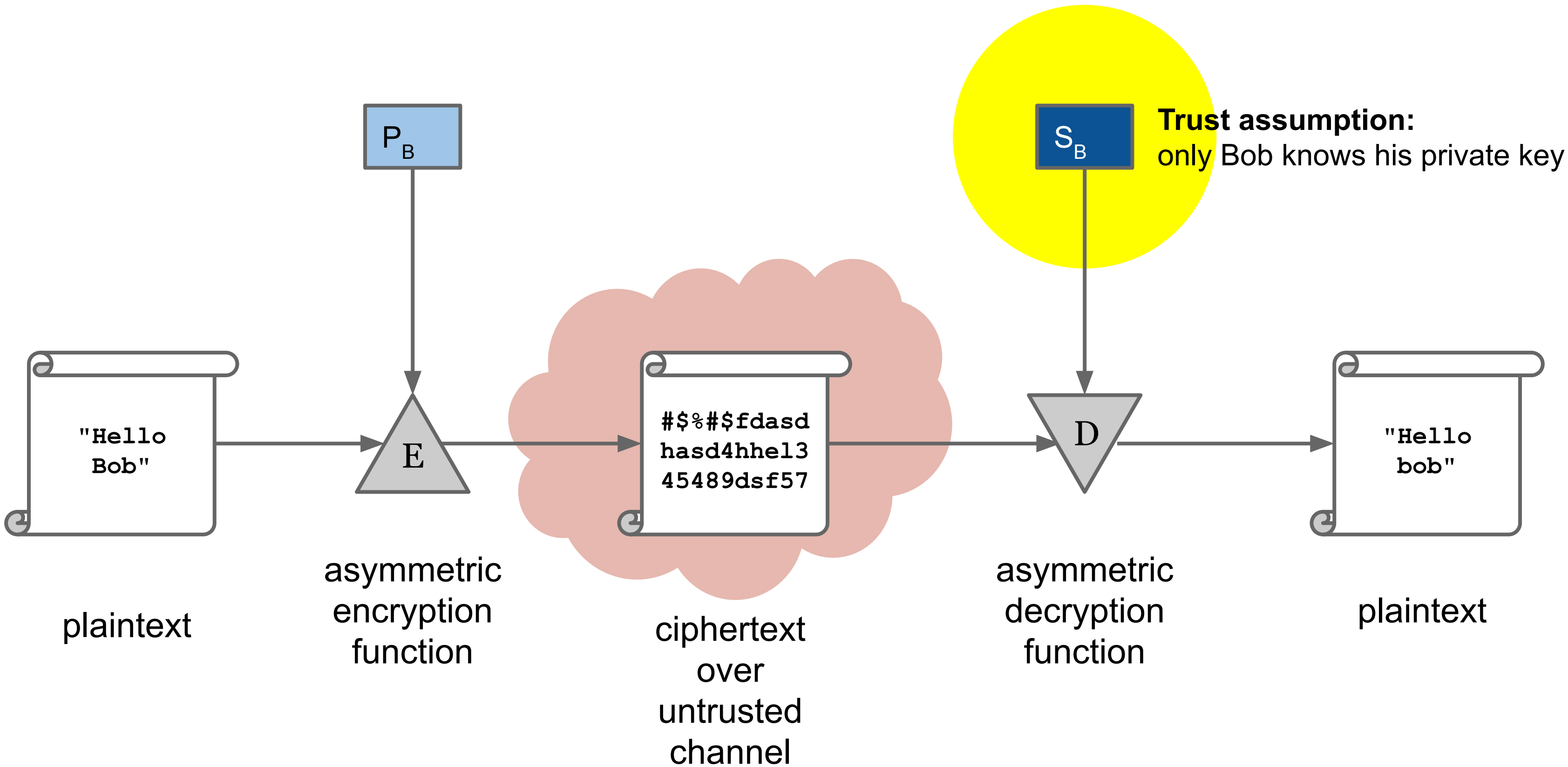
Asymmetric Encryption: Key Exchange



Asymmetric Encryption: Key Exchange



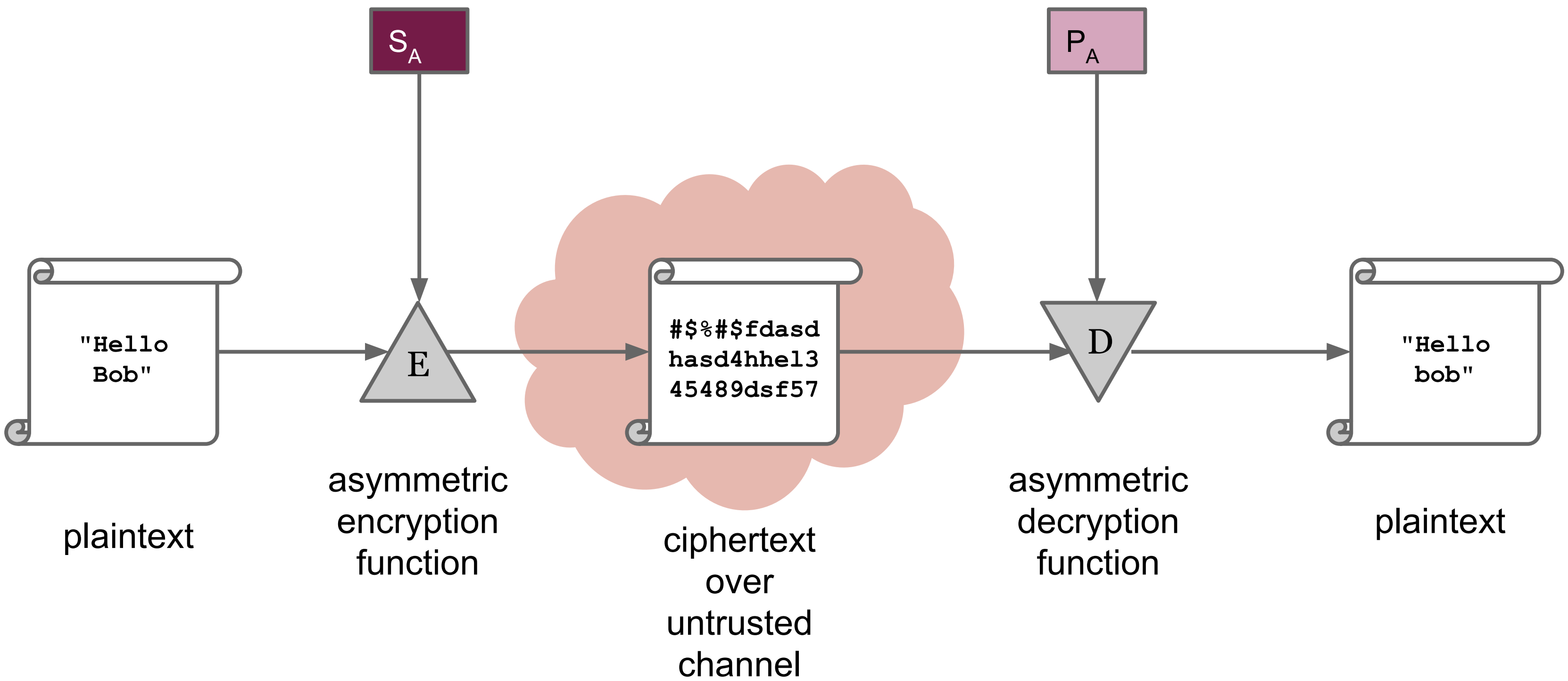
Asymmetric Encryption: Communication



Exercise: what is this instead?

Trust assumption:
only Alice knows her private key

Everybody knows Alice's public key



Common Asymmetric Ciphers

This is a brief list of names of commonly used asymmetric algorithms:

- Diffie-Hellman (1976)
- RSA (1977, Ron Rivest, Adi Shamir, Len Adleman)
- DSS (1991, FIPS PUB 186)
- ECC (IEEE P1363, elliptic curve cryptography)

Example: Diffie-Hellman Exchange

- 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: **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
  - $3^1 \bmod 7 = 3$ $3^2 \bmod 7 = 2$ $3^3 \bmod 7 = 6$
  - $3^4 \bmod 7 = 4$ $3^5 \bmod 7 = 5$ $3^6 \bmod 7 = 1$

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

How does D-H work (2) - Keys

Private key (undisclosed)**:**

They pick a number X in *[1, 2, ..., (p-1)]*

Alice X_A | $X_A = 3$

Bob X_B | $X_B = 1$

Public key (disclosed to everyone):

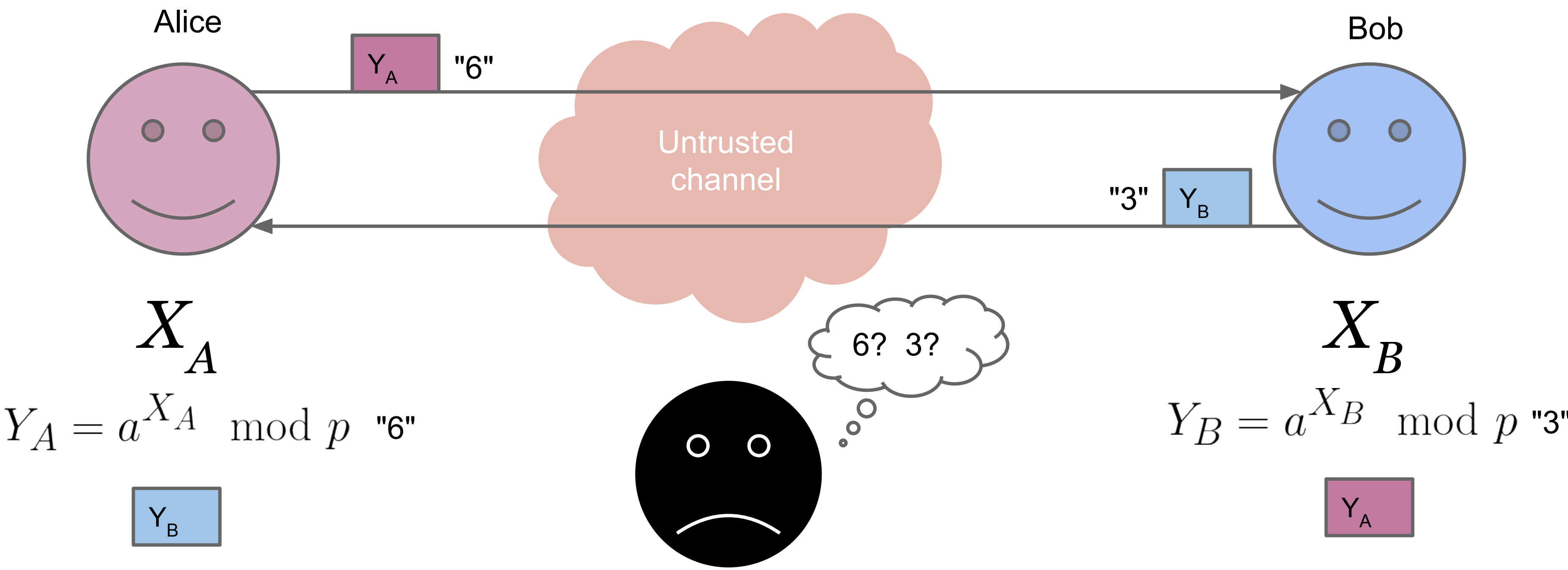
They obtain their public key as

Alice $Y_A = a^{X_A} \mod p$ | $Y_A = 3^3 \mod 7 = 6$

Bob $Y_B = a^{X_B} \mod p$ | $Y_B = 3^1 \mod 7 = 3$

How does D-H work (3)

$$a = 3, p = 7$$



How does D-H work (4) - Secret

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

- Since $Y_A^{X_B} \mod p = (a^{X_A} \mod p)^{X_B} \mod p = (a^{X_A})^{X_B} \mod p = (a^{X_B})^{X_A} \mod p = (a^{X_B} \mod p)^{X_A} \mod p = Y_B^{X_A} \mod p$

- Alice $K_A = Y_B^{X_A} \mod p = 3^3 \mod 7 = \boxed{6}$

$$\boxed{K_B = K_A = K}$$

- Bob $K_B = Y_A^{X_B} \mod p = 6^1 \mod 7 = \boxed{6}$

Anybody else can listen, but cannot compute K

- Because they miss the private key.

The RSA Algorithm (hints) - 1

Same trick, different base 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

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/3/

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)

Hash Functions

Integrity

What is a Hash Function

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

- **Collisions:** codomain “smaller” than domain.

Computationally infeasible to find:

- input x such that $H(x) = h$ (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**

Commonly Used Hash Functions

- SHA-1* (160 bit output)
    - "2017 Google: collision attack performed (pdf)"
    - 2020https://sha-mbles.github.io/?fbclid=IwAR0mjoqQ42ZWTpWTirnMp4-i05Qe-cD7WgTBNwj5piEckkw_2e84pRb8pt0
- SHA-2 / SHA-3 (256)
- MD5 ** (128) "cryptographically broken and unsuitable for further use"

```
$ echo "Hello how are you?" | md5                         # MD5
H(x1) = 1961b669810cb300abd9968f4f4f1530

$ echo "Hello how Are you?" | md5                         # MD5
H(x2) = 4e5a7a345bb647e5f4bef0ce37bf7b77

$ echo "Hello how are you?" | shasum                      # SHA-1
H(x1) = a04ff36f4c549920241353927805 23be88c50049  -

$ echo "Hello how Are you?" | shasum                      # SHA-1
H(x2) = 95e644da66e911faf6ac324a821c14f18f3bb8e7  -
```

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

$$x \text{ such that } H(x) = h$$

or, equivalently, given a specific input x can find

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

<u>faster than brute forcing</u>.

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

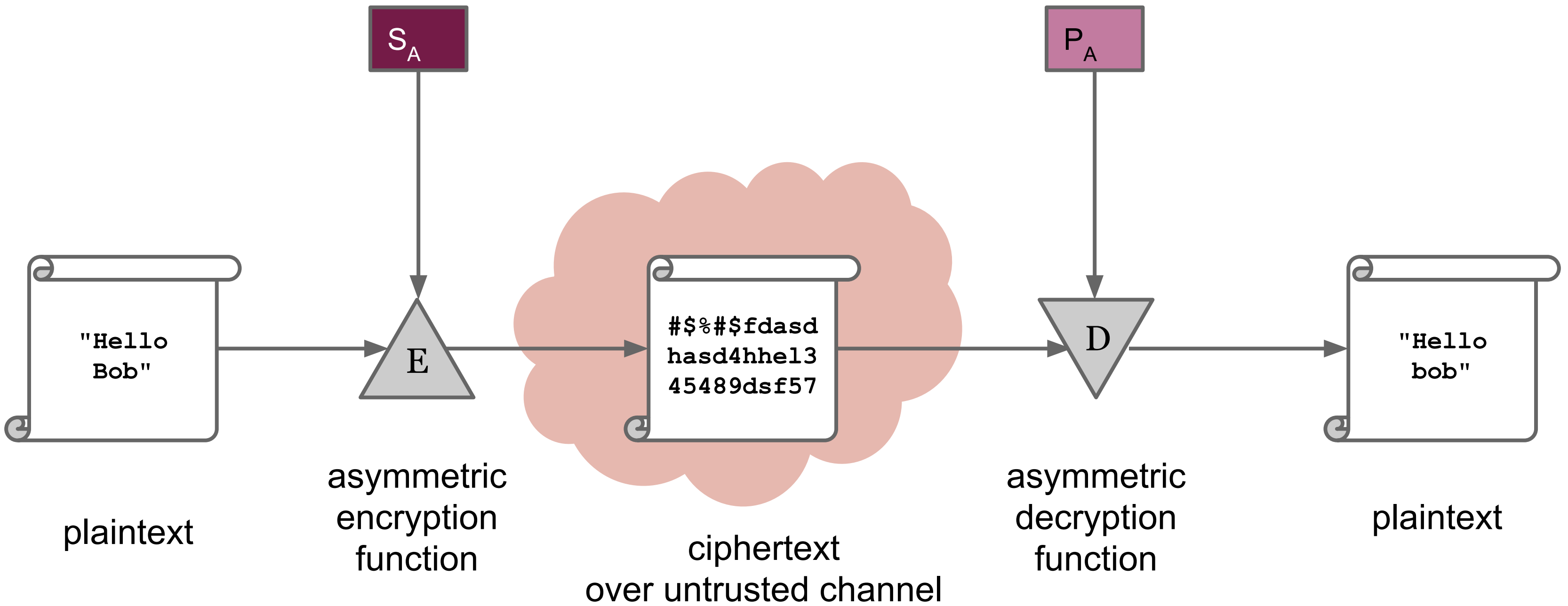
Digital Signature

Confidentiality, Integrity and Authentication
(not to be confused with Availability)

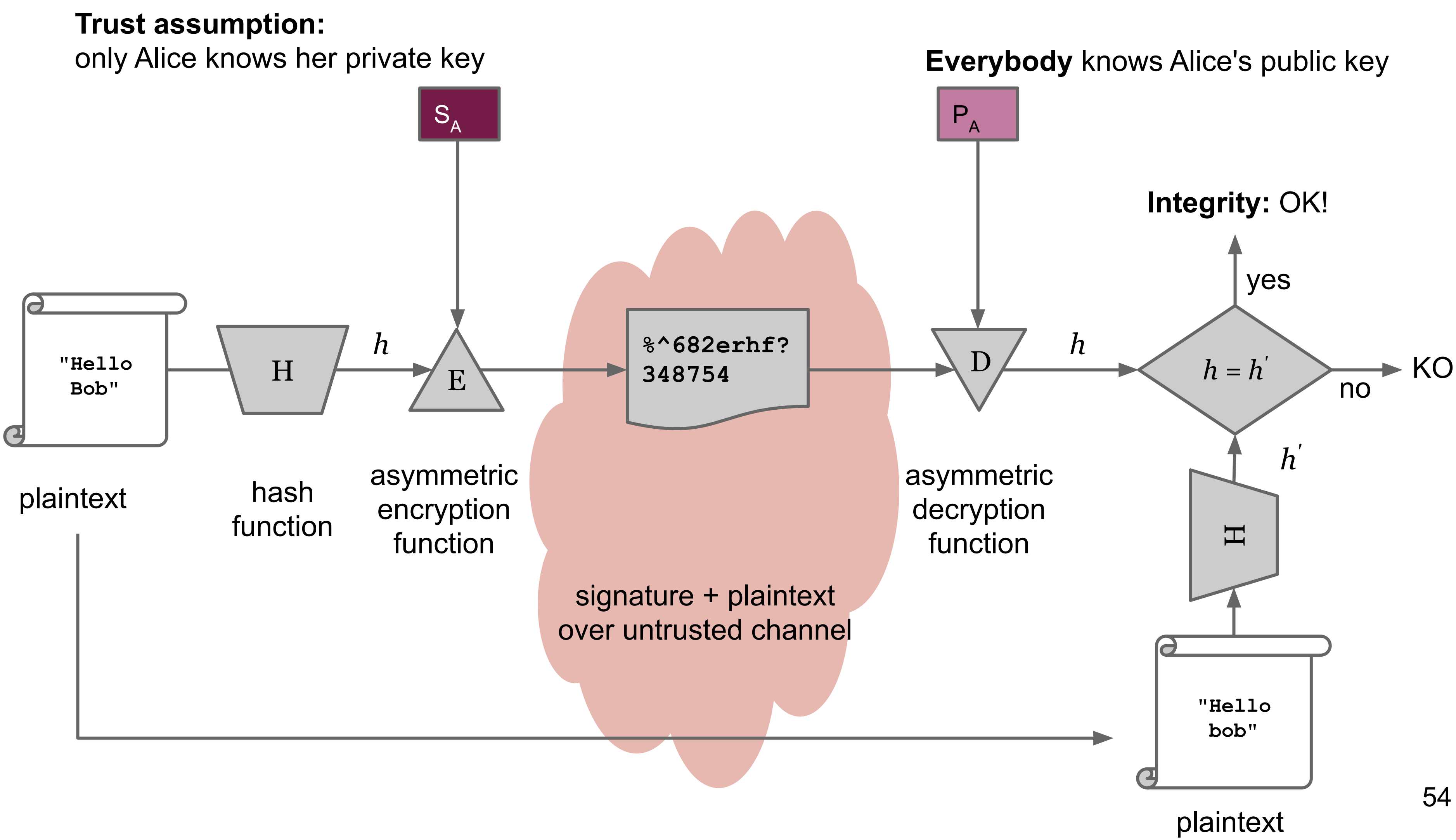
Digital Signature: Message Authentication

Trust assumption:
only Alice knows her private key

Everybody knows Alice's public key



Digital signature: Authentication and Integrity



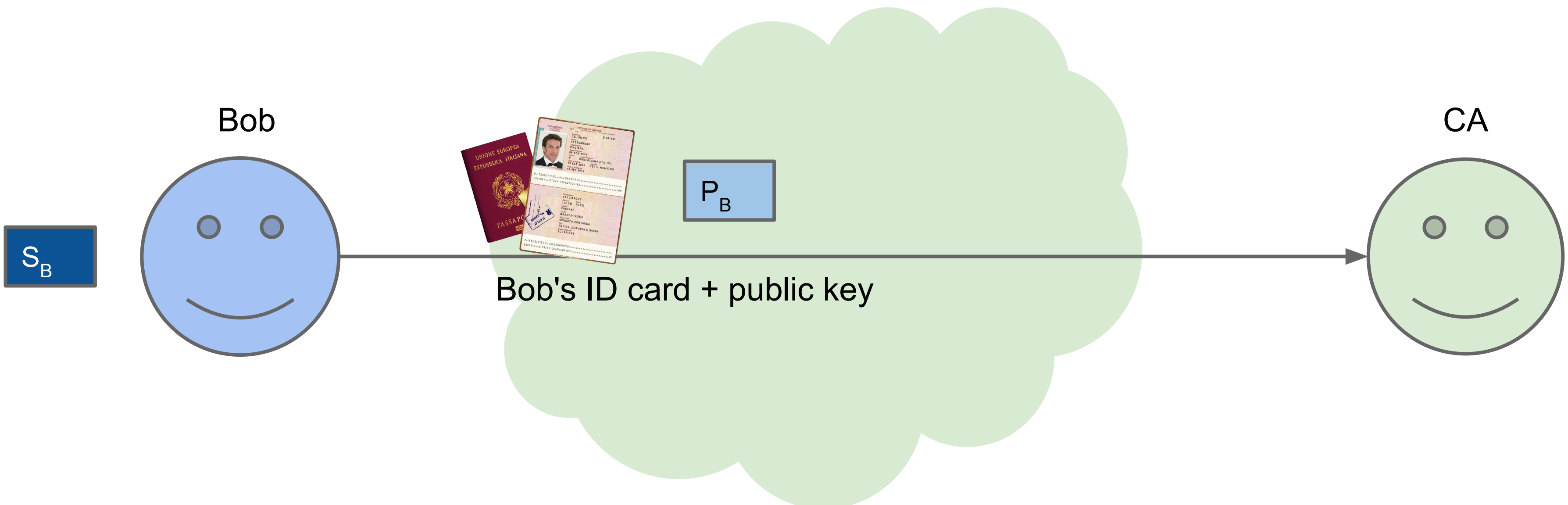
A case of identity

- 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

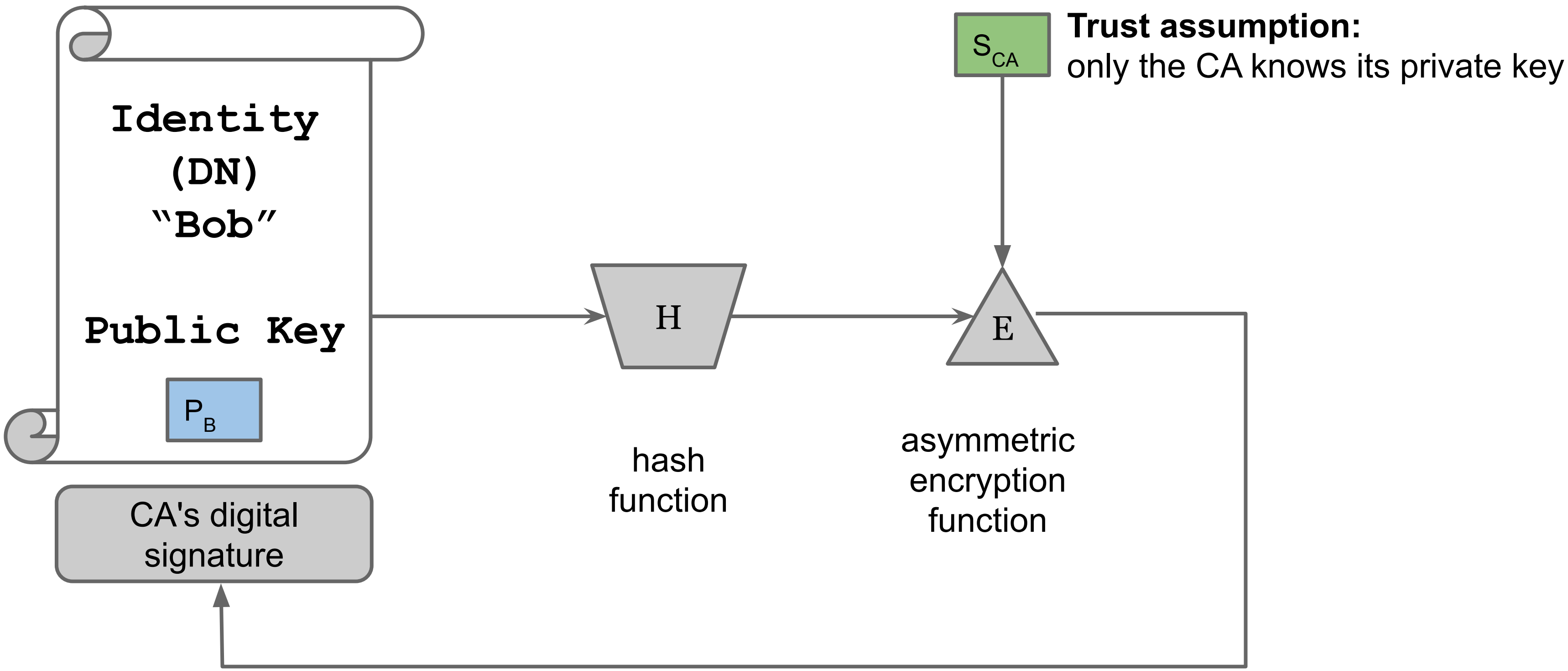
PKI

- 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...provided that we can obtain the **public key** of the CA

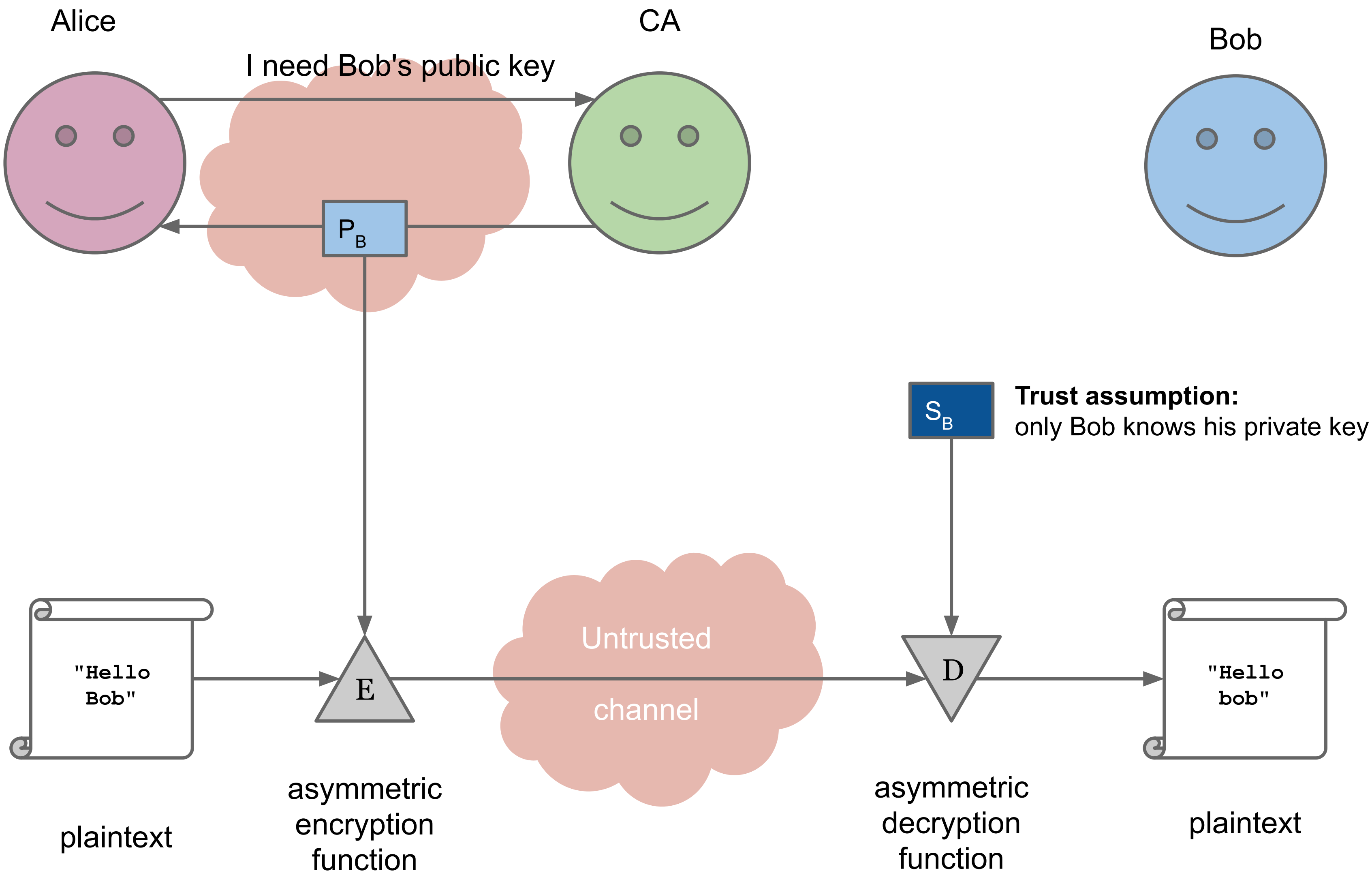
Bob's Digital Certificate (1)



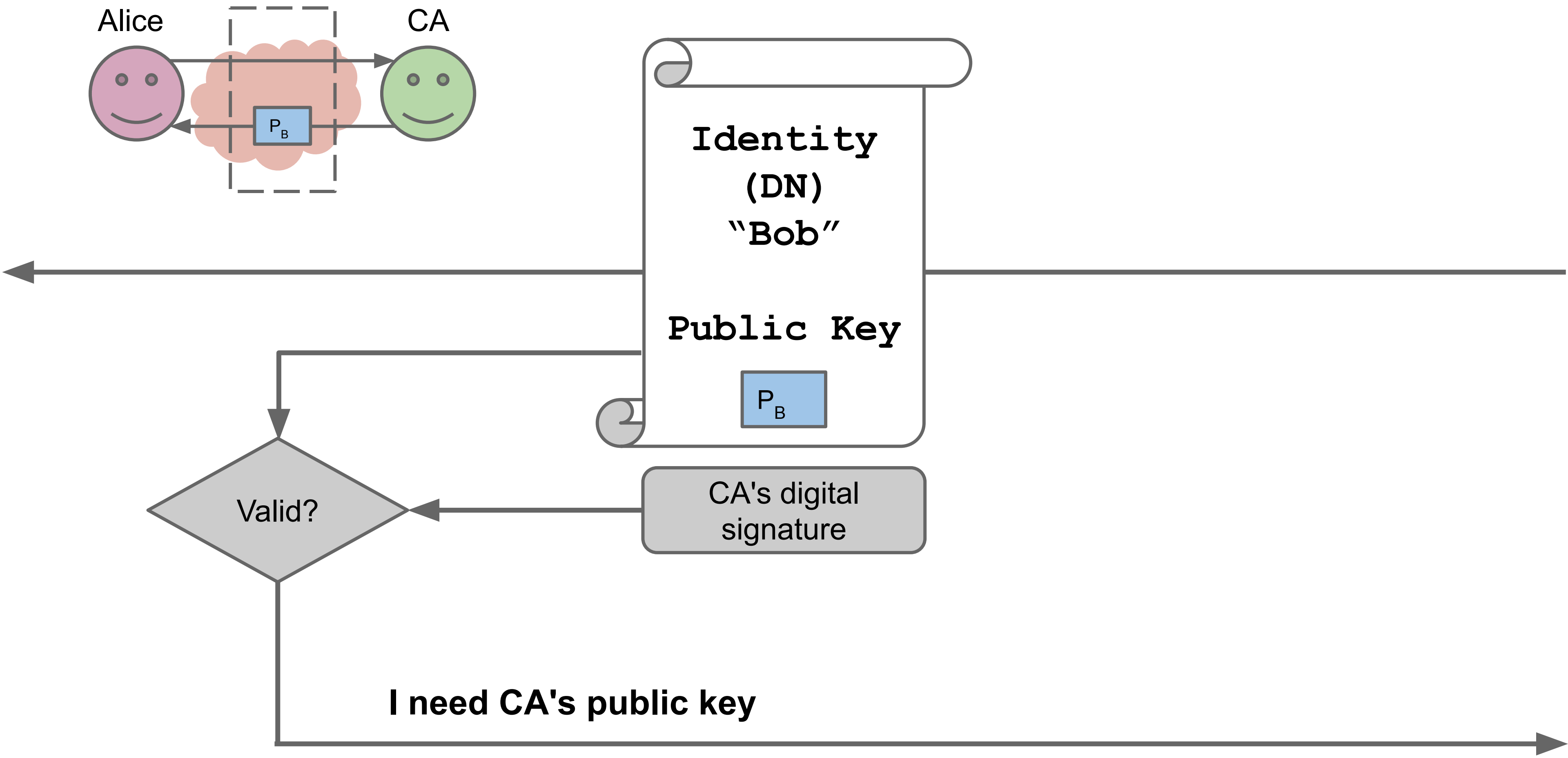
Bob's Digital Certificate (2)



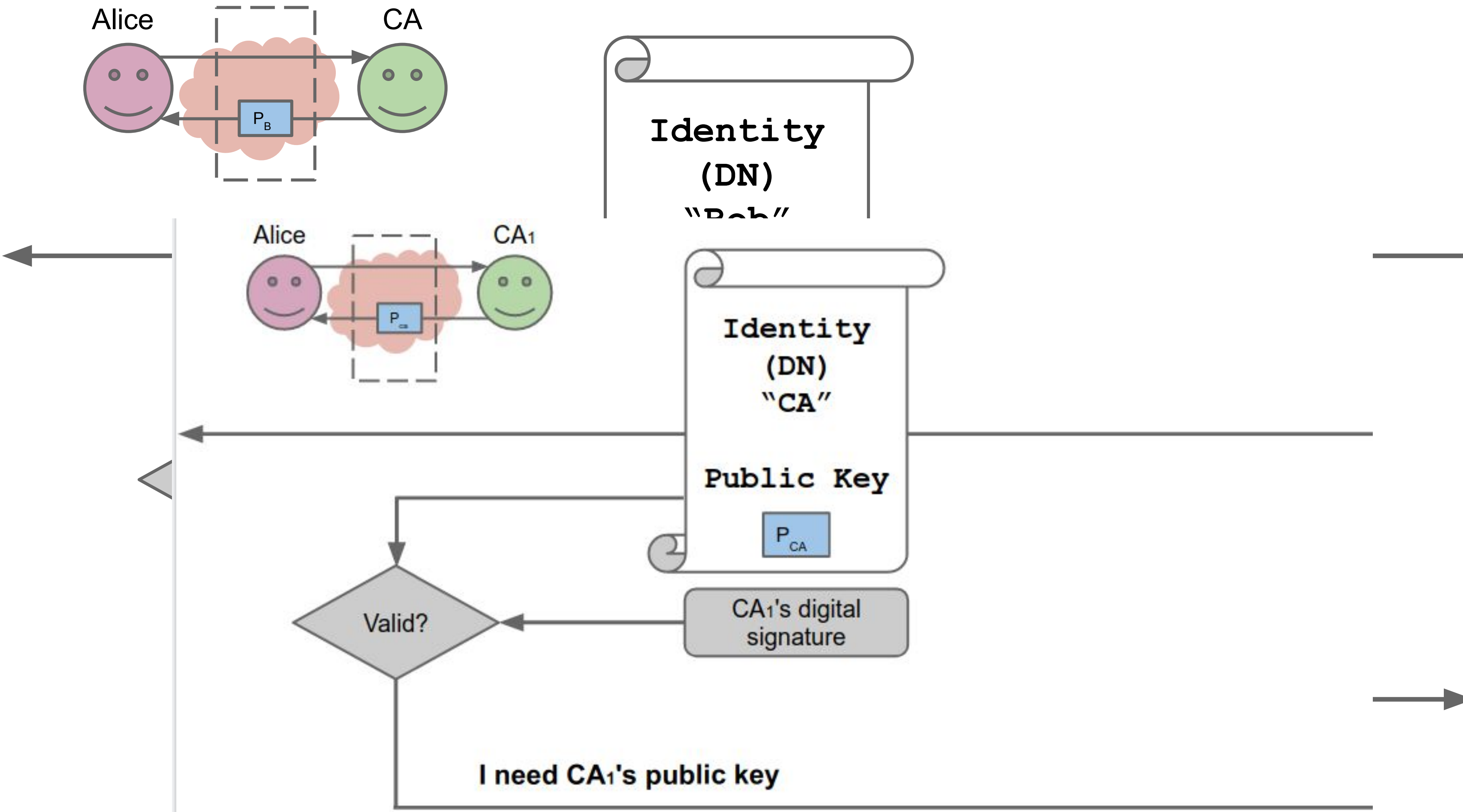
Retrieving Bob's Certificate



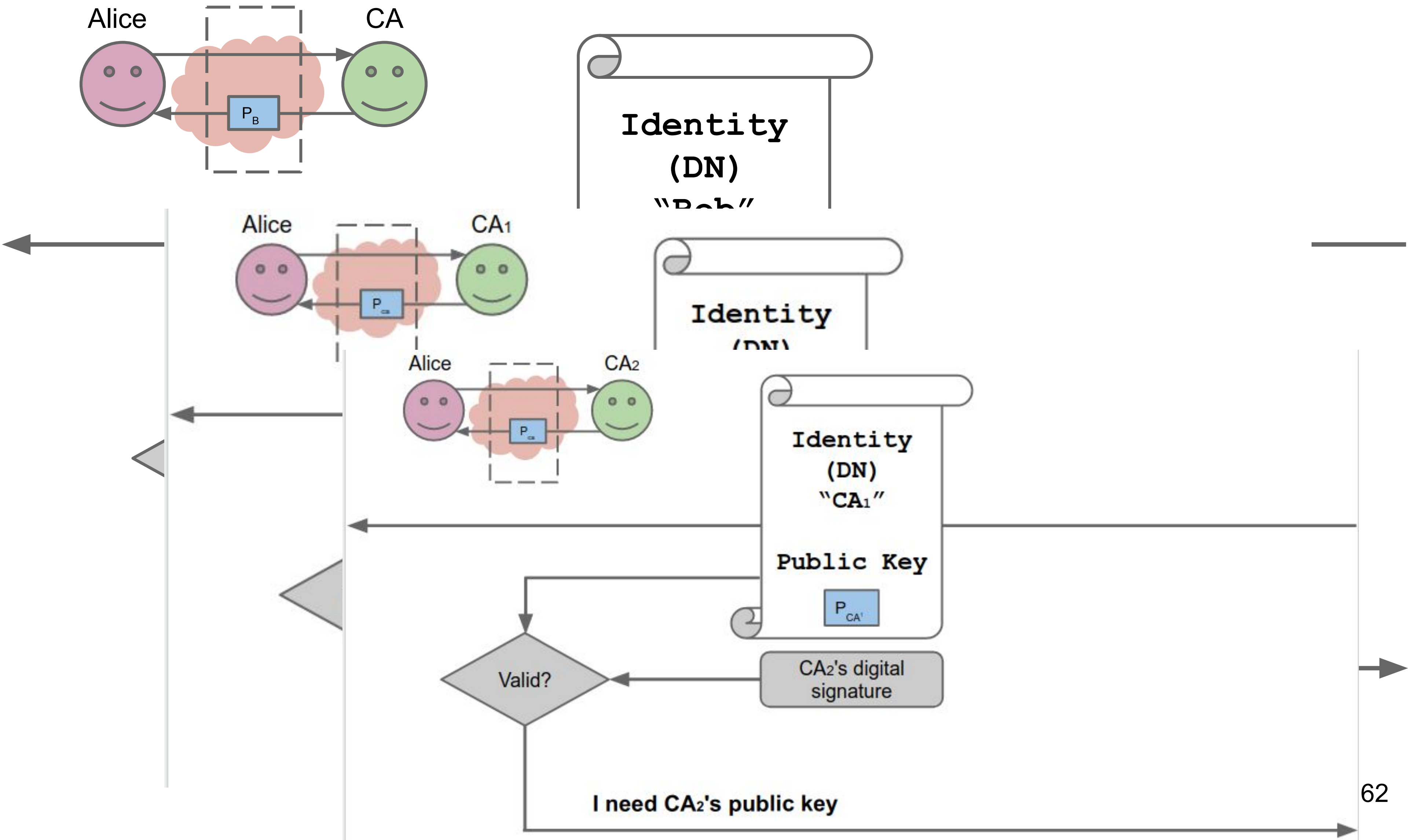
Zoom in: Is the public key valid?



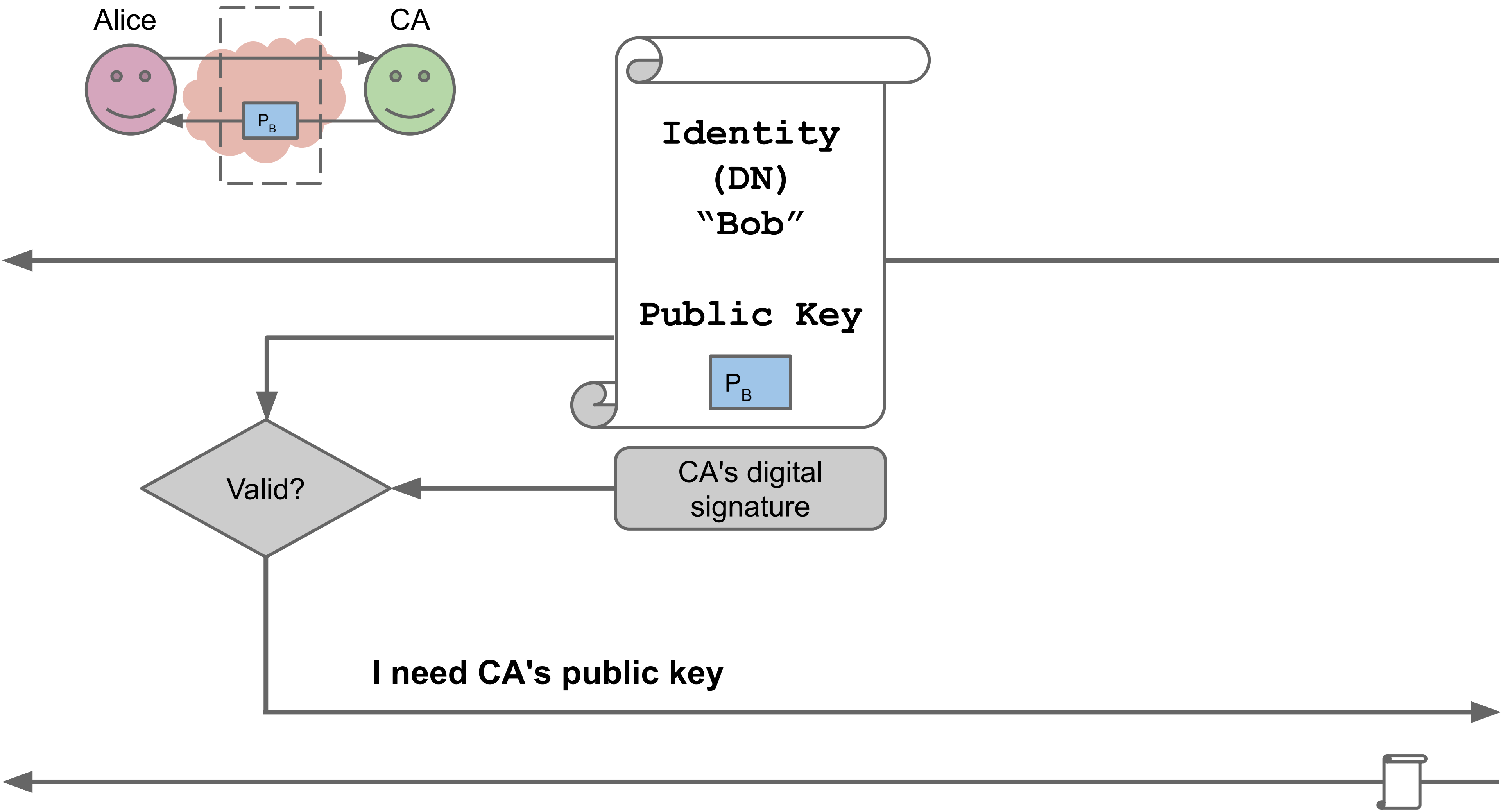
Zoom in: Is the public key valid?



Zoom in: Is the public key valid?



Zoom in: Is the public key valid?



Where is this going to end?

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!

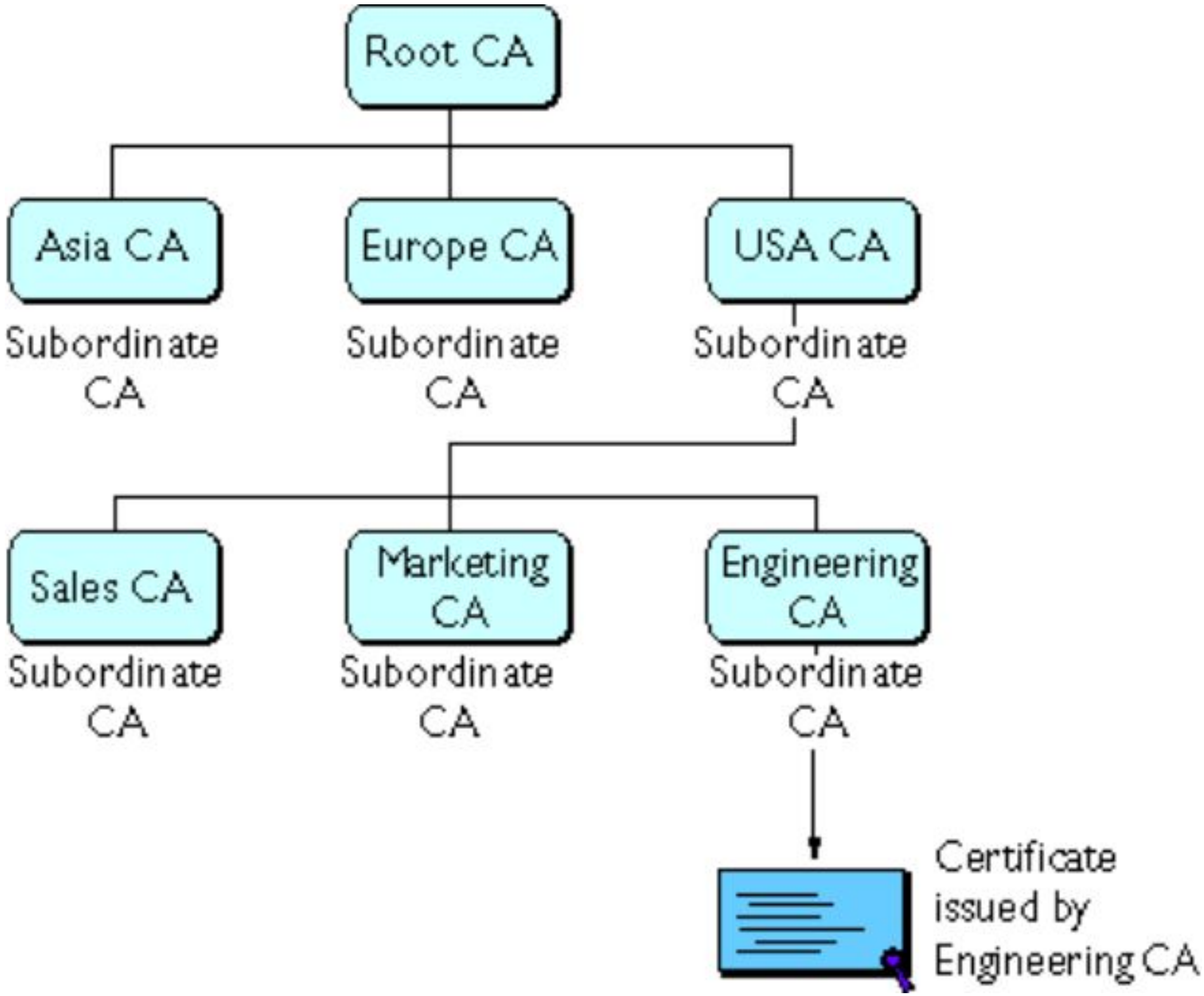
We Need a Trusted Element

Top-level CA (root CA, source CA)

Uses a self-signed certificate

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

The chain, or rather the tree



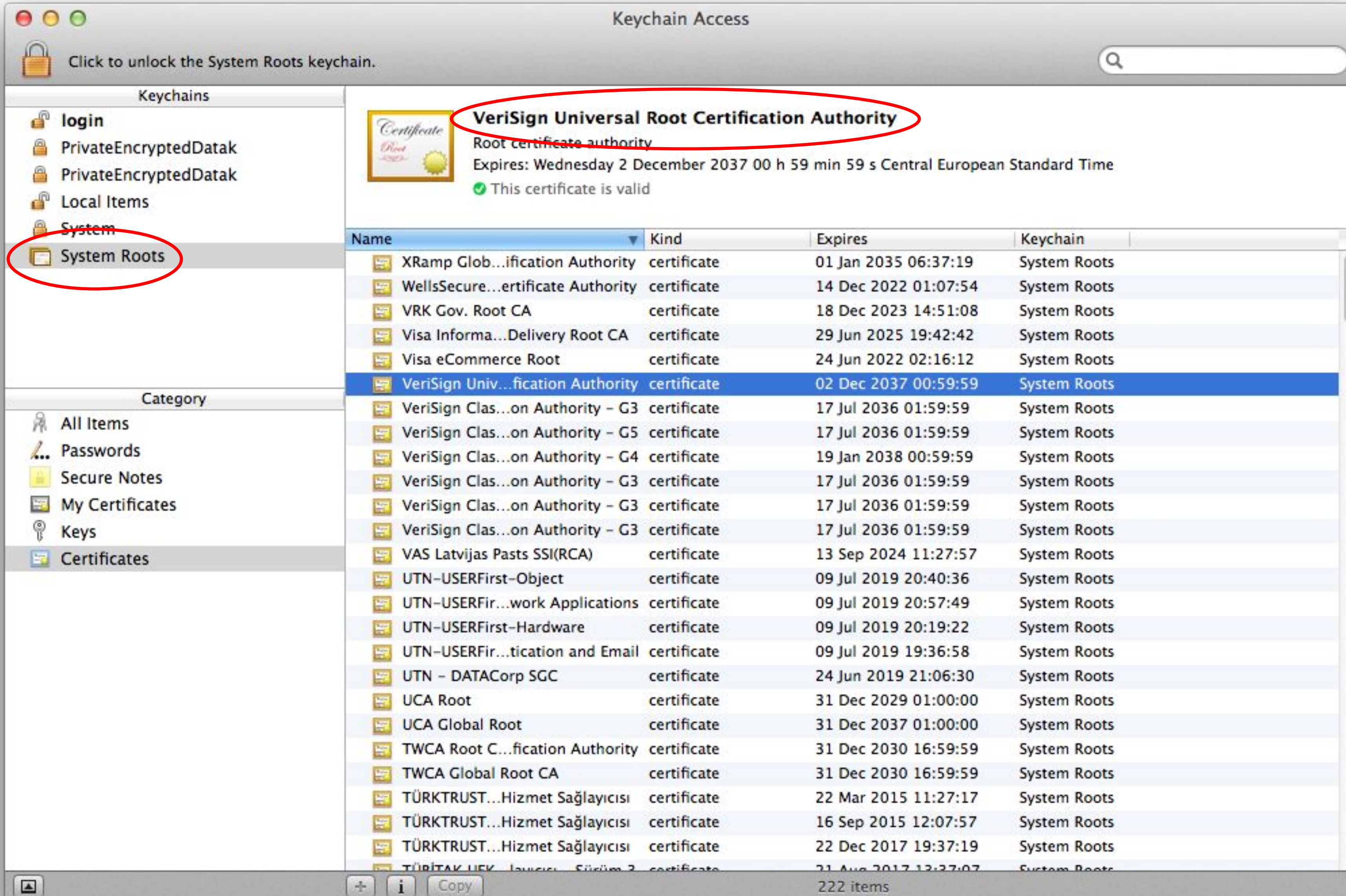
How to distribute the trusted element?

An **authority** releases it

- the state
- a regulator
- the organization management

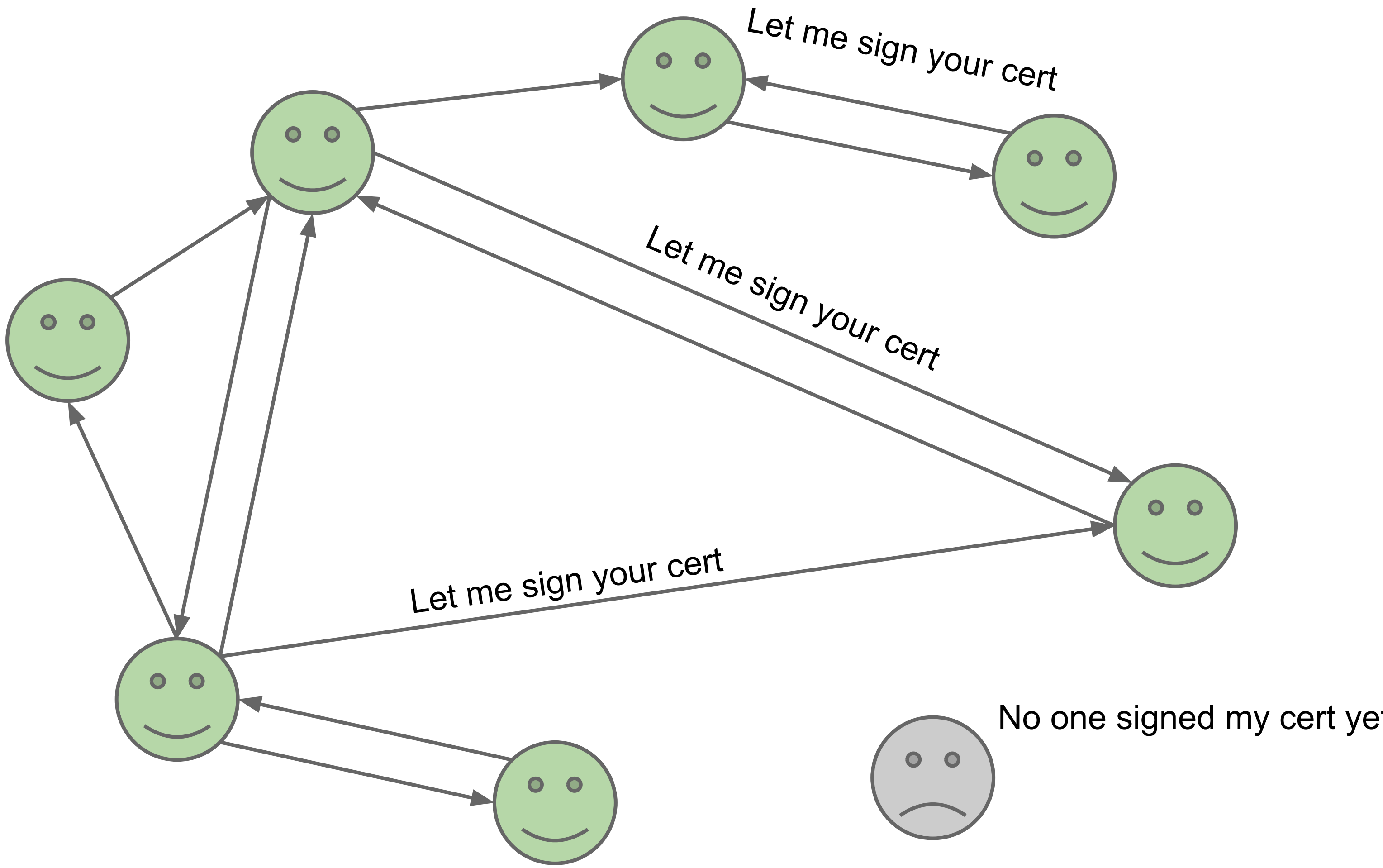
Decentralizing trust (e.g., PGP web-of-trust)

CA already (de facto standard)



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

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 omonimous 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**!

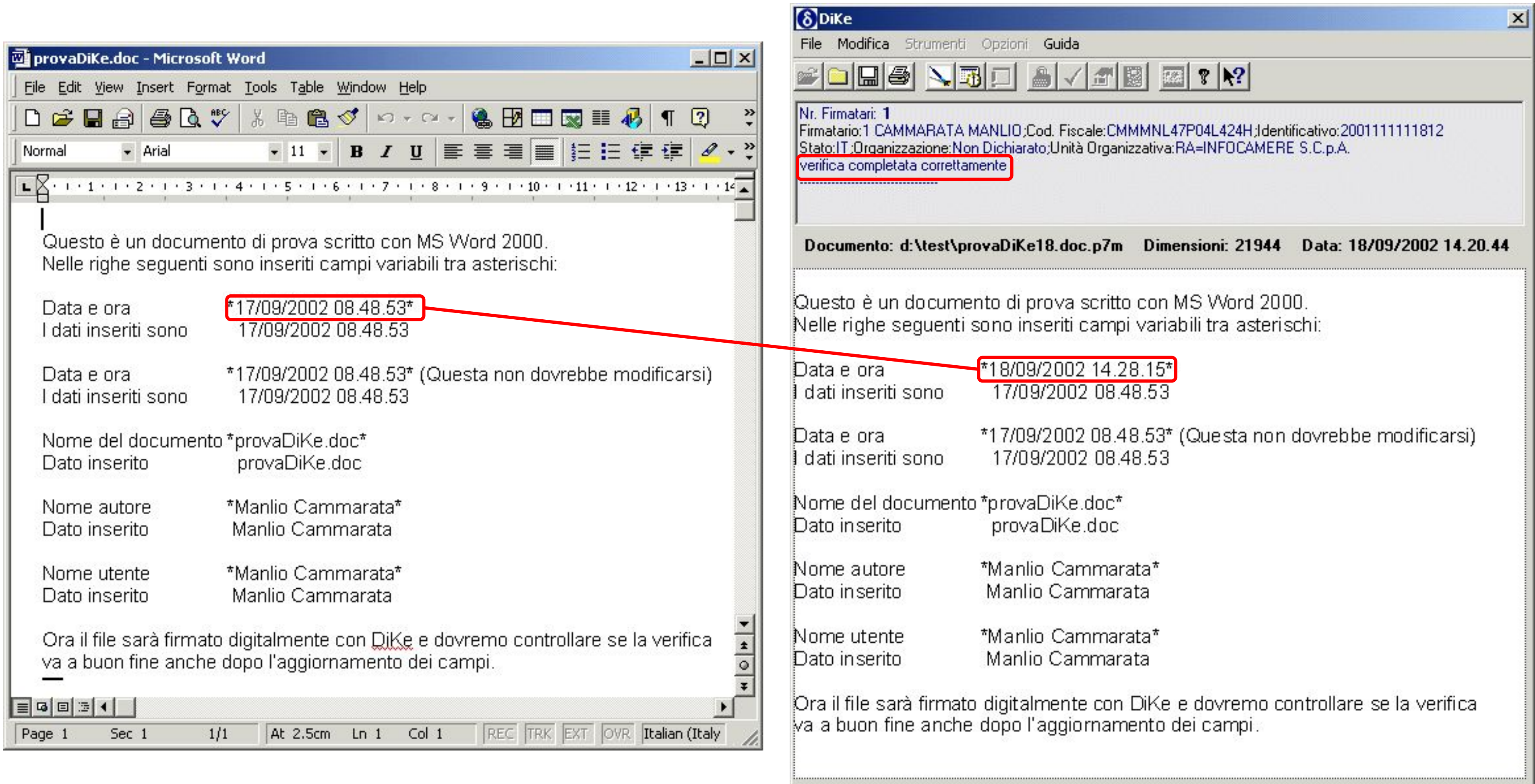
However, vulnerabilities did emerge

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
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*
Dato inserito provaDiKe.doc
Nome autore *Manlio Cammarata*
Dato inserito Manlio Cammarata
Nome utente *Manlio Cammarata*
Dato 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:CMMMNL47P04L424H;Identificativo:2001111111812
Stato:IT;Organizzazione:Non Dichiarato;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*
Dato inserito provaDiKe.doc
Nome autore *Manlio Cammarata*
Dato inserito Manlio Cammarata
Nome utente *Manlio Cammarata*
Dato 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”

Bug 2: Firma&Cifra

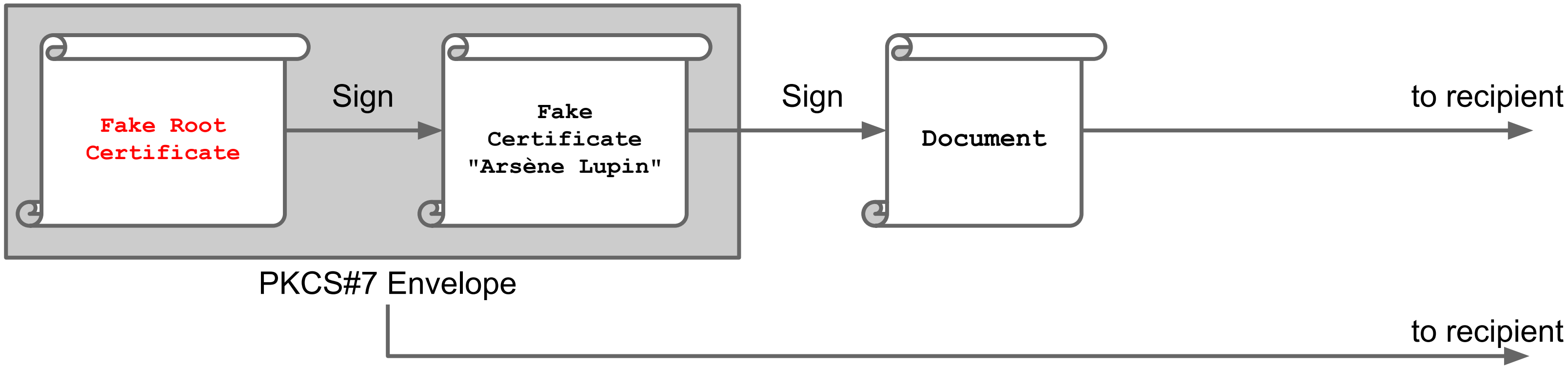
- Firma&Cifra was the digital signature application by PostECom
- Bug found by anonymous on 20/03/2003: http://www.interlex.it/docdigit/sikur159.htm
- **Result: creation** and **verification** of a **signature** with a **fake certificate**
- Also in this case: no cryptographic algorithm was broken to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

Vulnerability Description

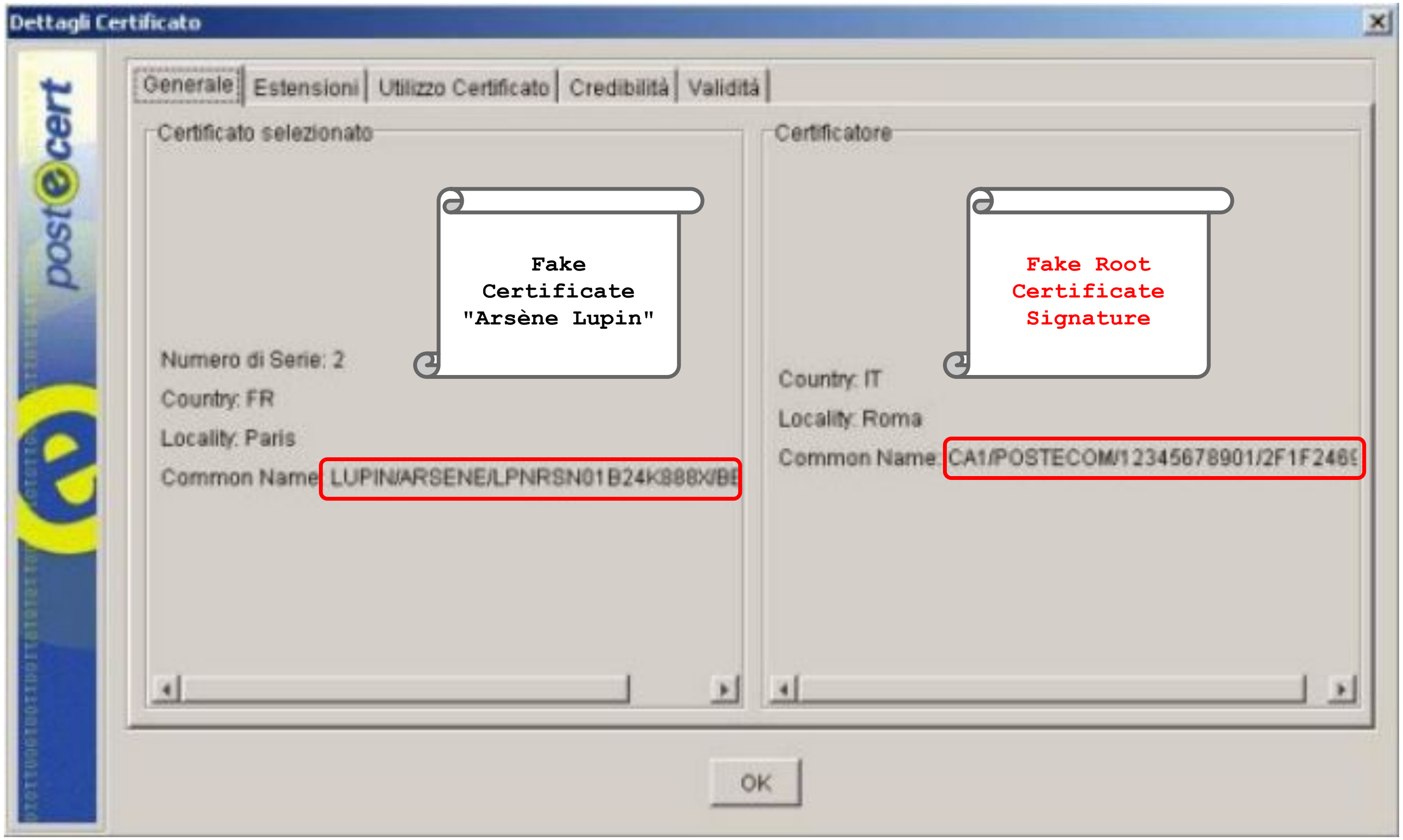
- In order to verify a signature, we need **author certificate** and the **certificate chain**
  - **Theoretically**, all available **online**
  - To allow **offline verification**, everything included with the document, in a PKCS#7 envelope
- Verification of root certificates must use preinstalled ones
  - 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

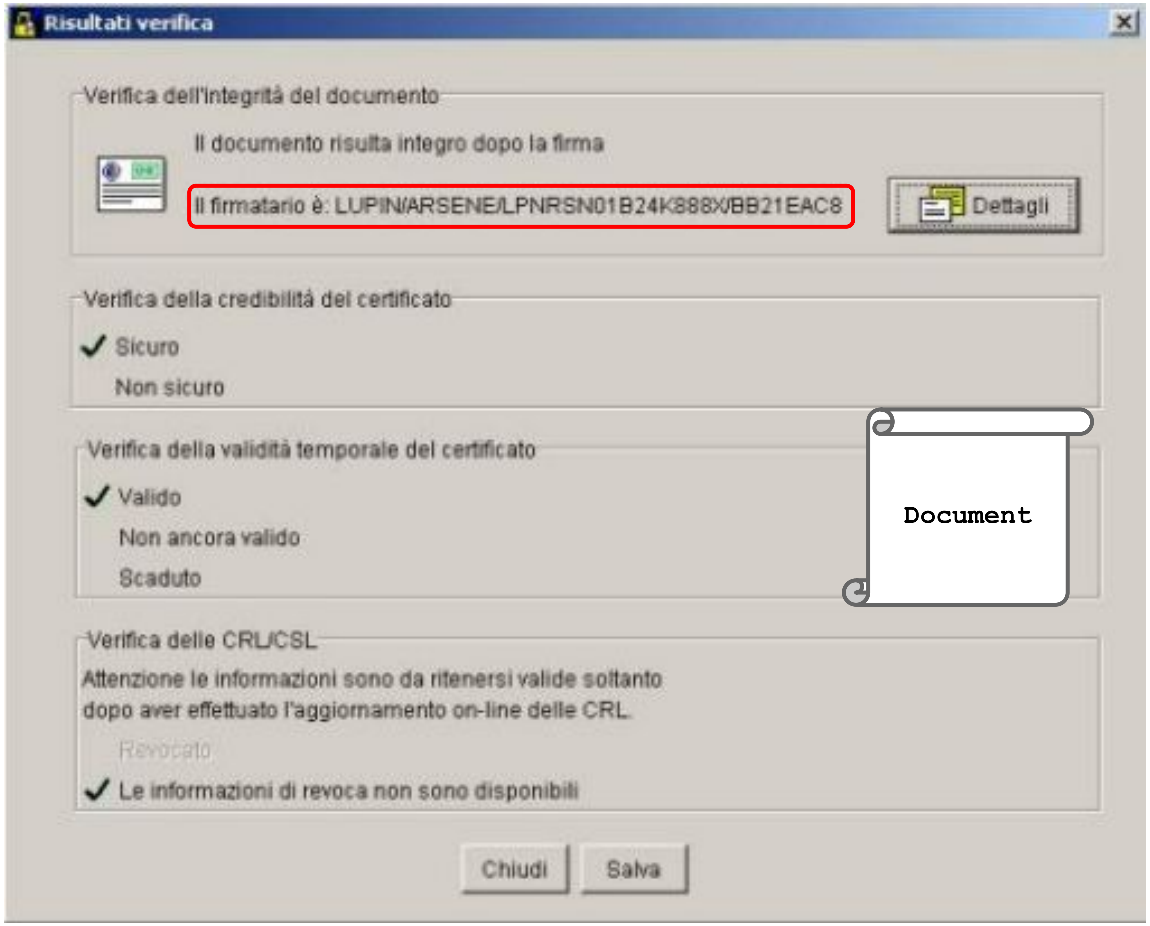
1. Generate a **fake *root* certificate** with the same name as a real one (e.g., PostECom itself)
2. Use this to generate a **fake *user* certificate** (in our example Arsène Lupin)
3. Use **Arsène Lupin's certificate** to sign theft and burglary confessions.
4. Include the fake root cert to the PKCS#7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



Les jeux sont faits: Lupin's Certficate



The Exploit: Arséne Lupin signature



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

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

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