Asymmetric Encryption

Confidentiality (plus something more)

(OR PUBLIC KEY ENCRYPTION)

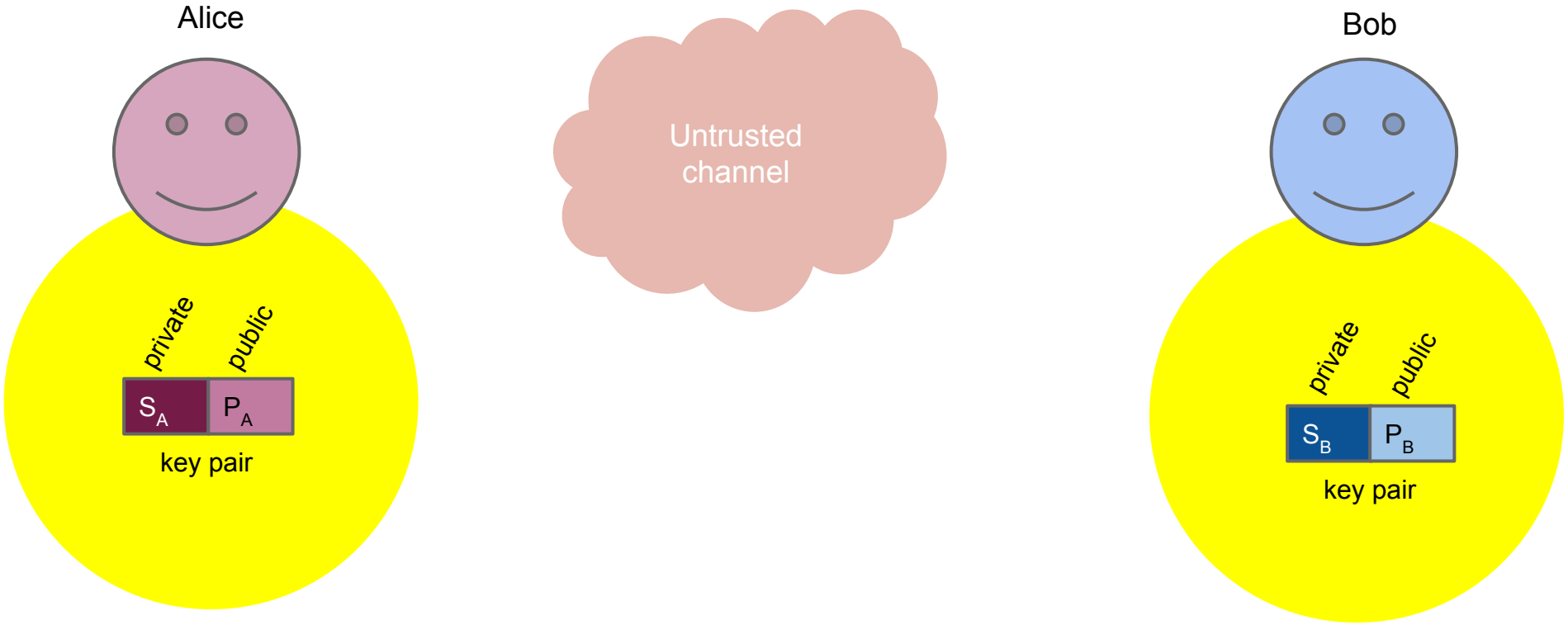
A PRIVATE KEY "FOR ME" AND A PUBLIC, SHARED KEY

Asymmetric Encryption

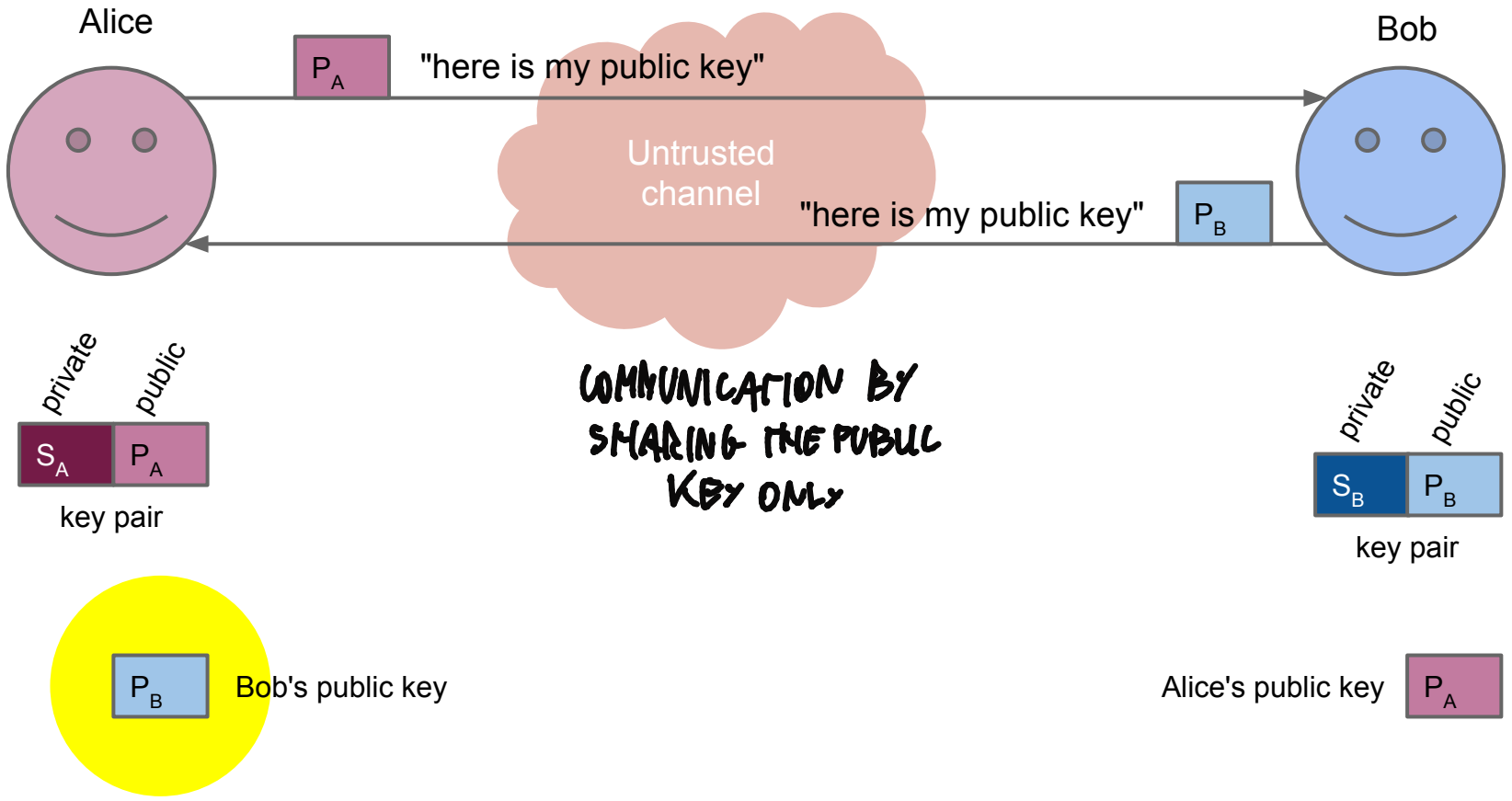
- **Concept:** a cipher that uses two keys
  - What is encrypted with **key1** can be decrypted only with **key2** (and not with key1 itself), and viceversa.
  - The keys cannot be retrieved from each other
- Introduced in 1976 (W. Diffie & M. Hellman)
- Also called *"public key cryptography"*
  - **Idea:** 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 **one-way function with a trapdoor**
  - They are usually computation-intensive

GENERALLY BASED ON HARD PROBLEMS

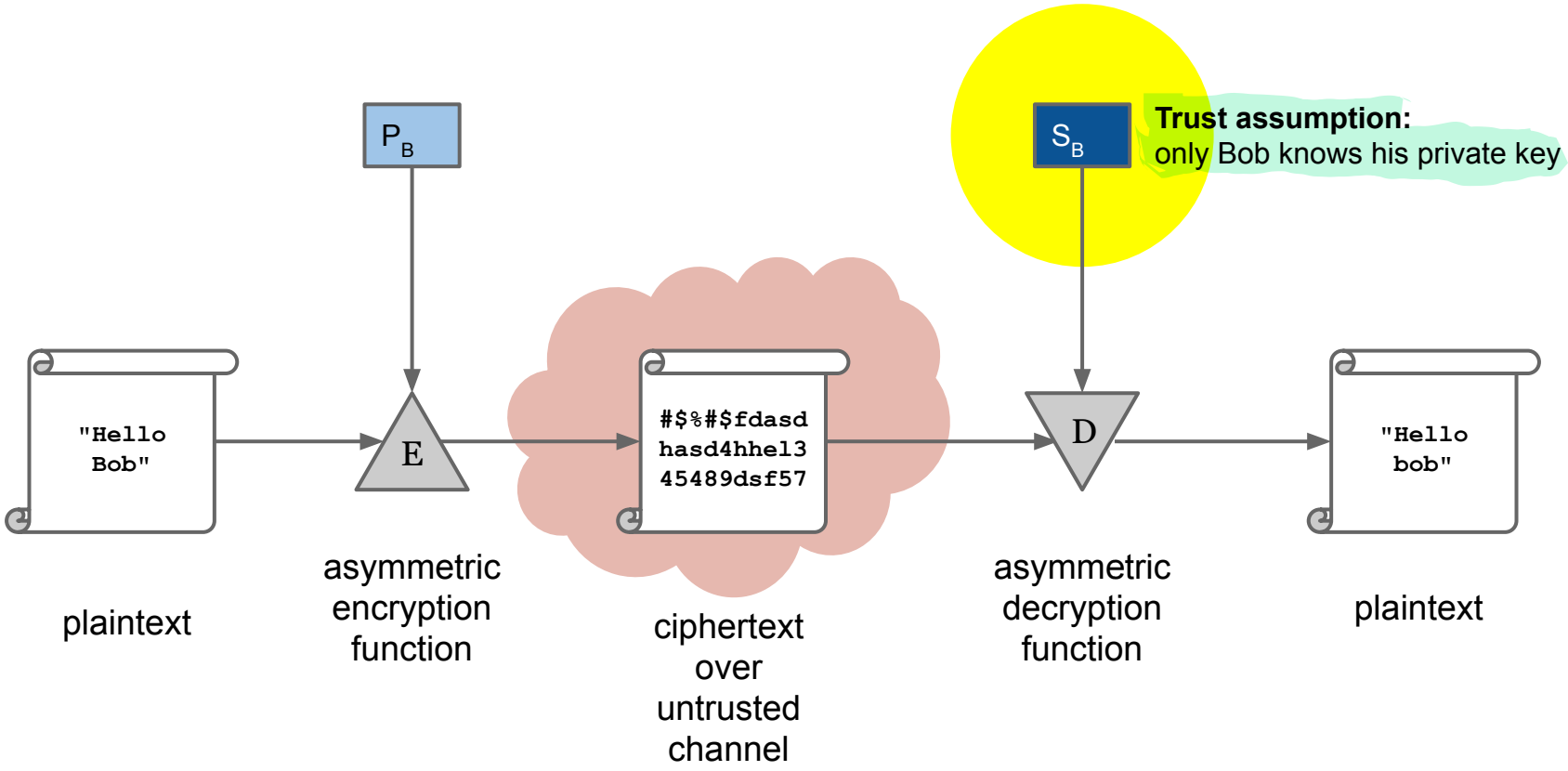
Asymmetric Encryption: Key Exchange



Asymmetric Encryption: Key Exchange



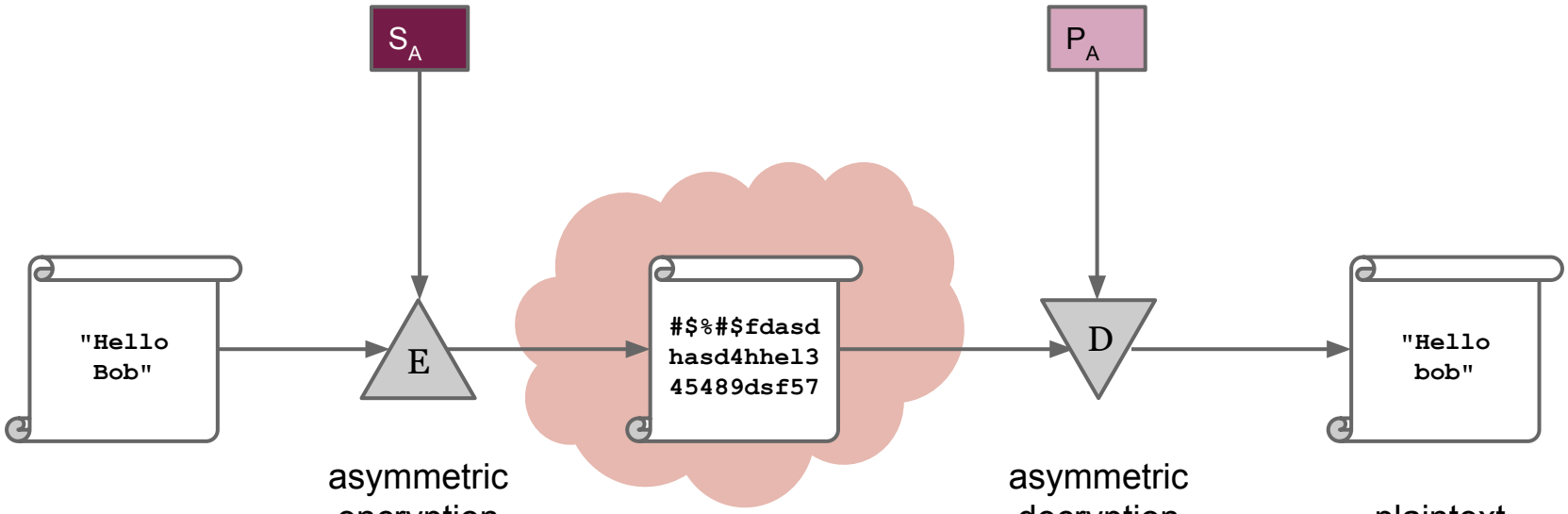
Asymmetric Encryption: Communication



Exercise: what is this instead?

Trust assumption:
only Alice knows her private key

Everybody knows Alice's public key



SOLUTION: WE KNOW WHO IS SENDING THE MESSAGE

↓

WE HAVE AUTHENTICATED MESSAGES

A cautionary note on security margins

Computational hardness

- Up to now, enumeration of the secret parameter was the best possible attack
- This is ok for modern block ciphers $\rightarrow$ best attack: $\mathcal{O}(2^{\lambda})$
- Asymmetric cryptosystems rely on hard problems for which bruteforcing the secret parameter is not the best attack $\rightarrow$ THIS PROBLEM HAS A COMPUTATIONAL COMPLEXITY $2^{\lambda/2}$ THAT IS SLOWER THAN THE EXPONENTIAL ONE 2^{λ}
  - Factoring a λ 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 λ depending on the cipher: `www.keylength.org`

AES (128 BITS) $\longrightarrow$ 2^{128}

RSA (128 BITS) $\longrightarrow$ $2^{20.30}$ $\longrightarrow$ EASIER TO BREAK $\Rightarrow$ RSA NEEDS LONGER KEYS

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/

The Diffie-Hellman key agreement

Goal

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

Attacker model

CONSIDER PASSIVE ATTACKERS ONLY

- 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 $\{0, \ldots, |\mathbf{G}| - 1\}$ ($\lambda = len(a) \approx \log_2 |\mathbf{G}|$)
- given g^a, g^b finding g^{ab} costs more than poly($\log |\mathbf{G}|$)
- Best current attack approach: find either b or a (discrete log problem)

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

Key agreement between Alice and Bob

- Alice: picks $a \xleftarrow{\$} \{0, \ldots, |\mathbf{G}| - 1\}$, sends g^a to Bob
- Bob: picks $b \xleftarrow{\$} \{0, \ldots, |\mathbf{G}| - 1\}$, sends g^b to Alice
- Alice: gets g^b from Bob and computes $(g^b)^a$
- Bob: gets g^a from Bob 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 $\mathcal{O}(2^{\frac{\lambda}{2}})$

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

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
- Bob X_B

$X_A = 3$
$X_B = 1$

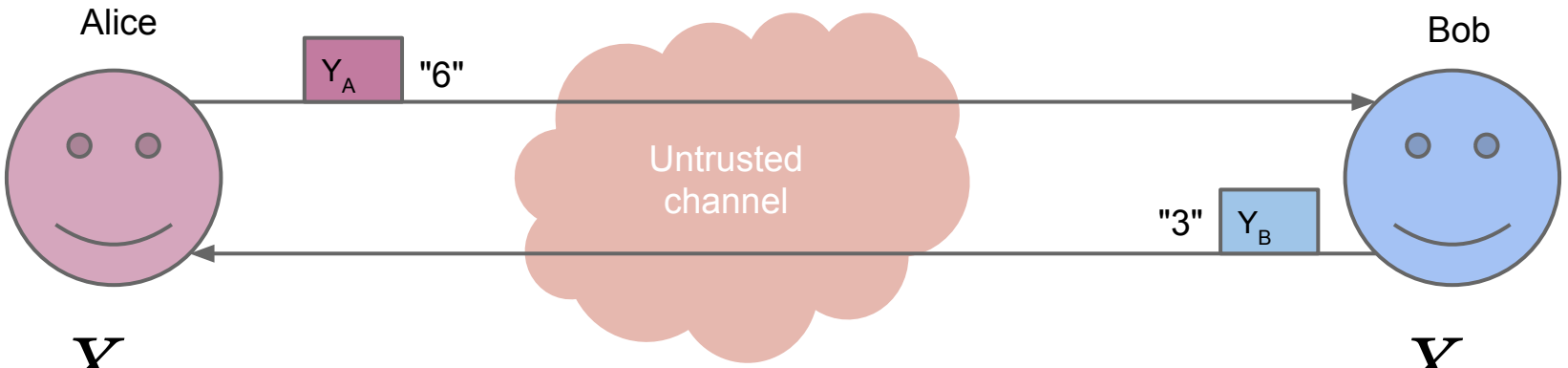
Public key (disclosed to everyone):

They obtain their public key as

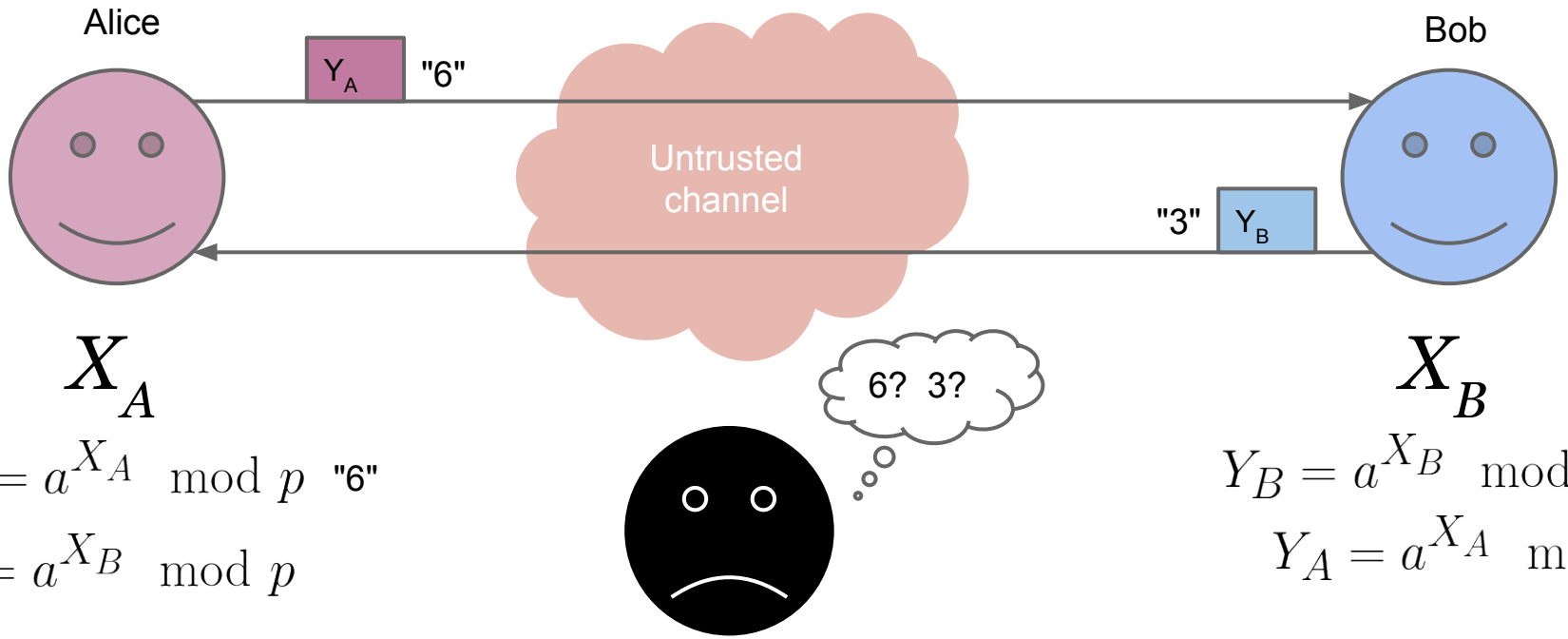
- Alice $Y_A = a^{X_A} \mod p$
- Bob $Y_B = a^{X_B} \mod p$

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

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

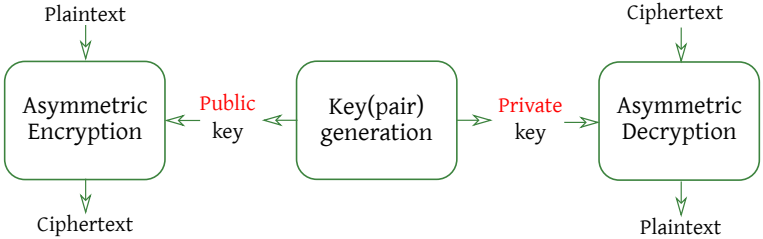
- Since $Y_A^{X_B} \mod p = (a^{X_A} \mod p)^{X_B} \mod p = (a^{X_A})^{X_B} \mod p = (a^{X_B})^{X_A} \mod p = (a^{X_B} \mod p)^{X_A} \mod p = Y_B^{X_A} \mod p$
- Alice $K_A = Y_B^{X_A} \mod p = 3^3 \mod 7 = 6$
- Bob $K_B = Y_A^{X_B} \mod p = 6^1 \mod 7 = 6$

$$K_B = K_A = K$$

Anybody else can listen, but cannot compute K

- Because they miss the private key.

Public Key Encryption



Components

- *Different* keys are employed in encryption/decryption
- It is computationally hard to: 2 HARD PROBLEMS TO SOLVE
  - Decrypt a ciphertext without the *private* key
  - Compute the *private* key given only the *public* key → PREFERRED PROBLEM ATTACKERS AIM TO BREAK

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

USING THE SAME KEY FOR PUBLIC AND PRIVATE MESSAGES

↓

PROVED PSEUDO-RANDOM GENERATOR

↳ BASED ON FACTORIZATION OF LARGE INTEGERS (COMPLEX MATH PROBLEM)

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

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 Encapsulation

Assumption

- A public channel between Alice and Bob is available (INTERNET)
- For the moment, the attacker model is "eavesdrop only" (PASSIVE ATTACKER)

Sharing a secret without agreement

- Alice: generates a keypair (k_{pri}, k_{pub}), sends to Bob
- Bob: gets $s \xleftarrow{\$} \{0,1\}^{\lambda}$, encrypts it with k_{pub}, sends ctx to Alice
- Alice: decrypts ctx with k_{pri} (HER KEY), 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

DIFFERENT ENDPOINTS S/INCRONOUS PROCESS ← KEY AGREEMENT VS KEY ENCAPSULATION

Efficiency considerations

Can't I skip a step?

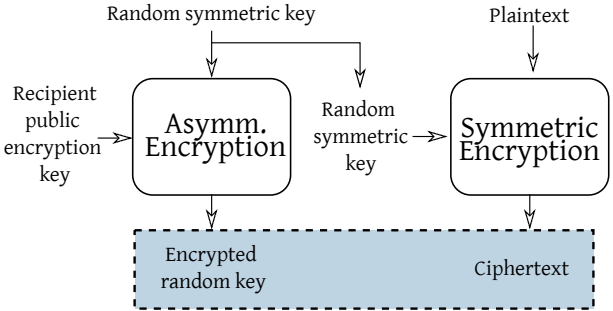
- Employing an asymmetric cryptosystem Bob encrypts a text for Alice without the need of sharing a secret beforehand
- In principle, Bob and Alice could employ *only* an asymmetric cryptosystem to communicate

Efficiency considerations

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 HOW MODERN SYSTEM WORKS

- 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

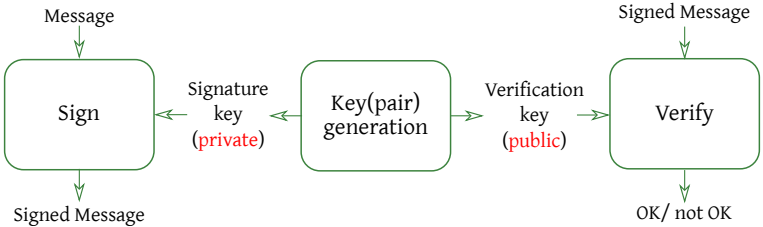
- To build a secure hybrid encryption scheme we need to be sure that the public key the sender uses is the one of the recipient
- We'd like to be able to verify the authenticity of a piece of data without a pre-shared secret → CREATE A "DIGITAL EQUIVALENT" SIGNATURE TO PROVIDE THE ENDPOINT

SOLUTION

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



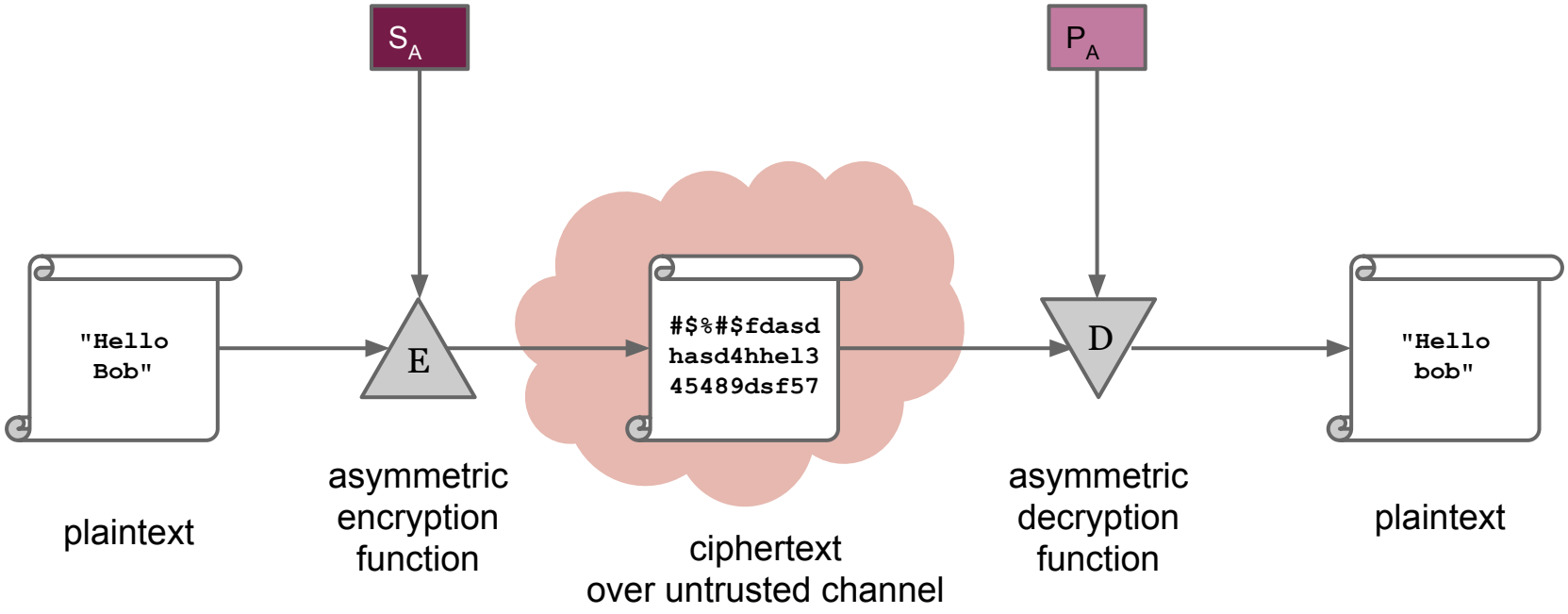
Computationally hard problems

- 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

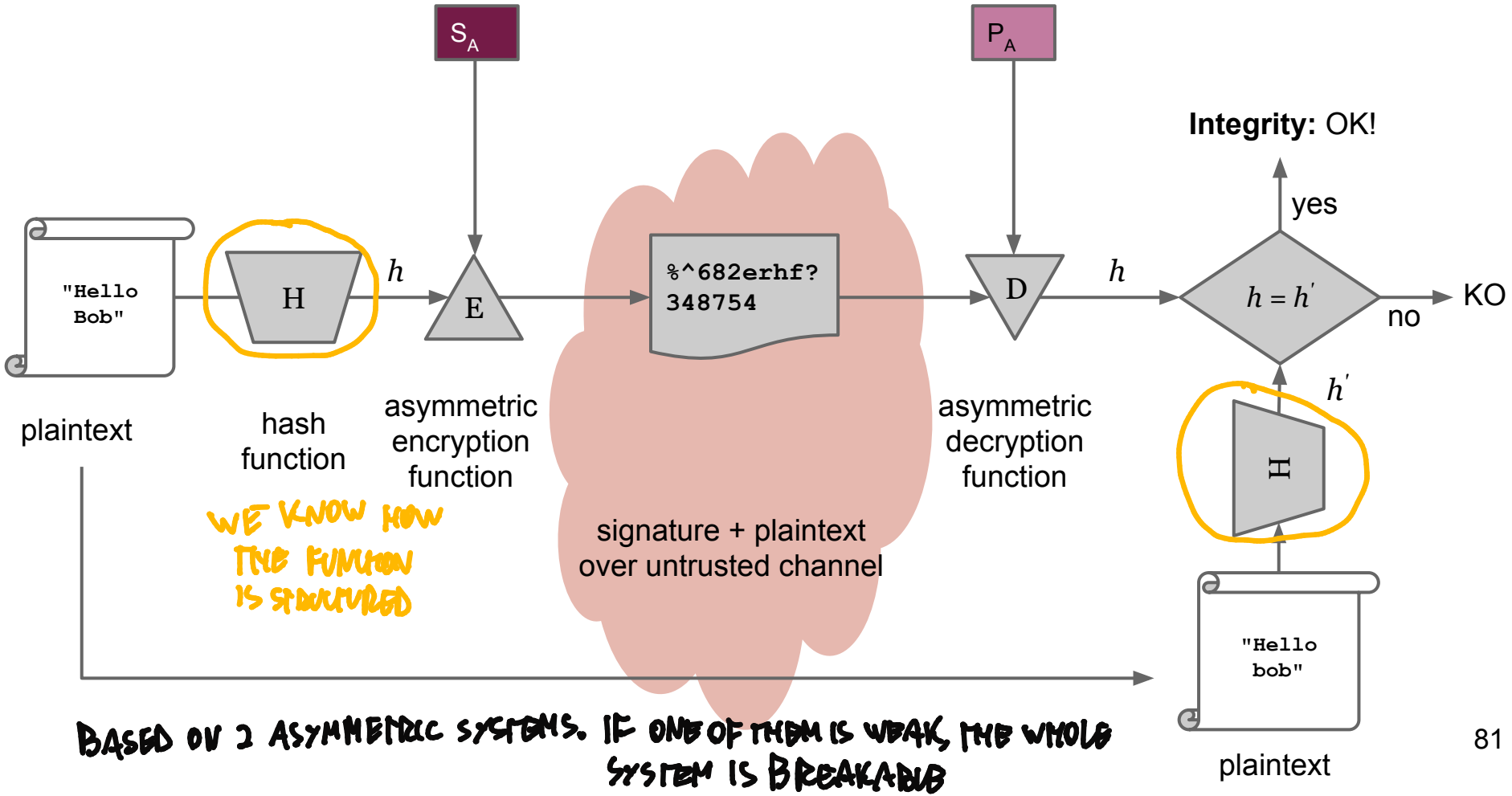
Everybody knows Alice's public key



Digital signature: Authentication and Integrity

Trust assumption:
only Alice knows her private key

Everybody knows Alice's public key



WE KNOW HOW THE FUNCTION IS STRUCTURED

BASED ON 2 ASYMMETRIC SYSTEMS. IF ONE OF THEM IS WEAK, THE WHOLE SYSTEM IS BREAKABLE

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

Horror story 1

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

The public key binding problem

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

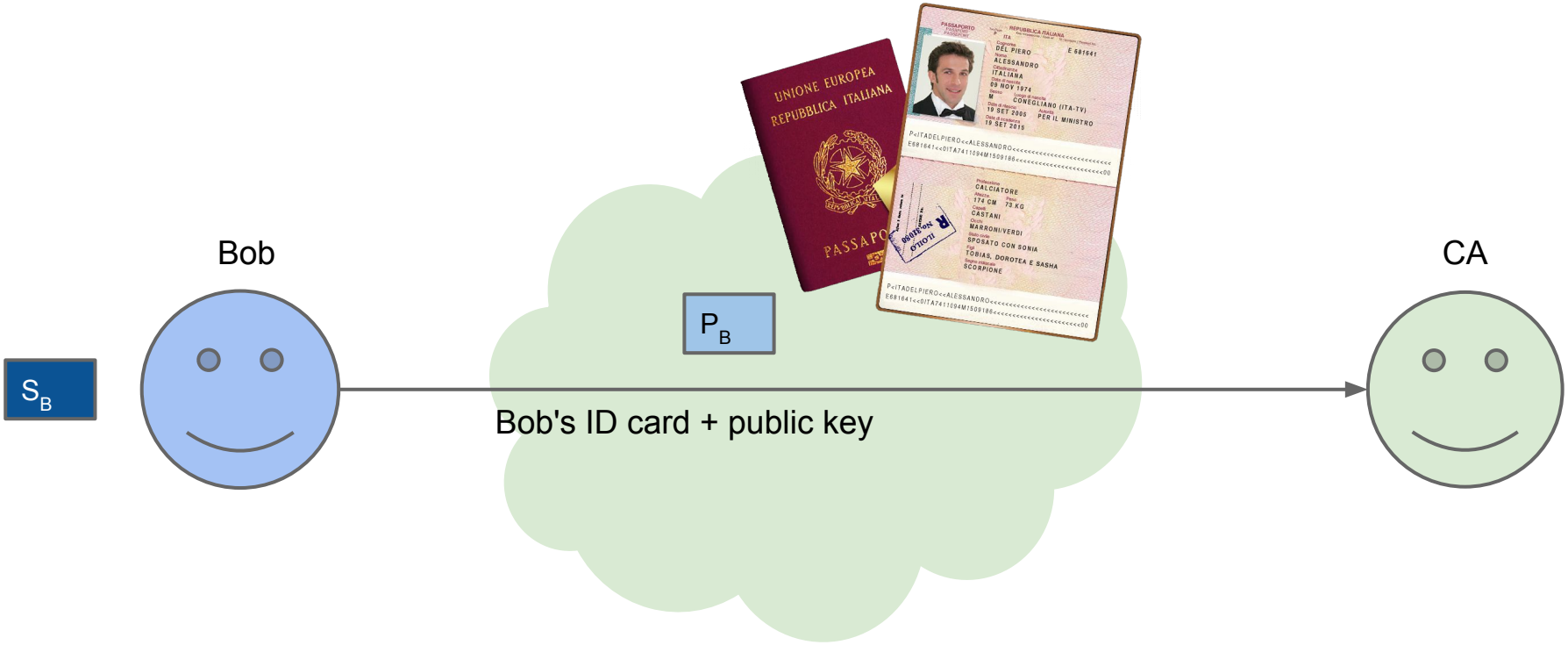
- A PKI uses a trusted third party called a **certification authority** (CA)
- The CA **digitally signs** files called **digital certificates**, which bind an identity to a public key
  - Identity = “Distinguished Name (DN)”
  - As defined in the X.509 standard (most used one)
- Now we can recognize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

Digital certificates

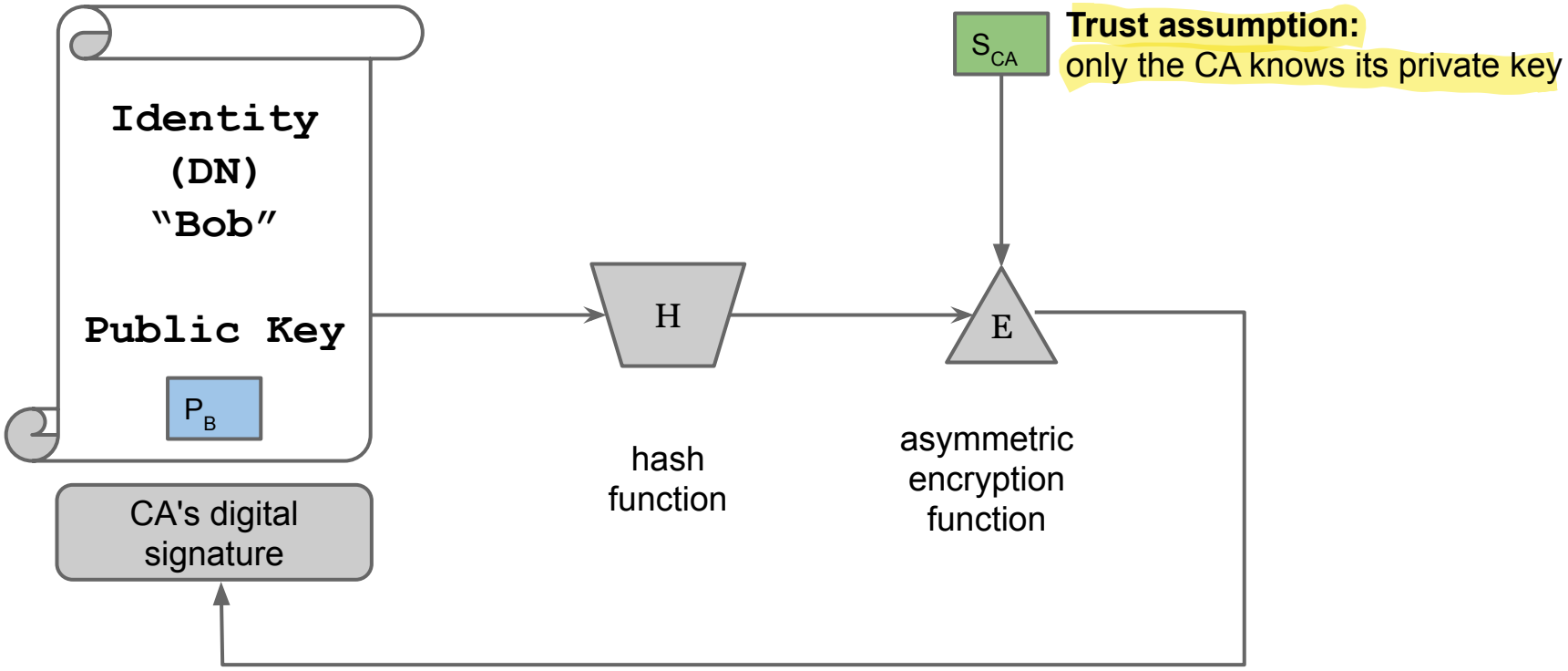
- They bind a public key to a given identity, which is:
  - for humans: an ASCII string
  - for machines: either the CNAME or IP address
- They specify the intended use for the public key contained
  - Avoids ambiguities when a key format is ok for both an encryption and a signature algorithm
- They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

IT CAN BE DEFINED AS A "DIGITAL DOCUMENT"

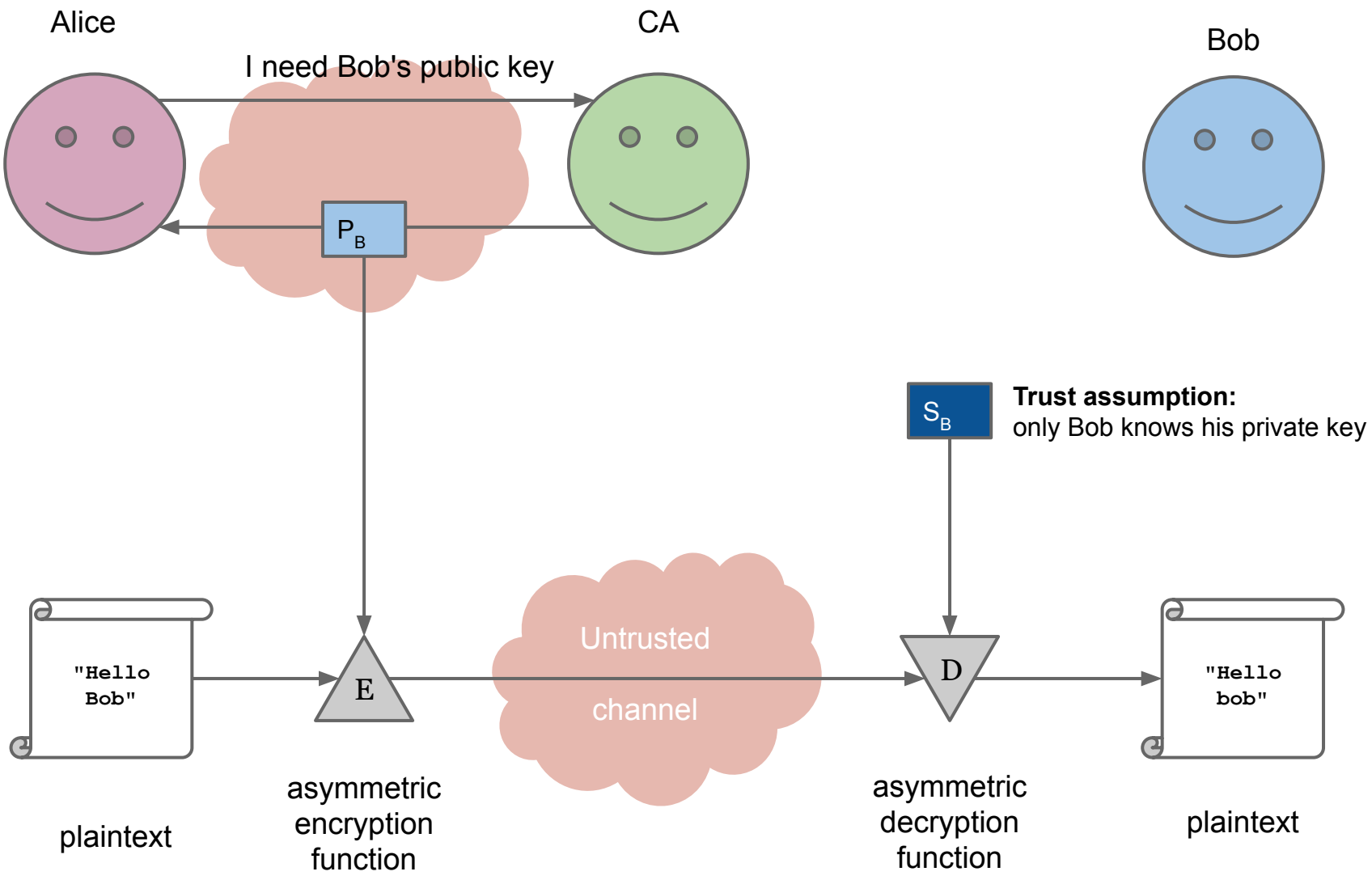
Bob's Digital Certificate (1)



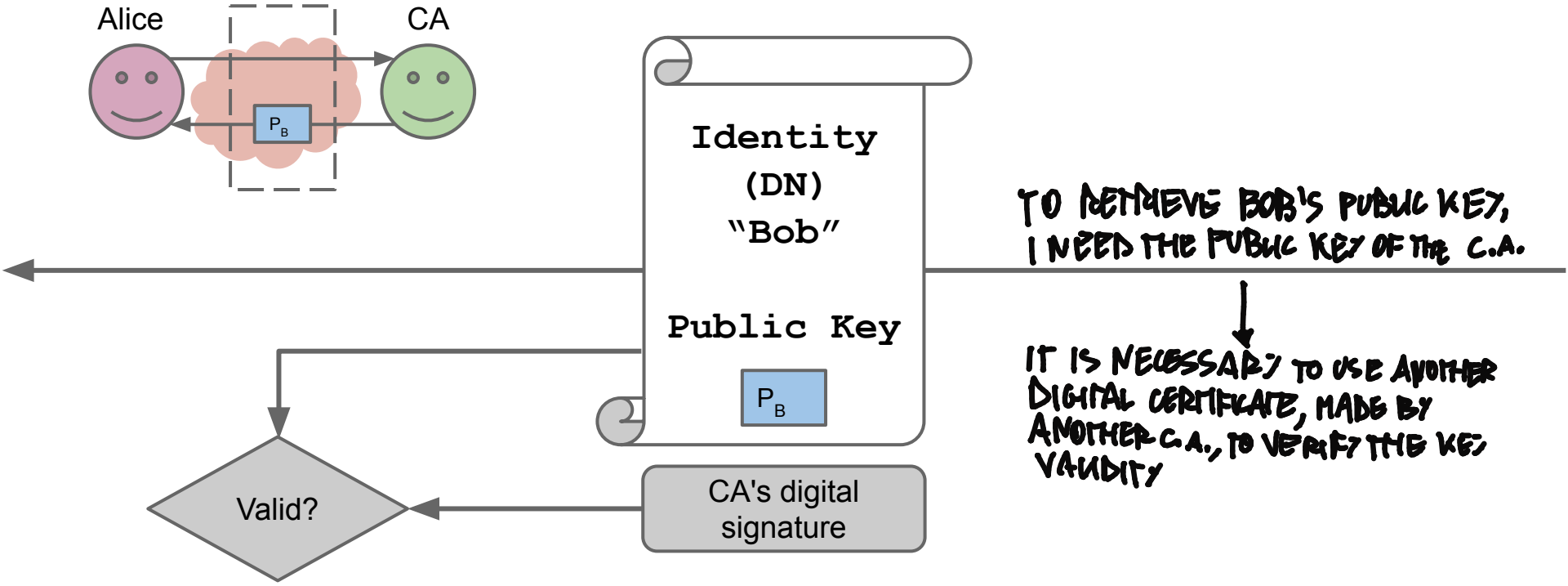
Bob's Digital Certificate (2)



Retrieving Bob's Certificate



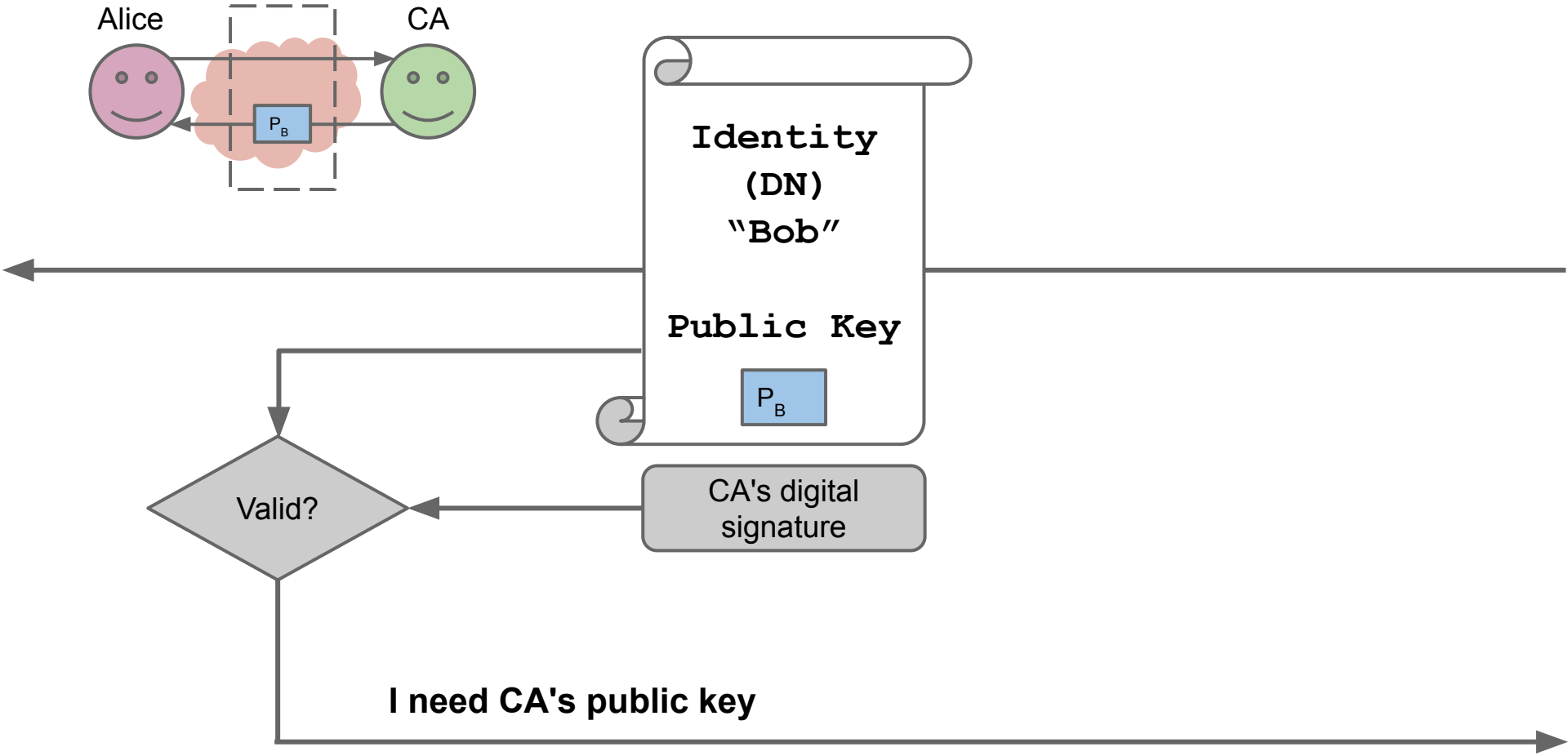
Zoom in: Is the public key valid?



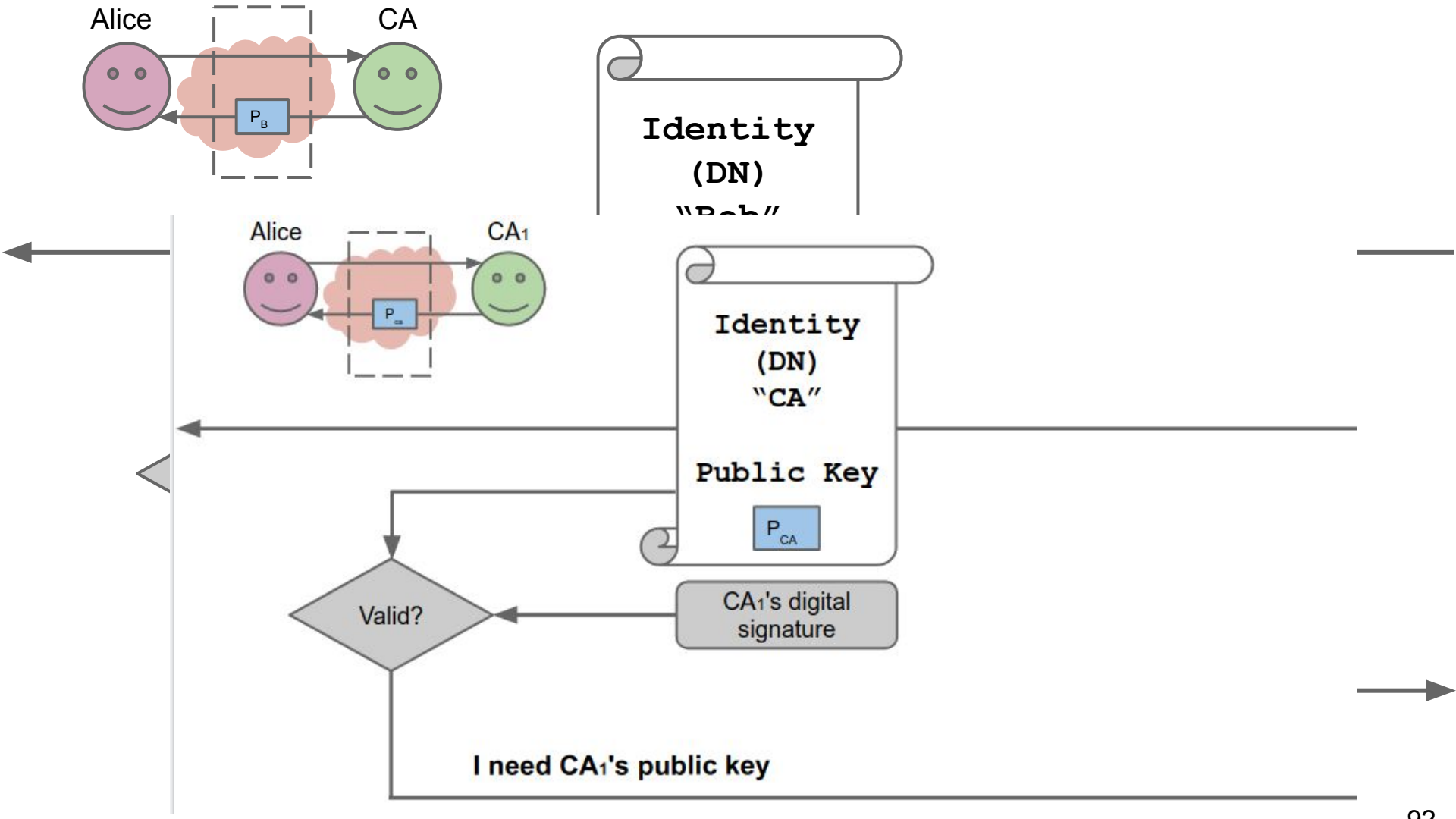
PKI

- 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

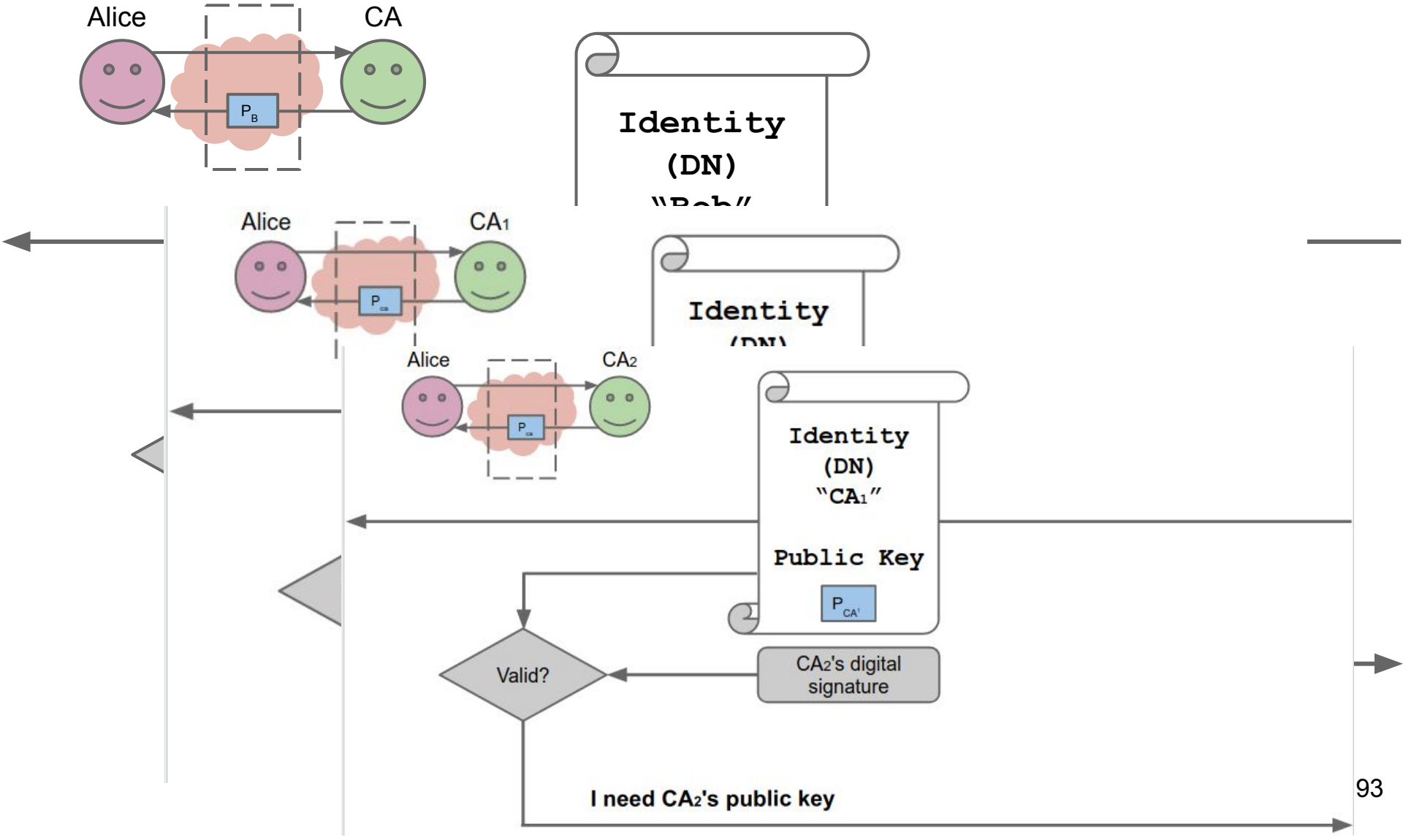
Zoom in: Is the public key valid?



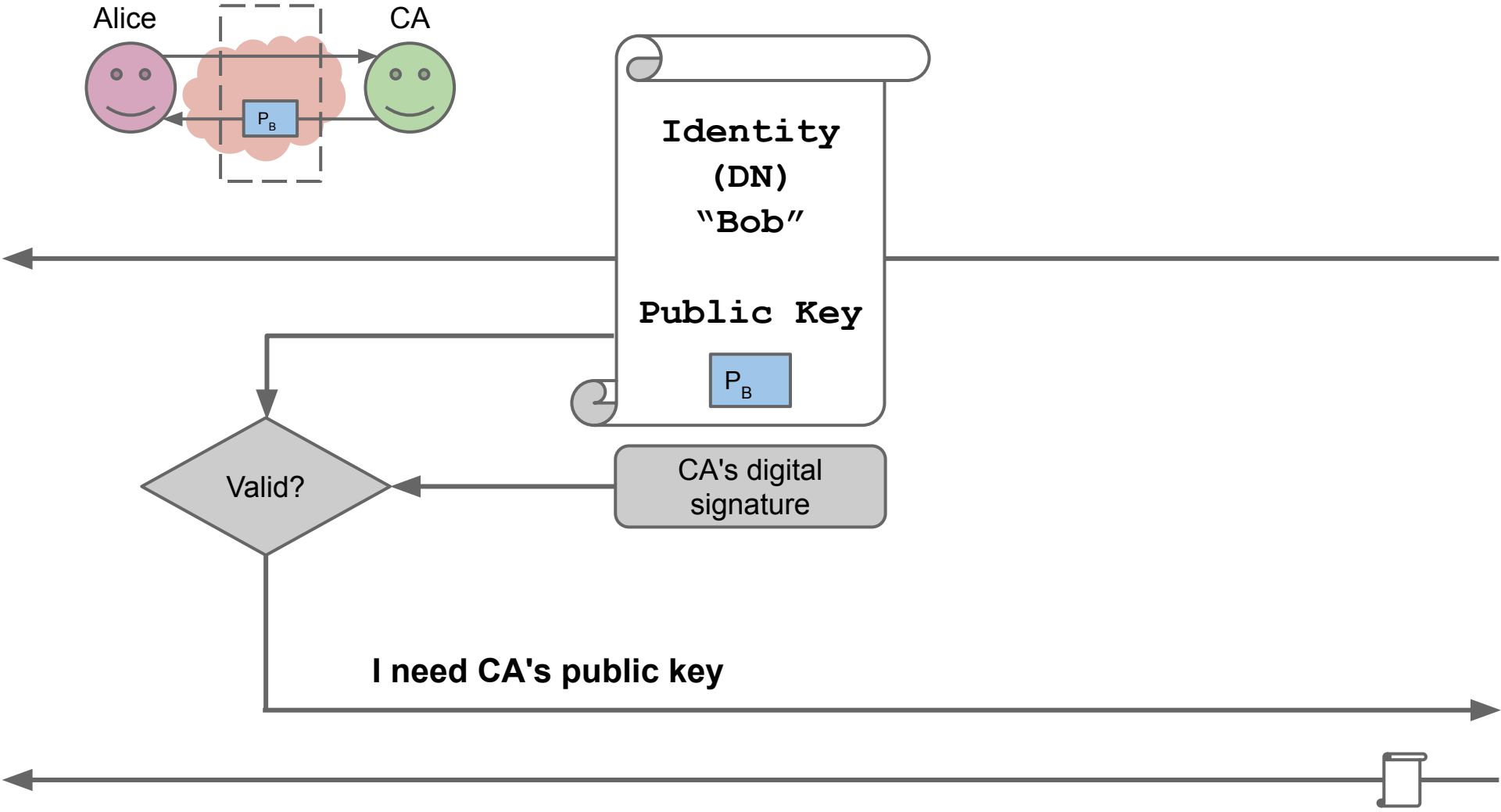
Zoom in: Is the public key valid?



Zoom in: Is the public key valid?



Zoom in: Is the public key valid?



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.

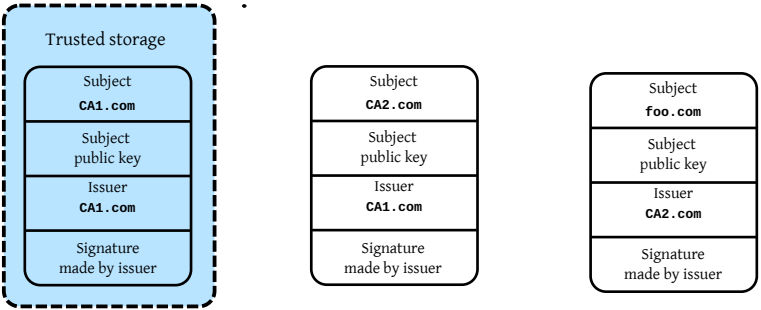
Someone else needs to sign **that** certificate

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

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



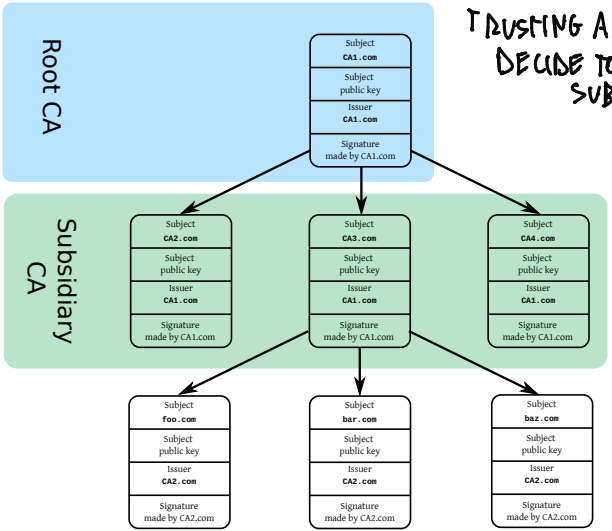
We Need a Trusted Element: Root of trust

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



TRUSTING A ROOT C.A., WE DECIDE TO TRUST ALL ITS SUBSIDIARIES

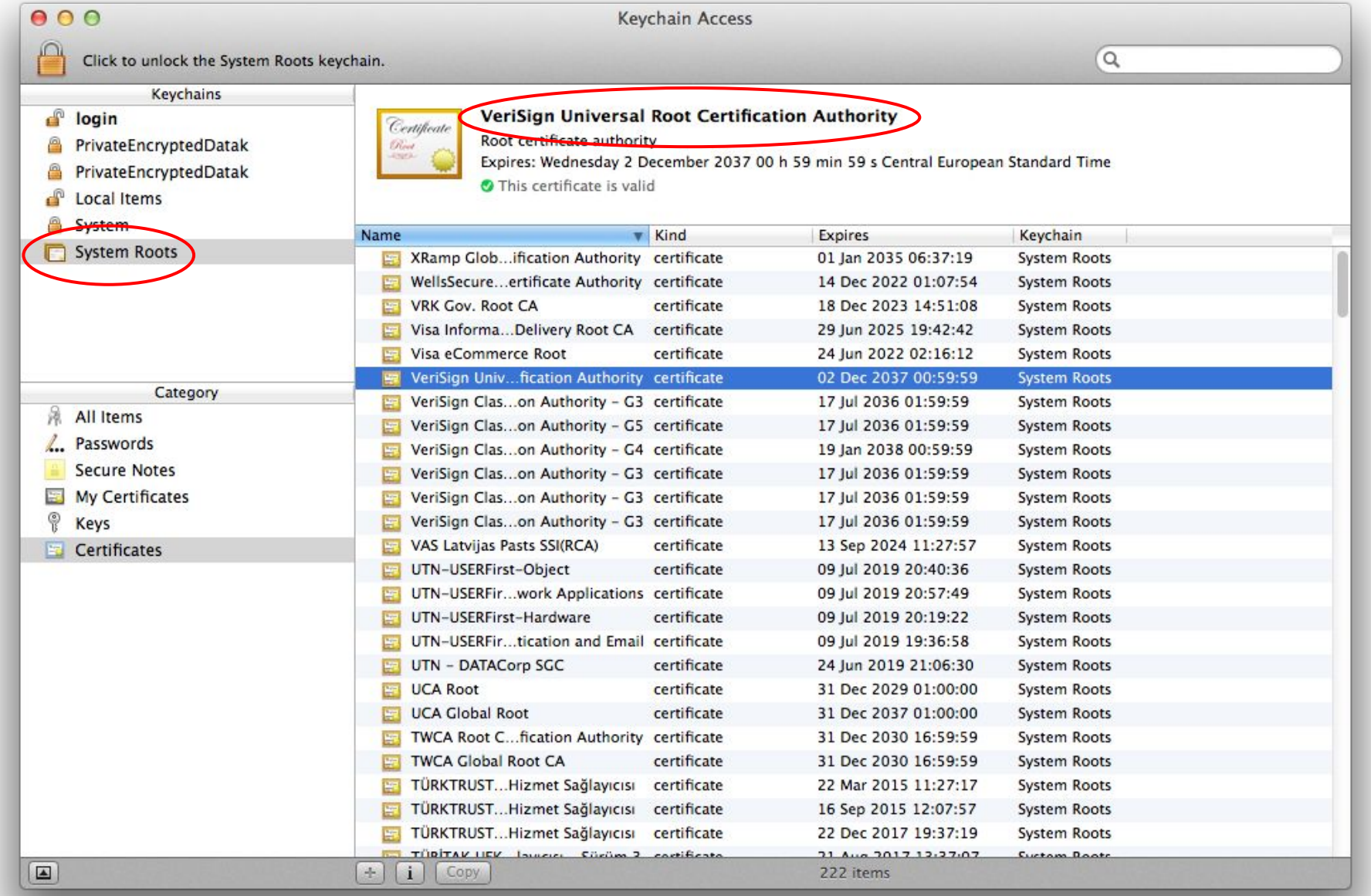
CAREFUL NOT NO TO TRUST UNTRUSTED SERVICES

How to distribute the trusted element?

An **authority** releases it

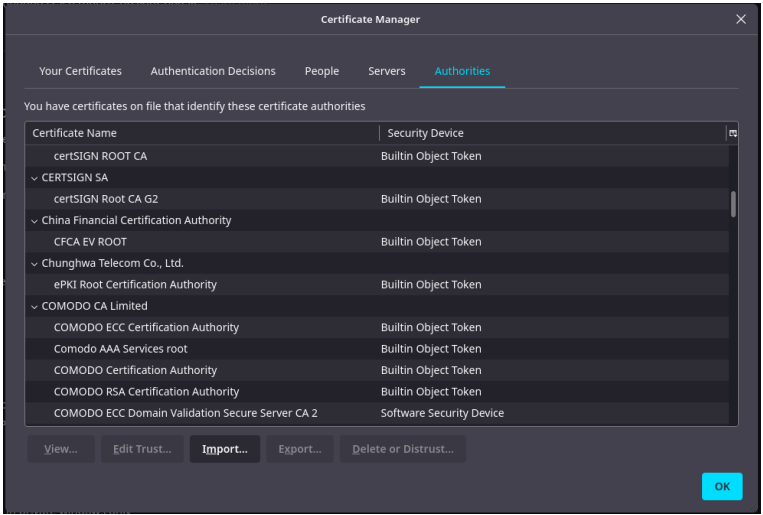
- the state
- a regulator
- the organization management

CA already (de facto standard)



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?

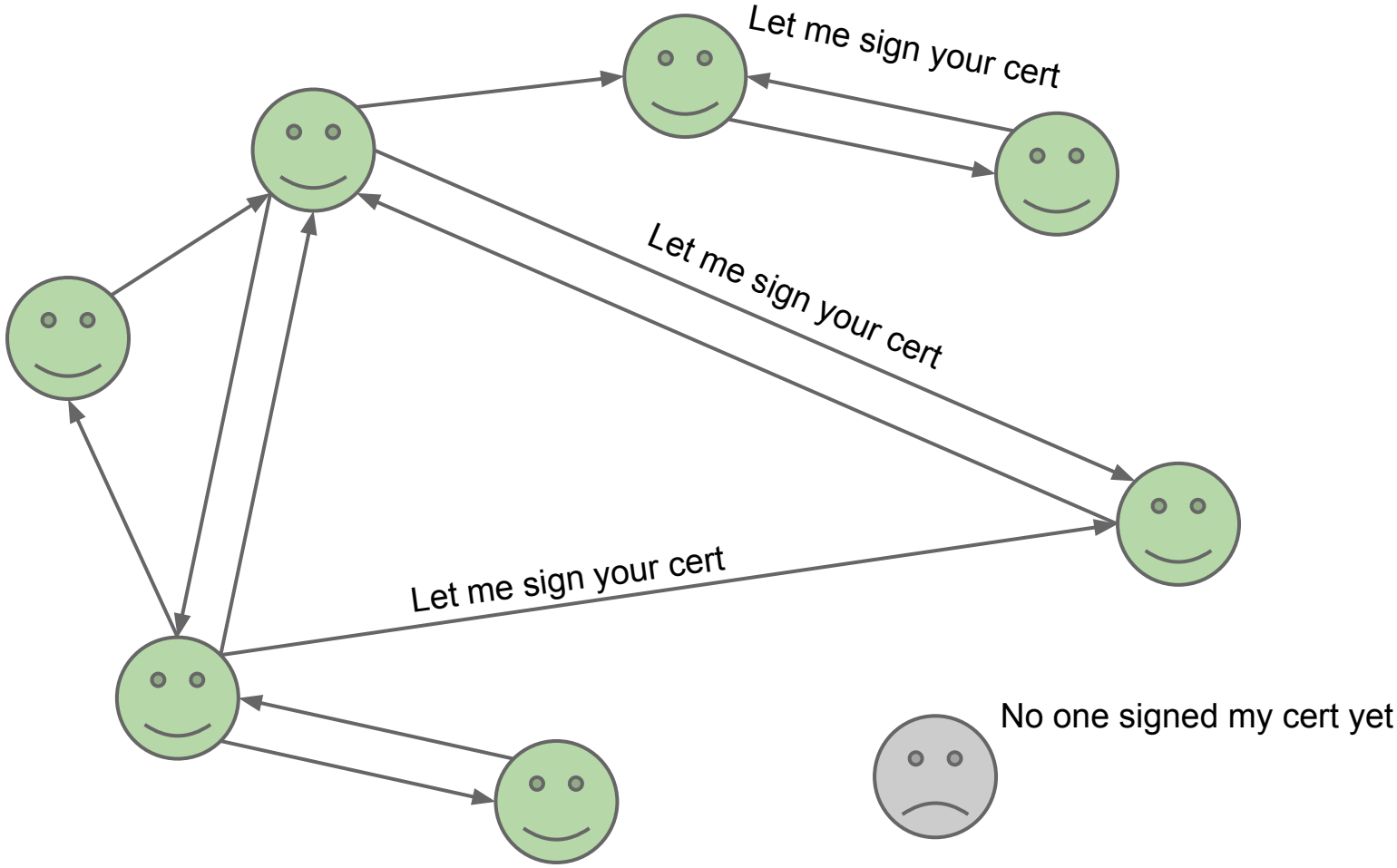
An **authority** releases it

- 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 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**?

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**?

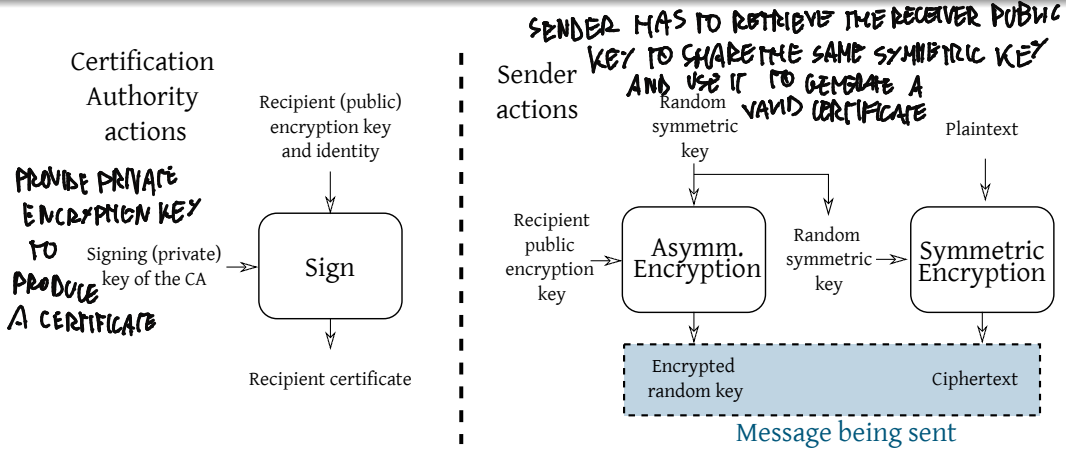
Any missing check = vulnerability!

Horror story 2

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? CHANGE IN COMPLEXITY
  - 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? → NO NEED TO DECRYPT; JUST PERFORMING OPERATIONS ON THE DATA WITHOUT ALWAYS KNOWING THEM
  - 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

Fundamentals of Information Theory

MATHEMATICAL MODEL OF HOW TWO ENDPOINTS COMMUNICATE TO EACH OTHER

What is Shannon's information theory?

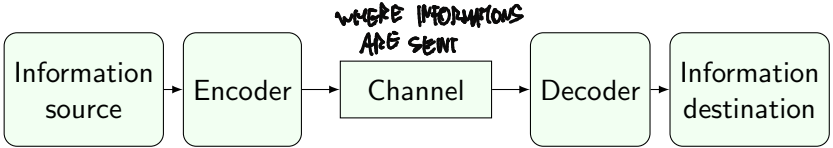
- Shannon's [3] way to mathematically frame communication
- A way to quantify information → AMOUNT OF INFORMATION CHANGED BETWEEN TWO ENDPOINTS

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

- Quantitatively frame "luck" and "guessing"

ESTIMATE HOW HARD IS TO GUESS A RANDOM STRING

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 dencoder
- Information is carried by a channel in the form of a sequence of symbols of a finite alphabet SEQUENCE OF BITS

GOAL: QUANTIFY THE AMOUNT OF INFORMATIONS SENT FROM A SENDER TO A RECEIVER

Transmitting and receiving

ASSUMPTION: ACQUIRING INFORMATIONS IS EQUIVALENT TO LOSING UNCERTAINTY. IN FACT, A RECEIVER RECEIVE A MESSAGE AND HE IS SURE OF THE INFORMATIONS ONLY WHEN THEY ARE ACTUALLY SAVED.

Losing uncertainty = Acquiring information

MEANWHILE, THE RECEIVER WAITS FOR COMUNICATIONS

- 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

A SOURCE THAT HAS A HUGE AMOUNT OF RANDOM SYMBOLS IS, GENERALLY, PROVIDING A LOT OF INFORMATIONS

Measuring uncertainty: Entropy

BECAUSE RECEIVING INFORMATIONS IMPLIES, FOR SURE, TO ACQUIRE KNOWLEDGE

Desirable properties

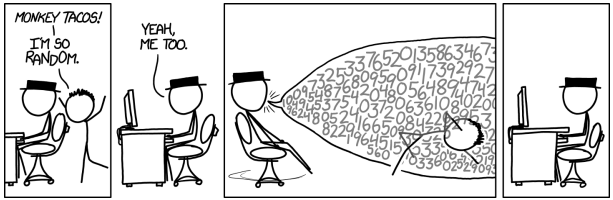
- Non negative measure of uncertainty
- "combining uncertainties" should map to adding entropies

INFORMATION RECEIVED FROM DIFFERENT SOURCES ⇓ SUM EVERYTHING

Definition

- Let $\mathcal{X}$ be a discrete r.v. with n outcomes in $\{x_0, \ldots, x_{n-1}\}$ with $\Pr(\mathcal{X} = x_i) = p_i$ for all $0 \leqslant i \leqslant n$
- 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 ($6^{26} \approx 3.1 \cdot 10^7$)
  - $H(\mathcal{X}) \approx \sum_{i=0}^{3.1 \cdot 10^7} -\frac{1}{3.1 \cdot 10^7} \log_b(\frac{1}{3.1 \cdot 10^7}) \approx 28.2b$
- $\mathcal{X}$ is a uniform pick from 6-letters English words
  - $H(\mathcal{X}) \approx \sum_{i=0}^{6300} -\frac{1}{6300} \log_b(\frac{1}{6300}) \approx 12.6b$

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. YOU CAN'T COMPRESS THE SOURCE IN LESS THAN $n \cdot H(\mathcal{X})$ (A MULTIPLE OF THE ENTROPY), OTHERWISE WE MIGHT LOSE SOME INFORMATIONS

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

WE DO NOT FOCUS ON HARDNESS

Min-Entropy

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

IDEA: WE DON'T WANT TO GUESS ALL THE POSSIBLE OUTCOMES, BUT ONLY THE MOST COMMONS

Example

A very biased r.v.

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

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)

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.

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**!

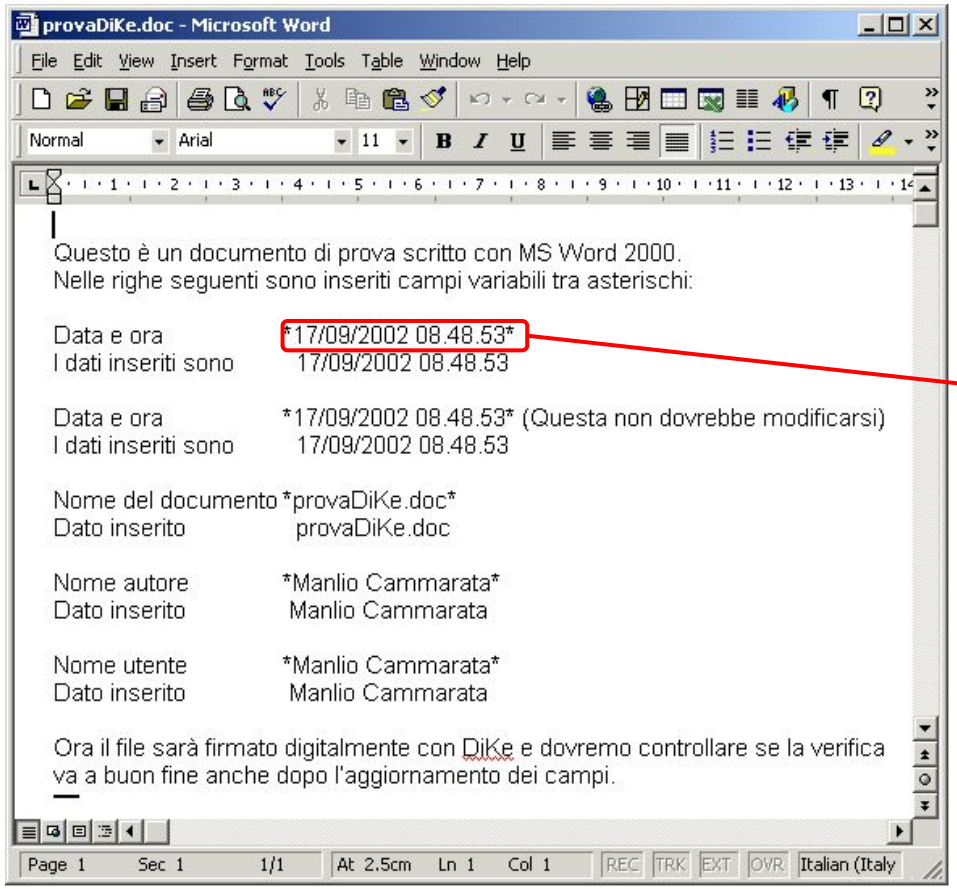
However, vulnerabilities did emerge

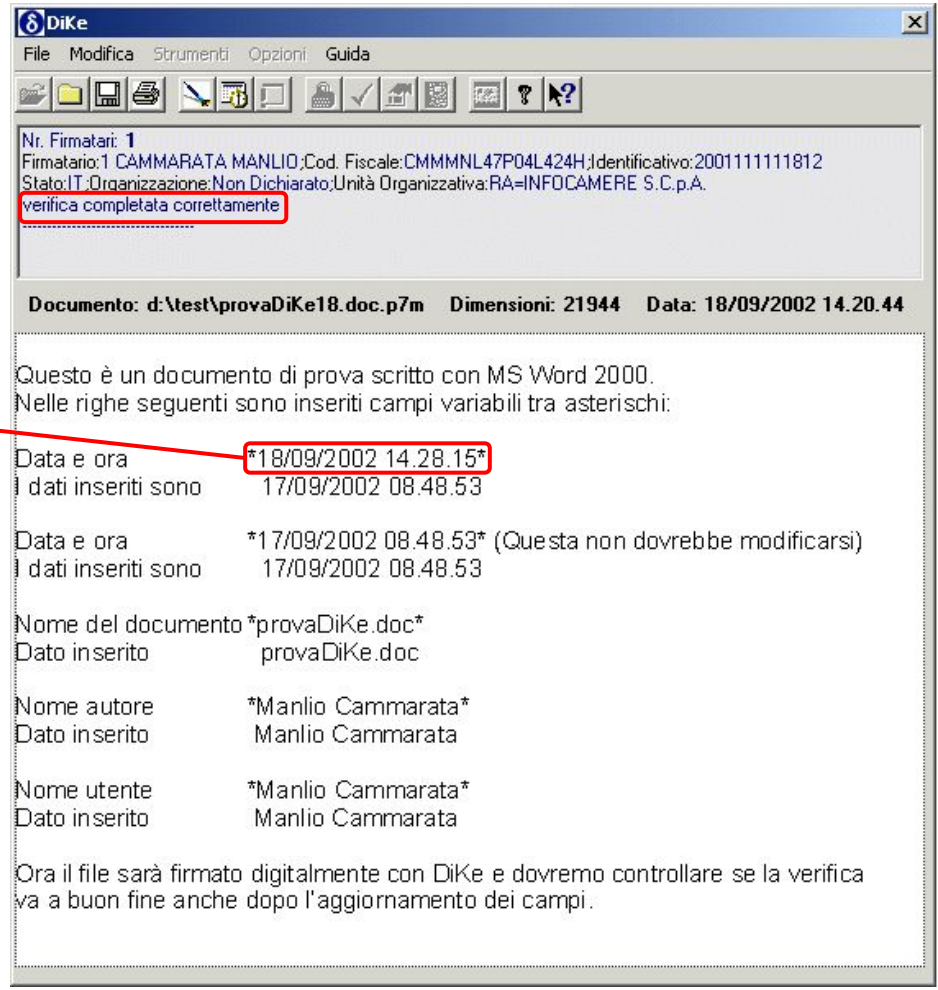
Do you remember the “bank vault door in a tent”?

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

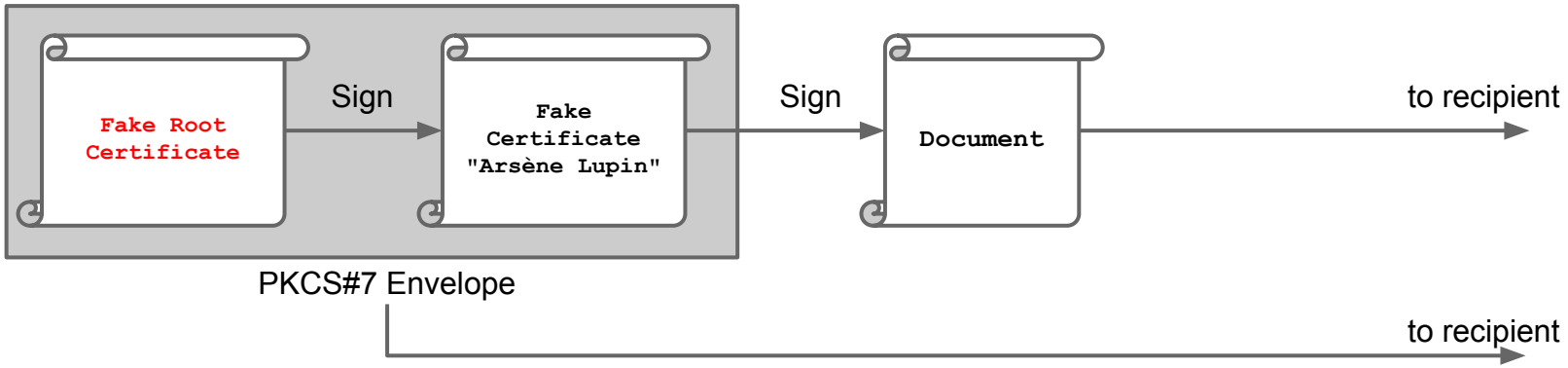
- Firma&Cifra was the digital signature application by PostECom
- Bug found by anonymous on 20/03/2003: http://www.interlex.it/docdigit/sikur159.htm
- **Result: creation** and **verification** of a **signature** with a **fake certificate**
- Also in this case: no cryptographic algorithm was broken to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

Vulnerability Description

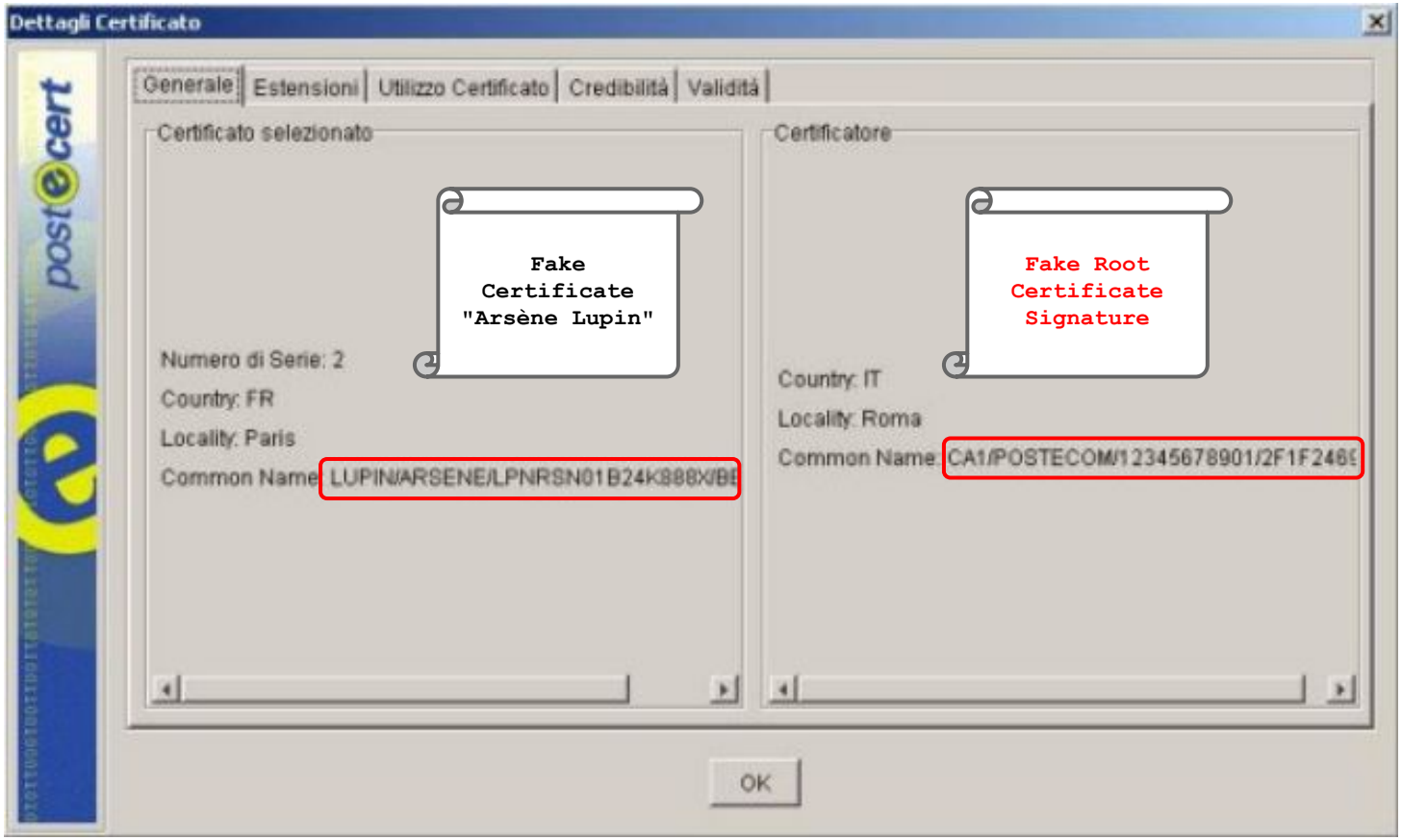
- In order to verify a signature, we need **author certificate** and the **certificate chain**
  - **Theoretically**, all available **online**
  - To allow **offline verification**, everything included with the document, in a PKCS#7 envelope
- Verification of root certificates must use preinstalled ones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

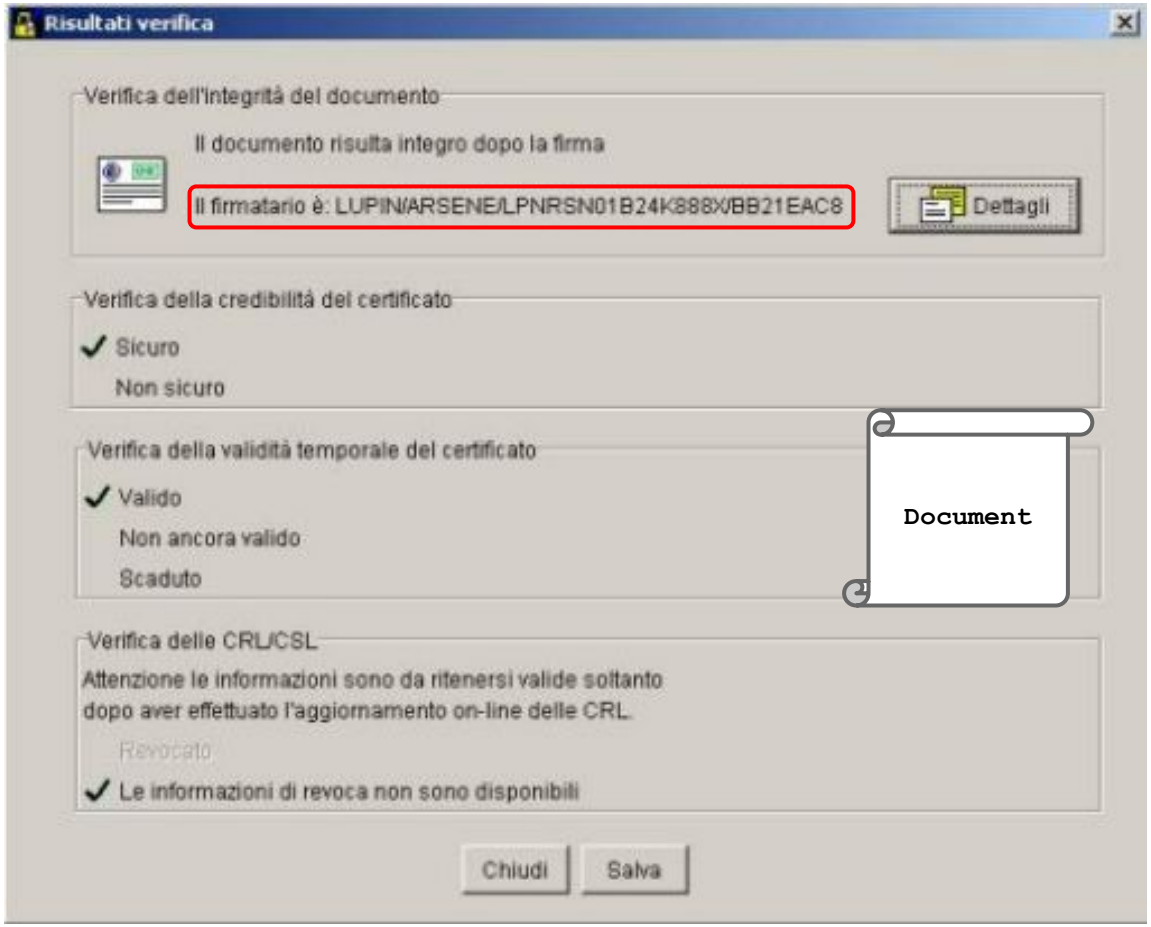
1. Generate a **fake *root* certificate** with the same name as a real one (e.g., PostECom itself)
2. Use this to generate a **fake *user* certificate** (in our example Arsène Lupin)
3. Use **Arsène Lupin's certificate** to sign theft and burglary confessions.
4. Include the fake root cert to the PKCS#7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