Lecture 6

Introduction to Public Key Cryptography

Stefan Dziembowski

www.crypto.edu.pl/Dziembowski

University of Warsaw

16.11.16 **version 1.0**

Plan

1. Public key cryptography – an overview
2. The key management problem
    1. qualified signatures
    2. public key infrastructure
3. Identity-based cryptography

Public-Key Cryptography

also called: **asymmetric cryptography**

Ralph Merkle (1974)

Whitfield Diffie and Martin Hellman (1976)

A little bit of history

Diffie and Hellman were the first to publish a paper containing the idea of the public-key cryptography:

W.Diffie and M.E.Hellman,
New directions in cryptography
IEEE Trans. Inform. Theory, IT-22, 6, 1976, pp.644-654.

A similar idea was described by **Ralph Merkle**:

in **1974** he described it in a project proposal for a Computer Security course at UC Berkeley
(it was rejected)
in **1975** he submitted it to the CACM journal (it was rejected)
(see www.merkle.com/1974/)

In **1997** the GCHQ (the British equivalent of the NSA) revealed that they new it already in **1973**.

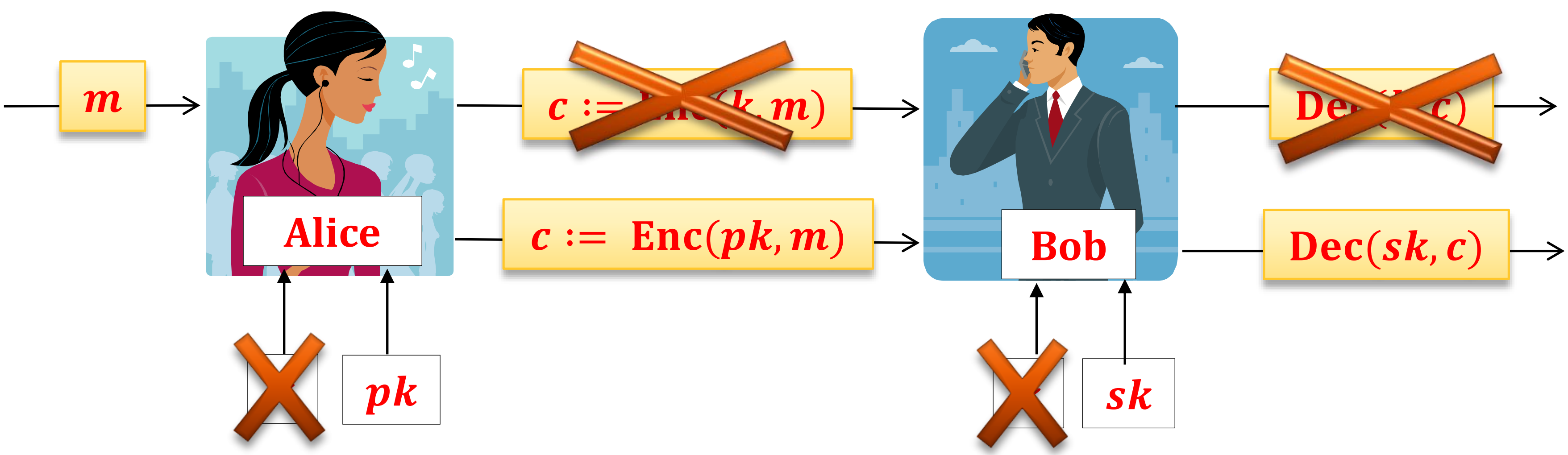
The idea

Instead of using one key $\boldsymbol{k}$,
use **2** keys $(\boldsymbol{pk}, \boldsymbol{sk})$, where

- $\boldsymbol{pk}$ is used for **encryption**,
- $\boldsymbol{sk}$ is used for **decryption**.

$\boldsymbol{pk}$ can be public, and only $\boldsymbol{sk}$ has to be kept secret!

That's why it's called: **public-key cryptography**

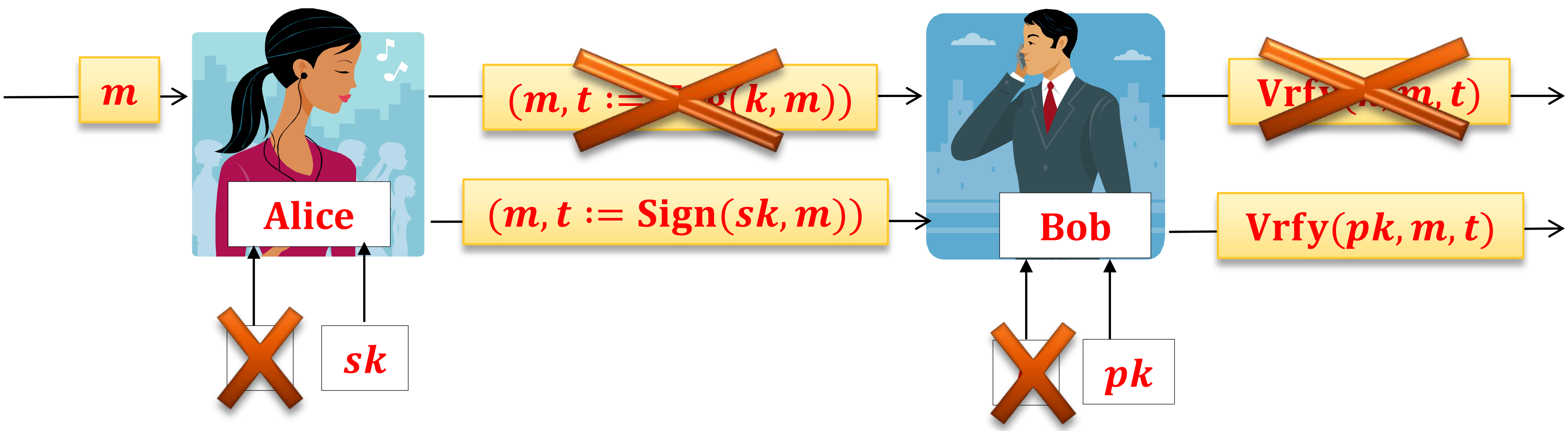


The same thing works for authentication

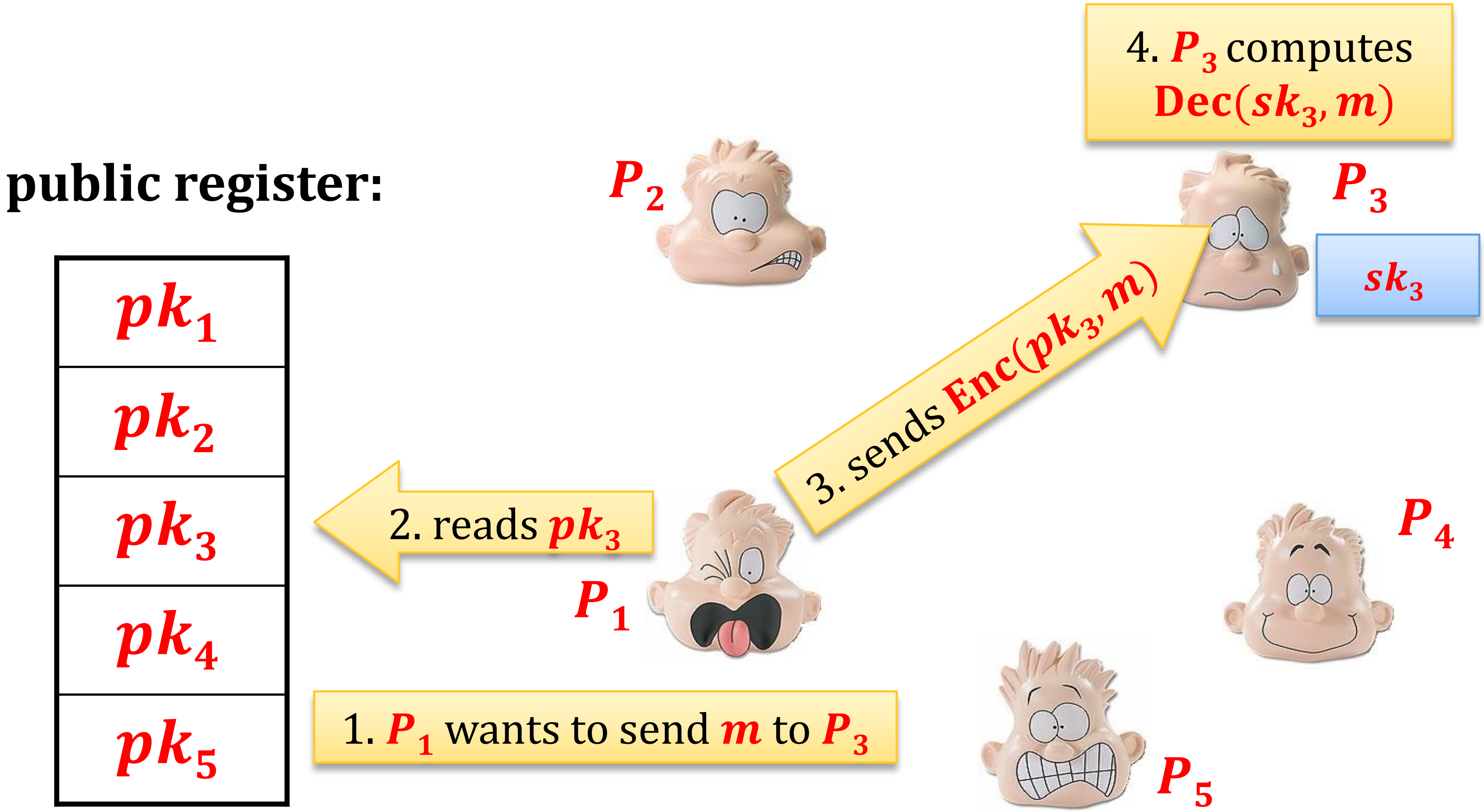
- sk is used for **computing a tag**,
- pk is used for **verifying correctness of the tag**.

this will be called "**signatures**"

Sign – the signing algorithm



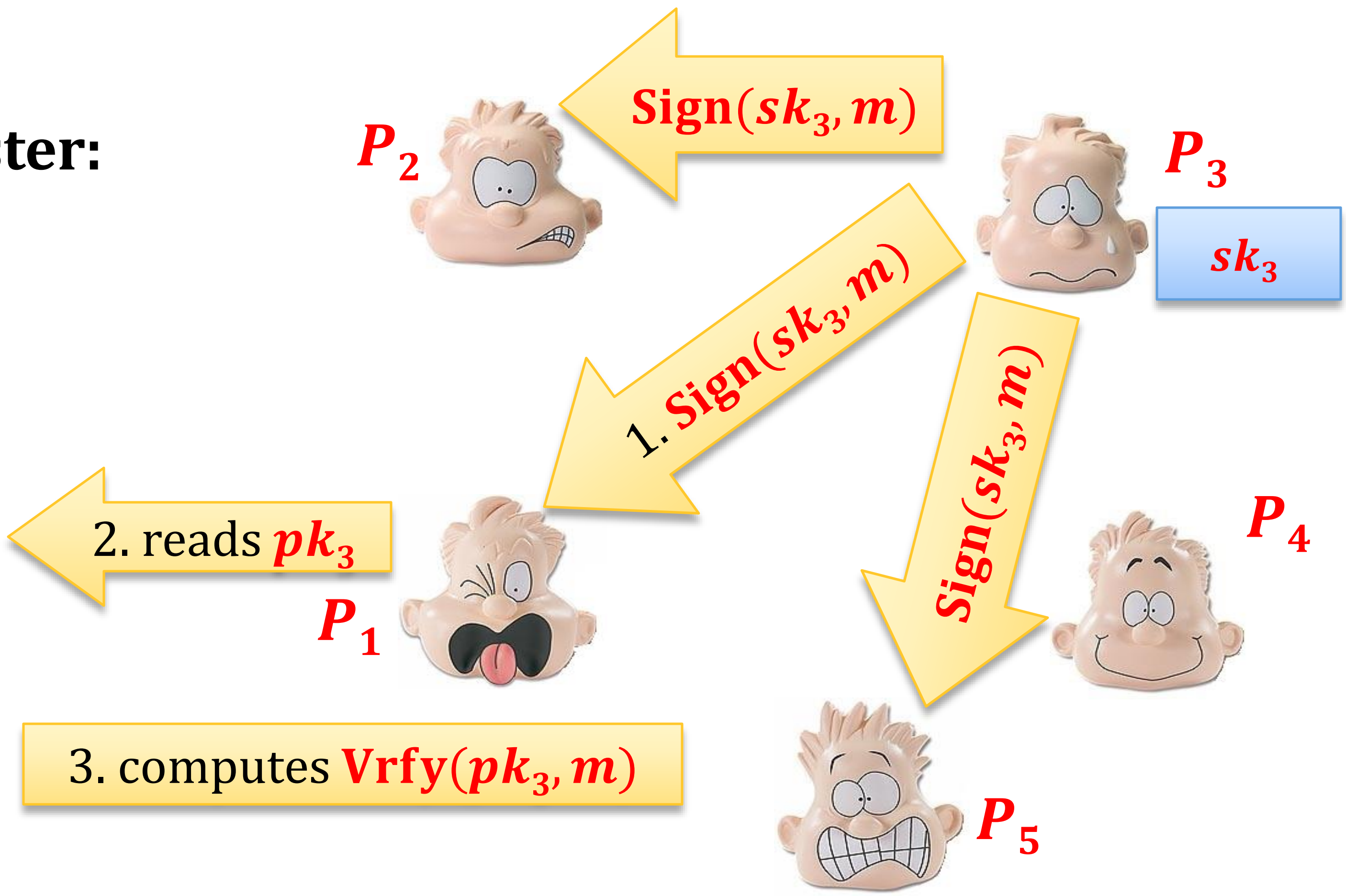
Anyone can send encrypted messages to anyone else



Anyone can verify the signatures

public register:

pk_1
pk_2
pk_3
pk_4
pk_5



Advantages of the signature schemes

Digital signatures are:

1. **publicly verifiable**,
2. **transferable**, and
3. provide **non-repudiation**

(we explain it on the next slides)

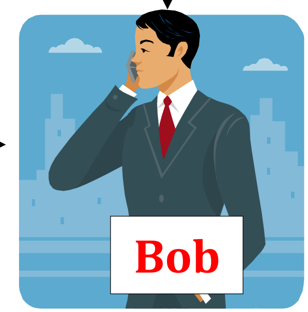
Look at the MACs...

k

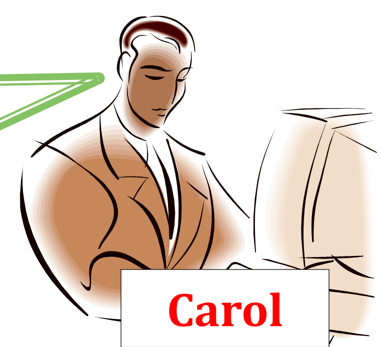
$m \in \{0, 1\}^*$

$(m, t = \mathrm{Tag}_k(m))$

k



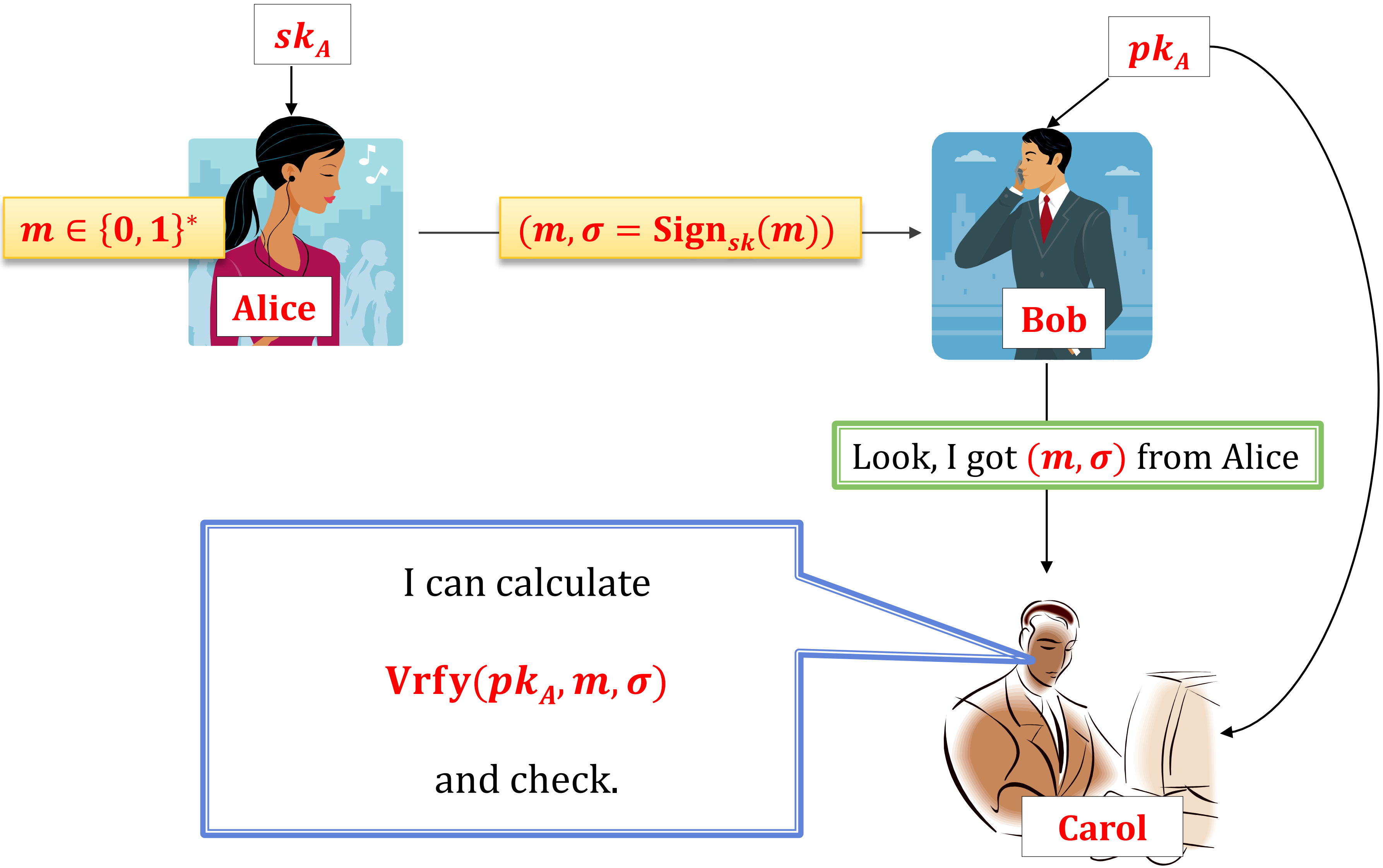
Look, I got (m, t) from Alice



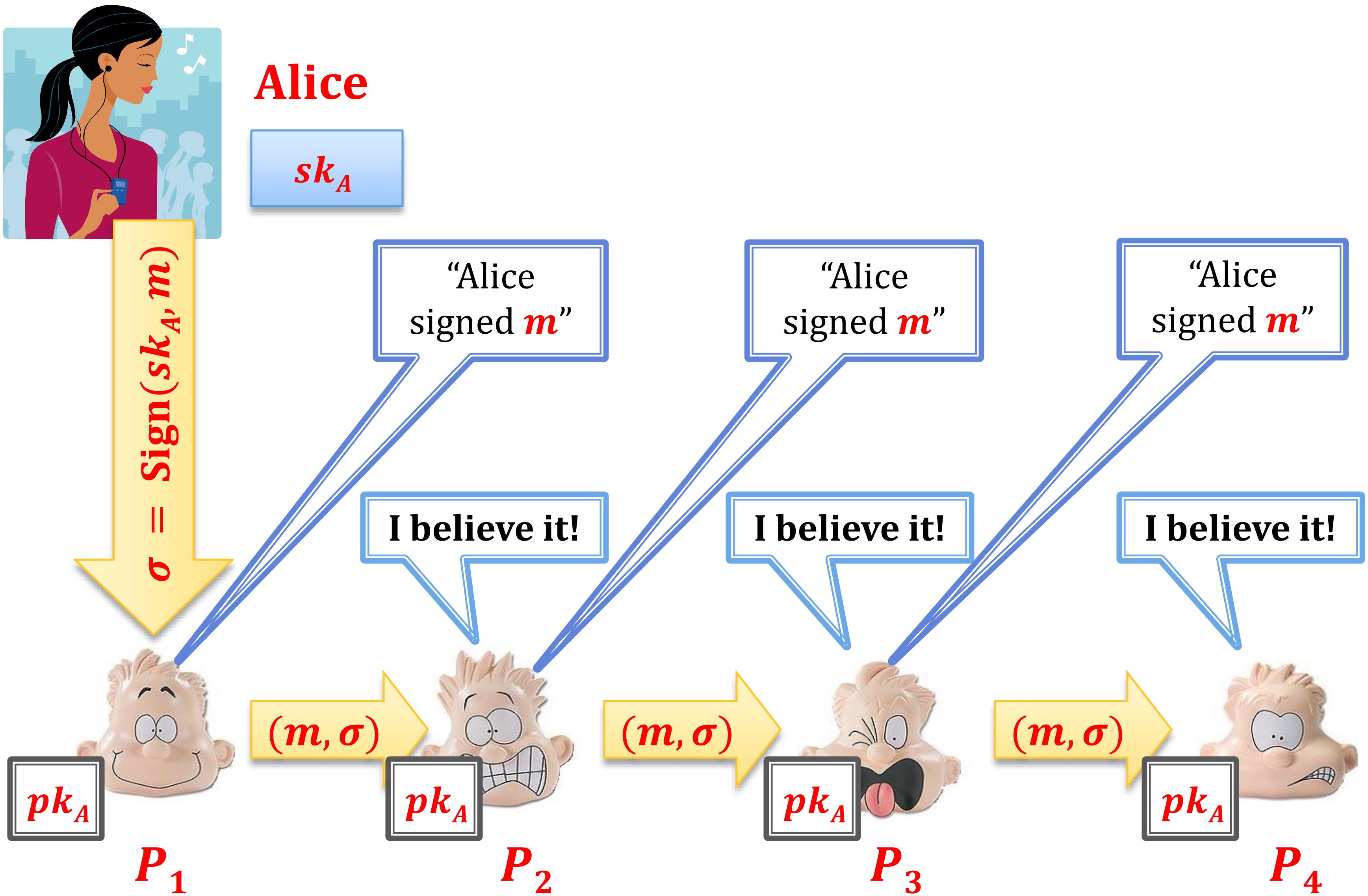
Why shall I trust you?

1. You could have created t yourself (because you know k)
2. I don't know k, so how can I verify the tag?

Signatures are publicly-verifiable!



So, the signatures are transferable



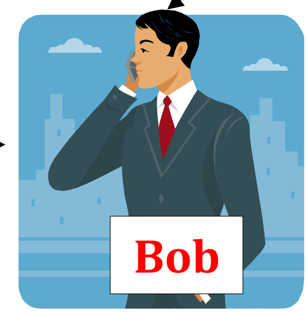
Non-repudiation

sk_A

$m \in \{0,1\}^*$

$(m, \sigma = \mathbf{Sign}_{sk}(m))$

pk_A



"I've got (m, σ) from Alice"

It's not true!
I never signed m!

$\mathbf{Vrfy}(pk, m, \sigma) \;=\; \textbf{yes}$
so you cannot **repudiate** signing m...

Things that need to be discussed

- Who **maintains "the register"**?
- How to **contact it securely**?
- How to **revoke the key** (if it is lost)?
- …

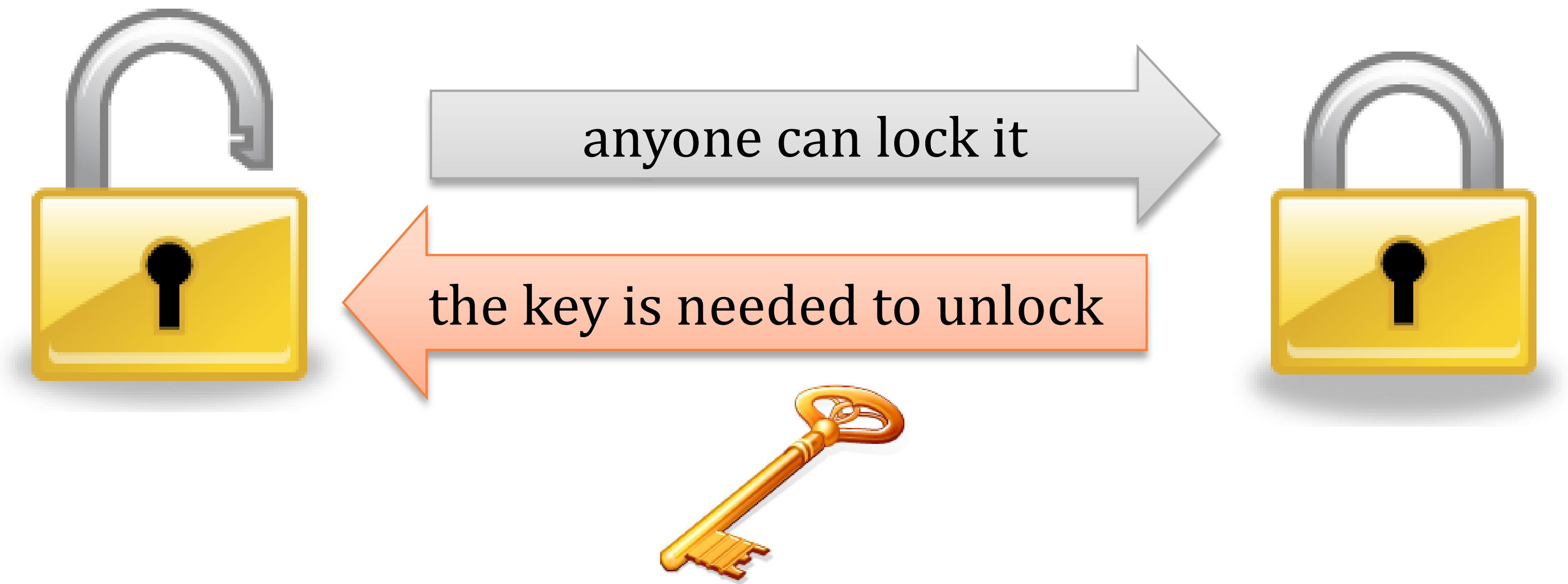
We will discuss these things later, when we will be talking about the
Public-Key Infrastructure

But is it possible?

In the “physical world”: **yes!**

Examples:

1. “normal” signatures
2. padlocks:



Diffie and Hellman (1976)

Diffie and Hellman proposed the public key cryptography in **1976**.

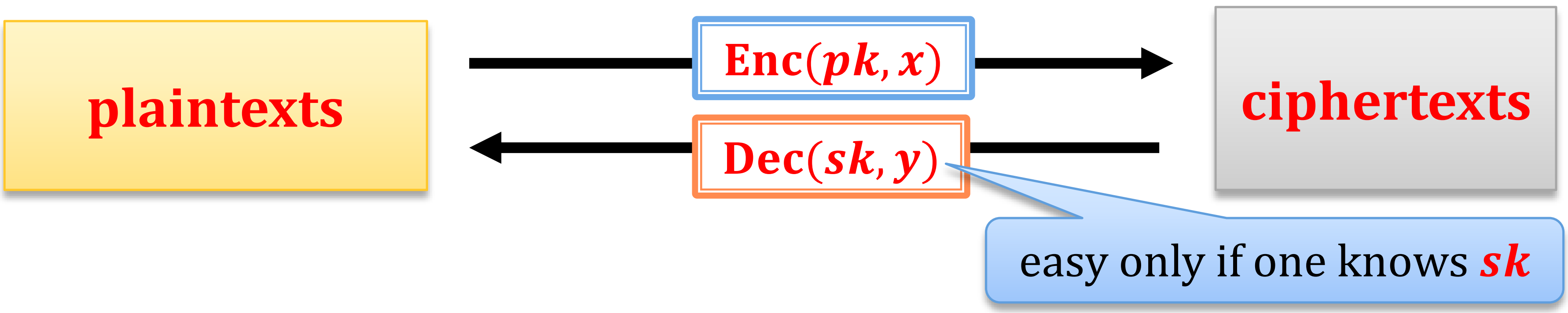
They just proposed the **concept**, not the **implementation**.

They have also shown a protocol for **key-exchange**.

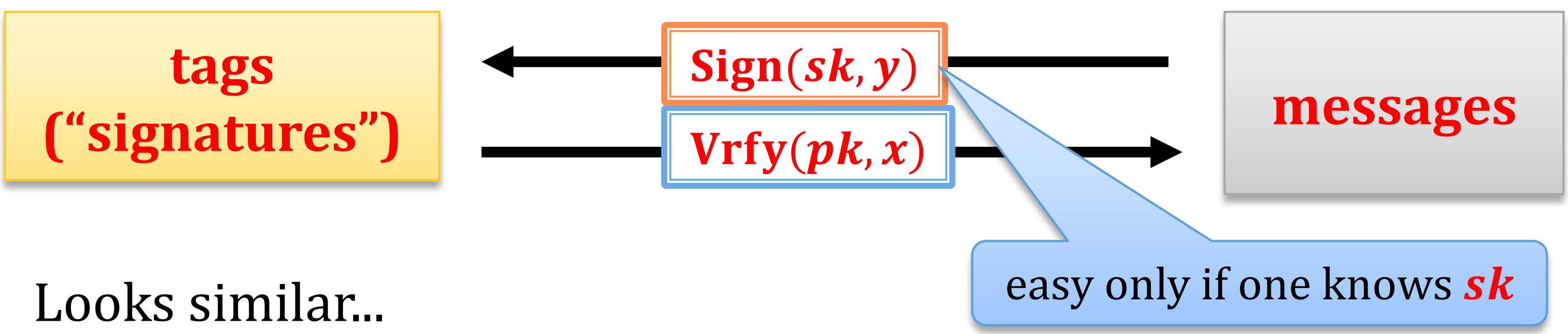
The observation of Diffie and Hellman:

(pk, sk) – the key pair

public-key encryption:



signature schemes:



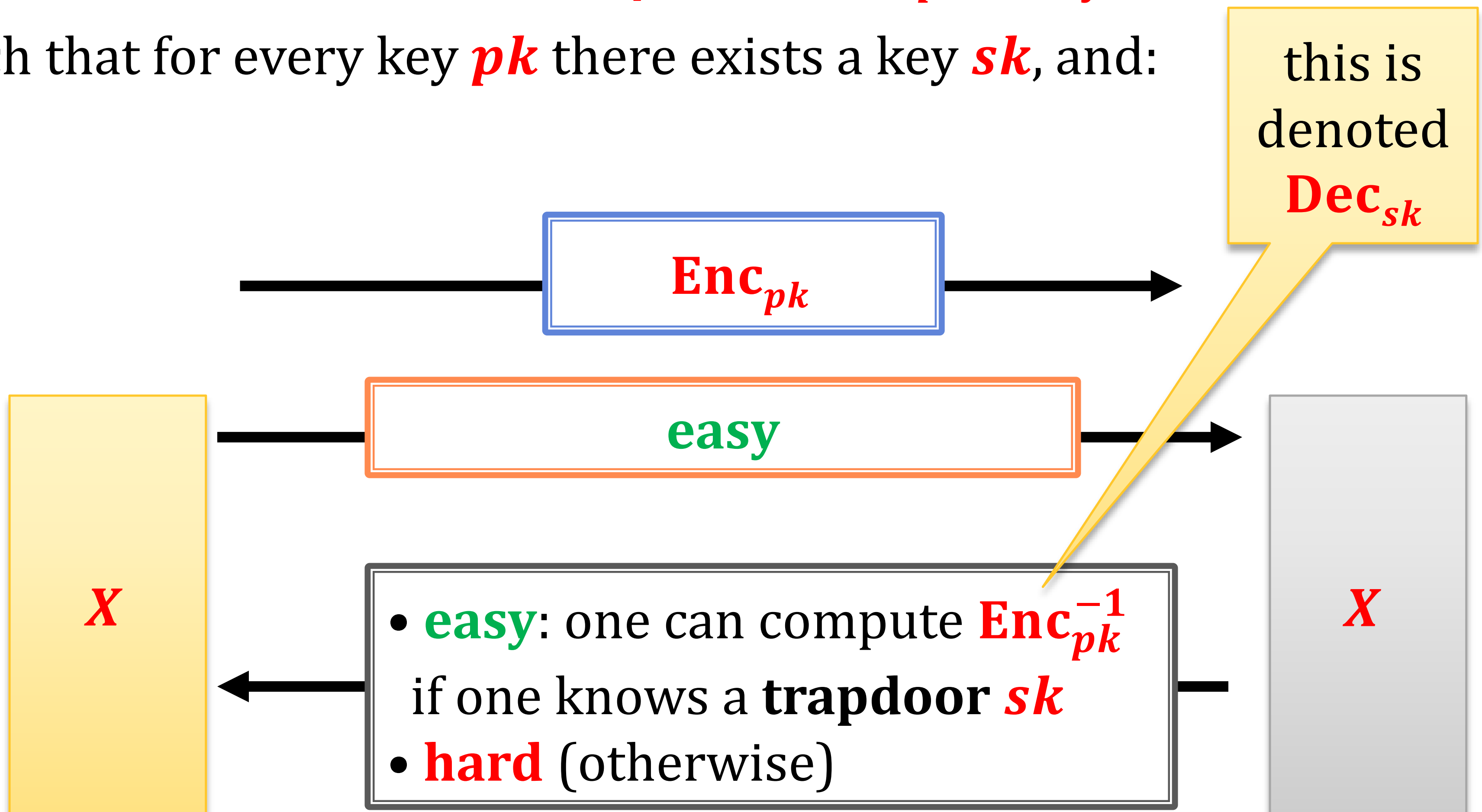
Looks similar...

Trapdoor permutations (informal definition)

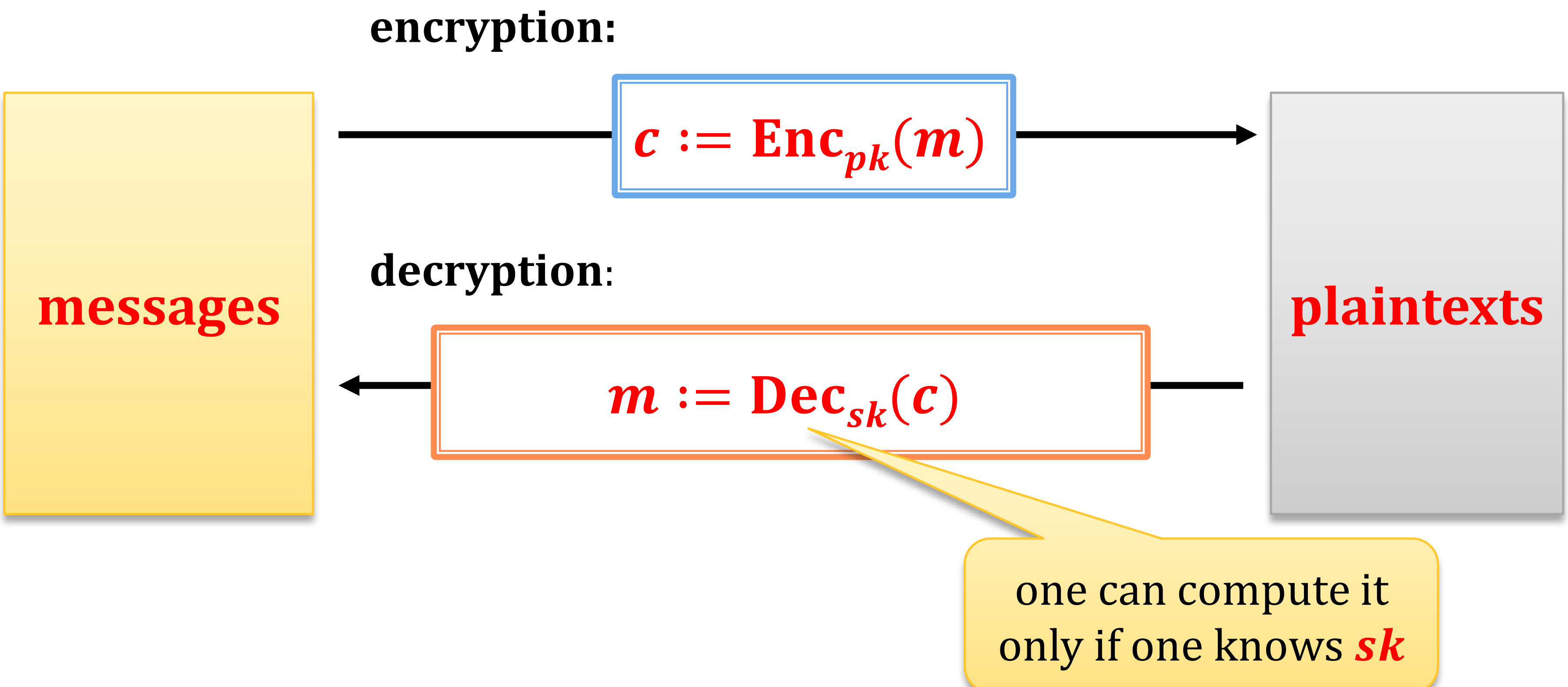
A family of permutations indexed by $\boldsymbol{pk} \in \mathbf{keys}$ **:**

$$\{\mathbf{Enc}_{pk} : X \rightarrow X\}_{pk \in \mathbf{keys}}$$

such that for every key $\boldsymbol{pk}$ there exists a key $\boldsymbol{sk}$, and:

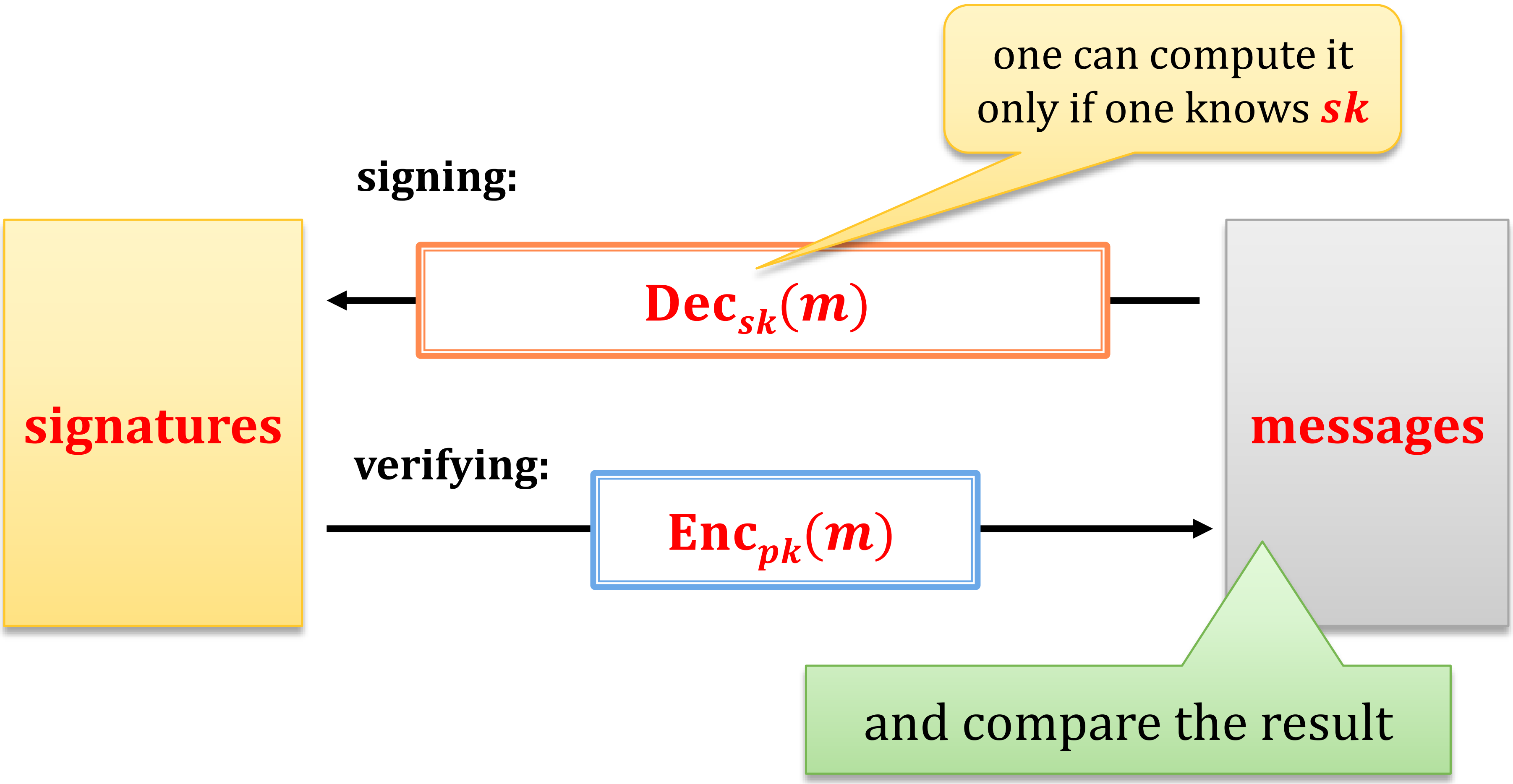


How to encrypt a message m



Warning: In reality it’s not that simple. We will explain it later.

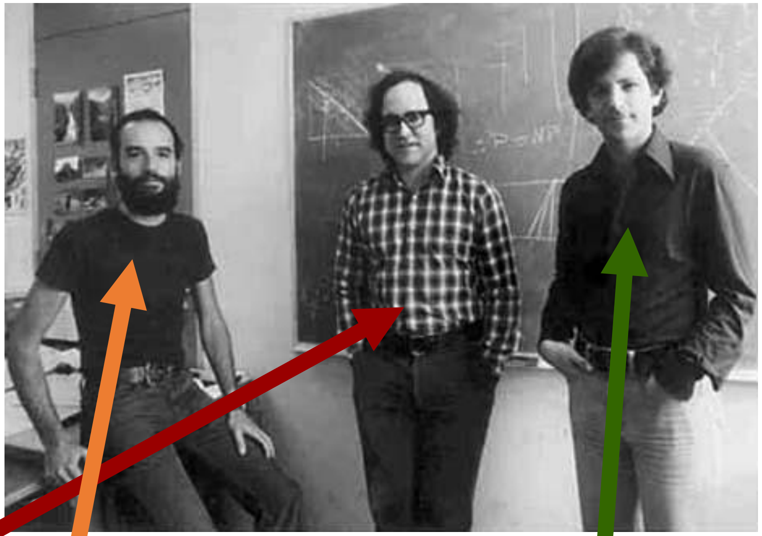
How to sign a message m



Warning: In reality it's not that simple. We will explain it later.

Do such functions exist?

Yes: exponentiation modulo N, where N is a product of two large primes.



Ron Rivest, Adi Shamir, and Leonard Adleman (1977)

RSA function is (conjectured to be) a trapdoor permutation!

The RSA function

$N = pq$, such that p and q are **random primes**, and $|p| = |q|$

e – **random such that** $e \perp (p-1)(q-1)$

d – **random such that** $ed = 1 \pmod{(p-1)(q-1)}$

$pk := (N, e) \quad sk := (N, d)$

$\mathbf{Enc}_{pk}\colon Z_N \to Z_N$ is defined as:

$$\mathbf{Enc}_{pk}(m) = m^e \bmod N.$$

$\mathbf{Dec}_{sk}\colon Z_N \to Z_N$ is defined as:

$$\mathbf{Dec}_{sk}(c) = c^d \bmod N.$$

Questions and doubts

$N = pq$, such that p and q are random primes, and $|p| = |q|$

e – random such that $e \perp (p-1)(q-1)$

d – random such that $ed = 1 \pmod{(p-1)(q-1)}$

$pk := (N, e) \quad sk := (N, d)$

$\mathbf{Enc}_{pk}\colon Z_N \to Z_N$ is defined as:

$\mathbf{Enc}_{pk}(m) = m^e \bmod N$.

$\mathbf{Dec}_{sk}\colon Z_N \to Z_N$ is defined as:

$\mathbf{Dec}_{sk}(c) = c^d \bmod N$.

How **large these primes** need to be? How to **sample** them?

where does this come from?

Can **exponentiation** be done **efficiently**?

$\mathbf{Enc}_{pk}(1) = 1^e \bmod N = 1$

Oops...

encryption is **deterministic**...

We will address them later...

(N, e, d) – as on the previous slide

"Handbook" RSA

Handbook RSA encryption scheme:

messages and **ciphertexts**: Z_N

- $\mathbf{Enc}_{N,e}(m) = m^e \bmod N$
- $\mathbf{Dec}_{N,d}(c) = c^d \bmod N$

Handbook RSA signature scheme:

messages and **signatures**: Z_N

- $\sigma := \mathbf{Sign}_{N,d}(m) = m^d \bmod N$
- $\mathbf{Vrfy}_{N,e}(m, \sigma) =$ **output yes iff** $\sigma^e \bmod N = m$

Is **RSA** secure?

Is **RSA** secure:

1. as an **encryption scheme**?
2. as a **signature scheme**?

The answer is not that simple.

First, we would need to define security!

We will do it on the next lectures.

Symmetric vs asymmetric crypto

Asymmetric cryptography (also called: private key cryptography) is **much more efficient**!

Example (Intel Core 2 1.83 GHz processor):

	MiB/Second	Cycles/Byte
AES/CTR (128-bit key)	**139**	**12.6**
HMAC(SHA-1)	**147**	**11.9**

	Operations/Second	Megacycles/Operation
RSA 2048 Encryption	**6,250**	**0.29**
RSA 2048 Signature	**165**	**11.06**

Source: https://www.cryptopp.com/benchmarks.html

Practical solutions

Typically **asymmetric cryptography** is **combined** with the **symmetric one**.

For example: asymmetric cryptography is used only for **agreeing on a symmetric key**.

Or: one can combine it directly using a "**hybrid approach**".

(we will discuss it later)

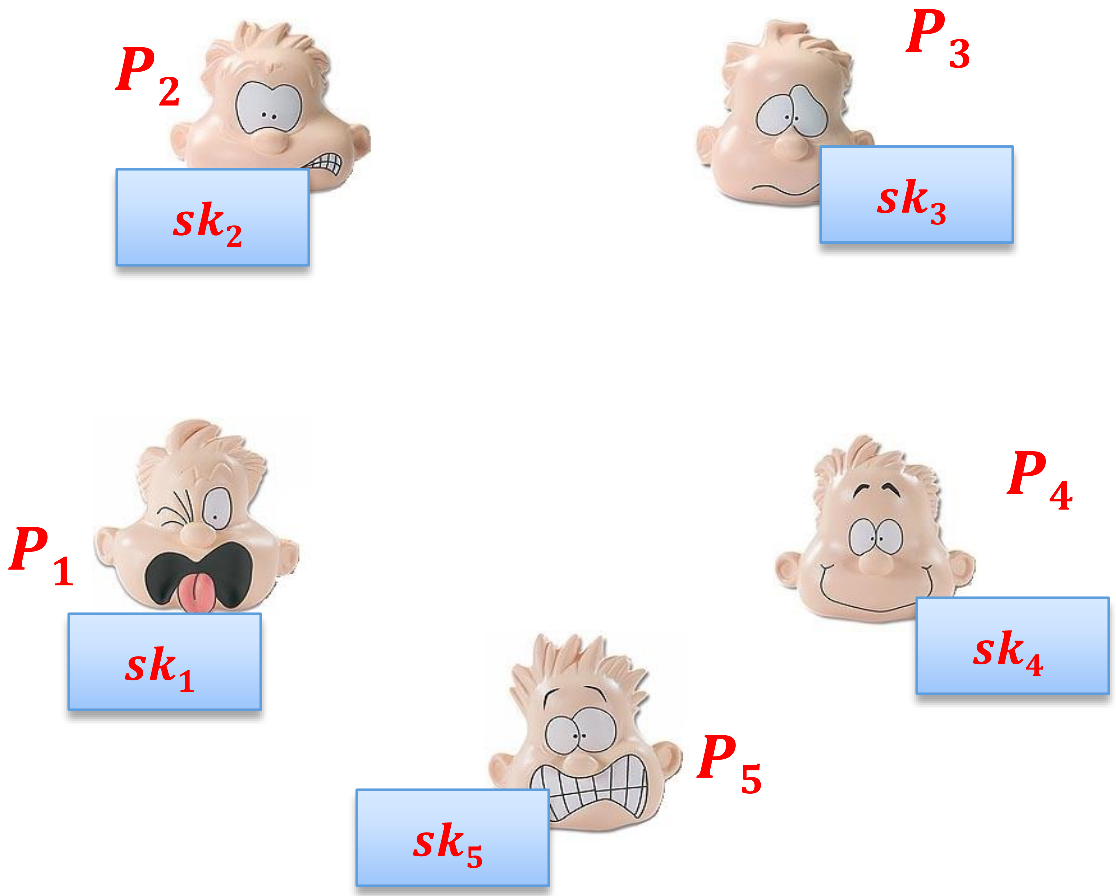
Plan

1. Public key cryptography – an overview
2. The key management problem
   1. qualified signatures
   2. public key infrastructure
3. Identity-based cryptography

Remember this slide?

public register:

pk_1
pk_2
pk_3
pk_4
pk_5

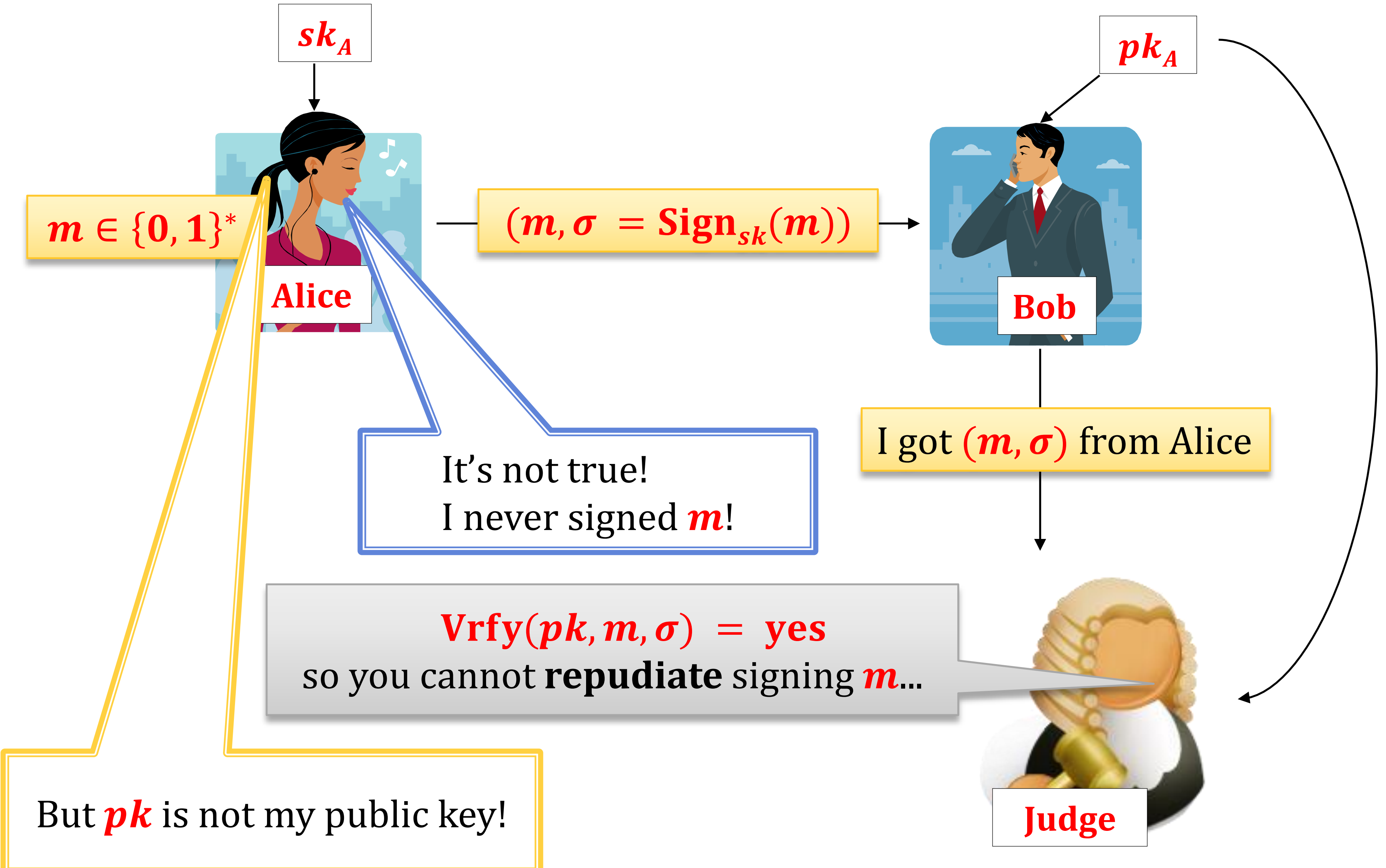


Question:
How to maintain the public register?

1. We start with the case when the public keys are used for **signing that is legally binding**.

2. Then we consider other cases.

A problem


sk_A
pk_A
$m \in \{0,1\}^*$
Alice
$(m, \sigma = \mathbf{Sign}_{sk}(m))$
Bob
It's not true!
I never signed m!
I got (m, σ) from Alice
$\mathbf{Vrfy}(pk, m, \sigma) = \mathbf{yes}$
so you cannot **repudiate** signing m...
But pk is not my public key!
Judge

Solution: certification authorities

A simplified view:

$(pk_{\text{Cert}}, sk_{\text{Cert}})$

Certification Authority

comes with her **ID** and pk_{Alice}

Alice

checks the **ID** of **Alice** and
issues a **certificate**:

$\mathbf{Sign}_{sk_{\text{Cert}}}$(“$pk_{\text{Alice}}$ is a public key of **Alice**”)

Now, **everyone** can verify that pk_{Alice} is a public key of **Alice**.
So **Alice** can attach it to every signature

really everyone?

What is needed to verify the certificate

To verify the certificate coming from **Cert** one needs:

1. to **know** the public key of the **Cert**
2. to **trust** **Cert**.

It is better if **Cert** also keeps a document:

*"I, **Alice** certify that pk_{Alice} is my public key"*

with a **written** signature of **Alice**.

How does it look from the legal point of view?

What matters at the end is if you can **convince the judge**.

Many countries have now a special law regulating these things.

In **Poland**:

Ustawa o podpisie elektronicznym, z dnia 18 września 2001 r.
(Dz.U.01.130.1450) 28 str. (ISIP), na podst. dyrektywy EU 1999/93/EC

This law defines the conditions to become an official **certification authority**.

A certificate issued by such an authority is called a **qualified certificate**.

A signature obtained this way is called a **qualified digital signature**.

The **qualified signature** is equivalent to the written one!

Polish Certificate Authorities:

NCCERT | NBP ENGLISH | ARCHIWUM | KONTAKT |

NBP Narodowy Bank Polski

Narodowe Centrum Certyfikacji (NCCert)

Strona główna | Dokumenty | Podmioty kwalifikowane | Zaświadczenia certyfikacyjne | Lista CRL | Lista TSL | Komunikaty

2015
2013
2012
2011
2010
2009
2008
2007
2006
2005

REJESTR KWALIFIKOWANYCH PODMIOTÓW ŚWIADCZĄCYCH USŁUGI CERTYFIKACYJNE

Narodowy Bank Polski prowadzi rejestr podmiotów kwalifikowanych od dnia 1 października 2005r.

Plik: NCCert.crt - zaświadczenie certyfikacyjne Narodowego Centrum Certyfikacji - (nowy root)

Wpisy uszeregowane pod kątem czasu uzyskania wpisu do rejestru - w kolejności od najwcześniejszego

Numer wpisu	Nazwa podmiotu	Rodzaj świadczonych usług	Czas dokonania wpisu
1.	UNIZETO TECHNOLOGIES Spółka Akcyjna	Wydawanie kwalifikowanych certyfikatów Wydawanie kwalifikowanych certyfikatów atrybutów	31 grudnia 2002 r.,godz. 12:00:00 13 września 2007 r.,godz. 10:00:00
		Znakowanie czasem	24 stycznia 2003 r., godz. 12:00:00

So, what to do if you want to issue the qualified signatures?

You have to go to one of these companies and **get a qualified certificate** (it costs!).

The certificate is **valid just for some period**.

What if the secret key is lost?

In this case you have to **revoke** the certificate.

Every authority maintains a list of **revoked certificates**.

The certificates come with some **insurance**.

In many case one doesn't want to use the qualified signatures

The certificates cost.

It's **risky** to use them:

How do you know what your computer is really signing?
Computers have **viruses, Trojan horses**, etc.

You can use **external trusted hardware** but it should have a display (so you can see what is signed).

Remember: qualified signatures are equivalent to the written ones!

Plan

1. Public key cryptography – an overview
2. The key management problem
    1. qualified signatures
    2. public key infrastructure
3. Identity-based cryptography

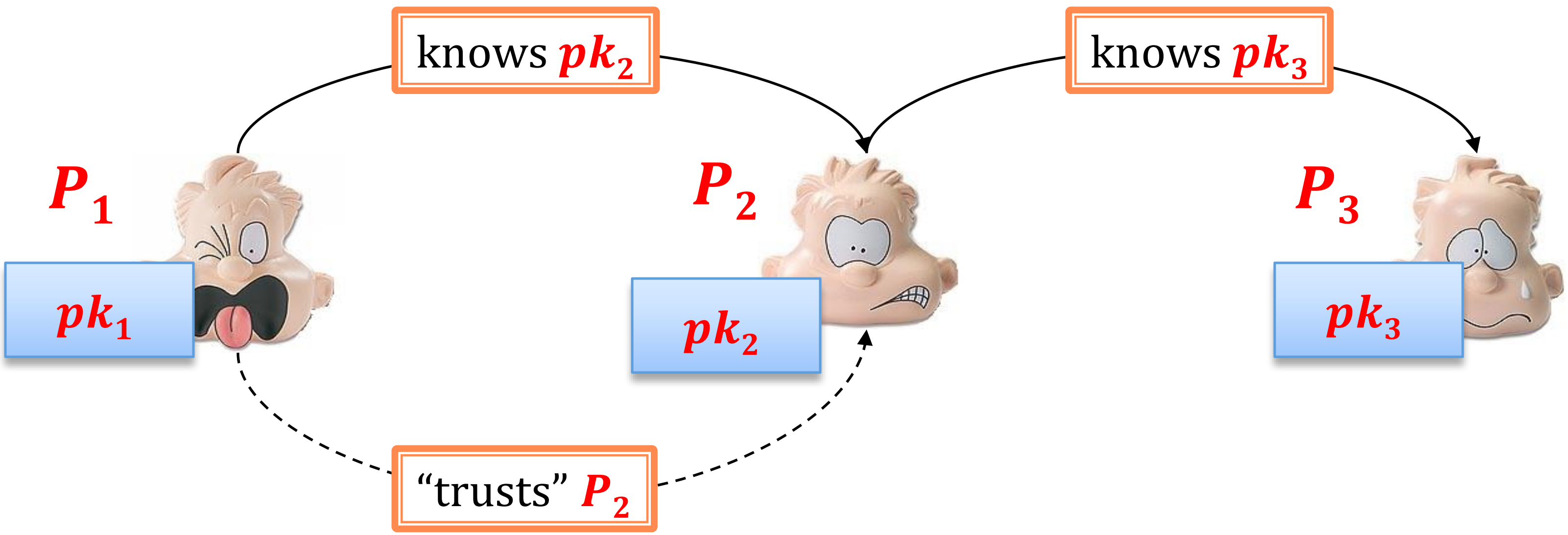
Practical solution

In many cases the **qualified signatures** are an overkill.

Instead, people use **non**-qualified signatures.

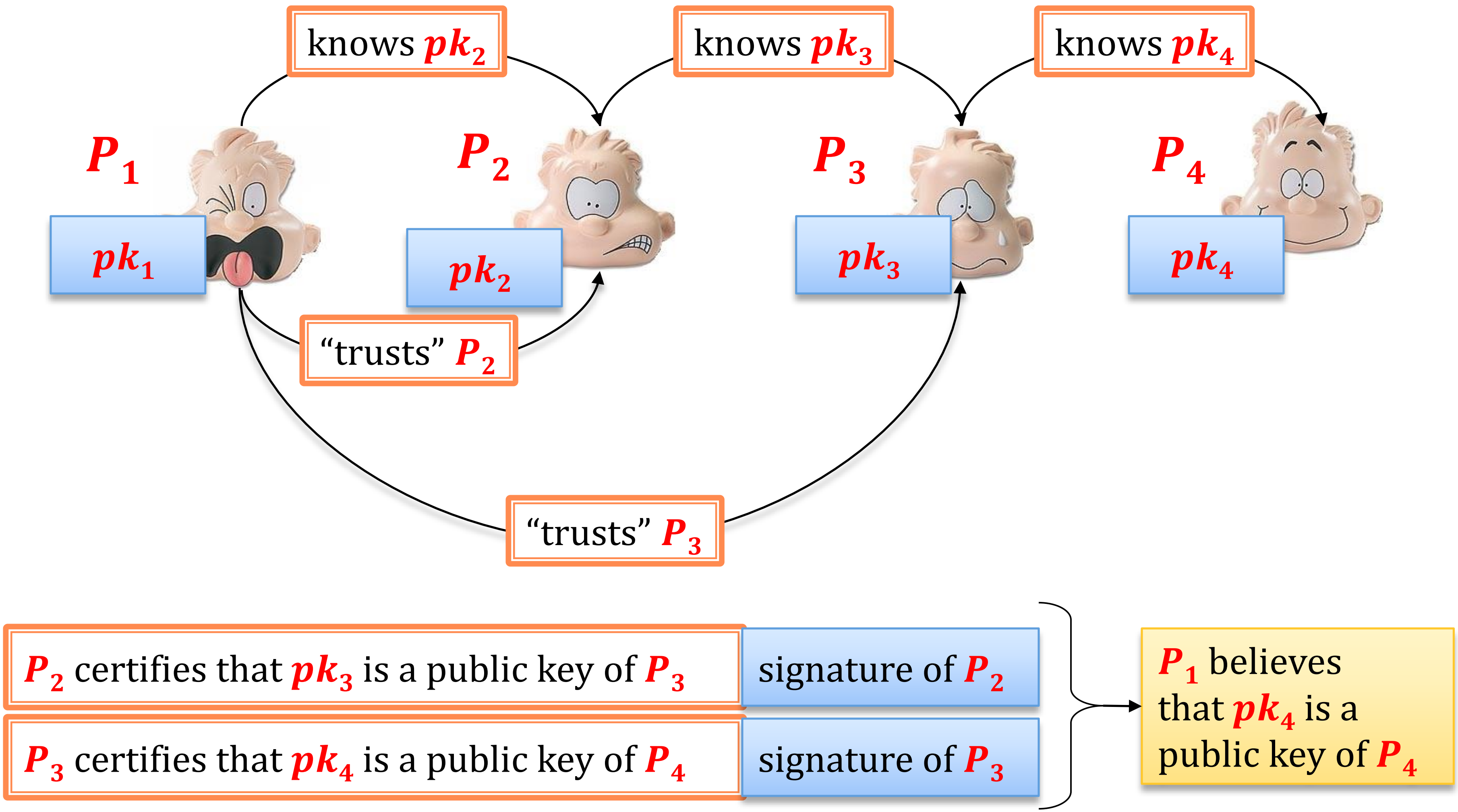
The certificates are distributed using a **public-key infrastructure (PKI)**.

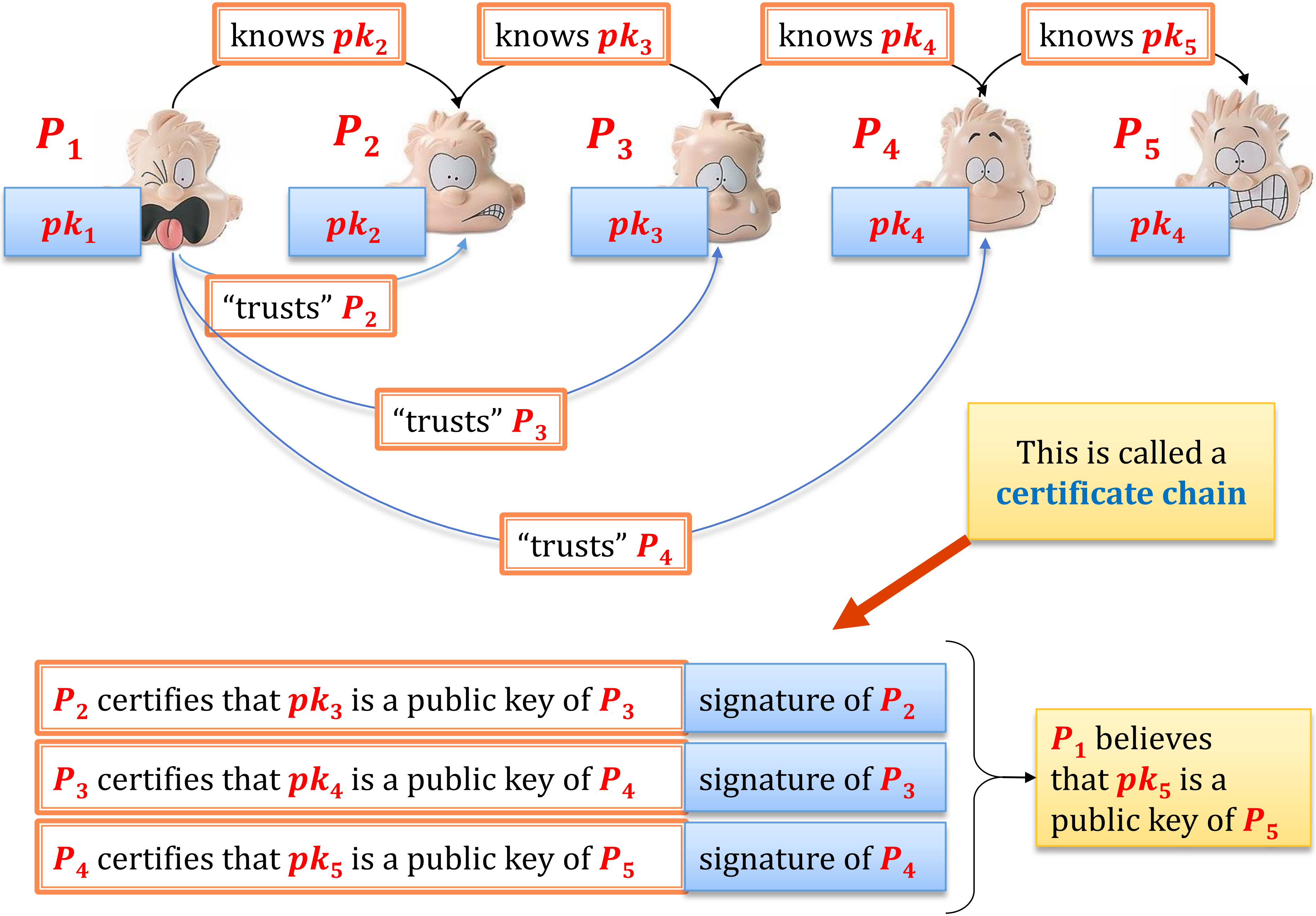
Users can certify keys of the other users



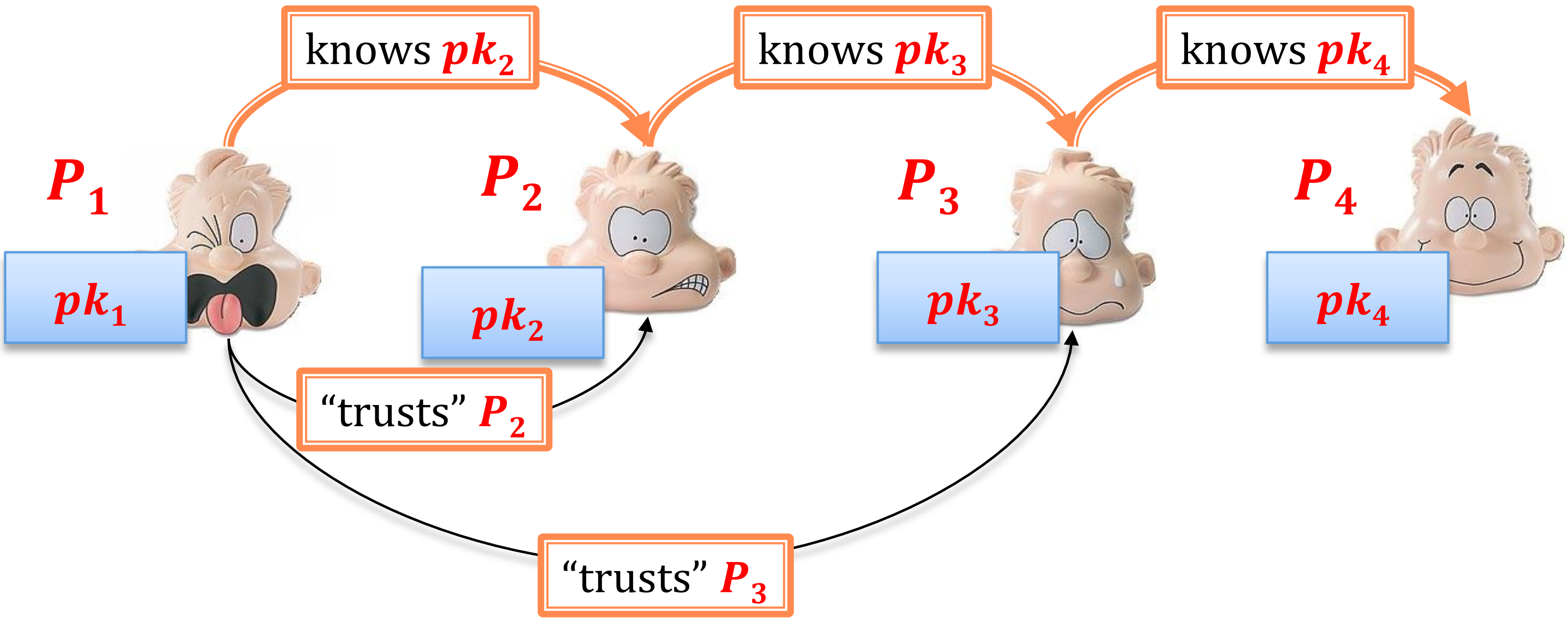
P_2 certifies that pk_3 is a public key of P_3 | signature of P_2 → P_1 believes that pk_3 is a public key of $P3$

this should be done only if P_2 really met P_3 in person and verified his identity


knows pk_2
knows pk_3
knows pk_4
P_1
P_2
P_3
P_4
pk_1
pk_2
pk_3
pk_4
“trusts” P_2
“trusts” P_3
P_2 certifies that pk_3 is a public key of P_3
signature of P_2
P_3 certifies that pk_4 is a public key of P_4
signature of P_3
P_1 believes that pk_4 is a public key of P_4


knows pk_2
knows pk_3
knows pk_4
knows pk_5
P_1
P_2
P_3
P_4
P_5
pk_1
pk_2
pk_3
pk_4
pk_4
“trusts” P_2
“trusts” P_3
“trusts” P_4
This is called a **certificate chain**
P_2 certifies that pk_3 is a public key of P_3
signature of P_2
P_3 certifies that pk_4 is a public key of P_4
signature of P_3
P_4 certifies that pk_5 is a public key of P_5
signature of P_4
P_1 believes that pk_5 is a public key of P_5

A problem



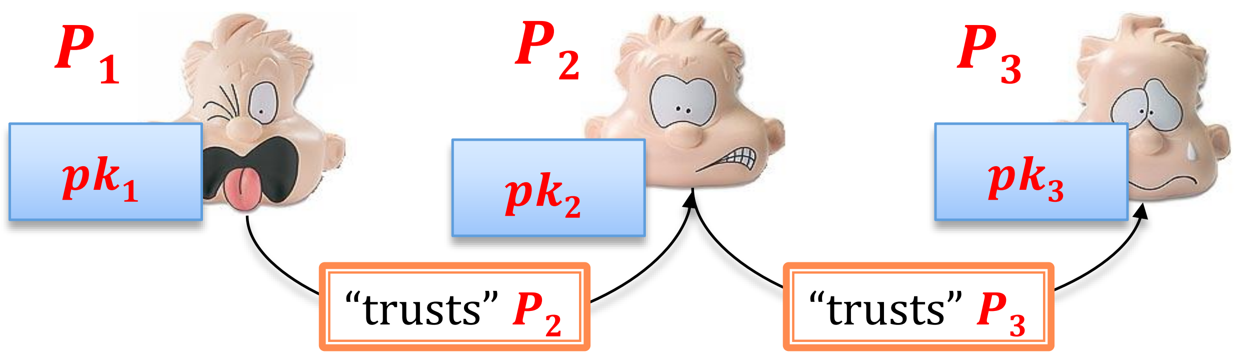
What if P_1 does not know P_3?

How can he trust him?

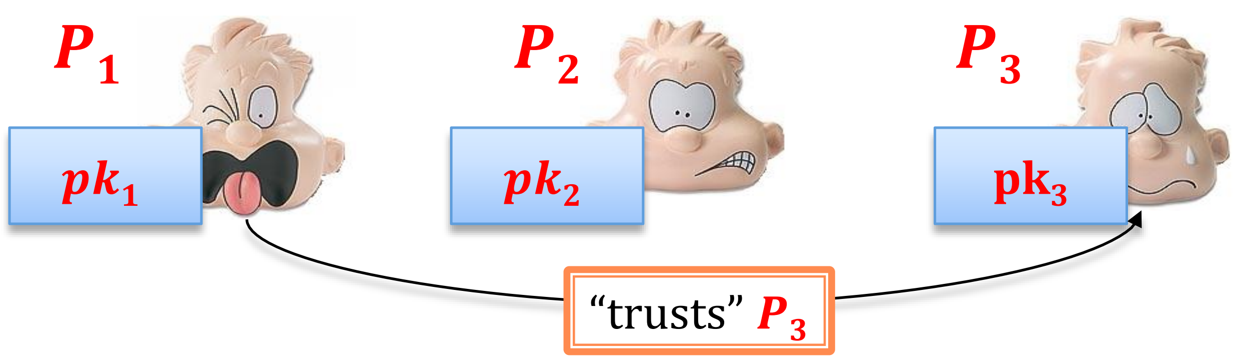
Answer: P_2 can **recommend** P_3 to P_1.

A question: is trust transitive?

Does:

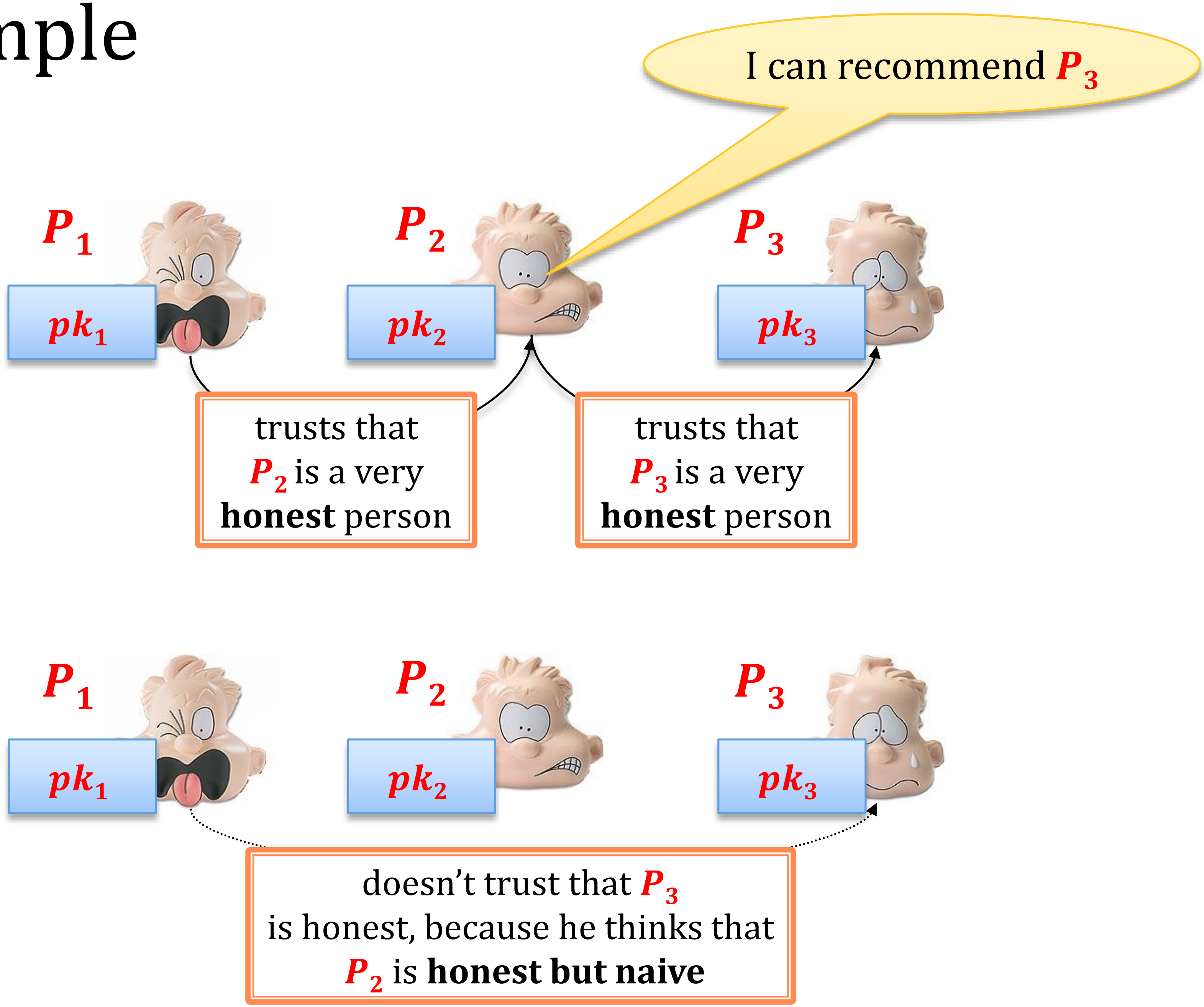


imply:



?

Example



I can recommend P_3
P_1
pk_1
P_2
pk_2
P_3
pk_3
trusts that P_2 is a very **honest** person
trusts that P_3 is a very **honest** person
P_1
pk_1
P_2
pk_2
P_3
pk_3
doesn't trust that P_3 is honest, because he thinks that P_2 is **honest but naive**

Moral

Trust is not transitive:

“P_1 trusts in the certificates issued by P_2”

is not the same as:

“P_1 trusts that
if
P_2 says: “you can trust the certificates issued by P_3”
then
one can trust the certificates issued by P_3”

Recommendation levels

level **1** recommendation:

A: *"you can trusts in all the certificates issued by **B**"*

level **2** recommendation:

A : *"you can trust that **all the level** **1** recommendations issued by **B**"*

level **3** recommendation:

B : *"you can trust that **all the level** **2** recommendations issued by **B**"*

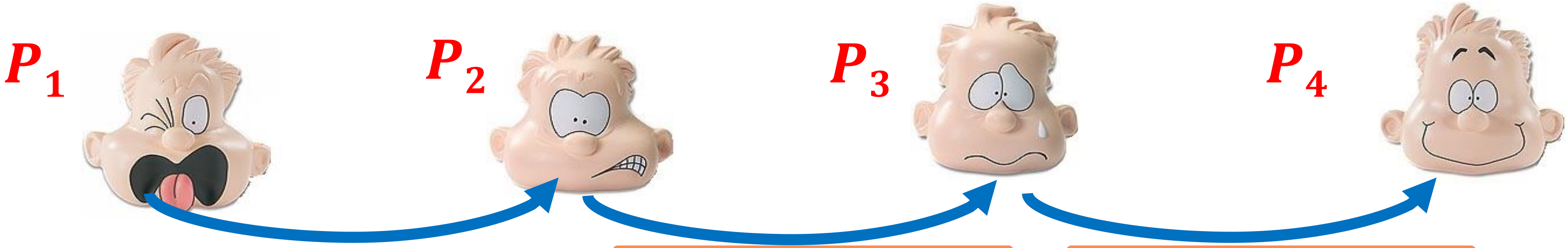
and so on. . .

Recursively:

level $\boldsymbol{i+1}$ recommendation:

A : *"you can trust that all **the level** **i** recommendations issued by **B**"*

Now, if:

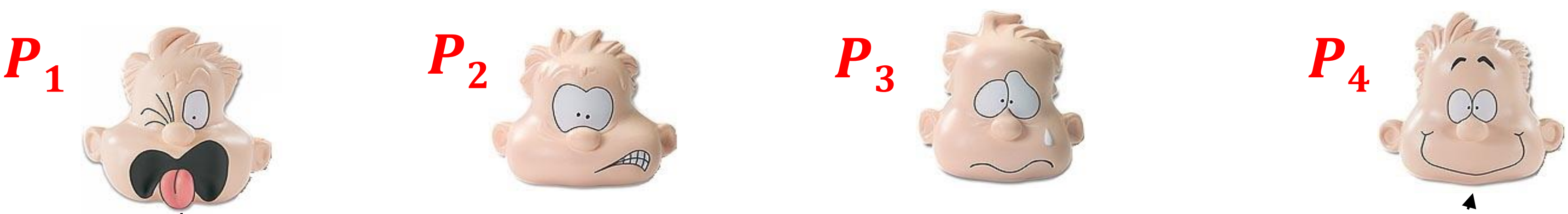


P_1 trusts in all the recommendations of level 2 issued by P_2

P_2 issues a recommendation of level 2 for P_3

P_3 issues a recommendation of level 1 for P_4

then



trusts the certificates issued by P_4

Of course the recommendations also need to be signed.

Starts to look complicated...

How is it solved in practice?

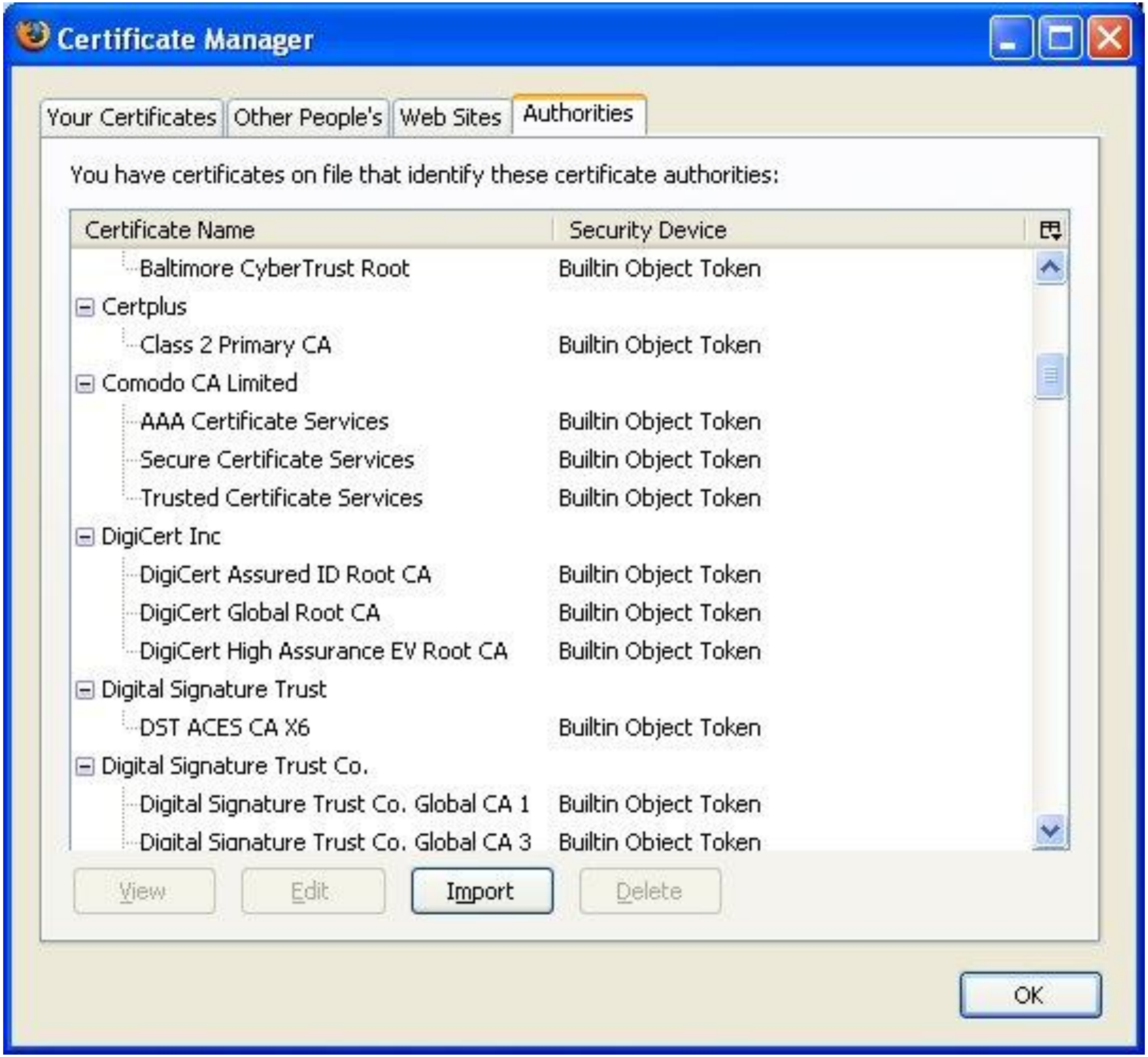
In popular standard is **X.509** the recommendation is included into a certificate.

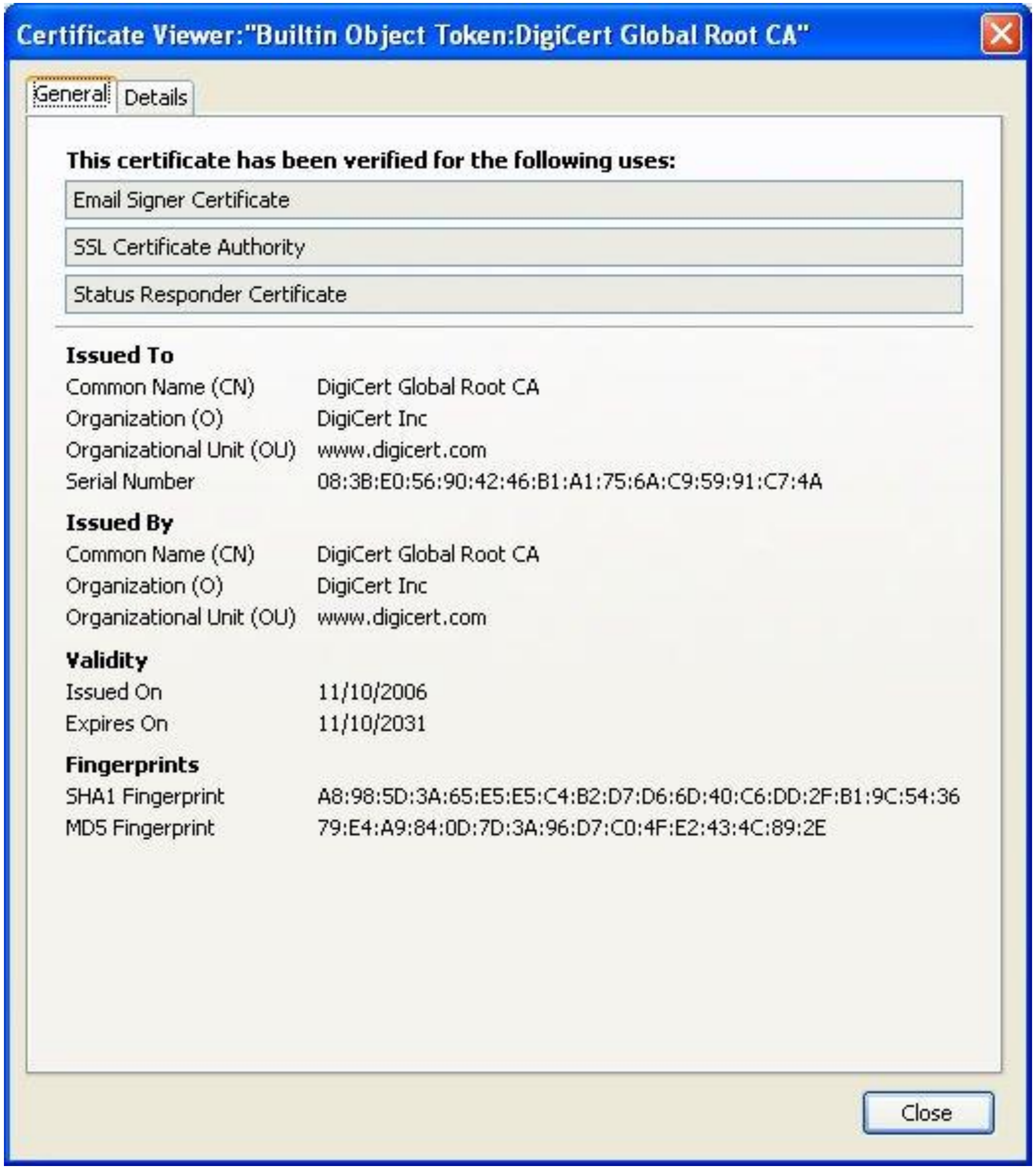
Here the level of recommendations is bounded using a field called ***basic constraints***.

X.509 is used for example in **SSL**.

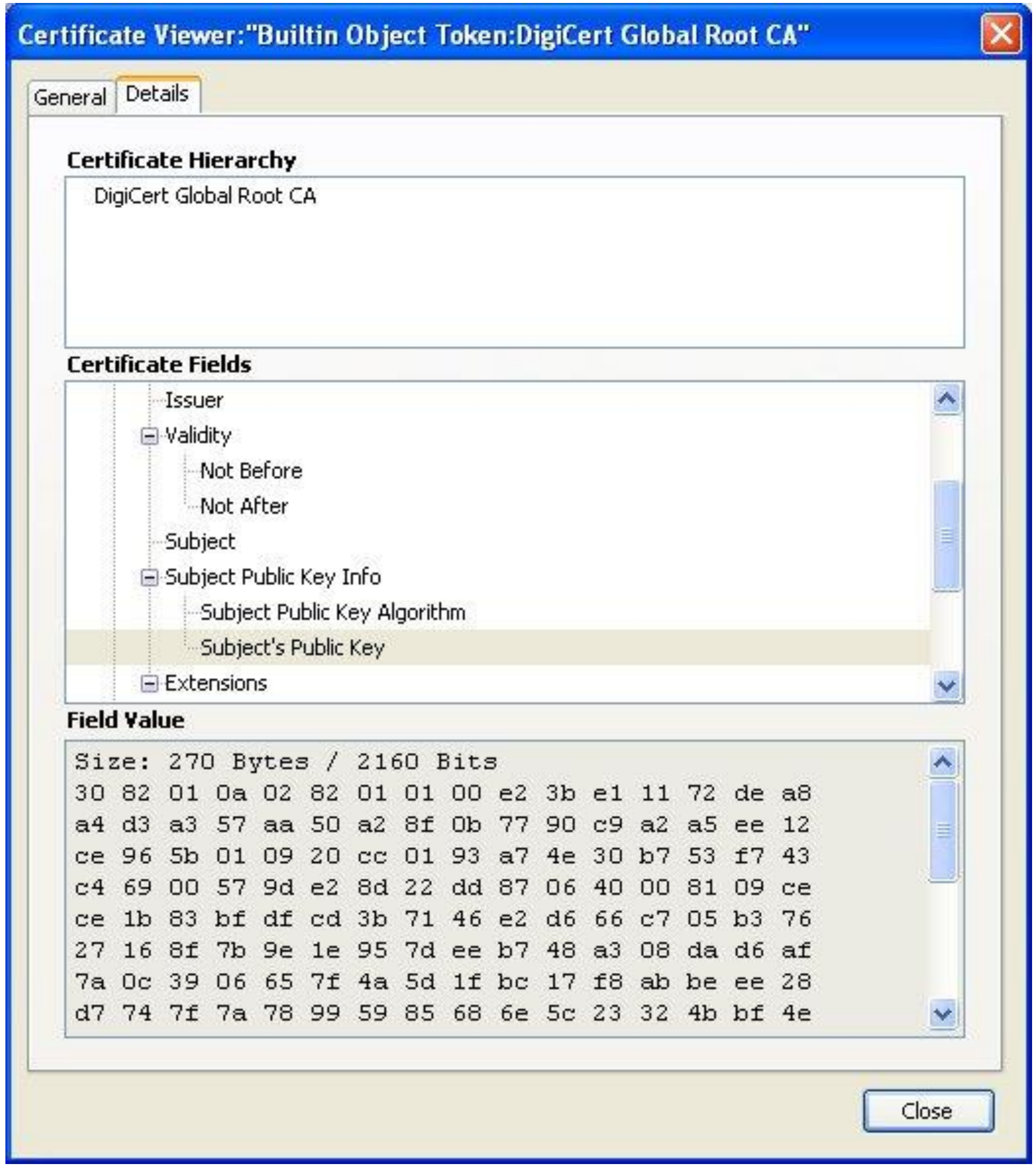
SSL is implemented is implemented in every popular web-browser.

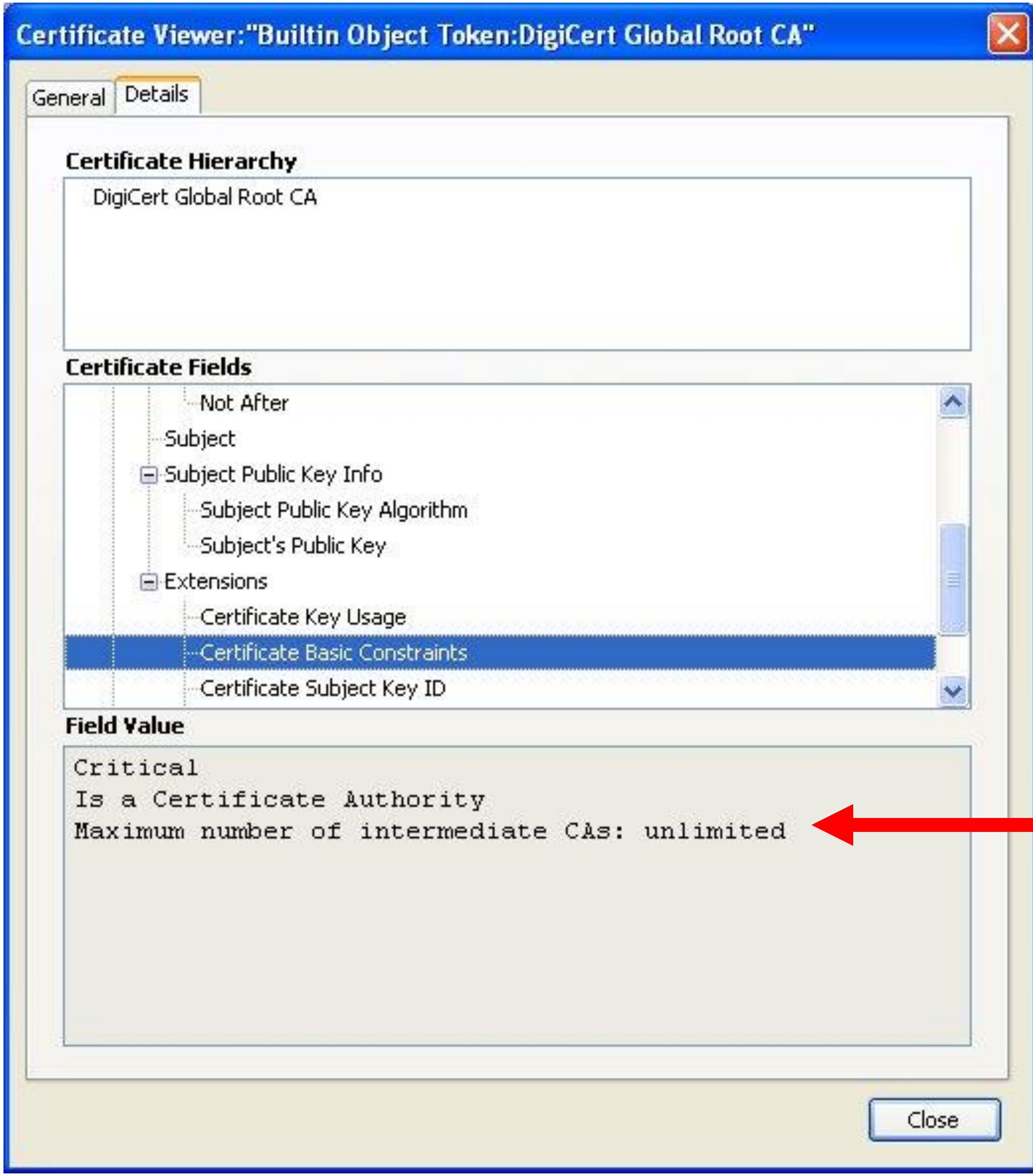
So, let’s look at it.


Certificate Manager
Your Certificates
Other People's
Web Sites
Authorities
You have certificates on file that identify these certificate authorities:
Certificate Name
Security Device
Baltimore CyberTrust Root
Builtin Object Token
Certplus
Class 2 Primary CA
Builtin Object Token
Comodo CA Limited
AAA Certificate Services
Builtin Object Token
Secure Certificate Services
Builtin Object Token
Trusted Certificate Services
Builtin Object Token
DigiCert Inc
DigiCert Assured ID Root CA
Builtin Object Token
DigiCert Global Root CA
Builtin Object Token
DigiCert High Assurance EV Root CA
Builtin Object Token
Digital Signature Trust
DST ACES CA X6
Builtin Object Token
Digital Signature Trust Co.
Digital Signature Trust Co. Global CA 1
Builtin Object Token
Digital Signature Trust Co. Global CA 3
Builtin Object Token
View
Edit
Import
Delete
OK


Certificate Viewer:"Builtin Object Token:DigiCert Global Root CA"
General
Details
This certificate has been verified for the following uses:
Email Signer Certificate
SSL Certificate Authority
Status Responder Certificate
Issued To
Common Name (CN) DigiCert Global Root CA
Organization (O) DigiCert Inc
Organizational Unit (OU) www.digicert.com
Serial Number 08:3B:E0:56:90:42:46:B1:A1:75:6A:C9:59:91:C7:4A
Issued By
Common Name (CN) DigiCert Global Root CA
Organization (O) DigiCert Inc
Organizational Unit (OU) www.digicert.com
Validity
Issued On 11/10/2006
Expires On 11/10/2031
Fingerprints
SHA1 Fingerprint A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
MD5 Fingerprint 79:E4:A9:84:0D:7D:3A:96:D7:C0:4F:E2:43:4C:89:2E
Close


Certificate Viewer:"Builtin Object Token:DigiCert Global Root CA"
General
Details
Certificate Hierarchy
DigiCert Global Root CA
Certificate Fields
Subject Public Key Algorithm
Subject's Public Key
Extensions
Certificate Key Usage
Certificate Basic Constraints
Certificate Subject Key ID
Certificate Authority Key Identifier
Certificate Signature Algorithm
Certificate Signature Value
Field Value
PKCS #1 SHA-1 With RSA Encryption
Close


Certificate Viewer:"Builtin Object Token:DigiCert Global Root CA"
General
Details
Certificate Hierarchy
DigiCert Global Root CA
Certificate Fields
Issuer
Validity
Not Before
Not After
Subject
Subject Public Key Info
Subject Public Key Algorithm
Subject's Public Key
Extensions
Field Value
Size: 270 Bytes / 2160 Bits
30 82 01 0a 02 82 01 01 00 e2 3b e1 11 72 de a8
a4 d3 a3 57 aa 50 a2 8f 0b 77 90 c9 a2 a5 ee 12
ce 96 5b 01 09 20 cc 01 93 a7 4e 30 b7 53 f7 43
c4 69 00 57 9d e2 8d 22 dd 87 06 40 00 81 09 ce
ce 1b 83 bf df cd 3b 71 46 e2 d6 66 c7 05 b3 76
27 16 8f 7b 9e 1e 95 7d ee b7 48 a3 08 da d6 af
7a 0c 39 06 65 7f 4a 5d 1f bc 17 f8 ab be ee 28
d7 74 7f 7a 78 99 59 85 68 6e 5c 23 32 4b bf 4e
Close

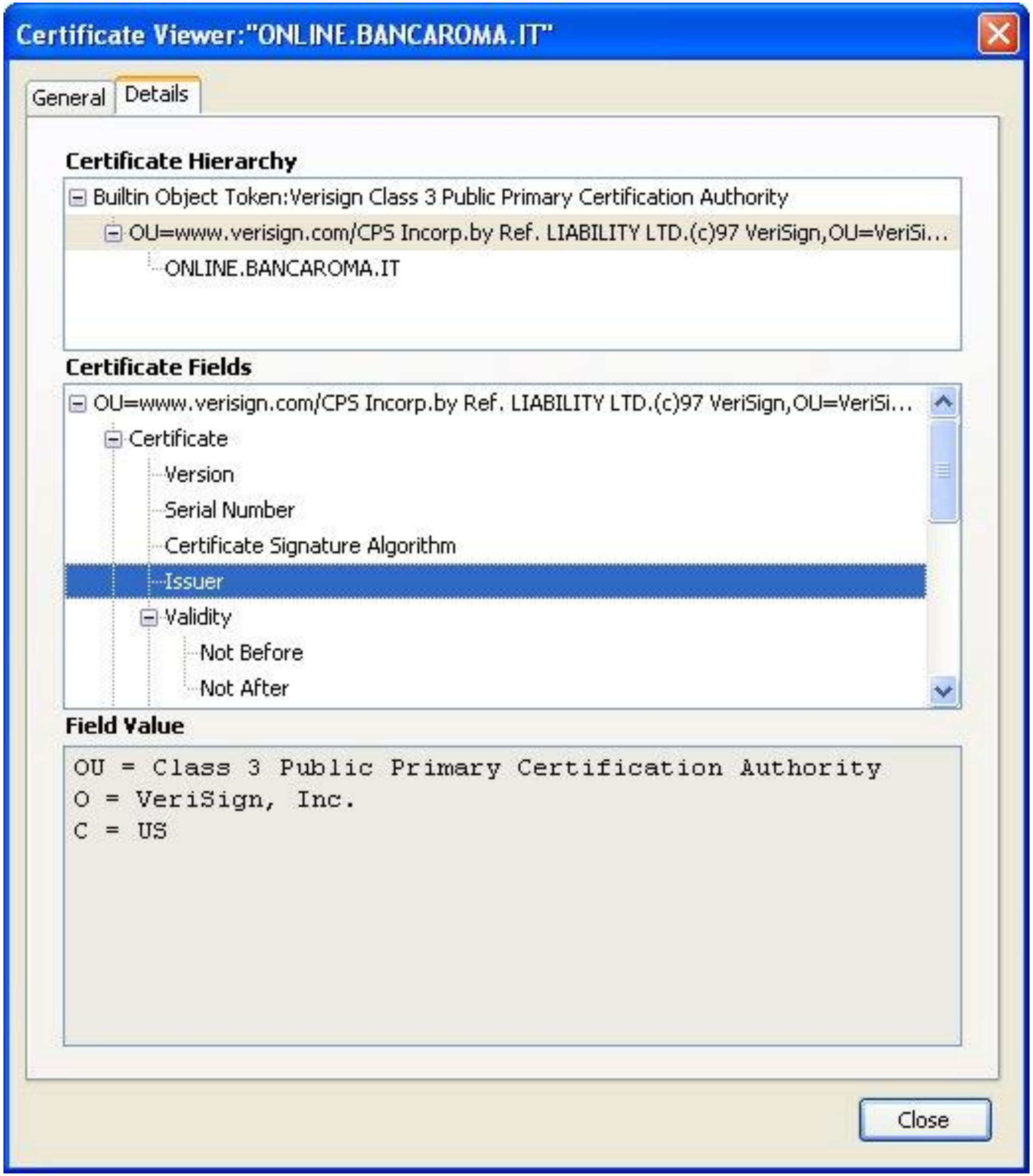


this field limits the recommendation depth
(here it's unlimited)

Concrete example

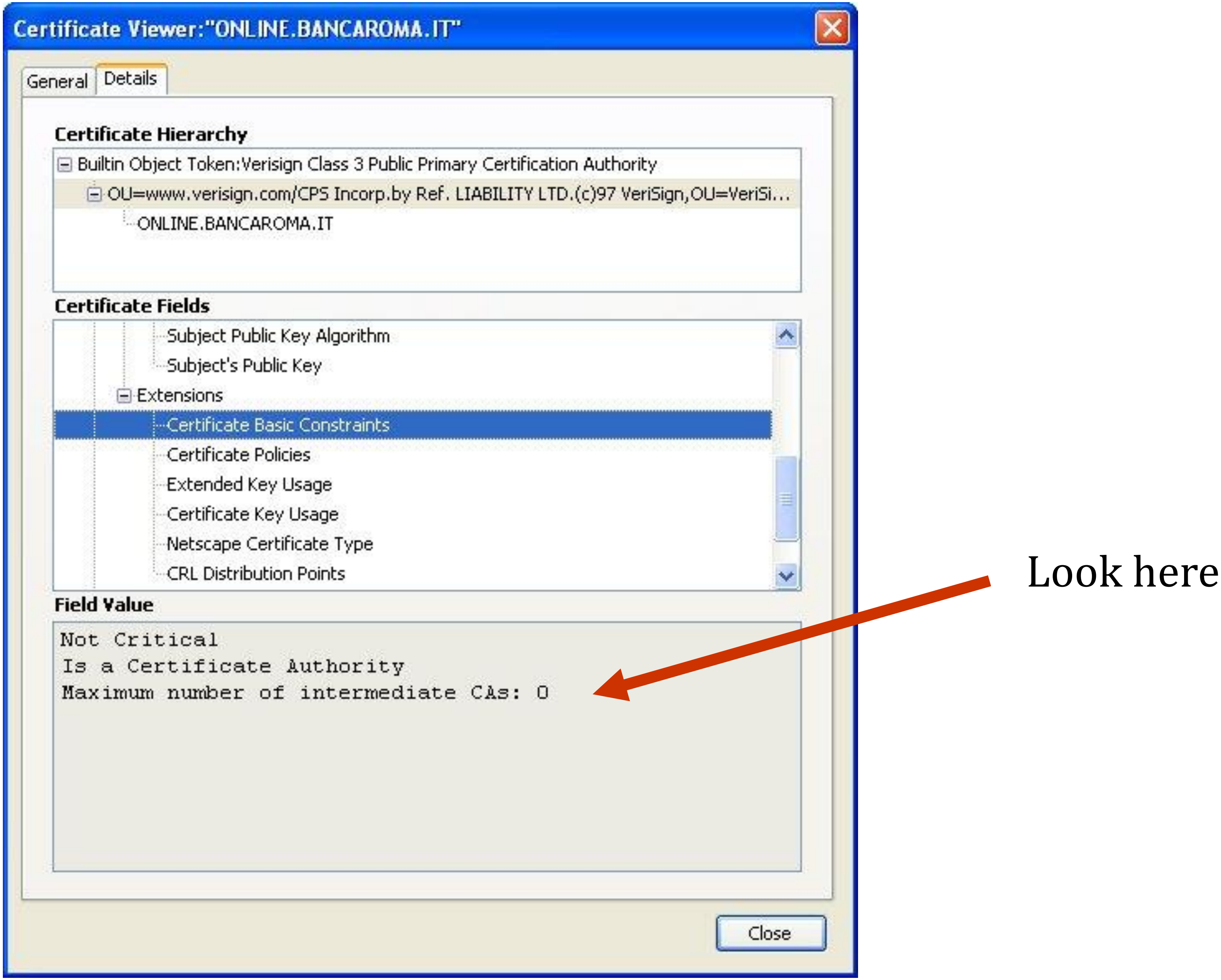
Let's go to the **Banca Di Roma** website

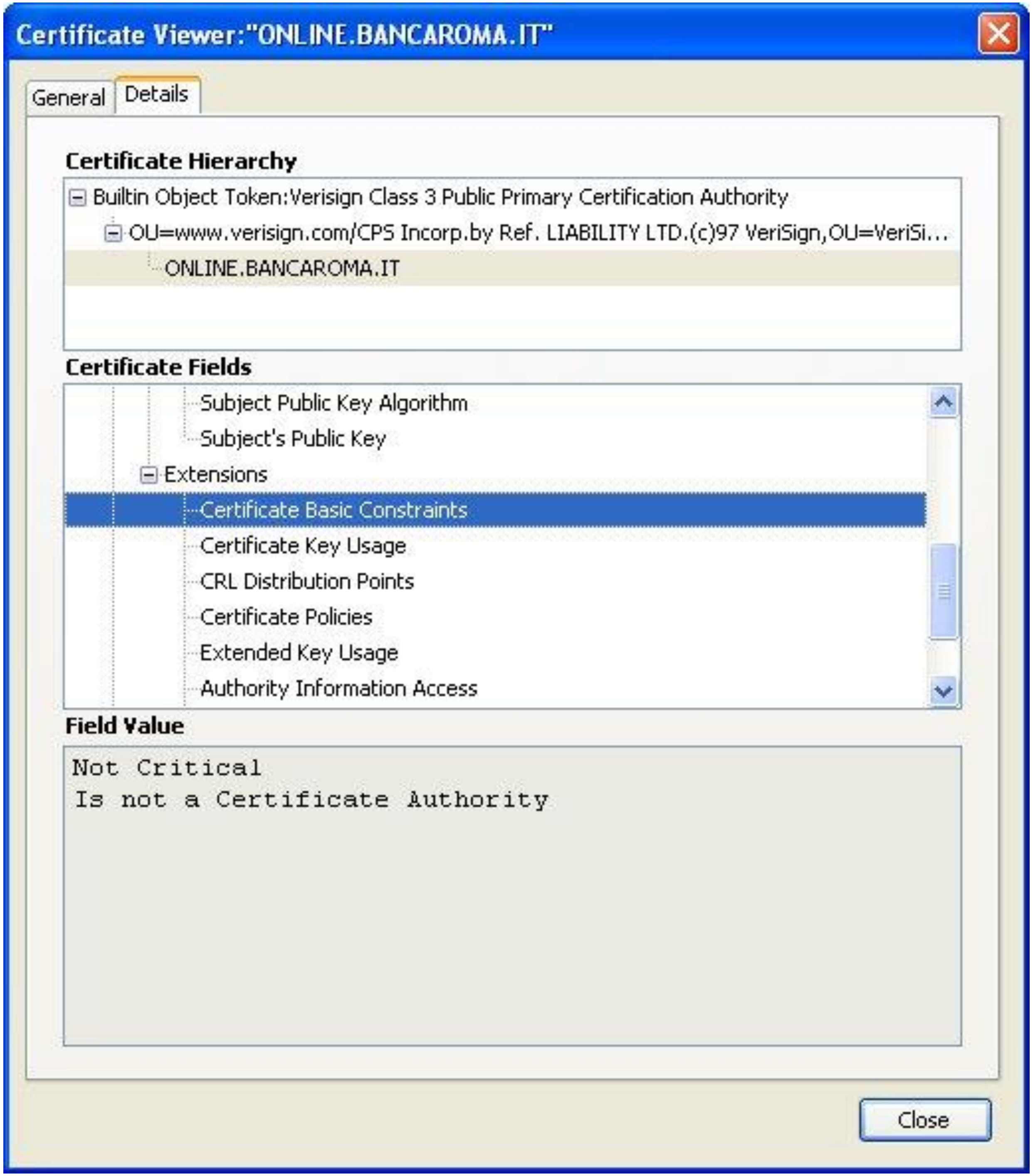
a certificate chain



the second certificate was signed by "**Verisign Primary Authority**" for "**Verisign Inc**".

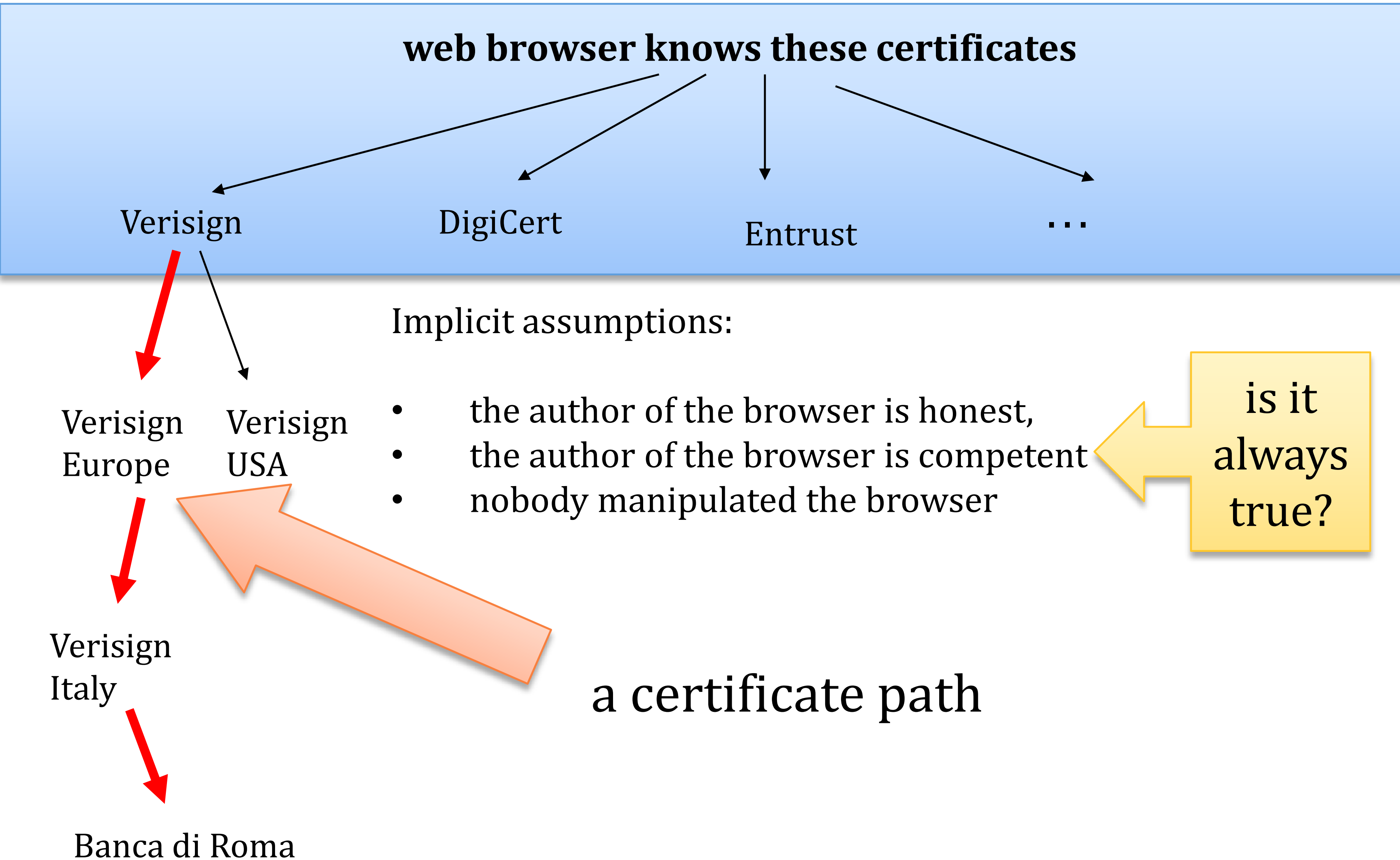
(it's not strange, we will discuss it)


Certificate Viewer:"ONLINE.BANCAROMA.IT"
General
Details
Certificate Hierarchy
Builtin Object Token:Verisign Class 3 Public Primary Certification Authority
OU=www.verisign.com/CPS Incorp.by Ref. LIABILITY LTD.(c)97 VeriSign,OU=VeriSi...
ONLINE.BANCAROMA.IT
Certificate Fields
Subject Public Key Algorithm
Subject's Public Key
Extensions
Certificate Basic Constraints
Certificate Policies
Extended Key Usage
Certificate Key Usage
Netscape Certificate Type
CRL Distribution Points
Field Value
Not Critical
Is a Certificate Authority
Maximum number of intermediate CAs: 0
Close
Look here



The third certificate was issued by **Verisign Inc.** for **Banca di Roma**

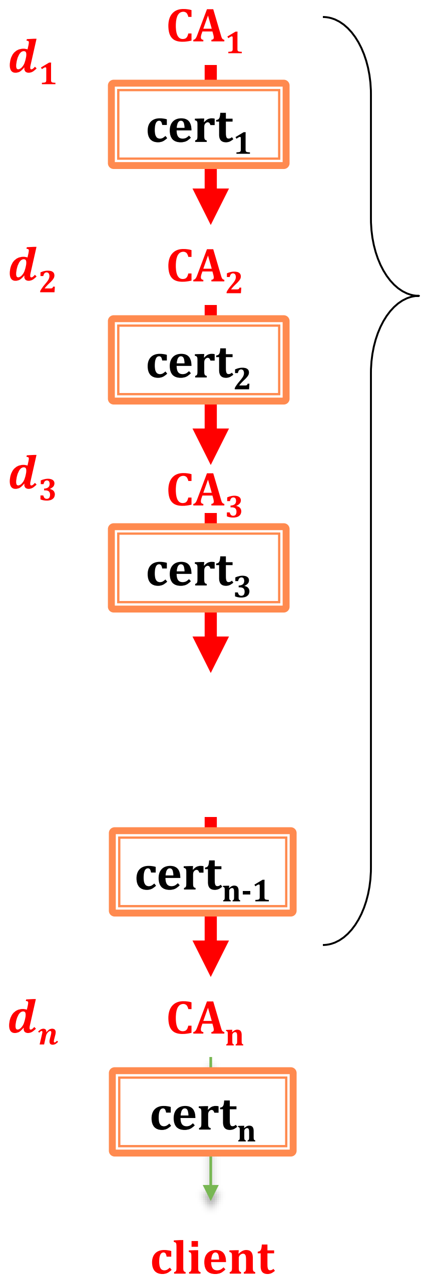
The typical picture



Implicit assumptions:

- the author of the browser is honest,
- the author of the browser is competent
- nobody manipulated the browser

is it always true?



All these certificates have to have a flag "**Is a Certification Authority**" switched on.

Moreover:

each **cert$_i$** has a number d_i denoting a maximal depth of certificate chain from this point (this limits the recommendation depth)

That is, we need to have:

$$d_i \geq n - i$$

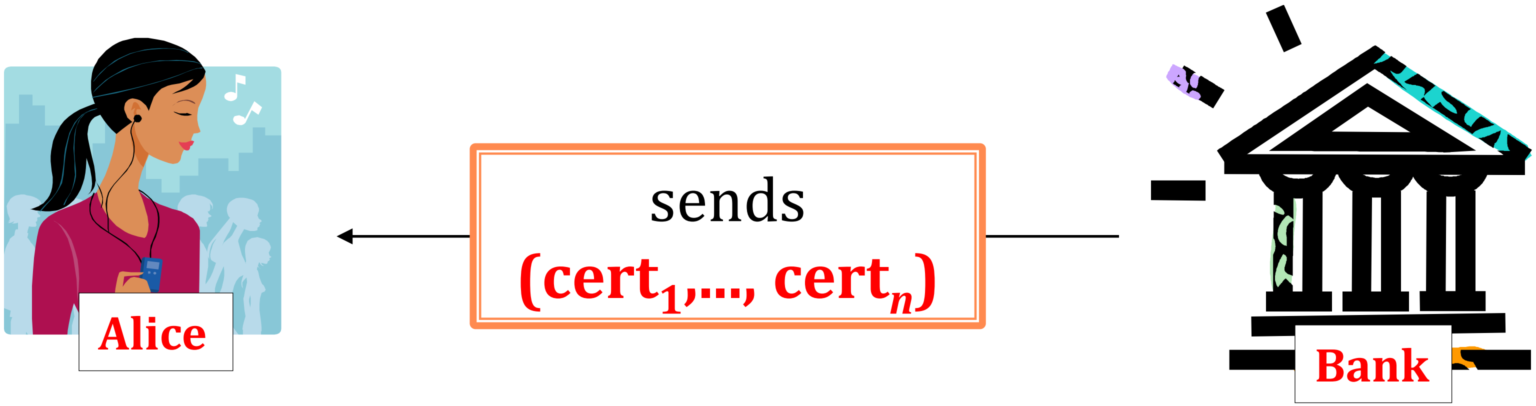
Is it so important to check it?

Yes!

For example: the last element in the chain can be anybody (who paid to **Verising** for a certificate).

For sure we do not want to trust the certificates issued by **anyone**.

So, what happens when a user contacts the bank?



If Alice's browser knows **$\mathbf{cert_1}$** it can verify the chain and read the public key of the bank from **$\mathbf{cert}_n$**.

Other information that the certificats contain

- information about the **signature algorithm**
- **validity** (dates)
- address of the **certificate revocation list**

Certificate Revocation List (CRL):
the list of revoked certificates
(need to access it before accepting the certificate)

Main problems with X.509

1. **Certificate revocation lists** work only **if you are online**.
2. **Revocation of root certificates** not addressed.
3. **CAs cannot restrict the domains** on which the subordinate CAs issue certificates.
4. It's enough into **hack one** of the popular CA's to impersonate any webpage.

Not only theoretical problems

DiGiNotar SSL certificate hack amounts to cyberwar, says expert

transactions,
se than

Google slaps Symantec for issuing fake web security certificates

by Jon Fingas | @jonfingas | October 29th 2015 At 8:22pm

Dan Kaplan, Executive Editor

Follow @dankaps

January 03, 2013

Google, Microsoft respond to fraudulent certificate

Share this article:

A Turkish certificate authority (CA) accidentally issued two intermediate, or chained, digital certificates, one of which was used by the holder to mimic legitimate websites.

A solution: “Public Key Pinning”:

- after the first connection **the web browser remembers the public keys on the certificate chain**,

- in each subsequent connection the browser **checks if the certificate chain is the same** as before.

Another problem

In practice:

the certificate **issuers do not check the identity** of their customers carefully

(due to the economical reasons).

Solution:
Extended Validation Certificates

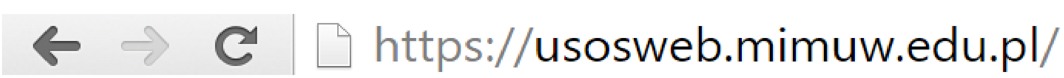
Some certificates are issued after **a more careful check**.

This is indicated in the web browser.

Example from Chrome:

EV certificate:

Non-EV certificate:



A different idea for a PKI

Namecoin

use Bitcoin's "blockchain" as a distributed register.

Another popular PKI (in the past)

Pretty Good Privacy (PGP) – every user can act as a certification authority.

Hence the name:

Web of Trust

Plan

1. Public key cryptography – an overview
2. The key management problem
    1. qualified signatures
    2. public key infrastructure
3. Identity-based cryptography

Identity based cryptography

Main idea:

the identifier **ID** of the user is it's public key.

(e.g. **ID** = user's email address).

message $\boldsymbol{M}$

$$\boldsymbol{C} = \mathbf{Enc}(\mathbf{ID}, \boldsymbol{M})$$

ID: alice@gmail.com

question:

What is the private key?

Solution

central authority

holds a **master secret key** **SK**

secret key $\mathbf{SK}_{\text{alice@gmail.com}}$:

Extract(SK, alice@gmail.com)

secret key of $\mathbf{SK}_{\text{bob@gmail.com}}$:

Extract(SK, bob@gmail.com)

secret key of $\mathbf{SK}_{\text{carol@gmail.com}}$:

Extract(SK, carol@gmail.com)

sent over a secure link

alice@gmail.com

bob@gmail.com

carol@gmail.com

How to decrypt

knows

$\mathbf{SK}_{\mathbf{alice@gmail.com}}$

message M

C = Enc(alice@gmail.com, M)

alice@gmail.com

calculates

M=Dec($\mathbf{Sk}_{\mathbf{alice}}$, C)

ID-based encryption

Main **advantage**:

- no need for an “infrastructure”

Drawbacks:

- users need to **trust an authority,**
- and they need to have a **secure link** to it,
- what about the **key revocation**?

ID-based encrypion

Proposed by **Adi Shamir** in **1984**.

(he only implemented the identity-based **signatures**)

First schemes were proposed by **Boneh** and **Franklin** (2001) and, independently **Cocks** (2001).

In **2002 Boneh** started a company

Voltage Security

that produces solutions based on his ID-based scheme.