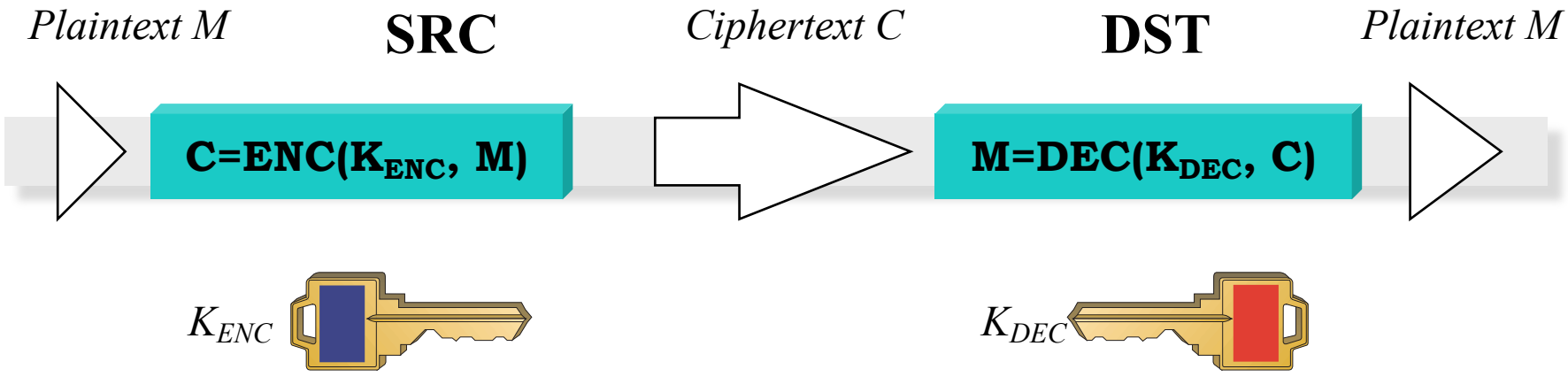
Introduction to Asymmetric Cryptography

Giuseppe Bianchi

Asymmetric cryptography

M	plaintext
C	ciphertext



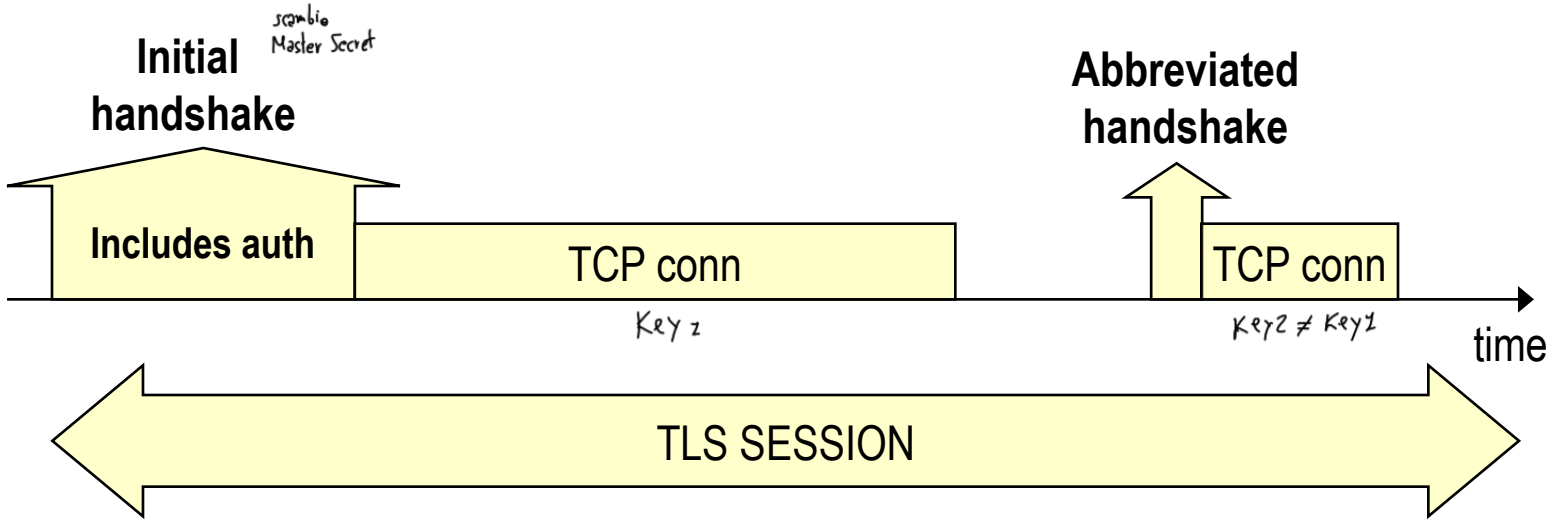
K_{ENC}	encryption key
ENC(.)	encryption algorithm

$K_{DEC} \neq K_{ENC}$	decryption key
DEC(.)	decryption algorithm

Encryption and decryption keys are obviously linked…
But CANNOT be determined from each other
(unless you know 'more')

Practical usage model #1: HTTPS/TLS-style

- **First phase: handshake**
  - ⇨ Signalling phase, sets up **shared secret** using asymmetric cryptography
- **Second phase: data transfer**
  - ⇨ Derives **symmetric encryption and message authentication keys** from shared secret derived in first phase *(use KDF, more later)*
- **Further phases: rekeying + data transfer**
  - ⇨ «**refresh**» symmetric encryption and message authentication **keys** still using shared secret derived in first phase *(use KDF with different nonces)*

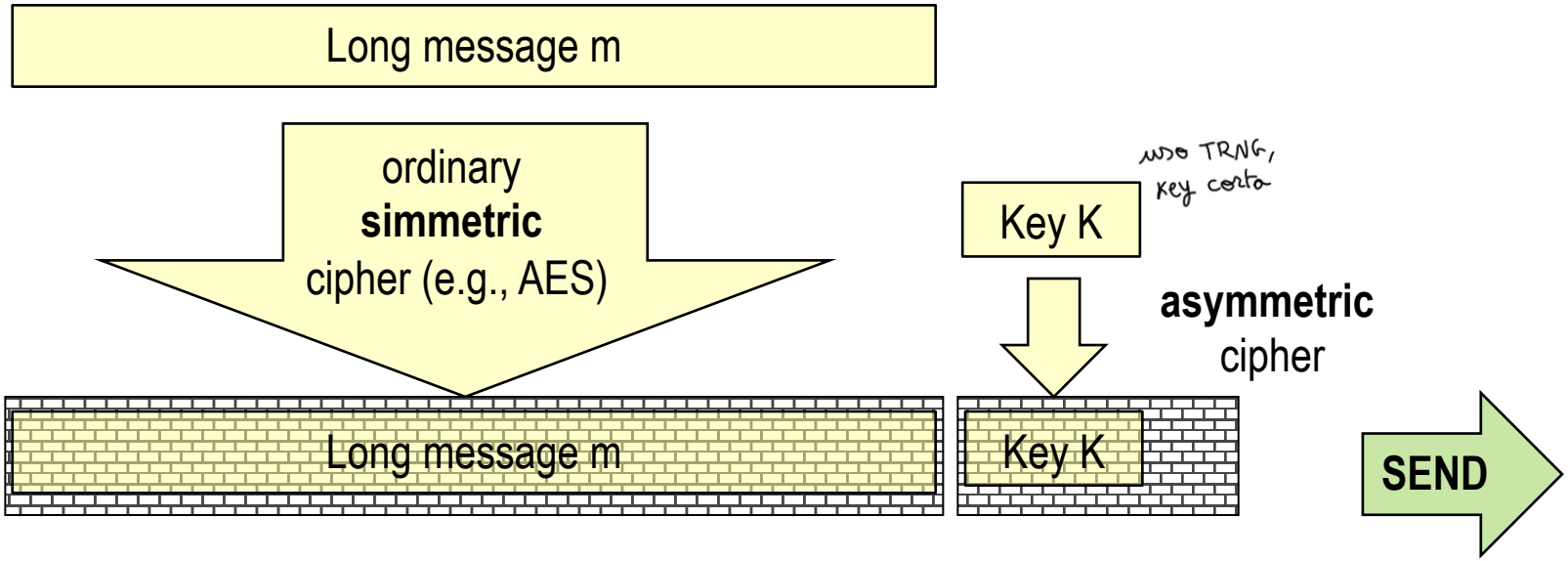


Practical usage model #2: Hybrid Encryption

➔ Generate (random) symmetric key K

➔ Use key K and symmetric cipher to encrypt (long) data

➔ Encrypt K with ASYMMETRIC cipher, and send it with message



(ECIES: recently introduced in 5G subscriber Identity Concealment – see 5G SUCI)

(usabile in 2 modi)

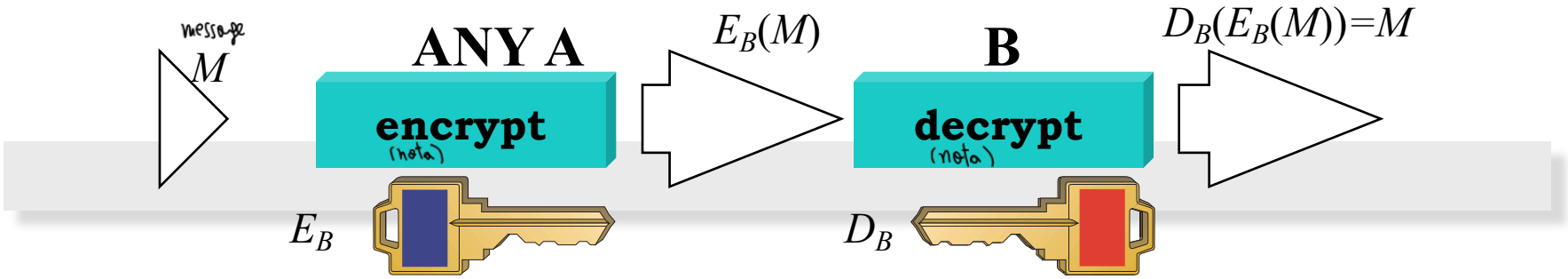
PubKey crypto: two ways to use it

↳ NON VALIDO ∀ schema, ma faremo finta di sì. (in RSA è così)

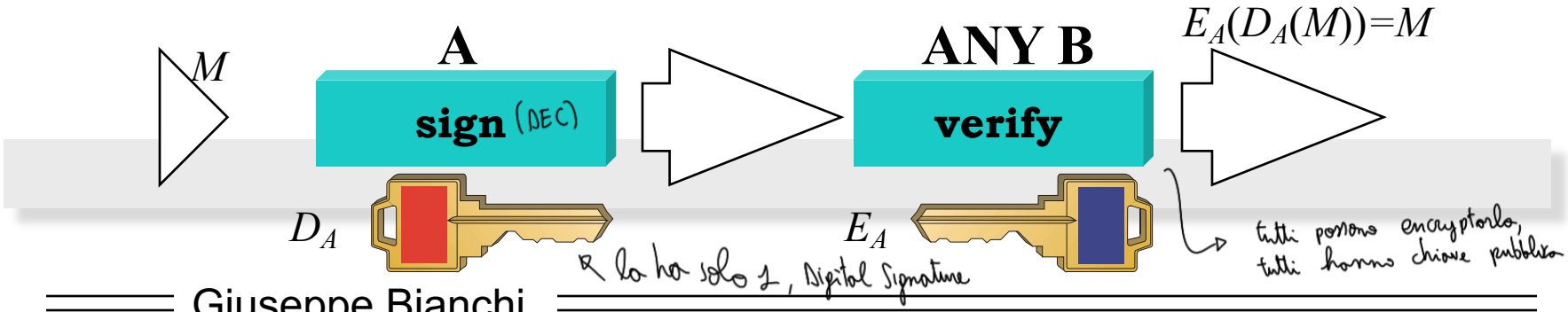
ENC and DEC are inverse operations: DEC(ENC(M))=M

assume commutativity: also ENC(DEC(M))=M

Public key encryption: anyone can encrypt M, but only B can decrypt



Digital signature: DUAL approach! Only A has K_{DEC} → can apply DEC (sign) but anyone else has K_{ENC} → can apply ENC (verify)!



digital signature (concettuale)

(encryption), nota

Public Key PK

CORRELATE

Private Key SK

(decryption), segreta

può essere lungo

M = «780 EUR per te»

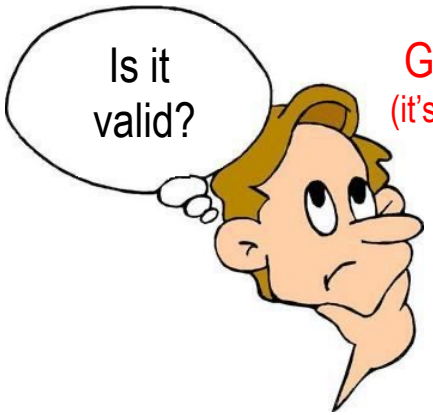
SIGNATURE

Crypto hash, short, è un TAG

H(M)

Use SK

è uno decrypt che usa SK, ma non è uno vero decrypt!



Get PK (it's public!)

M = «780 EUR per te»

H(M)

Use SK

applico PK, estraggo H(M) e applico Hash al msg per vedere se corrisponde. INVERTO IL TAG

digital signature

Private Key
SK

Public Key
PK

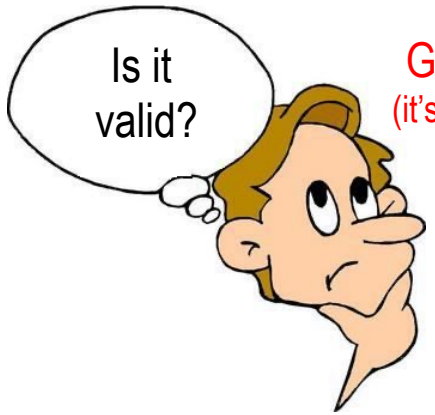
hash, oltre ad essere short, garantisce che non venga rotto. GARANTISCE SICURITY.

M = «780 EUR per te»

H(M) ①

Use SK ②

è un TAG



Get PK
(it's public!)

H(M) ? = H(M)

M = «780 EUR per te»

H(M)

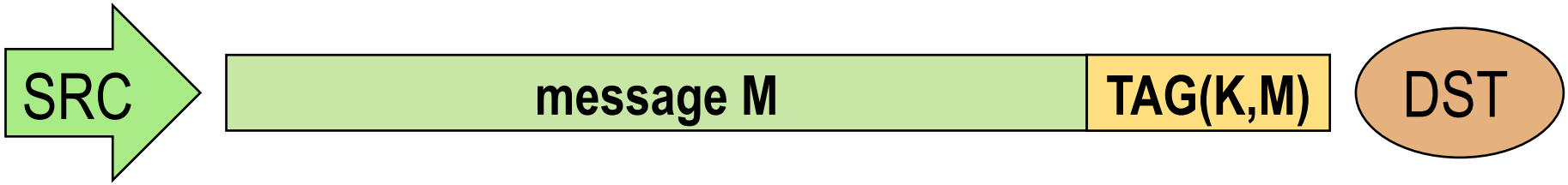
Use SK

Use PK to «invert» tag

Do they match?
If Y, message is valid

Digital signature: message integrity with source authentication

➔**Message integrity → auth tag**



➔**Symmetric MAC (e.g. HMAC):**

⇨K = symmetric key: shared by SRC and DST

⇨DST **must know SRC key = same key K** to verify

➔**Digital Signature**

⇨K = SRC private key

⇨DST **just needs to know SRC PubKey** to verify!

⇨Consequence: non-repudiation a.k.a. source authentication

Anymmetric Cryptography: Basic Algorithms

The Pioneers

IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-22, NO. 6, NOVEMBER 1976

New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER,

1976: invention of asymmetricy crypto
First-ever asymmetric algorithm:
Diffie-Hellman KEY AGREEMENT

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R.L. Rivest, A. Shamir, and L. Adleman*

1977: first ever public key encryption & digital signature: RSA crypto system

Basic ingredient: HARD (asymmetric) PROBLEM

- **Problem computationally EASY in one direction, but computationally HARD in the opposite direction**

- **Diffie-Hellman:** Se $y = 3^x \longleftrightarrow x = \log_3 y$, entrambe facili, poiché MONOTONE.
  - ⇨ DISCRETE LOGARITHM PROBLEM IN PRIME FIELDS
    - → Given large prime p, g, x, **compute** $y = g^x$ **mod p** → easy [ANCHE con P grande]
    - → Given $y = g^x$ mod p **compute** $x = \mathbf{DLog}_g(y)$ **mod p** → hard!!
      ("restringo", modulo num. primo)

- **RSA:**
  - ⇨ FACTORING A PRODUCT OF TWO LARGE PRIMES
    - → Given p and q large primes, **compute N=p q** → easy
    - → Given N, **find factors p, q** → hard

CAPITA SEMPRE ALL'ESAME

Why modular exponentiation is easy

"SQUARE & MULTIPLY" ALG.

- **Goal: compute g^{1437} mod p** (NUMERO qualunque (es 3))
- **Express x in bits**
  - $1437_{10}=10110011101_2$
- **list bits from lsb to msb** (meno importante → più importante)
- **First line: initialize**
  - Column Square = g
  - Column Multiply = 1 (if b=0) or g (if b=1)
- **Start from 2° lsb to msb**
  - For every bit
    - →Square;
    - →If 1: multiply to result

$b: lsb -> msb$	Square	Multiply
1	g	$1 \cdot g = g$
0	$(g)^2$	•
1	$(g^2)^4$	$1 \cdot g^4 \cdot g = g^5$
1	$(g^4)^2$	$1 \cdot g^8 \cdot g^5 = g^{13}$
1	g^{16}	$1 \cdot g^{16} \cdot g^{13} = g^{29}$
0		
0		
1		
1		
0		
1		

Why modular exponentiation is easy

- **Goal: compute g^{1437} mod p**
- **Express x in bits**
  - $1437_{10}=10110011101_2$
- **list bits from lsb to msb**
- **First line: initialize**
  - Column Square = g
  - Column Multiply = 1 (if b=0) or g (if b=1)
- **Start from 2° lsb to msb**
  - For every bit
    - →Square;
    - →If 1: multiply to result
- **Complexity:**
  - O(nbit) squares
  - O(nbit/2) multiplications

16 TOTALI, non 1436

10 op. 6 op

$b: lsb->msb$	*Square*	*Multiply*
1	g	$\times g \rightarrow g$
0	g^2	.
1	$\left(g^2\right)^2 = g^4$	$\times g^4 \rightarrow g^5$
1	$\left(g^4\right)^2 = g^8$	$\times g^8 \rightarrow g^{13}$
1	$\left(g^8\right)^2 = g^{16}$	$\times g^{16} \rightarrow g^{29}$
0	$\left(g^{16}\right)^2 = g^{32}$	.
0	$\left(g^{32}\right)^2 = g^{64}$	.
1	$\left(g^{64}\right)^2 = g^{128}$	$\times g^{128} \rightarrow g^{157}$
1	$\left(g^{128}\right)^2 = g^{256}$	$\times g^{256} \rightarrow g^{413}$
0	$\left(g^{256}\right)^2 = g^{512}$	.
1	$\left(g^{512}\right)^2 = g^{1024}$	$\times g^{1024} \rightarrow g^{1437}$

nbit + nbit/2 = 1.5(nbit)

(1.5) $\log_2(x)$

g^{1437} molte più operazioni

No algorithm such as this for the opposite problem!! That's why it is hard!

Quando ho "1" in lsb ho più calcoli → più tempo → più energia → rischio cose. Devo "peggiorare" le prestazioni

Esempio: $g^{167} \bmod 11$

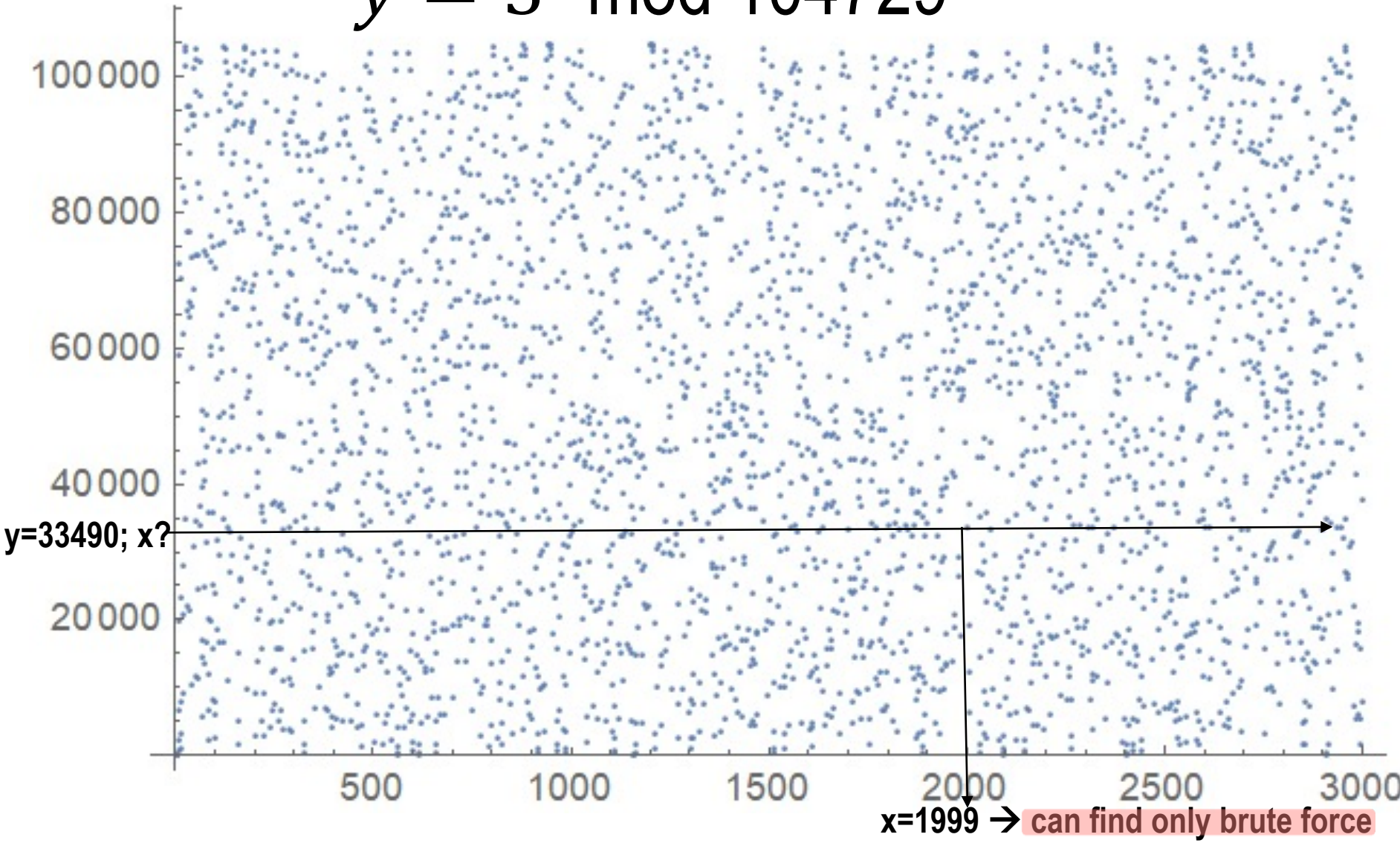
$g = 3$

b	Square	Multiply
1	3	3
0	$3^2 = 9$	•
1	$(3^2)^2 = 81 \bmod 11 = 4$	$(1 \cdot 4 \cdot 3) \bmod 11 = 1$
1	$(4^2) = 16 \bmod 11 = 5$	$(1 \cdot 5 \cdot 1) \bmod 11 = 5$
1	$(5^2) = 25 \bmod 11 = 3$	$(1 \cdot 3 \cdot 5) \bmod 11 = 4$
0	$(3^2) = 9$	•
0	4	•
1	$(4)^2 \bmod 11 = 5$	$(1 \cdot 5 \cdot 4) \bmod 11 = 9$
1	$(5)^2 \bmod 11 = 3$	$(1 \cdot 3 \cdot 9) \bmod 11 = 5$
0	$(3^2) \bmod 11 = 9$	•
1	$(9)^2 \bmod 11 = 4$	$(1 \cdot 4 \cdot 5) \bmod 11 = 9$

somma di potenze

And why DLOG is hard

$$y = 3^x \bmod 104729$$



Diffie-Hellman Key Agreement

- **Diffie-Hellman protocol, 1976**
  - Main crypto breakthrough in the last century
  - Invention of asymmetric cryptography
  - Though did NOT solve the PubKey cryptosystem problem
    - RSA solved it in the next year (1977)

- **What DH is: KEY AGREEMENT protocol**
  - **How to securely setup a shared secret (i.e., a symmetric key) at both ends, by only exchanging public values**
    - **i.e., WITHOUT having a secure channel**

- **Hard problem used in DH: DLOG**

Discrete

Vediamo: "ANONYMOUS DH" or "VANILLA DH", non funziona proprio così

Diffie-Hellman Key Agreement

non arbitrario ↑ g, p parametri pre-comunicati

$y = g^x \bmod p$ easy

$x = D\log_g(y)$ hard → logaritmo nel gruppo numeri primi

facile da computare

Alice

genero *Random* $x \approx$ PRIVATE DATA

riceve (g^y), bozza "g"

Alice invia:

SEND $[g^x] \bmod p$

PUBLIC coefficiente associato ad "x"

SEND $g^y \bmod p$

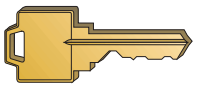
Bob

genero *Random* y

riceve (g^x)

COMPUTE K=$(g^y)^x \bmod p$

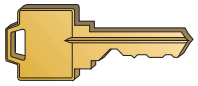
K=g^{xy}: SAME KEY at both ends!



COMPUTE K=$(g^x)^y \bmod p$

uguali

K=g^{xy}: SAME KEY at both ends!



Wait, I see all the exchange!!

ottiene g^{x+y}, NON g^{xy}

From g^x,g^y → hard to compute g^{xy}

Attacher must necessarily invert DLOG for one of the two terms: computationally hard!!

DH functional limitation

dove stanno ENC, DEC? fa solo Key agreement, cioè genero stesso segreto per Alice e Bob, null'altro.

➔ **Does NOT implement neither a public key cryptosystem**

⇨ You cannot «transfer» data encrypted with the receiver's PK

➔ **nor a digital signature!**

⇨ You cannot sign data using your SK

➔ **1977: RSA to the rescue!**

The RSA algorithm

- **Rivest, Shamir, Adleman, 1977**
- **Patented until 2000**
- **Widely deployed! Ultra-well known**

(stessa op) enc() = dec() ma ho PK e SK

COMMUTATIVO

- Preso $N = p \cdot q$ $(77 = 11 \cdot 7)$
- Preso $msg < N$, se faccio
  - $(MSG)^{PK} \bmod N = CT$
  - $(CT)^{PK} \bmod N$ da altro valore!
  - $(CT)^{SK} \bmod = N$

* msg trasformato in numero
- VANILLA RSA non garantisce semantic security

- **HARD (asymmetric) PROBLEM USED: prime number factoring**
  - Given $N = p \times q$ (p, q very large primes)
    HARD to «factor» N (= find the two values p, q)

composite number

risulto per...

- **Encrypt/decrypt: modular exponentiation**
  - ENC(message, {N, PK}) → ciphertext = $(message)^{PK} \bmod N$
  - DEC(ciphertext, {N,SK}) → message = $(ciphertext)^{SK} \bmod N$

SCORRELATI

2 valori; dato $msg < N = p \cdot q$ scorrelati tra loro. Devo vedere come trovarli.

- **Can support both encryption and digital signature**

The principle behind RSA

$m^x \bmod N$ is a periodic function

$$3^x \bmod 10 = \{3,9,7,1,3,9,7,1, \ldots\}$$

periodo è 4! in 9^x è 2, ma posso estenderlo

$$7^x \bmod 10 = \{7,9,3,1,7,9,3,1, \ldots\}$$

$$9^x \bmod 10 = \{9,1,9,1,9,1,9,1, \ldots\}$$

➔ **Euler Theorem (a.k.a. Euler-Fermat Theorem, a.k.a. Euler's Totient Theorem):**

⇨Explicit rule to compute the (maximum) period of $m^x \bmod N$

⇨Specifically, max period = **$\Phi(N)$ = Euler's Totient function**

→More formally: $9 = 3 \cdot 3$, $10 = 2^2 \cdot 5$ ⇝ 9, 10 coprime

if **m coprime with N**, i.e., GCD(m,N)=1,

then $\boldsymbol{m^{\Phi(N)} \bmod N = 1}$ → $9^4 \bmod (10) = 1$ = ultimo valore del periodo,

REGOLE:

Computing Euler's Totient Function

DA SAPERE

p prime → $\Phi(p) = p-1$

$p = 7, \Phi(7) = 6 \quad 3^6 \bmod 7 = 729 \bmod 7 = 1$

p,q primes → $\Phi(p \cdot q) = \Phi(p) \cdot \Phi(q) = (p-1)(q-1)$ **RSA case**

$N = 10, \Phi(10) = 4 \rightarrow 3^4 \bmod 10 = 81 \bmod 10 = 1$

SKIPPABILI

Prime p^k → $\Phi(p^k) = \Phi(p) \cdot p^{k-1} = (p-1)\, p^{k-1}$

$N = 25, \Phi(25) = \Phi(5^2) = 4 \cdot 5$

$\rightarrow 3^{20} \bmod 25 = 3486784401 \bmod 25 = 1$

general case: $N = p_1^{k_1} \cdot \cdots \cdot p_z^{k_z}$

$\Phi(N) = (p_1 - 1)p_1^{k_1 - 1} \cdot \ldots \cdot (p_z - 1)p_z^{k_z - 1}$

$N = 100, \Phi(100) = \Phi(5^2\, 2^2) = 4 \cdot 5 \cdot 1 \cdot 2$

$\rightarrow 3^{40} \bmod 100 = 12157665459056928801 \bmod 100 = 1$

Consequence of «periodicity»

IMPORTANTE !!!!

$m^x \bmod N$ is a periodic function **with period $\Phi(N)$**

$2^x \bmod 11 = \{2,4,8,5,10,9,7,3,6,1\}$ $\quad \Phi(11)=10$

$8 = 2^3$, $9 = 2^6$, $7 = 2^7$, $1 = 2^{10}$

Compute 9·7 mod 11=?

- Using multiplications mod 11 → 9·7 mod 11= 63 mod 11 = 8
- Alternative approach: since $9=2^6 \bmod 11$ and $7=2^7 \bmod 11$:

  $9 \cdot 7 \bmod 11 = (2^6 \cdot 2^7) \bmod 11 = 2^{6+7} \bmod 11 = 2^{13} \bmod 11 = \dots$

  $\dots = 2^{10+3} \bmod 11 = 2^3 \bmod 11 = 2^{13 \bmod 10} \bmod 11$

  → arithmetic @ exponent = modulo Φ(N), not N!!!

N base
Φ(N) esponente
!!

Consequence: $m^x = m \bmod N$ if $x = 1 \bmod \Phi(N)$

RSA construction

- ⇨Generate two LARGE primes ***p***, ***q*** **(must remain SECRET!)**
- ⇨Compute *RSA module*: $N = p \times q$ (is public!)
- ⇨Compute $\Phi(N) = (p - 1)(q - 1)$ **(must remain SECRET!)**
- ⇨Generate public key $1<e<\Phi(N)$ e must be coprime with $\Phi(N)$ (se 'e' primo perfetto) ← primi tra loro
- ⇨Generate private key ***d*** such that $e \times d = 1 \bmod \Phi(N)$

possible only if e invertible mod $\Phi(N)$ → that's why e coprime with $\Phi(N)$

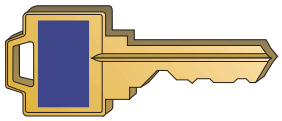
Security assumption: given *N*, must be hard to find factors p, q
- → **hence hard to find $\Phi(N)$**
- → **an without $\Phi(N)$ hard to compute *d* from *e***

$e \cdot d = 1 \bmod \Phi(N)$

$(M^e)^d = M^{ed} = M^1 = M$

$(M^e) = enc$; $(C^d) = decr$

scelto da me → (***N***, ***e***) → encrypt, RSA PUBLIC



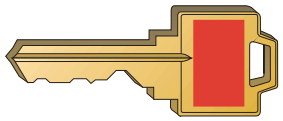
Pubic key

MAI RIUSARE "N"
Se rigenero 'e'

Private Key

nessuno me la da, è mia personale!

d → DECRYPT



$2^x \bmod 11 = \{2, 4, 8, 5, 10, 9, 7, 3, 6, 1\}$

$4^{-1} \bmod 11$? $x : 4 \cdot x = 1 \bmod 11$

$x = 3$, infatti $12 \bmod 11 = 1$

3 è l'inverso di 4 per mod 11

NB: trovo x se il numero che considero è coprimo del modulo.

$22^{-1} \bmod 77$ NON invertibile!

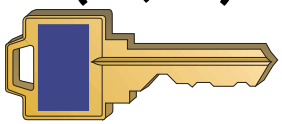
$77 = 11 \cdot 7$, 22 multiplo di 11

Nessuno ci da N, lo prendiamo NOI, altrimenti vulnerabili

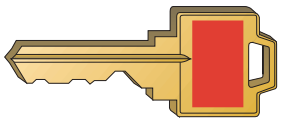
RSA construction

appena generate, butto N, non serve

Public key (***N***, ***e***)



Private Key ***d***



NOT semantic secure,
No IND_CPA secure

Encrypt: ENC(M): $C = M^e \bmod N$

Decrypt: DEC(C): $M = C^d \bmod N =$

$= (M^e)^d \bmod N =$

$= M^{ed \bmod \phi(N)} \bmod N =$

$= M^1 \bmod N = M$

Why RSA works? Trapdoor function!

Decryption challenge: given *N*, *e*, encrypted msg m^e
compute decryption key *x* s.t. $(m^e)^x \bmod N$ = m

non deve poter trovare 'x'

Normally: HARD problem
→ exponential complexity (brute force on possible values x)

Becomes EASY when you know Φ(N) TRAPDOOR
→ Equation $e \cdot x \bmod \Phi(N)$ can be solved in polynomial time

$x = e^{-1} \bmod \Phi(N)$
modular Inverse← Extended Euclidean Algorithm

Reminder: Ext Euclidean algo

La estesa

➔ **GCD[51,11]=1 coprime** ("1" è divisore più grande)

➔ **Find a,b s.t. 51 a + 11 b = 1**

a	b	val	rem	
1	0	51		
0	1	11	51/11=[4]	Subtract 4x
1	-4	7 = 51-11·4	1 (11/7)	Subtract 1x
-1	5	4 = 11-7	1	Subtract 1x
2	-9	3 = 7-4	1	Subtract 1x
-3	14	1 = 4-3	=	

Larger 1°

Smaller 2°

1 - 4·0

0 - 4·(1)

0 - 2 =

1 - (-4)·1

integer

51x(-3)+11x(14)=1

$51a + 11b = 1$

a	b	val	rem
1	0	51	
0	1	11	$\frac{51}{11} = 4$
$1 - 4 \cdot 0 = 1$	$0 - 4 \cdot 1 = -4$	7	$\frac{11}{7} = 1$
$0 - 1 \cdot 1 = -1$	$1 - 4(-1) = 5$	4	$\frac{7}{4} = 1$
$1 - 1 \cdot (-1) = 2$	$-4 - 1(5) = -9$	3	$\frac{4}{3} = 1$
$-1 - 1(2) = -3$	$5 - 1(-9) = 14$	1	

a

b

condizione iniziale!

, faccio linea 1 − 4 · linea 2

$51 - 11 \cdot 4$

$\frac{51}{11} = 4$ resto 7

$\frac{11}{7} = 1$ resto 4, faccio linea 2 − 1 · linea 3

$\frac{7}{4} = 1$ resto 3, linea 3 − 1 linea 4

$\frac{4}{3} = 1$ resto 1, linea 4 − 1 linea 5

Esempio esame

RSA con modulo $N = 143$ e chiave pubblica $e = 103$

1) trova $\phi(N)$, trova 'd' (decryption) con Extended Euclidean

How to compute RSA inverse

- **Public key: e**
  - Example: e=13
- **RSA modulus: 77=11×7**
  - $\Phi(77)=10\times6=60$
  - GCD(e, Φ)=1 (coprime, OK)
- **Find a,b such that** $\Phi\, a + e\, b = 1$
  - ExtendedGCD[60,13]={1,{5,-23}}
  - 60 × 5 + 13 × (-23) = 1 → 13 × (-23) = 1 - 60 × 5
  - Mod[13 × (-23), 60] = Mod[1 - 60 × 5, 60] = 1
- **Hence 13^{-1} mod 60 = -23 = 37**

- $e = 13$
- $N = 77 = 11 \cdot 7 \rightarrow \phi(N) = (11-1)\cdot(7-1) = 60$
- $GCD(e, \phi) = 1 \rightarrow$ primi tra loro

Voglio $\phi \cdot a + e \cdot b = 1$

Passo al modulo!

$(\phi a + eb) \bmod \phi(77) = 1 \bmod \phi(77)$

cioè $13 \cdot (-23) \equiv 1$ (e, d)

a	b	val	rem
1	0	60	
0	1	13	4
1	-4	8	1
-1	5	5	1
2	-9	3	1
-3	14	2	1
5	-23	1	

d la voglio $>0 \rightarrow d = -23 + \phi(77) = 37$

- Encrypto : $m^{e=13} \bmod n = 53$

  m: ④ (esempio), n: 77

- decrypto : $53^{d=37} \bmod n = ④$

  n: 77

Queste versioni sono "Vanilla", cioè semplificate, non gestiscono casi particolari (come msg = 000...)

RSA recap: toy example

- **p = 11, q = 17 (*secret*)**
- **N = p×q = 11×17 = 187 (*public*)**
- **Φ = (p–1)×(q–1) = 10×16 = 160 (*secret*)**

- **e = 7 (public, prime wrt 160)**
- **d = 7^{-1} mod 160 = 23 (secret)**
  - ⇨Note: 23×7 ≡ 161 = 160+1 = 1 (mod 160)

- **Public Key Encryption: $C=M^7$ mod 187**
  - ⇨Decrypt: $(M^7)^{23}$ mod 187= M^{7x23} mod 187= M

- **Digital Signature: send M, $TAG=H(M)^{23}$ mod 187**
  - ⇨Verify sign: TAG^7 mod 187= $(H(M)^{23})^7$ mod 187= H(M)
    - ⇨check hash matches with message M provided

Cosa succede se non applico Hash [NB. si applica sempre, non solo per ridurre len(msg)]

- dato M_1 mando M_1 con signature $(M_1)^d \bmod N$, ed M_2 con signature $(M_2)^d \bmod N$.

Data una coppia di tag precedenti $\begin{pmatrix} M_1 \to tag_1 \\ M_2 \to tag_2 \end{pmatrix}$ NON deve poter creare tag valido per msg a mia scelta.

- in RSA tuttavia, posso avere tag valido per $M_1 \cdot M_2$ e $tag_1 \cdot tag_2$ $(M_1 M_2)^d$ valido per $M_1 M_2$. Senza Hash non garantisco tale aspetto.
- con DS mando M e $H(M)_{SK}$, obbligatorio per RSA.

Digital Certificates and Public Key Infrastructure

digital signature: review

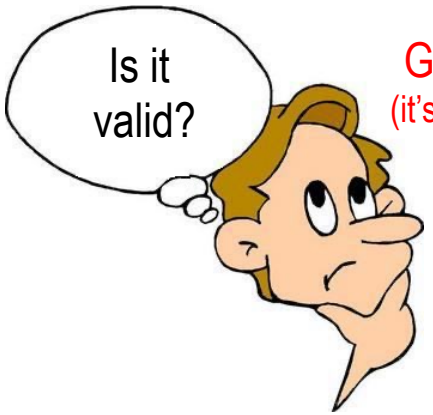
Real Problem of Public Encryption: attacco public key distribution, NON l'algoritmo.

Public Key
PK

Private Key
SK

M = «780 EUR per te»

H(M)
Use SK



Get PK
(it's public!)

M = «780 EUR per te»

H(M)
Use SK

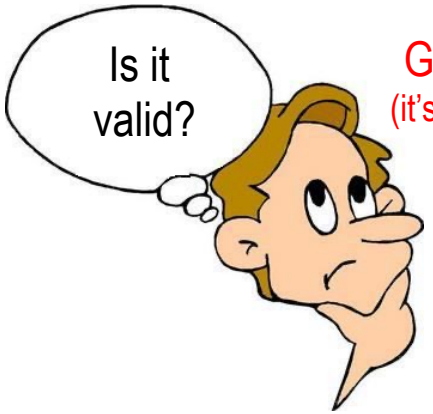
digital signature: review

Private Key
SK

Public Key
PK

M = «780 EUR per te»

H(M)
Use SK



Get PK
(it's public!)

M = «780 EUR per te»

H(M)
Use SK

Use PK to «invert» tag

Do they match?
If Y, message is valid

digital signature: problem

Private Key
flaviaSK

Public Key
flaviaPK

(impersonification attack)
convinco che Flavia ha come PublicKey la mia!

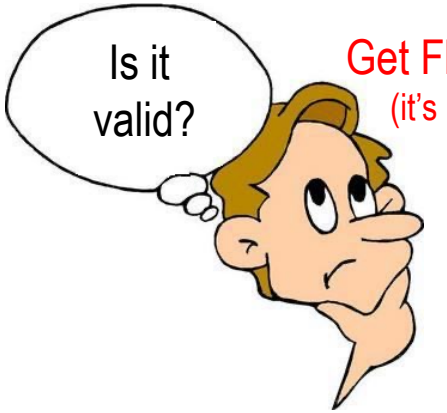
Public Key
gbPK

Mmmmh,
Let me try
to break this

Private Key
gbSK

I'm Flavia,
here is my PK: gbPK

Algoritmo ok, ma non attacco lui!



Get Flavia PK
(it's public!)

M' = «7800 EUR per te»

H(M')
gbSK

OK, The PK that Flavia (??!!) gave me
matches this signature, the message is valid (!!)

PK non mantiene info su chi sia il proprietario!!

RSA Key Transport, review

Flavia wants to access her bank account
using a secure communication

Hello dear, would you mind sending me you PK?

Here you are: *BankA* → pk_{bankA}

choose random K (simmetric secret K)

(RSA encrypt "k" con PKbankA

Here is the secret key we'll use in our session: ENC(K, pk_{BankA})

Data exchange: encrypted with symmetric key K

RSA Key Transport, MITM attack

Hello dear, would you mind sending me you PK?

I'm BankA, this is pk_{badguy}

I'm BankA, this is pk_{bankA}

choose random K

Here is key: ENC(K, pk_{badguy})

Here is key: ENC(K, pk_{bankA})

I have K!!

Data exchange: encrypted with symmetric key K

What about DH? MITM, again!!

ogni schema vulnerabile a questo attacco!!

Alice | **Eve!** | **Bob**

dovrei mandare (Alice, g^x mod p) CA-SIGN

per risolvere MITM in DH! (Fixed DH), anche questo ha difetti

genera "z" e "g^z"

attacco il binding nome - coeff. DH, non chiave!

choose random
$x \in \{0,1\}^s$

choose random
$y \in \{0,1\}^s$

Alice's

① $(g^x) \bmod p$

③ $(g^z) \bmod p$

Alice's

$(g^z) \bmod p$ ③

② $(g^y) \bmod p$

Alice computa: $K_1 = (g^{xz}) \bmod p$ ≠ Bob

$K_2 = (g^{yz}) \bmod p$: computato da Bob ≠ Alice

Data exchange: encrypted with K1 (con Alice)

Data exchange: encrypted with K2 (con Bob)

Three different scenarios, SAME problem!!
We need a way to 'cryptographically bind' a public key to an identity!

voglio linkarli !!

I'm Flavia, my public key is 876543

I'm Flavia, my public key is 876544

NO

YOU'RE NOT, I CAN SPOT YOU

The solution:
Digital Certificates

Main Role of a digital certificate

(legare con associazione che non posso rompere, i.e. crypto)

➔To Bind a public key to a subject

⇨A person, a company, a legal entity, …

Subject NAME	Subject Public Key

➔Cryptographic binding

(scarico la responsabilità su qualcuno fidato)

⇨Digital signature by a TRUSTED THIRD PARTY: a "certification Authority" that guarantees integrity of the binding

registro id-key, ma è statico, se lo aggiorno tutti devono cambiarlo, NON SCALA.
Conviene "preinstallarlo" o "aggiornarlo" (ma non è standardizzato)

Why we need certificates?

➔**User X knows ALL public keys of all remaining users**

⇨No need for certificates (direct trust model)

⇨BUT: unviable, does not scale

➔**User X trusts TTP, who guarantees for other users**

⇨Public Key Infrastructure PKI

A) Issuing a Certificate (offline)

CERTIFICATION AUTHORITY

4

a loro interessa controllare nome ed identità, non tanto PK, lo faccio in carole sicure (evito MITM, che potrebbe associare ID vittima - PK MITM)

CERT = (BankName, BankPK)$_{CA\_sign}$

3

Here is my BankName and here my PK (via secure channel) fisicamente!

CERT = (BankName, BankPK)$_{CA\_sign}$

SK

1

Generate PK and SK pair (le creo Io!)

2

Store SK locally (never diclosed or TXed!)

B) Verifying Certificate validity

CERTIFICATION AUTHORITY

1 check if is not in CRL (block list)

Here I am: name and PK
$(BankName, BankPK)_{CA\_sign}$

$CERT = (BankName, BankPK)_{CA\_sign}$

SK

non mi dice niente su chi è, mi fido della signature? deve stare nel PC! (in Chrome → privacy → gestisci certificati) (1° controllo)

2 **Do I trust the CA you have chosen? (Is in my list of trusted ones?) → OK**

3 **Is the CA signature correct? → OK**

4 chiedo alla banca di provare che possieda private SK associata!
→ sign qualcosa di fresh
→ decrypt qualcosa di fresh

DOES THIS MEAN THAT I'M REALLY TALKING TO THE BANK?

Mi dice solo mapping BANK-PA è corretto! Non mi dice se sto parlando con la BANK!

B) Verifying Certificate validity

CERTIFICATION AUTHORITY

Here I am: name and PK
$(BankName, BankPK)_{CA\_sign}$

$CERT = (BankName, BankPK)_{CA\_sign}$

3 Is the CA signature correct? → OK

DOES THIS MEAN THAT
I'M REALLY TALKING TO THE BANK?

NOT YET – what about cert replay?!

B) Verifying Certificate validity

CERTIFICATION AUTHORITY

SK

TWO WAYS TO DO THIS:

1) Ask Bank to sign something fresh
2) Ask bank to decrypt something fresh

4 Let's «ask» the bank to PROVE possession of the private key SK associated to the cert!

Proof of knowledge of SK via digital signature

Generate fresh NONCE (challenge)

$\text{Certificate(BankName, BankPK)}_{\text{CA\_sign}}$

Prove knowledge of SK by signing a fresh NONCE (chiaro)

$(\text{NONCE})_{\text{Bank\_sign}}$

non è certification authority, ma con la SK di BANK

If bank signature «opens» with certificate PK, then Bank is authenticated!

Proof of knowledge of SK via PubKey encryption

Generate fresh NONCE (challenge)

Certificate(BankName, BankPK)$_{CA\_sign}$

encripto 'NONCE'
solo io ho NONCE in clear! Solo 'BANK' può decriptarlo

Prove you know SK by decrypting this: ENC (NONCE)$_{Bank\_PK}$

Here you are: NONCE

If bank returns «decrypted» NONCE
then Bank is authenticated!

Practical TLS approach

(when RSA Key Transport used)

Generate fresh Key K

authentication to the BANK

Certificate(BankName, BankPK)$_{CA\_sign}$

se fosse "NONCE" è come lo slide di prima!

ENC (key K)$_{Bank\_PK}$

implicit authentication

solo BANK lo può decriptare!

Key derivation

Data exchange: encrypted with symmetric key K

Bank can communicate only if it is Authentic!! If Rogue/MITM, cannot decrypt key K

Certificate chains

UNIVERSETRUST

ROOT authority, mi fido "ciecamente" di lei! normalmente non da' garanzie!

a sicura per lei! (aumenta scalabilita')

("intermediate")

LUNATRUST

LUNATRUST? Not in my list of trusted.

Who «certifies» LUNATRUST? NON vedo il suo certificato, quindi mi aspetto certificate (LUNATRUST, PK LUNATRUST) UNIVERSETRUST

UNIVERSETRUST: OK, this is in my list, let's go on!

CERTIFICATE(BankName,PK)$_{LUNATRUST}$

Il certificato necessita di altro! Ovvero vedere se Luna ha il permesso di rilasciare certificazioni e GB invece no. ∃ browser che non verificano tale condizione. (BASIC CONSTRAINTS Flag)

Very brief intro to PKI
Public Key Infrastructure

(ad oggi PKI NON Basta!)

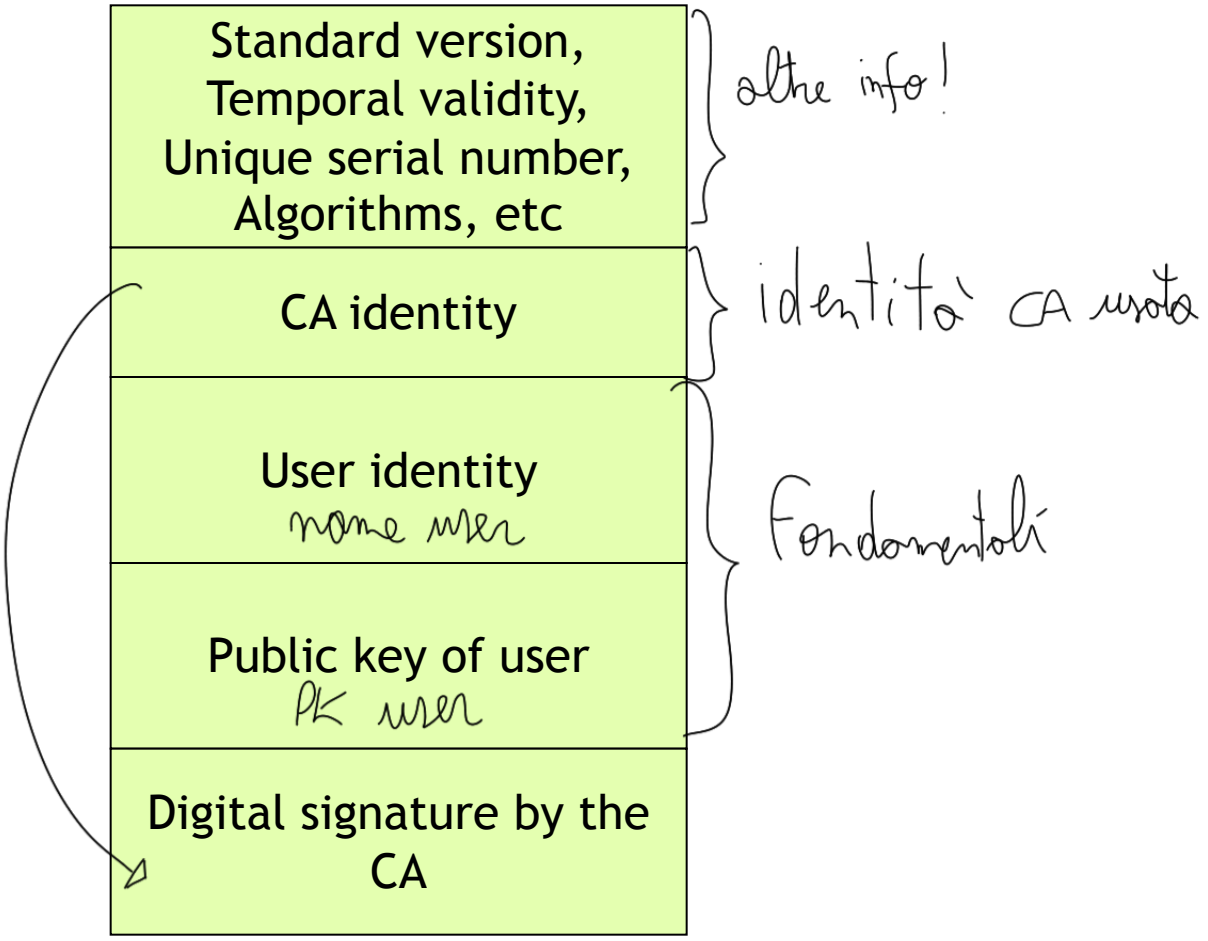
Public Key Infrastructure (PKI)

- **A PKI specifies protocols, policies, and technical mechanisms needed to support exchange of public keys**
- **A PKI architecture requires:**
  - Standard format for certificates
  - Relation among CAs, and with end users
  - Policies for issuing and REVOKING (!!) certificates
  - *Directory services*

↳ certificate revocation list

- **Typical certificate format: X.509**

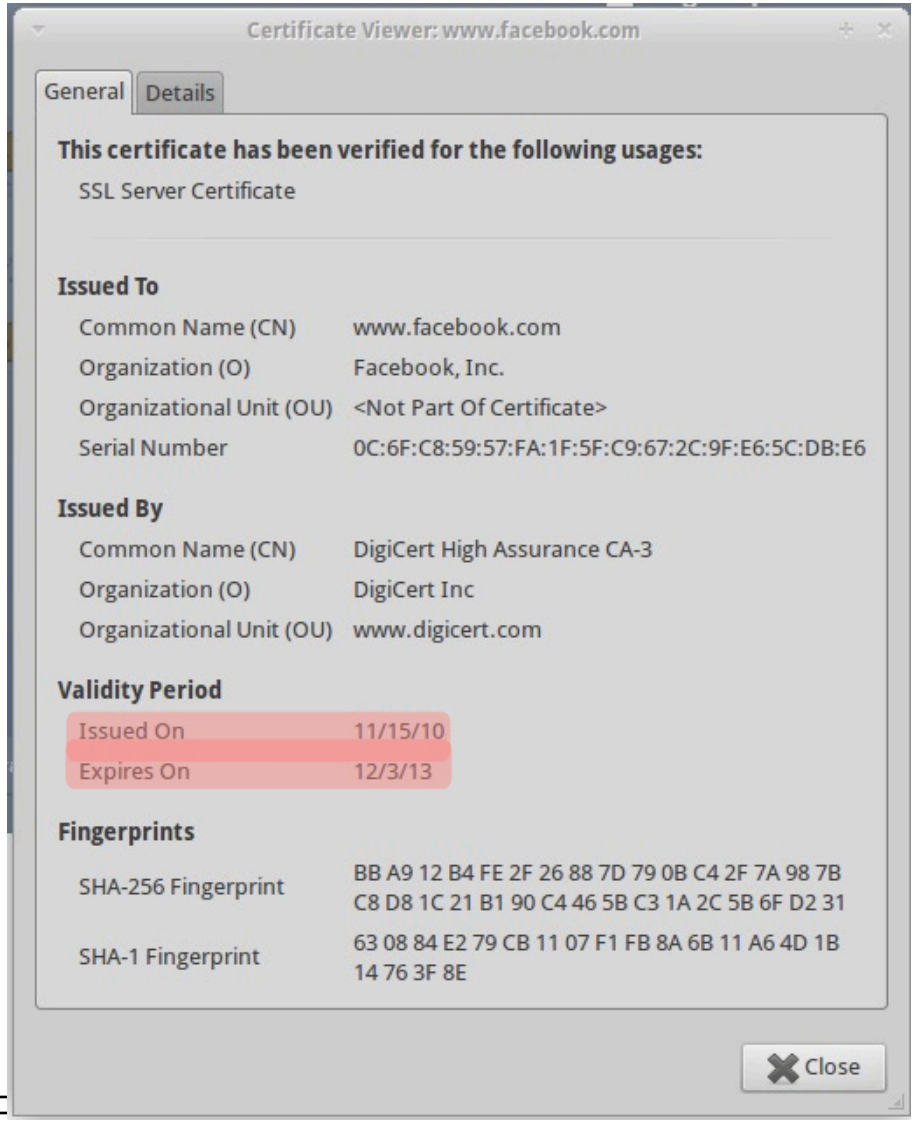
X.509 certificates (high level)



X.509 certificates: a real example

facebook

```
marlon@ubuntu:~/Desktop$ openssl x509 -in www.facebook.com
-----BEGIN CERTIFICATE-----
MIIGMjCCBRqgAwIBAgIQDG/IWVf6H1/JZyyf5lzb5jANBgkqhkiG9w0BAQUFADBm
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSUwIwYDVQQDExxEaWdpQ2VydCBIaWdoIEFzc3VyYW5j
ZSBDQS0zMB4XDTEwMTExNTAwMDAwMFoXDTEzMTIwMjIzNTk1OVowajELMAkGA1UE
BhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExEjAQBgNVBAcTCVBhbG8gQWx0bzEX
MBUGA1UEChMORmFjZWJvb2ssIEluYy4xGTAXBgNVBAMTEHd3dy5mYWNlYm9vay5j
b20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMHffWNBvcTk+mUzE3jVYjeW
p2HzsZa/I466h6ftB/neLeuox7ytd6ZejQMDNuNN99Dxq2byty4zFr4mD11BFn//
twCe+g6ZFWxSGtcKxq375AciP9sEpLZppe3Wh7aIxYP16Maz/8AOH52jhXDtonYU
e3A+77BCCzjWggAj3WN1AgMBAAGjggNaMIIDVjAfBgNVHSMEGDAWgBRQ6nOJ2yn7
EI+e5QEg1N55mUiD9zAdBgNVHQ4EFgQUqldKM7bs1W6BE6Y2XvR7Q1jzjOQwKQYD
VR0RBCIwIIIQd3d3LmZhY2Vib29rLmNvbYIMZmFjZWJvb2suY29tMHsGCCsGAQUF
BwEBBG8wbTAkBggrBgEFBQcwAYYYaHR0cDovL29jc3AuZGlnaWNlcnQuY29tMEUG
CCsGAQUFBzAChjlodHRwOi8vY2FjZXJ0cy5kaWdpY2VydC5jb20vRGlnaUNlcnRI
aWdoQXNzdXJhbmNlQ0EtMy5jcnQwDgYDVR0PAQH/BAQDAgWgMAwGA1UdEwEB/wQC
MAAwZQYDVR0fBF4wXDAsoCqgKIYmaHR0cDovL2NybDMuZGlnaWNlcnQuY29tL2Nh
My0yMDEwaS5jcmwwLKAqoCiGJmh0dHA6Ly9jcmw0LmRpZ2ljZXJ0LmNvbS9jYTMt
MjAxMGkuY3JsMIIBxgYDVR0gBIIBvTCCAbkwggG1BgtghkgBhv1sAQMAATCCAaQw
OgYIKwYBBQUHAgEWLmh0dHA6Ly93d3cuZGlnaWNlcnQuY29tL3NzbC1jcHMtcmVw
b3NpdG9yeS5odG0wggFkBggrBgEFBQcCAjCCAVYeggFSAEEAbgB5ACAAdQBzAGUA
IABvAGYAIABOAGgAaQBzACAAQwBlAHIAdABpAGYAaQBjAGEAdABlACAAYwBvAG4A
cwB0AGkAdAB1AHQAZQBzACAAYQBjAGMAZQBwAHQAYQBuAGMAZQAgAG8AZgAgAHQA
aABlACAARABpAGcAaQBDAGUAcgB0ACAAQwBQAC8AQwBQAFMAIABhAG4AZAAgAHQA
aABlACAAUgBlAGwAeQBpAG4AZwAgAFAAYQByAHQAeQAgAEEAZwByAGUAZQBtAGUA
bgB0ACAAdwBoACAAbABpAG0AaQB0ACAAbABpAGEAYgBpAGwAaQB0AHkA
IABhAG4AZAAgAGEAcgBlACAAaQBuAGMAbwByAHAAbwByAGEAdABlAGQAIABoAGUA
cgBlAGkAbgAgAGIAeQAgAHIAZQBmAGUAcgBlAG4AYwBlAC4wHQYDVR0lBBYwFAYI
KwYBBQUHAwEGCCsGAQUFBwMCMA0GCSqGSIb3DQEBBQUAA4IBAQAlM16QP6OC/t6S
0p4S9+8Wao26ZqBarmZ2vEoSE+OS1vcPlLwBlSDo8P2sZt4kGK/uor9fo+xectYg
GtLGjwcNev+lj30HfO6ZgQBqjYCnjcAAFsUd2AY39+wD6KK0QFyVdQwUAdF1p1aY
8DggH3cVeau14wQKd8nDtZlXdk8ObncaYTdvmrpTUT9RPpXAtMQgl+kmE0DDGeRB
2Sb3OUvyoaTDtQXFvuJlhcspgGHW14e6yCX+hXG7OmZjUkkLHWqAYkM8J/w8Khwu
gqeCEJjrS1oyfLGPXDkAxC9xtb3+v2DdAEOj8xCWg/hvleSrYh1SBXmU1zHyHHVE
yieOb6nD
-----END CERTIFICATE-----
```



X.509 certificates: a real example

Version: 3 (0x2)
Serial Number:
0c:6f:c8:59:57:fa:1f:5f:c9:67:2c:9f:e6:5c:db:e6
Signature Algorithm: sha1WithRSAEncryption viene specificato!
Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert High Assurance CA-3
Validity
Not Before: Nov 15 00:00:00 2010 GMT
Not After : Dec 2 23:59:59 2013 GMT
Subject: C=US, ST=California, L=Palo Alto, O=Facebook, Inc., CN=www.facebook.com
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (1024 bit)
Modulus (1024 bit):
00:c1:df:7d:63:41:bd:c4:e4:fa:65:33:13:78:d5: **(... deleted ...)** 0b:38:d6:82:00:23:dd:63:75
Exponent: 65537 (0x10001)

X509v3 extensions: **(deleted)**
X509v3 Subject Key Identifier:
AA:57:4A:33:B6:EC:D5:6E:81:13:A6:36:5E:F4:7B:43:58:F3:8F:44
X509v3 Subject Alternative Name:
DNS:www.facebook.com, DNS:facebook.com
X509v3 Key Usage: critical
Digital Signature, Key Encipherment
X509v3 Basic Constraints: critical
CA:FALSE
Posso generare certificato? Se in mezzo alla catena allora si, se alla fine controllo.
X509v3 Extended Key Usage:
TLS Web Server Authentication, TLS Web Client Authentication

Signature Algorithm: sha1WithRSAEncryption
25:33:5e:90:3f:ad:02:fe:de:92:d2:9e:12:f7:ef:16:6a:8d: **(... deleted ...)** 8e:6f:a9:c3

Public Key Cryptography Standards

- **PKCS family**
  - PKCS #1 -- #15
  - Some obsoleted
- **Standardize many PKI related aspects (and more)**
  - PKCS#1: RSA
  - PKCS#3: DH
  - PKCS#13: ECC
  - PKCS#6: X.509 extensions
  - PKCS 10: Certification Request Syntax Standard
  - PKCS 11, 15: Cryptographic Token Interface Standard - API for signing and verifying data by a device that holds the key
  - PKCS 12: Personal Information Exchange Syntax Standard - file format for storing certificate and private key - used to move private information between browsers

Example: Certificate Signing Request

- **A certificate signing request (also CSR or certification request) is a message sent from an applicant to a certificate authority in order to apply for a digital identity certificate**
- **The most common format for CSRs is the PKCS#10 specification**
- **Operations:**
  - the applicant first generates a key pair, keeping the private key secret
  - the applicant generates a CSR contains information identifying herself (X.509 subject field), optional X.509 extensions (e.g. key usage: RSA authentication for web servers) and the public key chosen by the applicant
  - The CSR may be accompanied by other credentials or proofs of identity required by the certificate authority, and the certificate authority may contact the applicant for further information

X509v3 extensions

- An X.509 v3 certificate contains an extension field that permits any number of additional fields to be added to the certificate

- Certificate extensions provide a way of adding information such as alternative subject names and usage restrictions to certificates

Some standard extensions

- **Authority Key Identifier**
  - The authority key identifier extension provides a means of identifying the public key corresponding to the private key used to sign a certificate
- **Subject Key Identifier**
  - The subject key identifier extension provides a means of identifying certificates that contain a particular public key
- **Key Usage**
  - The key usage extension defines the purpose (e.g., encipherment, signature, certificate signing) of the key contained in the certificate.
  - digitalSignature, nonRepudiation, contentCommitment, keyEncipherment , dataEncipherment, keyAgreement, keyCertSign, cRLSign, encipherOnly, decipherOnly
- **Subject Alternative Name**
  - The subject alternative name extension allows identities to be bound to the subject of the certificate. These identities may be included in addition to or in place of the identity in the subject field of the certificate
- **Extended Key Usage**
  - This extension indicates one or more purposes for which the certified public key may be used, in addition to or in place of the basic purposes indicated in the key usage extension.
  - TLS WWW server authentication, TLS WWW client authentication, Signing of downloadable executable code, Email protection, Timestamping

See http://tools.ietf.org/html/rfc5280 for the complete list

Certificate revocation

➔ **A PKI MUST include mechanisms for handling compromised certificates!**

- ⇨ Examples:
  - →Public keys whose private keys have been disclosed
  - →Public keys whose private keys have been lost
  - →Public keys not anymore used

➔ **Two major (and coexisting) approaches**

- ⇨ Validity period for a certificate
- ⇨ Explicit revocation

➔ **Classical analogy: a credit card**

Revocation approach: *Certificate Revocation List* (CRL)

- **EVERY CA must regularly publish a list of certificates the CA has revoked**

- **List format: must include**
  - Issuer, Last update date, Next update date
  - List of serial numbers revoked, along with revocation date
  - CA digital signature

- **In principle, you SHOULD verify that the certificate of the site you are accessing has NOT been revoked**
  - Overhead:
    - Online access, or offline periodic download of CRL
  - In practice… frequently overlooked by non critical applications
    - But remember, whenever you are involved in a security critical application!