

# **Cours 5MMSIC – WMMBESIC**

## **Sécurité Informatique et confidentialité**

**Jean-Louis ROCH**

## **Cours « 2 » Architectures de confiance**

# Intégrité

- Alice utilise OTP pour communiquer avec Bob.  
Elle veut envoyer  $M$ , et calcule  $C = M \text{ xor } K_{\text{AliceBob}}$
- Bob reçoit  $C'$ , qu'il décode en  $M' = C' \text{ xor } K_{\text{AliceBob}}$
- Quelle “confiance” Bob et Alice peuvent-ils avoir dans le fait que  $M = M'$  ? (*valeur de la confiance*)
- Comment augmenter cette confiance ?
  - Contrôle d'intégrité
  - Naïf: répétition

# **Architectures de confiance**

- 1. Intégrité et redondance - Signatures**  
(codes détecteurs d'erreurs et hash)
- 2. Confiance centralisée:**
  - Infrastructure à clef privée (Kerberos)
  - Infrastructure à clef publique ( PKI )
- 3. Confiance distribuée:**
  - Hiérarchie de confiance (Web of trust, Open PGP)
  - Registre = liste unique (ledgers)

# Intégrité

- Principe : ajout de redondance
  - Message  $M$  + information redondante sur  $M$   
Exemple: répéter  $M$  2 fois:  $M \parallel M$
- Redondance chiffrée = signature
  - Bob publie «  $M \parallel D_{Bob}(M)$  »
  - Chacun peut vérifier que c'est bien Bob qui a écrit le message

# « Parités » usuelles pour la simple détection d'erreur

- LUHN10 pour les cartes bleues : dernier chiffre = chiffre de parité
  - Doubler modulo 9 un chiffre sur deux du n°  
La somme des chiffres obtenus doit être multiple de 10.



- Exemple : 4561 0032 4001 236**c**
  - $(4*2\%9)+5+(6*2\%9)+1+(0*2\%9)+0+(3*2\%9)+2+ (4*2\%9)+0+(0*2\%9)+1+ (2*2\%9)+3+(6*2\%9)$   
 $= 8 +5+ 3 +1+ 0 +0+ 6 +2+ 8 +0+ 0 +1+ 4 +3+ 3 =44$
  - Donc  $c = 10 - (44 \% 10 ) = 6 \Rightarrow n° \text{ valide} = 4561 0032 4001 2366$

# « Parités » usuelles pour la simple détection d'erreur

- Billets  $\epsilon = (1_{\text{lettre}} \parallel 11_{\text{chiffres}})$ : code ascii lettre = somme(chiffres)  $\pmod{9}$ 
  - Lettre: code Ascii A=65=2(mod 9), B=3,..., U<sub>France</sub>=85=4, ..., X<sub>Allemagne</sub>=2, ...

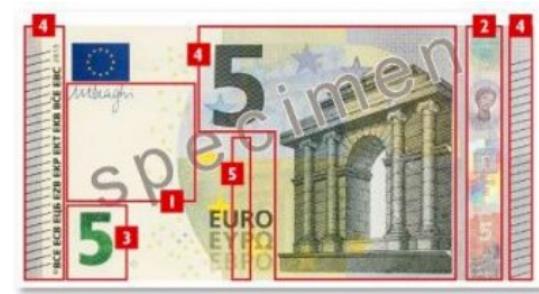


$$\begin{array}{cccccccccc}
 & X & 8 & 2 & 0 & 0 & 5 & 0 & 3 & 8 & 9 & 1 & 2 \\
 \hline
 & 1 & + & 5 & & & & 2 & + & 3 & & & \\
 \hline
 & 2 = & & & & & & & & & & & 2 \pmod{9}
 \end{array}$$



$$\begin{array}{cccccccccc}
 & X & 1 & 4 & 9 & 0 & 2 & 2 & 2 & 4 & 3 & 4 & 7 \\
 \hline
 & 5 & + & 2 & & & + & 8 & + & 5 & & & \\
 \hline
 & 2 = & & & & & & & & & & & 2 \pmod{9}
 \end{array}$$

**Billets 5€ depuis Janvier 2013**  
2 lettres || 10 chiffres



# « Parités » usuelles pour la simple détection d'erreur

- Clefs (sécurité sociale, RIB, etc.)

- Contraventions:  $(n^{\circ} \text{ 14 chiffres} \parallel \text{clef}_2 \text{ chiffres})$  tel que  $\text{clef} = n^{\circ} \pmod{97}$



- Sécu: numéro =  $(n^{\circ} \text{ NIR}_{13} \text{ chiffres} \parallel \text{clef}_2 \text{ chiffres})$  tel que:  $\text{NIR} + \text{clef} = 0 \pmod{97}$ 
  - Chiffres 6 et 7 du NIR = dépt.. (Corse 2A, 2B : A = -1 et B = -2)



- RIB : clef calculée pour que  $(\text{numéro} \parallel \text{clef})_{5+5+11+2 \text{ chiffres}} = 0 \pmod{97}$
- IBAN :  $(A_4 \text{ caractères} \parallel N)$  : soit  $n=(N||A)$  avec A=10, B=11, .. Z=35; alors  $n=1 \pmod{97}$

Relevé d'Identité Bancaire/IBAN

Ce relevé évite les erreurs ou les retards concernant les opérations au débit (prélèvements...) ou au crédit (versements de salaire...) de votre compte. Il suffit de vous assurer le bon enregistrement des opérations qui concernent votre compte.

N'hésitez pas à le remettre aux organismes concernés par ces opérations.

RIB	Code Banque (1)	Code Agence (2)	Numéro de compte (3)	IBAN	Votre agence de domiciliation (4)
	30004	00988	00010137370	FRT6 3000 4009 8800 0101 3737 084	BNP PARIBAS PARIS EUROC (00988)
IBAN					IBC: BNPAPRFFFG 0

(1) Code de BNP Paribas (2) Code de votre agence d'origine (3) Votre numéro de compte (4) Agence BNP Paribas (5) International Bank Account Number (6) Bank Identifier Code

- Exercice: Pourquoi pas mod 100 ? Pourquoi mod 97 ?

Indications: René Camille en 1934

(et Babbage en 1834



# Code barre EAN-13

- EAN-13 (European Article Numbering)
- Numéro sur 13 chiffres + motif graphique barres noires/blanches

$$c_{12} - c_{11} \dots c_6 - c_5 \dots c_0$$

- $c_0$  chiffre de parité calculé comme suit:

- Soient  $a = c_{12} + c_{10} + c_8 + c_6 + c_4 + c_2 \bmod 10$   
et  $b = c_{11} + c_9 + c_7 + c_5 + c_3 + c_1 \bmod 10$
- Alors  $c_0 = 10 - (a+3b \bmod 10)$
- Exemple:  $a = 3+9+1+3+5+7 \bmod 10 = 8 ; b = 2+0+2+4+6+8 \bmod 10 = 2 ; c_0 = 10 - (a+3b \bmod 10) = 10 - 4 = 6$

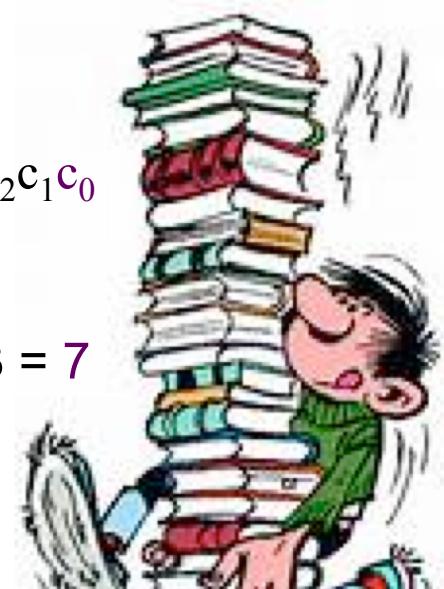
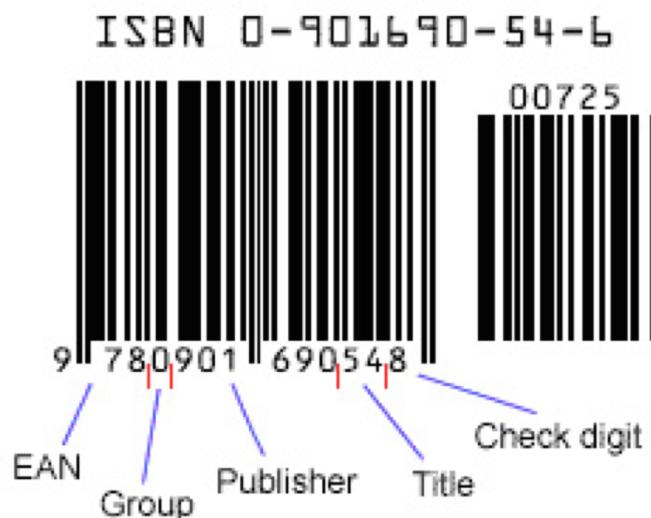


- Le code barre graphique code le même numéro:
  - chaque colonne de 2,31mm code un seul chiffre, par 4 barres de largeur différentes chaque colonne est divisée en 7 barres N/B de largeur élémentaire 0,33 mm
- EAN13 permet de détecter des erreurs mais pas de corriger.

# « Parités » usuelles pour la simple détection d'erreur

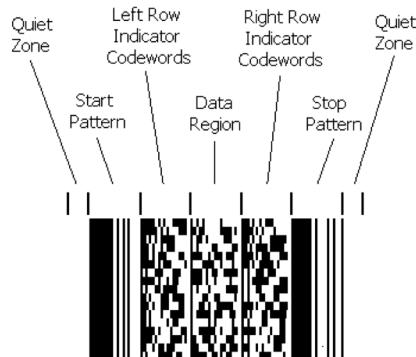
- De 1972 à 2007: Code ISBN sur les livres sur 10 chiffres  $a_{10}a_9a_8a_7a_6a_5a_4a_3a_2a_1$   
 $a_1$  = chiffre de contrôle appartient à  $\{0, 1, 2, \dots, 9, X\}$  avec  $X=10$  et tel que :  
 $10.a_{10} + 9.a_9 + 8.a_8 + 7.a_7 + 6.a_6 + 5.a_5 + 4.a_4 + 3.a_3 + 2.a_2 + 1.a_1 \equiv 0 \pmod{11}$

- Depuis 2007: Code ISBN sur les livres=EAN-13:  $c_{12}c_{11}c_{10}c_9c_8c_7c_6c_5c_4c_3c_2c_1c_0$ 
  - Avec pour les livres:  $c_{12}c_{11}c_{10}=978$ 
    - Ex: **978-2-10-050692-7**  
 $c_0 = 10 - [9+8+1+0+0+9+3 \times (7+2+0+5+6+2) \pmod{10}] = 10 - 3 = 7$



# Codes bidimensionnels correcteurs

- Code PDF417: 1108 octets, grâce à un code correcteur de Reed-Solomon

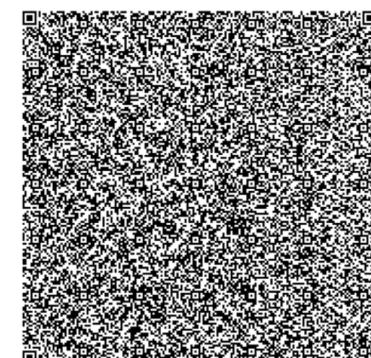


- Codes QR : 2953 octets, code correcteur de Reed-Solomon (jusqu'à 30% redondance)

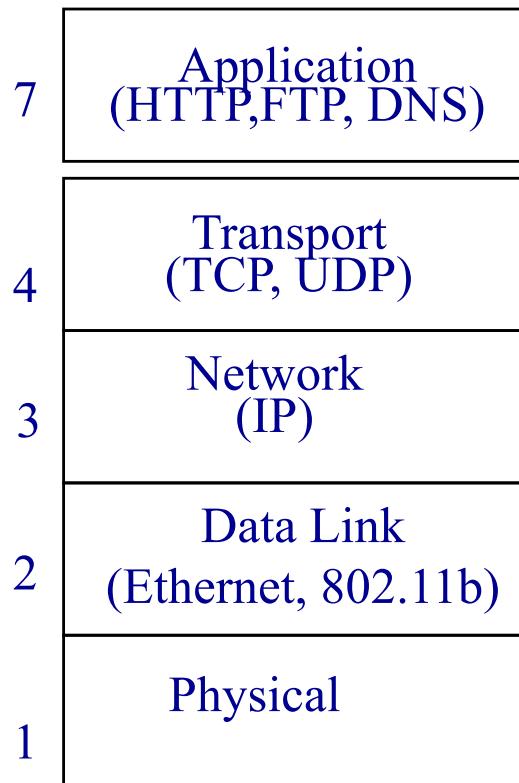
V2 : 25x25  
de 20 à 47 caractères



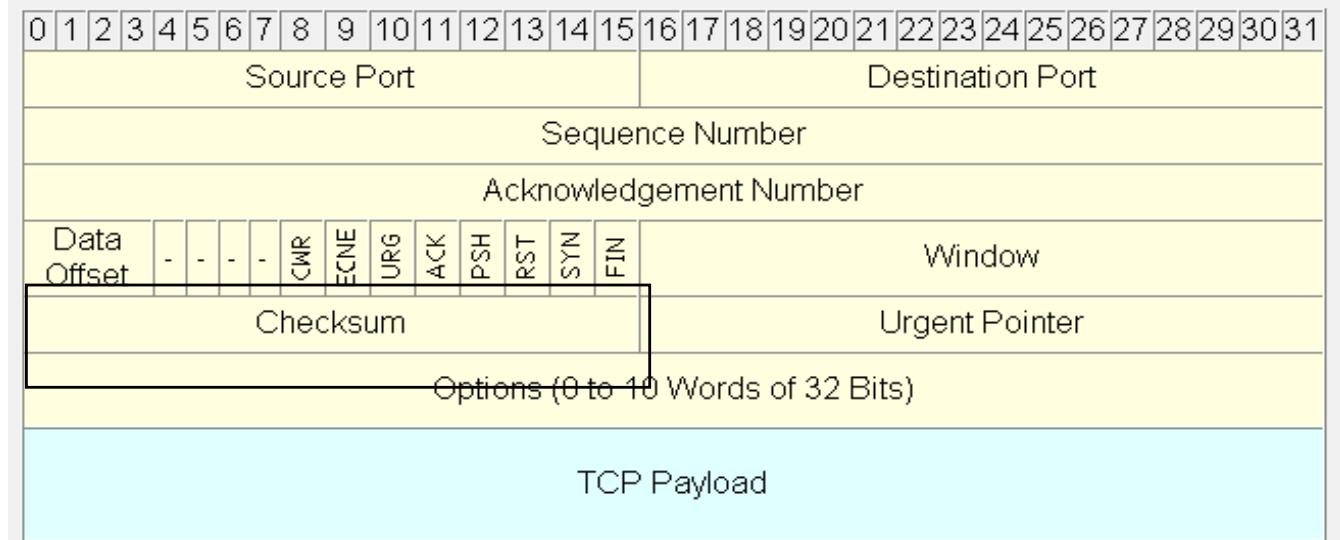
V40 : 177x177  
2953 octets = 4296 caractères



# Exemple: contrôle de parité dans TCP

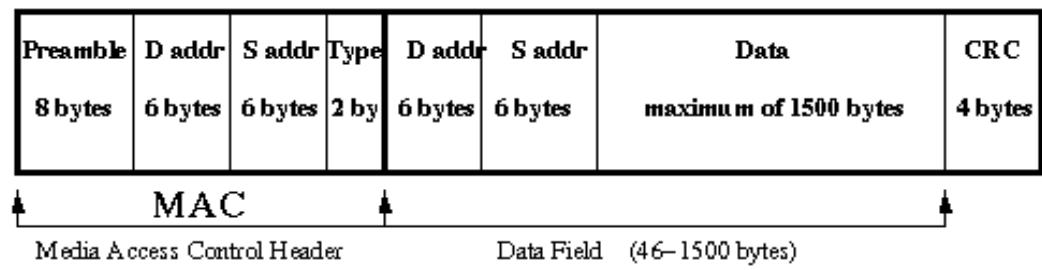


TCP Packet Format



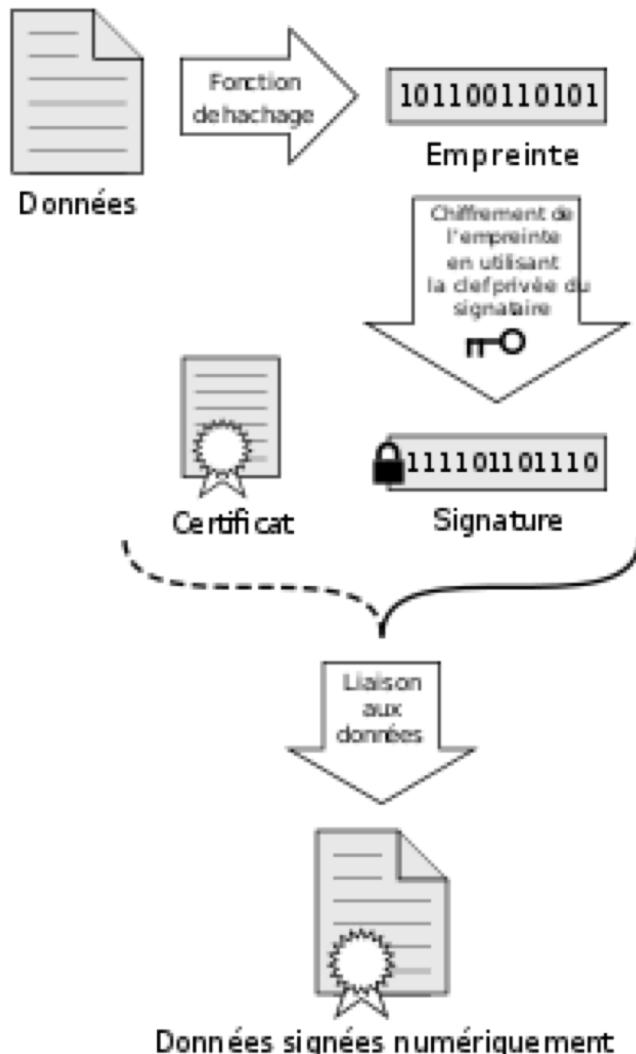
- TCP Checksum : bits de contrôle, calculés par l'émetteur à partir de l'en-tête et des données, et vérifiés lors de la réception.

## Exemple: Ethernet CRC-32

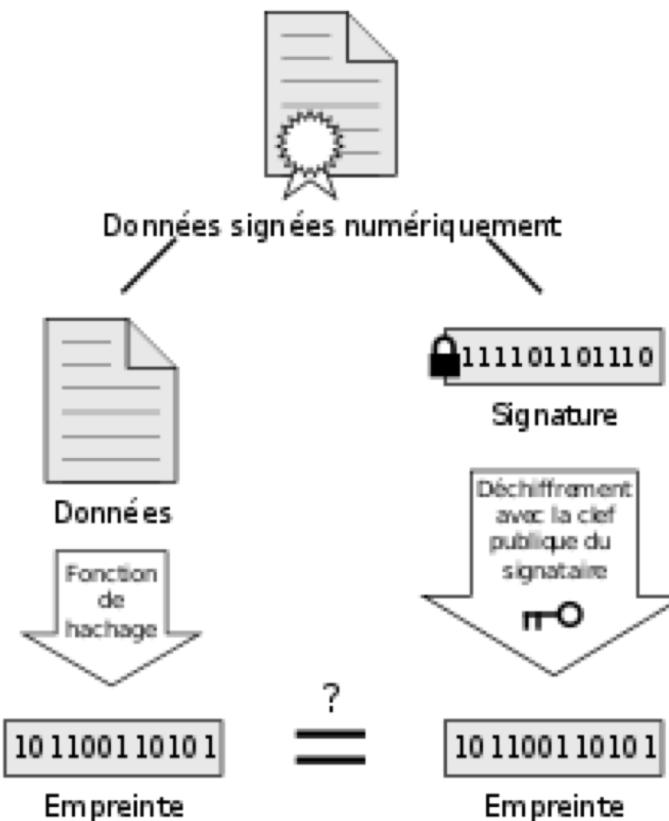


# Signature numérique

## Signature



## Vérification



Si les empreintes sont identiques, la signature est valide

- [https://fr.wikipedia.org/wiki/Signature\\_numérique](https://fr.wikipedia.org/wiki/Signature_numérique)

## Signature DSA, ECDSA

- Algorithme de signature standard
- Génération de clefs (clef secrète + clef publique)
  - Signature avec la cle privée
    - Seul le hash  $H(m)$  du message  $m$  est signé
  - Vérification de la signature avec la cle publique
  - Initialement défini sur  $Z_p^*$  ;  
mais d'autres groupes sont possibles (exemple: EC DSA)
    - Affecte la génération des clefs (et leurs tailles = paramètre de sécurité)

Cf [http://fr.wikipedia.org/wiki/Digital\\_Signature\\_Algorithm](http://fr.wikipedia.org/wiki/Digital_Signature_Algorithm)

# Hash function

- Hash functions take a variable-length message and reduce it to a shorter *message digest* with fixed size ( $k$  bits)  
$$h: \{0,1\}^* \rightarrow \{0,1\}^k$$
- Many applications: “Swiss army knives” of cryptography:
  - Digital signatures (with public key algorithms)
  - Random number generation
  - Key update and derivation
  - One way function
  - Message authentication codes (with a secret key)
  - Integrity protection
  - code recognition (lists of the hashes of known good programs or malware)
  - User authentication (with a secret key)
  - Commitment schemes
- Cryptanalysis changing our understanding of hash functions
  - [eg Wang’s analysis of MD5, SHA-0 and SHA-1 & others]

# Hash Function Properties

- *Preimage resistant*
  - Given only a message digest, can't find any message (or *preimage*) that generates that digest. Roughly speaking, the hash function must be one-way.
- Second preimage resistant
  - Given one message, can't find another message that has the same message digest. An attack that finds a second message with the same message digest is a *second pre-image* attack.
    - It would be easy to forge new digital signatures from old signatures if the hash function used weren't second preimage resistant
- *Collision resistant*
  - Can't find any two different messages with the same message digest
    - Collision resistance implies second preimage resistance
    - Collisions, if we could find them, would give signatories a way to repudiate their signatures
  - Due to birthday paradox, k should be large enough !

- Collision\_attack  $\leq_p$  2<sup>nd</sup>-Preimage\_attack
- Careful: Collision\_resistance NOT  $\leq_p$  Preimage\_resistance
  - Let  $g : \{0,1\}^* \rightarrow \{0,1\}^n$  be collision-resistant and preimage-resistant.
  - Let  $f : \{0,1\}^* \rightarrow \{0,1\}^{n+1}$  defined by  $f(x) :=$  if ( $|x|=n$ ) then “ $0\|x$ ” else “ $1\|g(x)$ ”.
  - Then  $f$  is collision resistant but not pre-image resistant.
- But :
   
 $(\text{Collision\_resistance and one way}) \Rightarrow_p \text{Preimage\_resistance}$

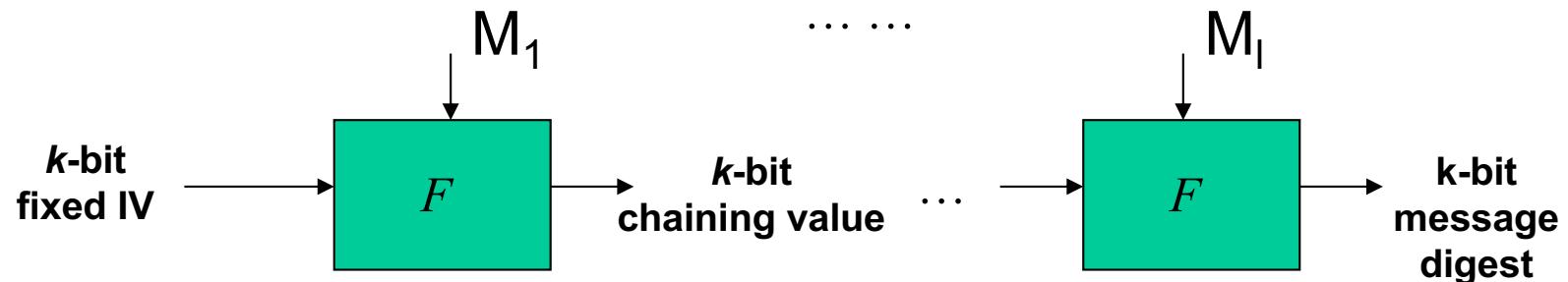
# Finding Collisions by Brute Force : *Birthday paradox and Yuval's attack*

- Choose n random elements in a finite set E :
  - $\Pr[\text{ all distincts}] = \#E . (\#E - 1) . \dots (\#E - n+1) / \#E^n$
  - $\Pr[\#\text{collisions} \geq 1] = 1 - ((\#E !) / (\#E - n)!) / \#E^n$   
 $\sim 1 - \exp(-n(n-1) / (2 \cdot \#E))$
- Conversely, the number of elements n to choose to get a collision with probability p is :  $n \sim (2 \cdot \#E \cdot \ln(1-p)^{-1})^{1/2}$   
For  $p = 0.5$  we have  $n \sim \sqrt{(2 \cdot \#E \cdot \ln 2)} \sim 1.2 \sqrt{\#E}$
- **Yuval's attack** : to build a collision from 2 messages  $M_1$  and  $M_2$  for a h functions with digests of k bits :
  - build a set of  $t=2k$  messages  $M_1'$  close to  $M_1$
  - build random messages  $M_2'$  close to  $M_2$  untill collision :  $h(M_1') = h(M_2')$
- Digests of size 128 bits (MD5) or 160 bits (SHA-1) do not resist to collisions. But 256 bits digest resist.

# Building hash functions: *compression* + *extension*



- Let  $F$  be a basic “*compression function*” that takes in input a block of fixed size ( $k+r$  bits) and delivers in output a digest of size  $k$  bits :
  - For some fixed  $k$  and  $n$ ,  $F$  “compresses” a block of  $n$  bits to one of  $k=n-r$  bits  
 $F: \{0,1\}^{k+r} \rightarrow \{0,1\}^k$  (eg. for SHA2-384  $k=384$  bits and  $r=640$  bits)
- One-to-one padding:**  $M \rightarrow M \parallel \text{pad}(M)$  to have a bit length multiple of  $r$  :
  - $M \parallel \text{pad}(M) = M_1, M_2, M_3, \dots, M_l$  [one-to-one padding:  $M \neq M' \Leftrightarrow M \parallel \text{pad}(M) \neq M' \parallel \text{pad}(M')$ ]
    - Ex.1:  $\text{pad}(M) = "0\dots0" \parallel s$ , where  $s=64$  bits that encode the bitlength of  $M$
    - Ex.2:  $\text{pad}(M) = "0\dots0" \parallel u \parallel 1 \parallel v$ , where  $u=\text{bitlength}(M)$  and  $v="0"^{\log(u)}$
- $F$  is extended to build  $h: \{0,1\}^* \rightarrow \{0,1\}^k$   
based on a provable secure *extension scheme*.
  - Eg: Merkle scheme: last output of compression function is the  $h$ -bit digest.



# Provable compression functions

- **Example:** Chaum-van Heijst - Pfitzmann

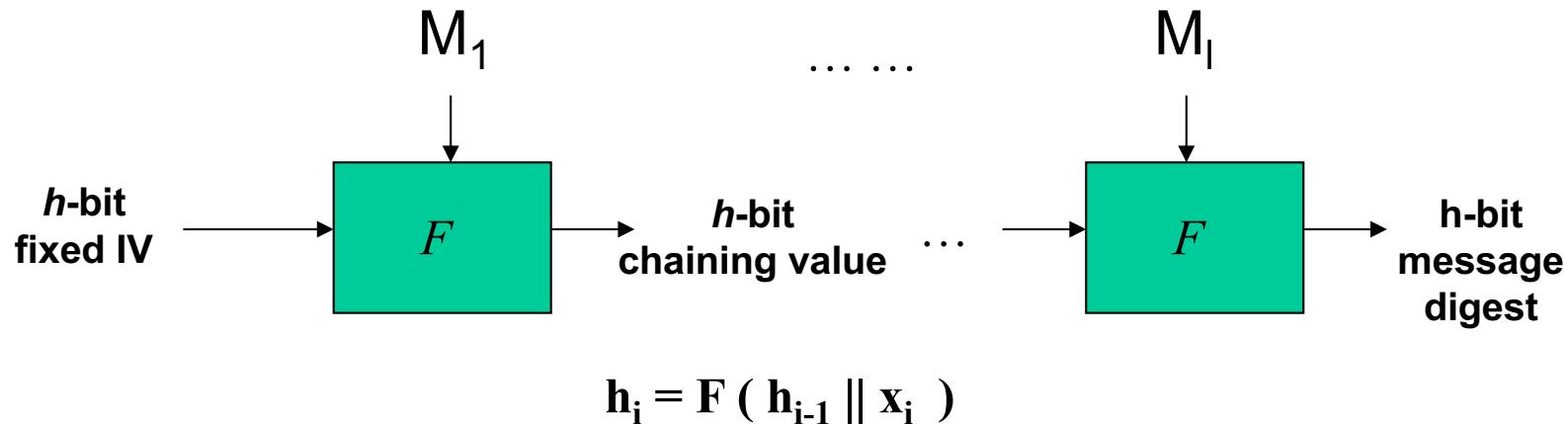
- two prime numbers  $q$  and  $p=2q+1$ .
  - $\alpha$  and  $\beta$  to primitive elements in  $F_p$ .
  - Compression function  $h_1$

$$\begin{aligned} h_1 : \quad & F_q \times F_q \rightarrow F_p \\ & (x_1, x_2) \mapsto \alpha^{x_1} \cdot \beta^{x_2} \mod p \end{aligned}$$

- **Theorem:** If  $\text{LOG}_\alpha(\beta) \mod p$  is impossible to compute (i.e. to find  $x$  such that  $\alpha^x = \beta \mod p$ ),  
then  $h_1$  is resistant to collision.
    - Proof ?
- > Training exercises 4 : building a provable secure compression function  $F$  and a provable secure parallel extension scheme.

# Provable Extension schemes

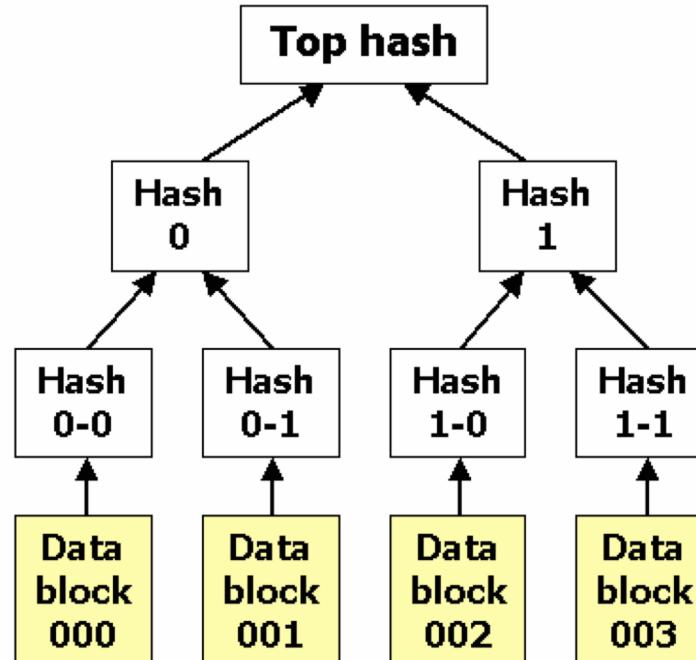
- Example: Merkle-Damgard scheme:
  - Preprocessing step: add padding to injectively make that the size of the input is a multiple of r:  
Compute the hash of  $x \parallel \text{Pad}(x)$ .



- **Theorem:** If the compression function  $F$  is collision resistant then the hash function  $h$  is collision resistant .
  - Proof: by contradiction (reduction) and induction.
- Note: Drawback of Merkle-Damgard: pre-image and second preimage
  - There exist  $O(2^{k-t})$  second-preimage attacks for  $2^t$ -blocks messages [Biham&al. 2006]

## Other extension schemes

- Merkle tree:



- Variants: Truncated Merkle-tree, IV at each leave
- HAIFA :  $h_i = F(h_{i-1} \parallel x_i \parallel i_{encoded\ on\ 64\ bits})$ 
  - where compression  $F: \{0,1\}^{k+r+64} \rightarrow \{0,1\}^k$
  - Lower bound  $W(2^k)$  for 2nd-preimage[Bouillaguet&al2010]
- ...

# MD5

- The message is divided into blocks of  $n = 512$  bits
  - Padding: to obtain a message of length multiple of 512 bits
    - $[B_1..B_k] \Rightarrow [B_1..B_k 10..0 k_0..k_{63}]$  where  $[k_0..k_{63}]$  is the length  $k$  of the source (in 32 bits words)

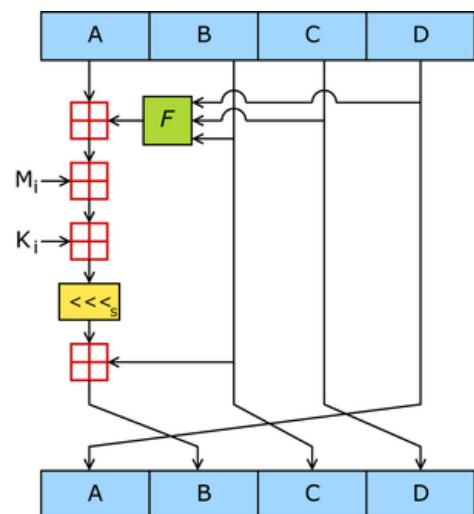
- One step: 4 rounds of 16 operations of this type:
  - $M_i$  plaintext (32 bits):  $16*32=512$  bits
  - $A, B, C, D$ : current hash -or IV-:  $4*32=128$  bits

$K_i$ : constants

$F$ : non linear box,

$$\begin{array}{|c|c|}\hline & + \mod 2^{32} \\ \hline \end{array}$$

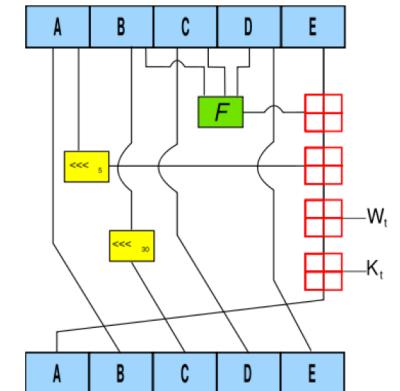
- First collisions found in 2004 [Wang, Fei, Lai, Hu]
  - No more security guarantees
  - Easy to generate two texts with the same MD5 hash



# Secure Hash Algorithms SHA

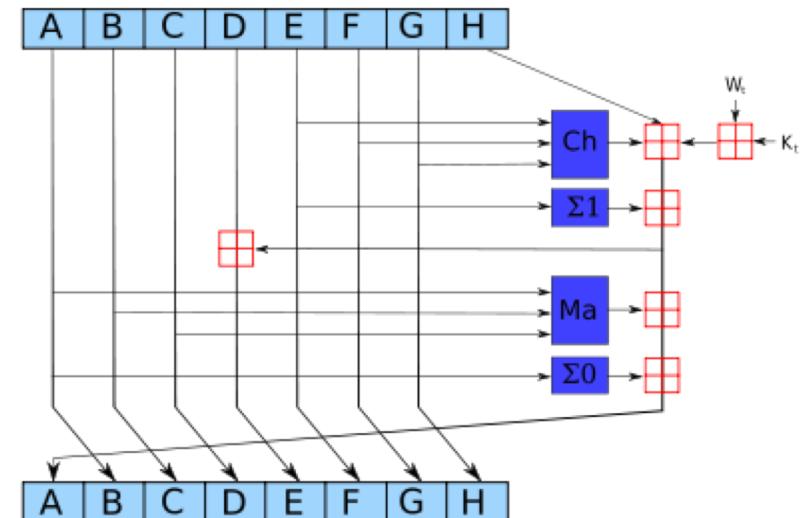
- SHA1: n=512, k=160; 80 rounds with 32 bits words:

- $W_t$  plaintext (32 bits;  $16 \times 32 = 512$  bits)
- A,B,C,D,E: current hash -or IV-:  $5 \times 32 = 160$  bits
- $K_t$ : constants
- F: non linear box,  $+ \text{mod } 2^{32}$
- Weaknesses found from 2005
  - $2^{35}$  computations [BOINC...]



- SHA2: 4 variants: k=224/384/256/512

- k=Size of the digest
- SHA-256: n=512, k=256
  - 64 rounds with 32 bits words
  - Message length  $< 2^{64}-1$
  - SHA-224: truncated version
- SHA-512: n=1024, k=512
  - 80 rounds with 64 bits words
  - Message length  $< 2^{128}-1$
  - SHA-384: truncated version

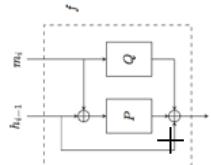


# **SHA-3 initial timeline (the Secure Hash Standard)**

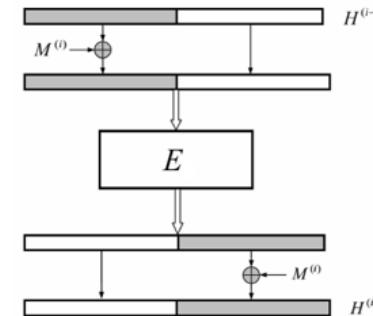
- **April 1995** FIPS 180-1: SHA-1 (revision of SHA, design similar to MD4)
- **August 2002** FIPS 180-2 specifies 4 algorithms for 160 to 512 bits digest message size  $< 2^{64}$ : SHA-1, SHA-256 ;  $< 2^{128}$  : SHA-384, and SHA-512.
- **2007** FIPS 180-2 scheduled for review
  - **Q2- 2009** First Hash Function Candidate Conference
  - **Q2- 2010** Second Hash Function Candidate Conference
- **Oct 2008** FIPS 180-3 [http://csrc.nist.gov/publications/fips/fips180-3/fips180-3\\_final.pdf](http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf) specifies 5 algorithms for SHA-1, SHA-224, SHA-256, SHA-384, SHA-512.
- **2012:** Final Hash Function Candidate Conference
- **2 October 2012 :** SHA-3 is **Keccak** (pronounced “catch-ack”).
  - Creators: Bertoni, Daemen, Van Assche (STMicroelectronics) & Peeters (NXP Semiconductors)

# The five SHA3 finalists

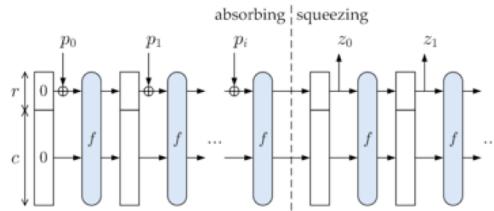
- BLAKE
  - New extension scheme (HAIFA) + stream cipher (Chacha)



- Grøstl
  - Compression function (two permutations)  
Merkle-Damgård extension + output transformation (Matyas-Meyer-Oseas)

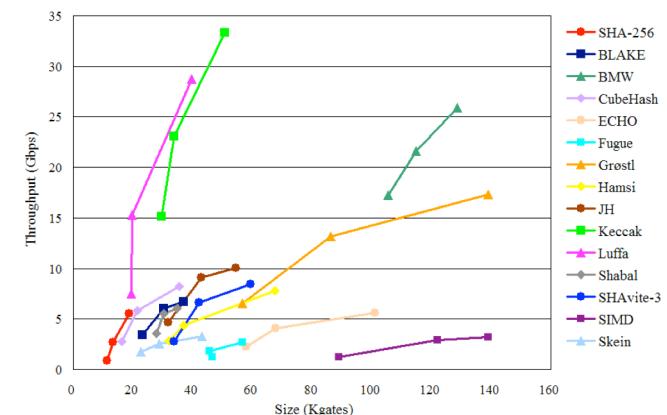


- JH
  - New extension scheme + AES/Serpent cipher



- Keccak
  - Extension « sponge construction » + compression

- Skein
  - Extension « sponge construction » + Threefish block cipher



## SHA-3 : Keccak

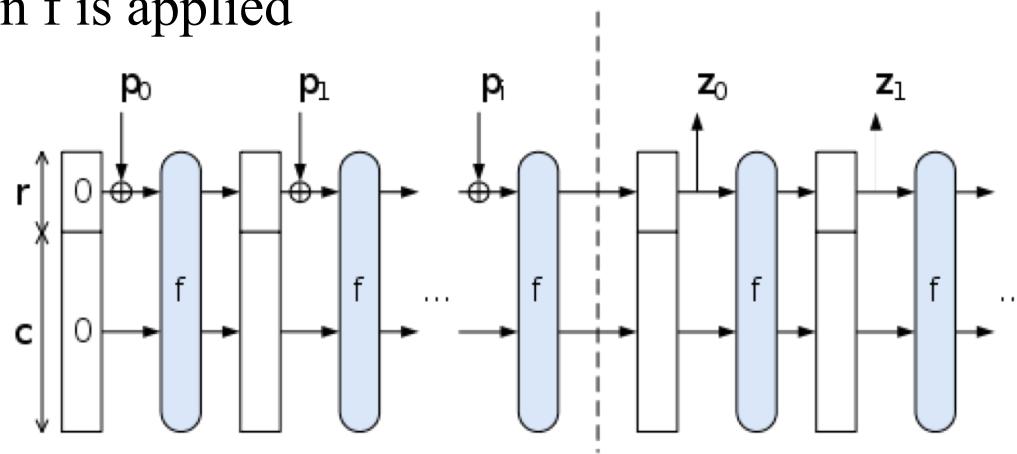
- Alternate, non similar hash function to MD5, SHA-0 and SHA-1:
  - Design : block permutation + Sponge construction
- But not meant to replace SHA-2
- Performance 12.5 cycles per byte on Intel Core-2 cpu; efficient hardware implementation.
- Principle (sponge construction):
  - message blocks XORed with the state which is then permuted (one-way one-to-one mapping)
  - State = 5x5 matrix with 64 bits words = 1600 bits
  - Reduced versions with words of 32, 16, 8, 4, 2 or 1 bit

# Keccak block permutation

- Defined for  $w = 2^\ell$  bit ( $w=64$ ,  $\ell = 6$  for SHA-3)
- State =  $5 \times 5 \times w$  bits array : notation:  $a[i, j, k]$  is the bit with index  $(i \times 5 + j) \times w + k$  (*arithmetic on i, j and k is performed mod 5, 5 and w*)
- block permutation function =  $12+2\ell$  iterations of 5 subrounds :
  - $\theta$ : xor each of the  $5w$  columns of 5 bits parity of its two neighbours :  
$$a[i][j][k] \oplus= \text{parity}(a[0..4][j-1][k]) \oplus \text{parity}(a[0..4][j+1][k-1])$$
  - $\rho$ : bitwise rotate each of the 25 words by a different number, except  $a[0][0]$  for all  $0 \leq t \leq 24$ ,  $a[i][j][k] = a[i][j][k-(t+1)(t+2)/2]$  with
$$\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 1 & 0 \end{pmatrix}^t \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$
  - $\pi$ : Permute the 25 words in a fixed pattern:  $a[3i+2j][i] = a[i][j]$
  - $\chi$ : Bitwise combine along rows:  $a[i][j][k] \oplus= \neg a[i][j+1][k] \& a[i][j+2][k]$
  - $\iota$ : xor a round constant into one word of the state. In round  $n$ , for  $0 \leq m \leq \ell$ ,  
$$a[0][0][2^m-1] \oplus= b[m+7n]$$
 where  $b$  is output of a degree-8 LFSR.

# Sponge construction = absorption+squeeze

- To hash variable-length messages by  $r$  bits blocks ( $c = 25w - r$ )
- Absorption:
  - The  $r$  input bits are XORed with the  $r$  leading bits of the state
  - Block function  $f$  is applied



- Squeeze:
  - $r$  first bits of the states produced as outputs
  - Block permutation applied if additional output required
- « Capacity » :  $c = 25w-r$  bits not touched by input/output
  - SHA-3 sets  $c=2n$  where  $n = \text{size of output hash}$  (1 step squeeze only)
- Initial state = 0. Input padding = 10\*1

# Provable secure hash functions

- Due to birthday paradox, the expected number of k-bit hashes that can be generated before getting a collision is  $2^{k/2}$ 
  - Security of a hash function with 128 bits digest cannot be more than  $2^{64}$
- Choose a provable secure compression function  $F : \{0,1\}^{k+r} \rightarrow \{0,1\}^k$ 
  - eg Chaum-van Heijst-Pfitzmann (discrete logarithm, cf exercise)
  - Or based on a (provably secure) symmetric block cipher  $E_K$   
eg Matyas-Meyer-Oseas; Davies-Meyer; Miyaguchi-Preneel; Meyer-Shilling (MDC2)
  - Or ...
- Choose a provable secure extension scheme to build  $h_F$  from  $F$ 
  - Eg: Merkle scheme:  $h_F(x \parallel b_1..b_r) = F(h(x) \parallel b_1..b_r)$  [cf course]
  - Or (usually when  $k=r$ ):  $h_F(x \parallel y) = F(h_F(x) \parallel h_F(y))$  [cf exercise]
  - And use an initial value IV of k bits to initialize the scheme  
$$h_F(b_1..b_r) = F(IV \parallel b_1..b_r)$$

## **Annexe**

### **Partage de secret**

Comment stocker une information critique de manière distribuée?

## Problème du Partage de secret « k parmi n »:

- S secret partagé entre n entités :
  - S est connu d'un tiers de confiance
  - S est représenté par  $D_1, \dots, D_n$  avec  $D_i$  secret de l'entité i
  - La connaissance de k (ou plus) valeurs  $D_i$  permet de calculer facilement S
  - La connaissance de moins de  $k-1$   $D_i$  rend S complètement inconnu (toutes les valeurs de S sont équiprobables : « secret parfait »)

# Protocole de partage de secret de Shamir

- Soit  $F$  un corps fini (de grande taille) tel que  $S$  est représenté de manière unique dans  $F$ 
  - On suppose  $\text{Prob}(S=x) = 1/\text{card}(F)$  [pourquoi ?]
- **Proocole de Shamir:**
  - Soit  $f(X) = S + a_1 \cdot X + a_2 \cdot X^2 + \dots + a_{k-1} \cdot X^{k-1}$  avec  $a_1, \dots, a_k$  choisis uniformément dans  $F$  (on pose  $a_0=S$ )
  - Soient  $n$  points distincts  $w_i \neq 0$  dans  $F$   
(par exemple  $w_i = i$  si  $F$  est de caractéristique  $> n$ , ou  $w_i = g^i$  etc)
  - Chaque entité  $i$  reçoit du tiers  $(w_i, f(w_i))$
- **Reconstruction du secret par  $k$  entités :**
  - Par interpolation de  $f$  (de dgré  $k-1$ ) à partir de  $k$  valeurs : TCR
  - Si moins de  $k-1$  valeurs: alors toutes les valeurs de  $S$  sont équiprobables
- De plus: possibilité de reconstruction en présence d'erreurs: avec  $k+r$  valeurs si  $r \geq 2.\#\text{erreurs}$

## Propriétés du protocole de Shamir

- **Secret parfait** (indistingabilité, comme OTP)
- Minimal: la taille de chaque  $D_i$  n'est pas plus grande que la taille de  $S$
- **Dynamique** On peut changer de polynôme de temps en temps
- **Extensible** : ajout possible de participants
- **Flexible**: les entités prioritaires peuvent avoir plus de valeurs
- Mais requiert la confiance dans le tiers qui distribue les valeurs !

# Architectures de confiance

1. Intégrité et redondance - Signatures  
(codes détecteurs d'erreurs et hash)
2. **Confiance centralisée:**
  - Infrastructure à clef privée (Kerberos)
  - Infrastructure à clef publique ( PKI )
3. Confiance distribuée:
  - Hiérarchie de confiance (Web of trust, Open PGP)
  - Registre = liste unique (ledgers)

# Certificats Numériques

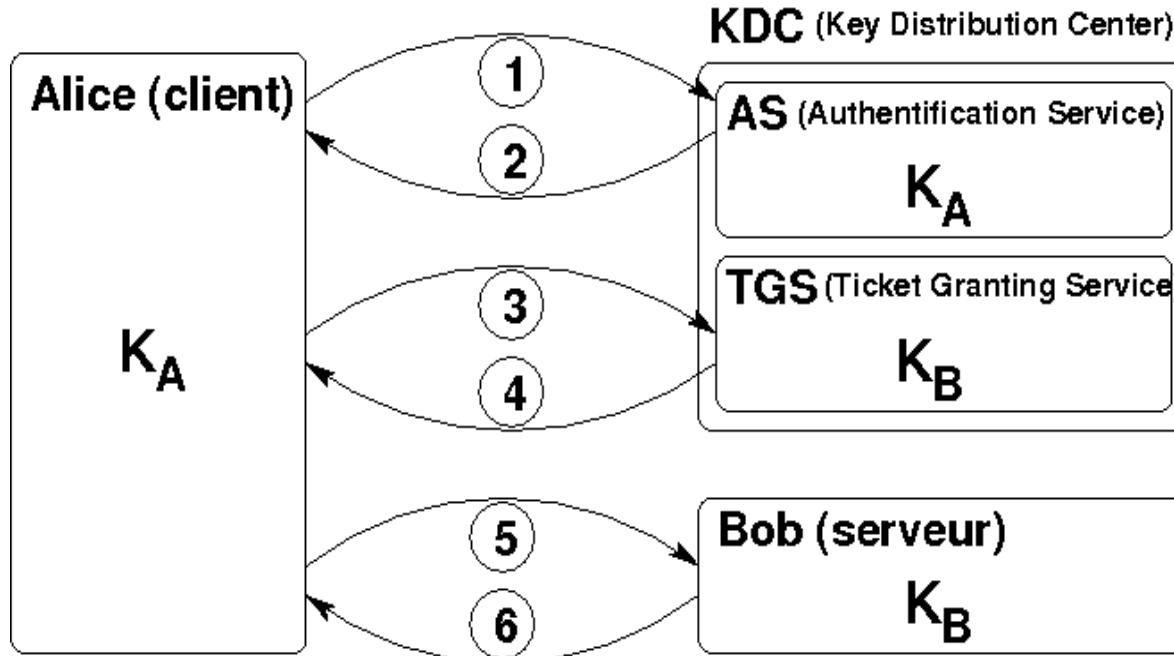
- Comment rattacher une clef publique à son propriétaire ?
  - Certificat numérique : association identité + clef
- Comment authentifier un certificat ?
  - Les certificats sont signés par des Autorités
- Comment récupérer un certificat ?
  - Des annuaires sont maintenus
- Comment gérer / sécuriser ce mécanisme
  - Auto signatures des Autorités
  - Architectures à clef publiques : PKI ...

# Architecture de confiance

- Utilisation d'un serveur de clef central pour éviter les attaques par le milieu : KDC (*Key Distribution Center*)
  - Architecture PKI (Public Key Infrastructure) qui atteste l'authenticité des clefs publiques
  - Ou Serveur central qui délivre des clefs de session
- Challenge
  - KDC envoie un « challenge », nombre aléatoire que le client doit (dé)crypter avec sa clef secrète : authentification
- Tiers de confiance
  - Pas besoin d'échanger entre tous les utilisateurs possibles, juste avec le KDC
  - Le KDC peut alors distribuer des clefs de session ciblées
- Horodatage
  - Empêche les re-jeux

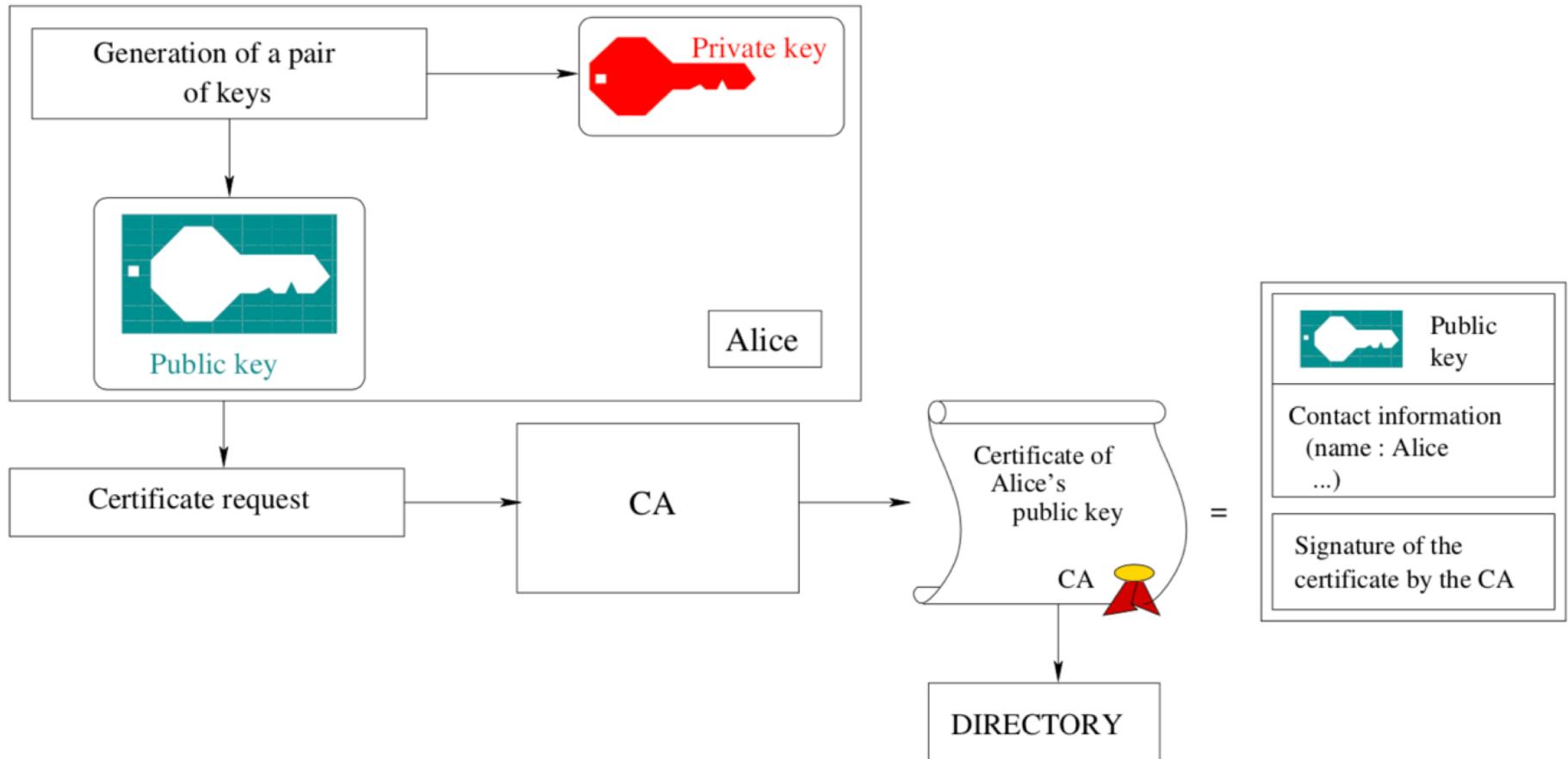
# Infrastructure à clefs secrètes (symétriques)

## Exemple: Kerberos



- Kerberos = Challenge + Tiers + Horodatage
  1. Bonjour, je suis A et je veux avoir un ticket de TGS pour cette transaction avec B
  2. Voici la clef et ton ticket d'authentification à TGS :  $E_{KA}(K_{A,TGS}) \parallel E_{KT}(T_{A,B})$  avec  $T_{A,B} = B \parallel A \parallel \text{date} \parallel \text{échéance}$  : ticket qu'Alice utilise pour communiquer avec Bob.
  3. Je veux parler à Bob || réponse challenge datée
  4.  $K_{A,B} \parallel$  Ticket daté pour Bob contenant  $E_{KB}(K_{A,B})$
  5. Ticket pour Bob || Je suis Alice daté
  6. Bonjour Alice daté

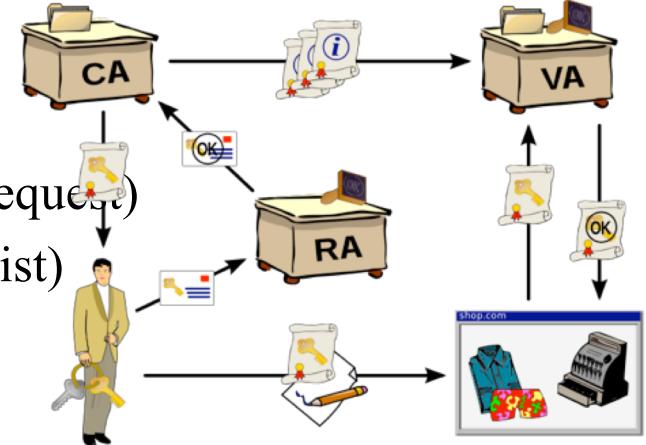
# Infrastructure à clef publique (PKI) : principe



# Infrastructure à clefs publiques (PKI)

Caractérisée par 4 entités [norme IETF]

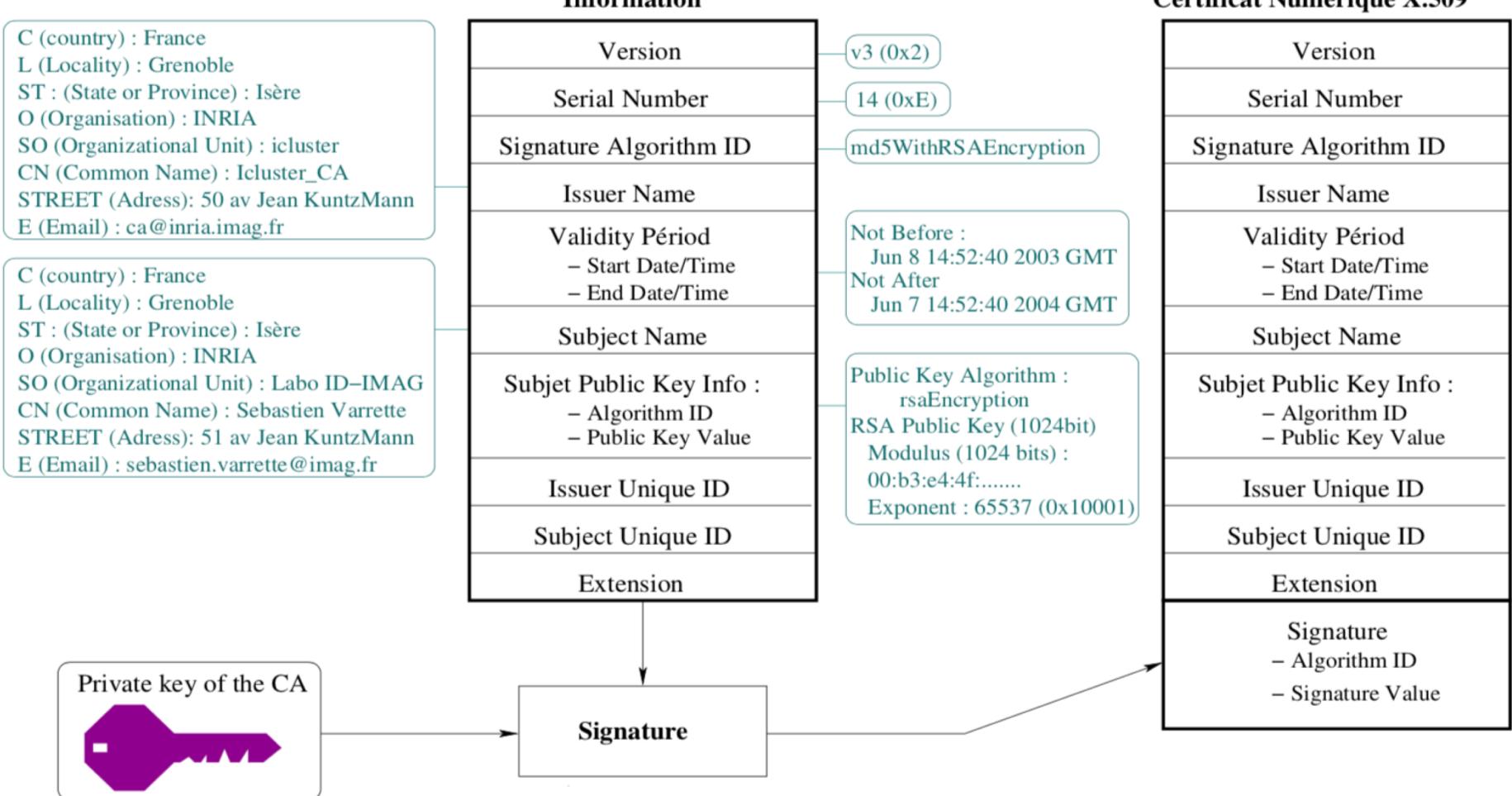
- **Autorité de certification (CA: Certificate authority)**
  - signer les demandes de certificat (CSR Certificate Signing Request)
  - signer les listes de révocation (CRL Certificate Revocation List)
- **Autorité d'enregistrement (RA Registration authority)**
  - générer les certificats (RA)
  - Valider les informations du certificat (VA): identité utilisateur, ...  
(NB les certificats numériques sont nominatifs et uniques pour l'ensemble de la PKI).
- **Autorité de dépôt (Repository)**
  - stocker les certificats numériques et les listes de révocation (CRL).
- **Entité finale (EE : End Entity) : le « sujet »**
  - Utilisateur ou système qui est le sujet d'un certificat



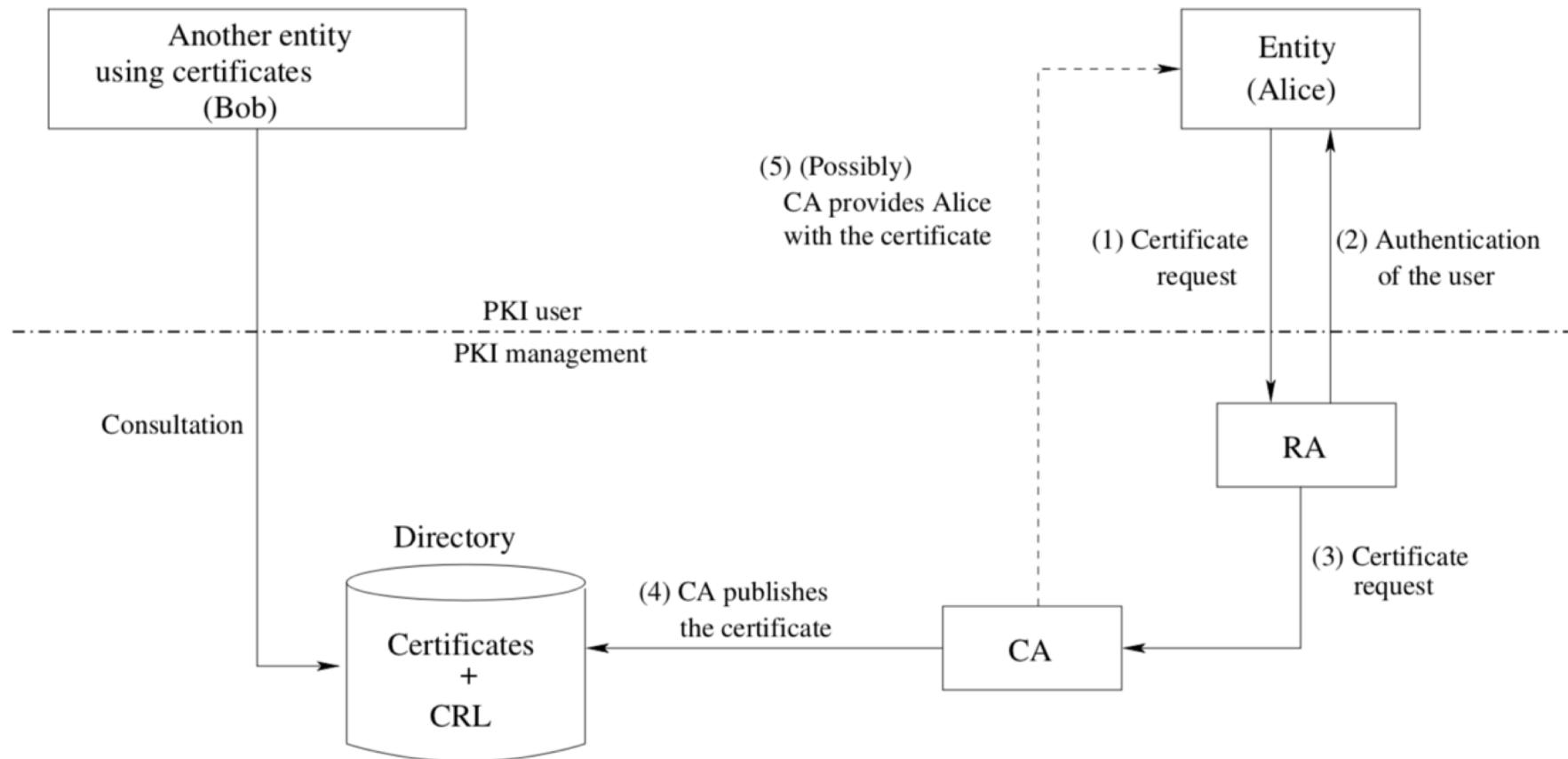
*En complément* (non spécifié par IETF) **Autorité de séquestre (Key Escrow) :**

- stocker de façon sécurisée les clés de chiffrement générées par la PKI pour les restaurer éventuellement.
- France: obligation de fournir aux autorités un moyen de déchiffrer les données chiffrées d'un utilisateur de la PKI.

# Certificat X509

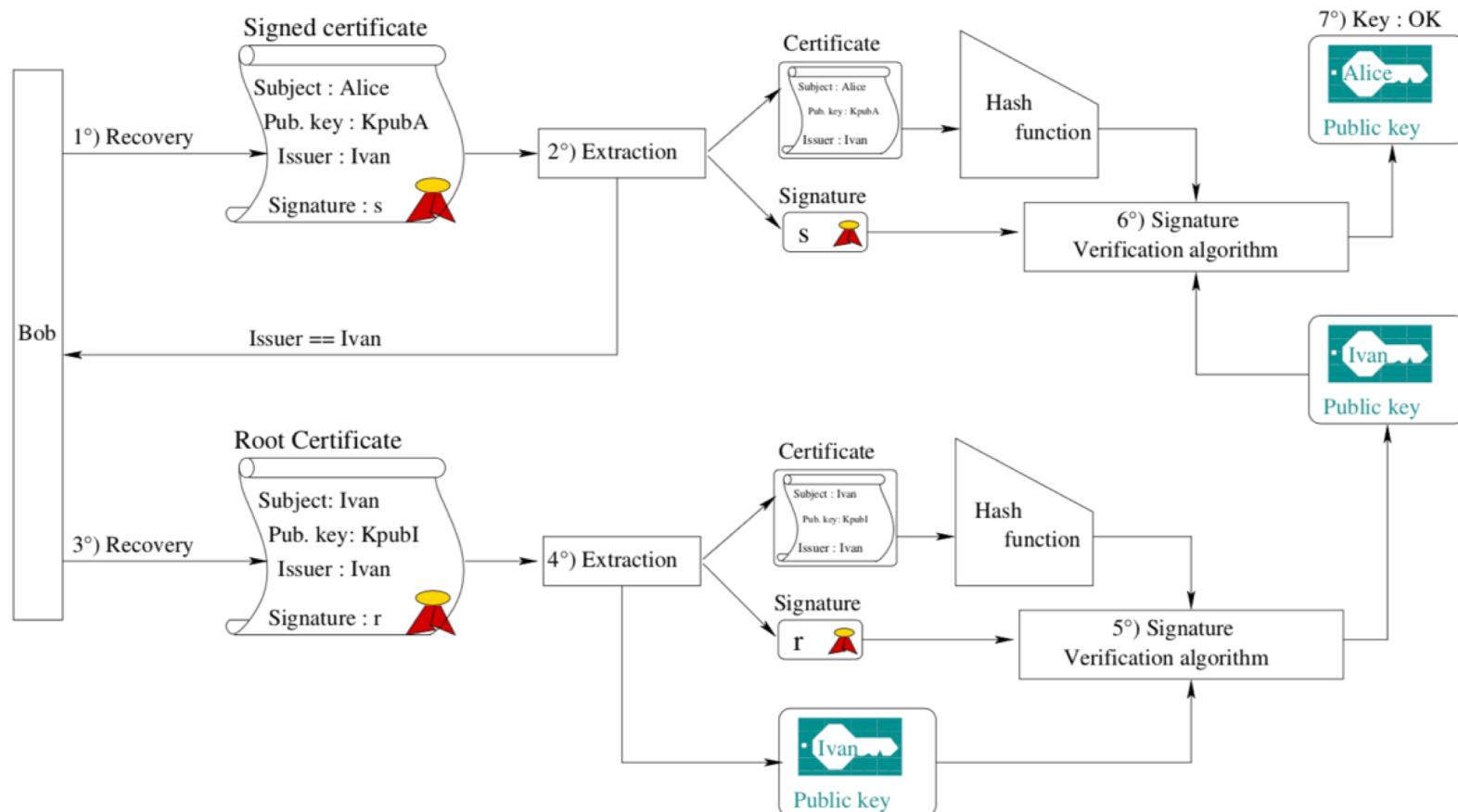


# Certificat : création, vérification



# Certificats et Root of trust

- PKI à clef publique (classique)
  - PKIX (certificats X509)
- Confiance en la racine du système (Root of Trust)



# Architectures de confiance

1. Intégrité et redondance - Signatures  
(codes détecteurs d'erreurs et hash)
2. Confiance centralisée:
  - Infrastructure à clef privée (Kerberos)
  - Infrastructure à clef publique ( PKI )
3. **Confiance distribuée:**
  - Hiérarchie de confiance (Web of trust, Open PGP)
  - Registre = liste unique (ledgers)

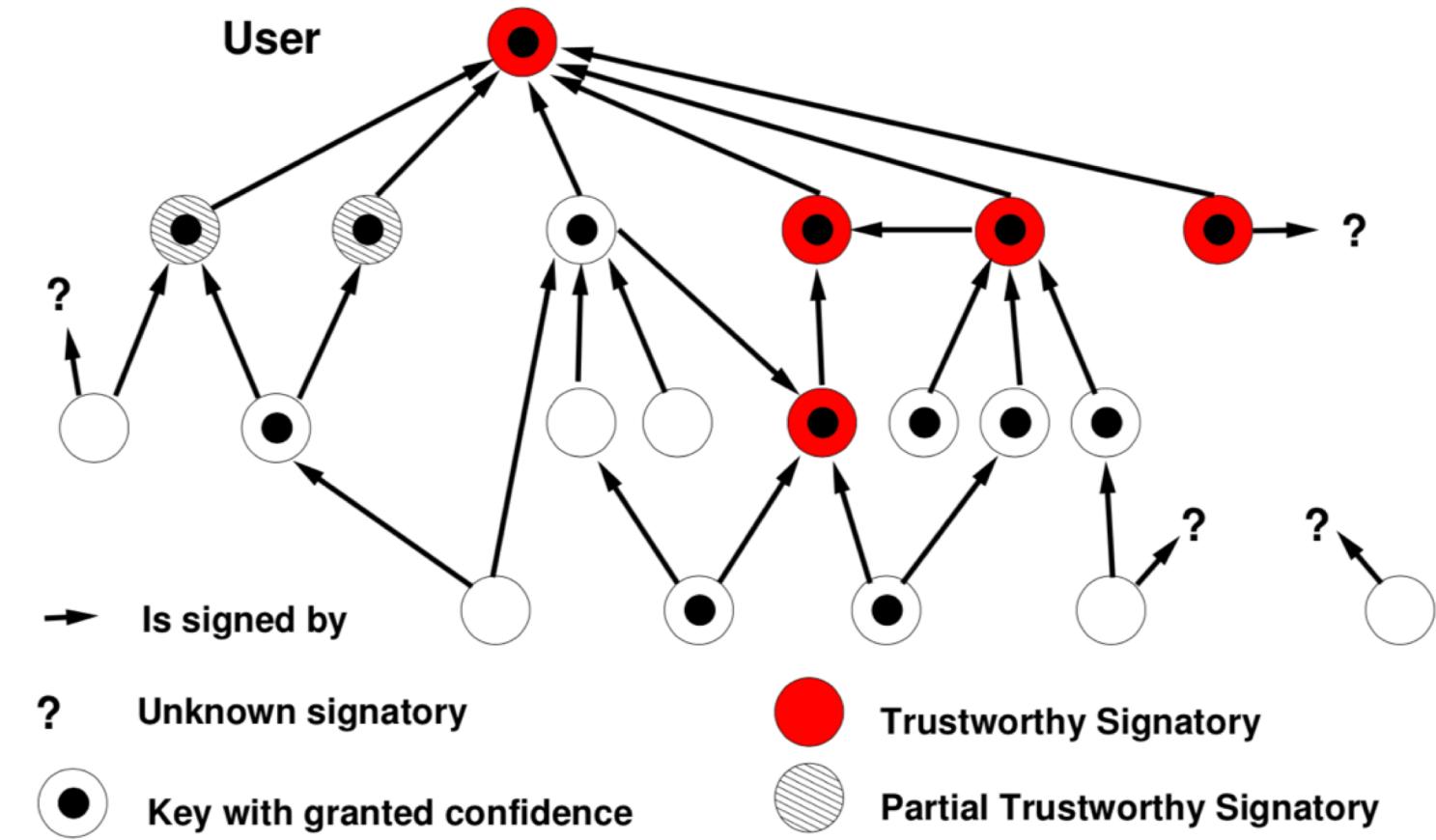
# Système hybride : clef secrète / clef publique

- Initiée par PGP [Zimmermann]
  - Cryptographie secrète : rapide mais problème de l'attaque par le milieu
  - Cryptographie publique : lente
- SSH
  1. Client contacte le serveur
  2. Serveur envoie ses clefs publiques H fixe et S changée régulièrement + challenge
  3. Client vérifie H correspond bien au Serveur (dans sa BD côté client)
  4. Client double crypte clef secrète de session par H et S
  5. Confirmation du serveur

⇒ Connexion sécurisée établie

# Infrastructures distribuées (Open PGP)

- Open PGP : chaque utilisateur peut signer les certificats
- Confiance aux certificats signés par des utilisateurs en qui on a confiance
- « Web of Trust »



# Architectures de confiance

1. Intégrité et redondance - Signatures  
(codes détecteurs d'erreurs et hash)
2. Confiance centralisée:
  - Infrastructure à clef privée (Kerberos)
  - Infrastructure à clef publique ( PKI )
3. Confiance distribuée:
  - Hiérarchie de confiance (Web of trust, Open PGP)
  - **Registre = liste unique (ledgers)**