Les Algorithmes Cryptographiques Asymétriques

Omar Cheikhrouhou
Omar.cheikhrouhou@isetsf.rnu.tn

ISET SFAX, 2009-2010

Plan

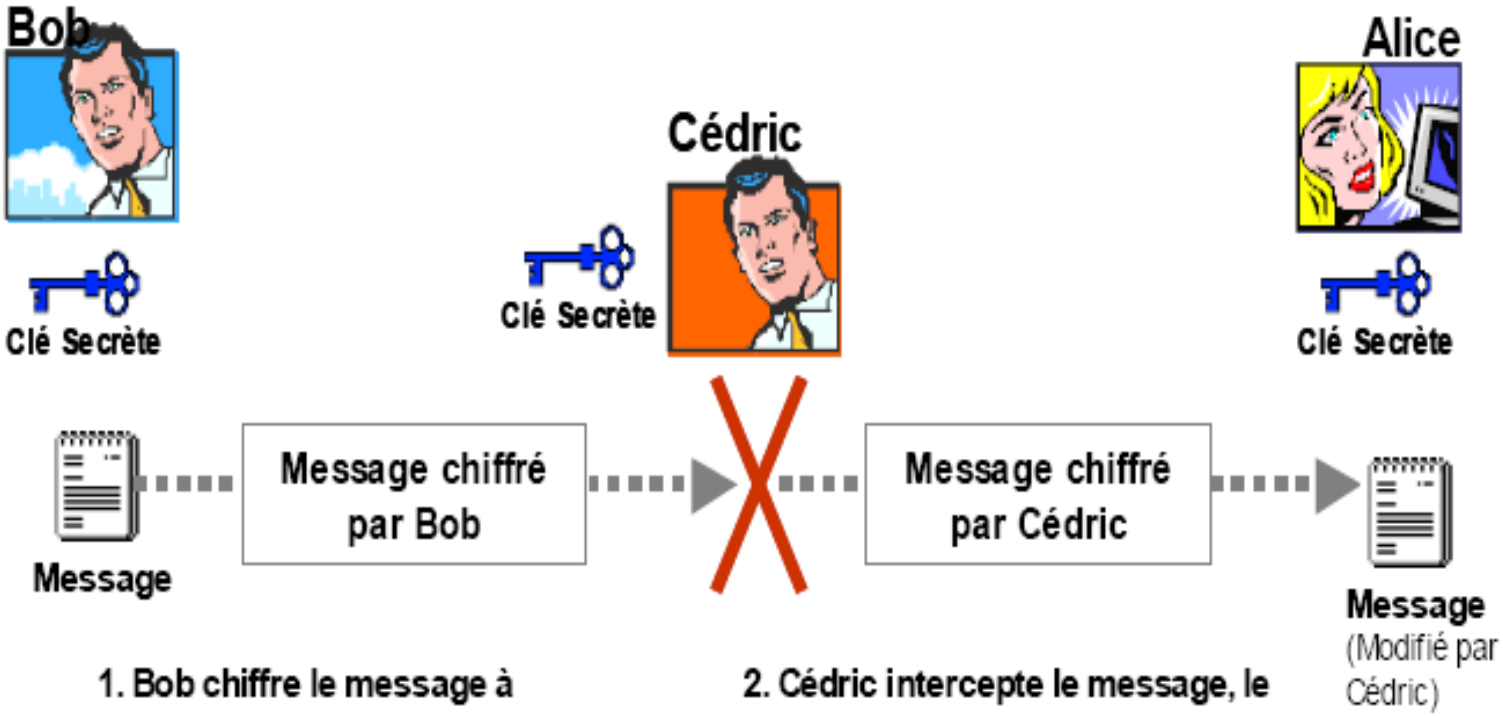
- *Introduction*
- *Principe*
- *Description*
- *Performances*
- *Modes d'utilisation*
- Algorithmes Cryptographiques Asymétriques
  - RSA
  - ElGamel
  - ECC
- Public Key Infrastructure (PKI)
  - Certificats Électroniques
  - Signature numérique

Principe de Kerckhoffs:

La sécurité d'un système cryptographique ne doit pas reposer sur la non divulgation des fonctions de chiffrement et de déchiffrement utilisées mais sur la non divulgation des clés utilisées pour les paramétrer.

Cryptage symétrique

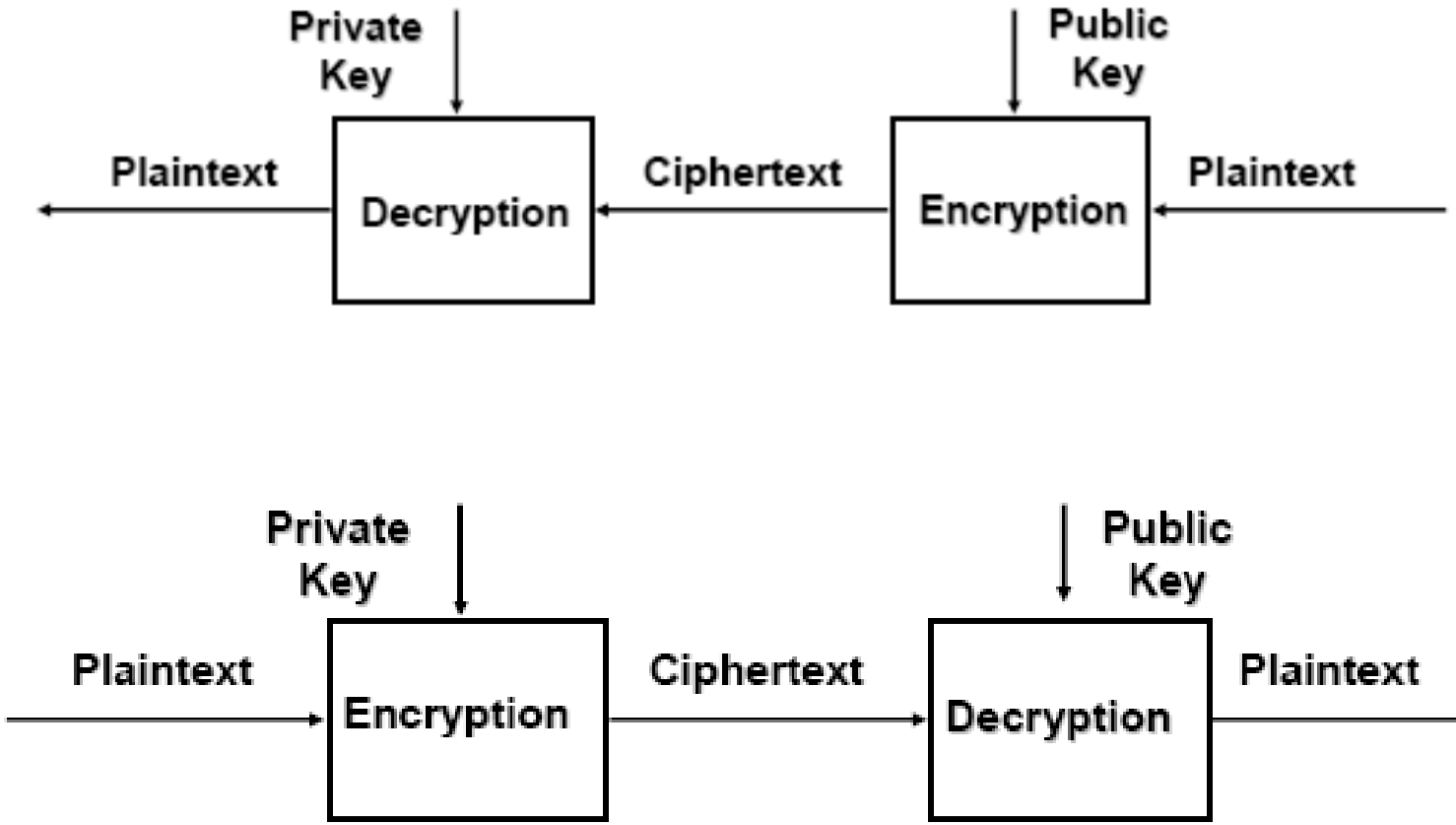
- Limitation: Pas d'intégrité et d'identification de l'auteur
- Si Alice, Bob et Cédric partage le même lien de communication alors ils partagent la même clé de chiffrement symétrique.



Cryptage asymétrique (à clé publique)

- Utilisation d'une paire de clés:
  - Publique: Connue par tout le monde, utilisée généralement pour **crypter** ou **vérifier** la signature des messages.
  - Privée: Connue uniquement par le détenteur, utilisée pour **décrypter** et **signer** des messages.
- Impossible de trouver la clé privée à partir de la clé publique.
- Exemples: RSA, Diffie-Hellman, El Gamal.
- Généralement dix fois plus lent que le cryptage symétrique.
- Utilisé généralement pour
  - Cryptage / décryptage: assurer la confidentialité.
  - Signature numérique: assurer l'authentification et la non répudiation.
  - Distribution de clés: se mettre d'accord sur une clé de session.
- Clés à grande taille (ex: RSA: 1024-2048-…)

Cryptage asymétrique



RSA

RSA

- Développé par Rivest, Shamir & Adleman à MIT en 1977, publié en 1978
- Le plus connu et le plus utilisé comme algorithme de cryptage asymétrique.
- Breveté par RSA, et cette patente a expiré en 2000.
- Utilise des entiers très larges 1024+ bits
- Le fonctionnement du cryptosystème RSA est basé sur la difficulté de factoriser de grands entiers.

Rappel

- ***A mod B*** est le reste de la division entière de A par B
- **La multiplication et le modulo**

$$(A \bmod B)\,(C \bmod B) = A * C \bmod B$$

- **L'exponentielle et le modulo**

$$a^n \bmod m = (a \bmod m)^n \bmod m$$

- Pierre **Fermat** :

Si on utilise un nombre premier comme module, alors quand on élève un nombre à la puissance (nombre premier -1), on obtient 1

  - Pour n'importe quel nombre ***m*** et pour ***p*** premier :

$$m^{(p-1)} \bmod p = 1$$

  - Exemple : $7^{10} \bmod 11 = 1$ …pas besoin de calcul car 11 est premier

Rappel

- Leonhard **Euler** :
  - Lorsqu'on utilise un module comme étant le produit de deux nombres premiers on a :

    Soit n = p * q, avec p et q premiers, et quelque soit m

    $$m^{(p-1)(q-1)} \bmod n = 1$$

  - Exemple : soit p = 11 et q = 5, n = 55 et (p – 1)(q – 1) = 10 * 4 = 40

    $$38^{40} \bmod 55 = 1$$

  - Si on manipule le résultat d'Euler en multipliant par m l'équation :

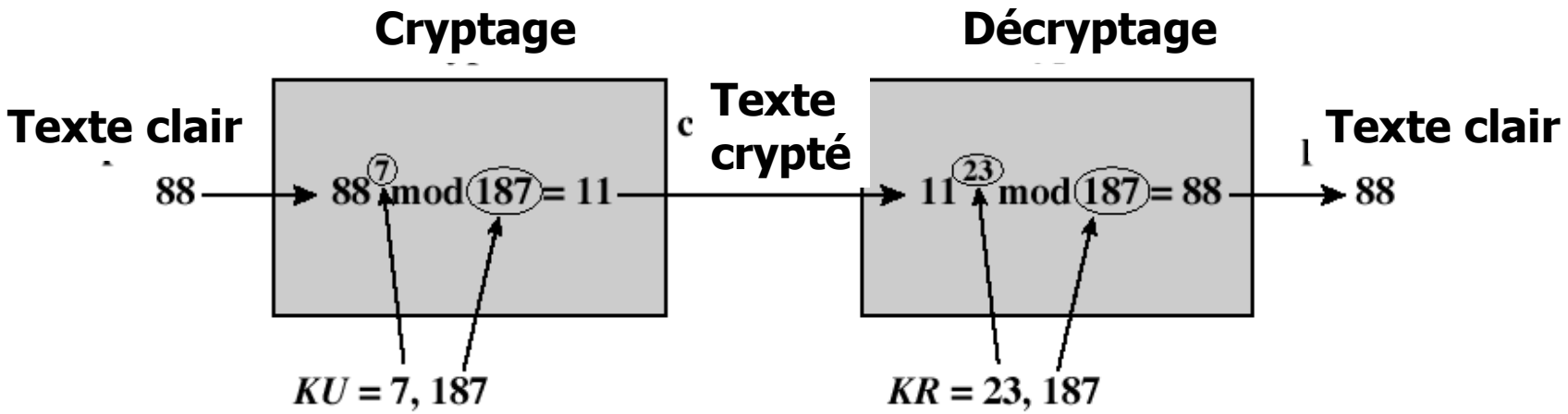
    $$m * m^{(p-1)(q-1)} \bmod n = m$$

    $$m^{(p-1)(q-1)+1} \bmod n = m$$

RSA: Algorithme

- Étapes
  1. Sélectionner deux entiers premiers entre eux « p » et « q »
  2. Calculer n = p x q
  3. Calculer $\varphi(n)=(p-1)(q-1)$
  4. Sélectionner « e » tel que: *pgcd*$(\varphi(n),e)=1$; $1<e<\varphi(n)$
     - *En général « e » est un entier de petite taille.*
  5. Calculer $d=e^{-1} \bmod \varphi(n)$. En d'autre terme: $d.e = 1 \bmod (\varphi(n))$
  6. Clé publique: $K_{pu}=\{e,n\}$
  7. Clé privée $K_{pr} = \{d,n\}$
- Pour crypter un message M < n, l'émetteur:
  - Obtient une clé publique du récepteur et calcule « $C= M^e \bmod n$ »
- Pour décrypter un message crypté C le récepteur
  - Utilise sa clé privée et calcule « $M = C^d \bmod n$ »

RSA: Algorithme



- p = 17, q = 11, n = p x q= 187
- Φ(n) = 16 x 10 =160,
- Choisir e = 7,
- d.e =1 (mod Φ(n)) ➔ d = 23

Preuve de RSA

- $D(E(M)) = (M^e \bmod n)^d \bmod n$

  $= M^{e.d} \bmod n$

- On a: $e.d = 1 \pmod{\varphi(n)}$

  $= z \times \varphi(n) + 1$

- $M^{e.d} = M^{z \times \varphi(n) + 1}$

  $= (M^z)^{\varphi(n)} \times M$

  $= 1 \times M \pmod n$

- Par hypothèse RSA crypte des blocks de données de taille inférieure à n (décomposition en blocks)

  ➔ $D(E(M)) = M$

Exemple d'utilisation de RSA

- **Création de la paire de clés:**
  - Soient deux nombres premiers au hasard: ***p*** = 29, ***q*** = 37, on calcule ***n*** = ***pq*** = 29 * 37 = 1073.
  - On doit choisir ***e*** au hasard tel que ***e*** n'ai aucun facteur en commun avec ***(p-1)(q-1):***

    (p-1)(q-1) = (29-1)(37-1) = 1008
  - On prend ***e*** = 71
  - On choisit ***d*** tel que 71*d mod 1008 = 1, on trouve d = 1079.
  - L'instruction ***d=PowerMod[e,-1,(p-1)(q-1)]*** de *Mathematica* permet de calculer ***d*** facilement.
  - On a maintenant les clés :

- la **clé publique** est (***e,n***) = (71,1073) (*=clé de chiffrement*)

- la **clé privée** est (***d,n***) = (1079,1073) (*=clé de déchiffrement*)

Exemple d'utilisation de RSA

- **Chiffrement du message 'HELLO'.**
- On prend le code ASCII de chaque caractère et on les met bout à bout:

  m = 7269767679
- Il faut découper le message en blocs qui comportent moins de chiffres que n.
- n comporte 4 chiffres, on découpe notre message en blocs de 3 chiffres:

  726 976 767 900 (on complète avec des zéros)
- On chiffre chacun de ces blocs :
  - 726^71 mod 1073 = 436
  - 976^71 mod 1073 = 822
  - 767^71 mod 1073 = 825
  - 900^71 mod 1073 = 552
- *Le message chiffré est 436 822 825 552.*

Problèmes de RSA

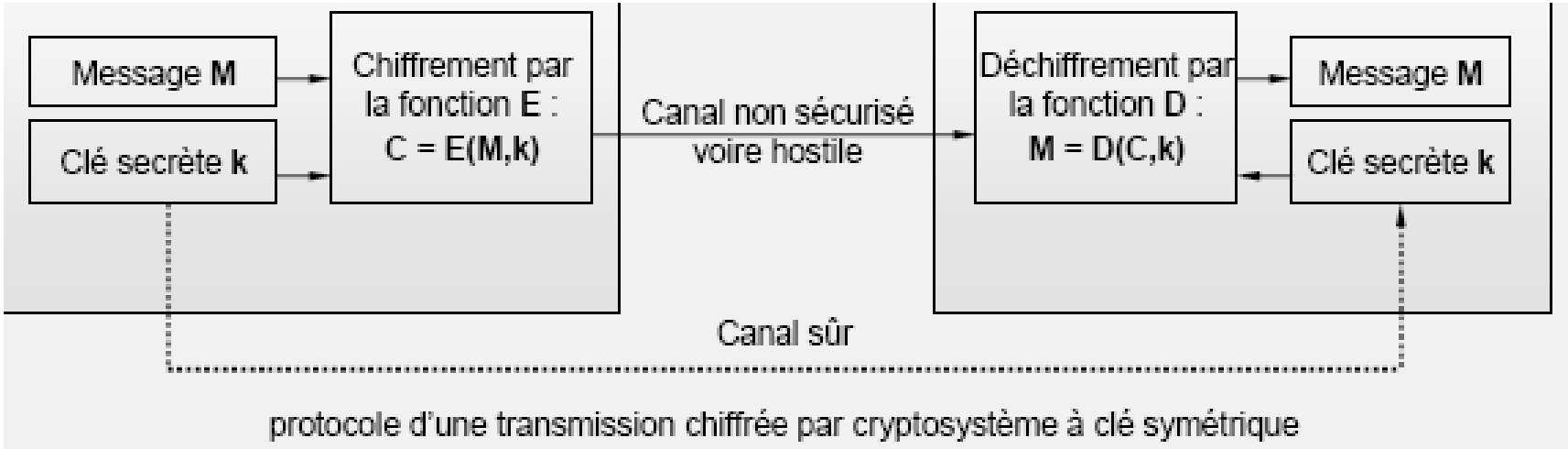
- Complexité algorithmique de la méthode:
  - recherche de nombres premiers de grande taille, et choix de clés très longue
  - Réalisation des opérations modulo n.
- Problème d'implémentation sur les équipements disposants de faible puissance de calcul (ex: cartes bancaire, stations mobiles, etc.)
- La méthode est officiellement sûre si des contraintes de longueur des clés et d'usage sont respectées.
- ➔ Solution: Utilisation de RSA pour l'échange des clés secrètes de session d'un algorithme symétrique à clés privées.

Comparaisons entre RSA et DES

- **RSA**
  - — clé de 40 bits
  - — chiffrement matériel : 300 Kbits/sec
  - — chiffrement logiciel : 21,6 Kbits/sec
  - — **Inconvénient majeur** : un pirate substitue sa propre clé publique à celle du destinataire, il peut alors intercepter et décrypter le message pour le recoder ensuite avec la vraie clé publique et le renvoyer sur le réseau. **«L'attaque» ne sera pas décelée.**
  - — **usage** : on ne les emploiera que pour transmettre des données courtes telles que les clés privées et les signatures électroniques.
- **DES**
  - — clé de 56 bits
  - — chiffrement matériel : 300 Mbits/sec
  - — chiffrement logiciel : 2,1 Mbits/sec
  - — **Inconvénient majeur** : attaque «brute force» rendue possible par la puissance des machines.
  - — **Usage** : chiffrement rapide, adapté aux échanges de données de tous les protocoles de communication sécurisés.

Échange sécurisé

- **Résolution du problème de l'échange des clés secrètes :** utilisation d'une méthode **hybride** combinant à la fois chiffrement **symétrique** et **asymétrique**.

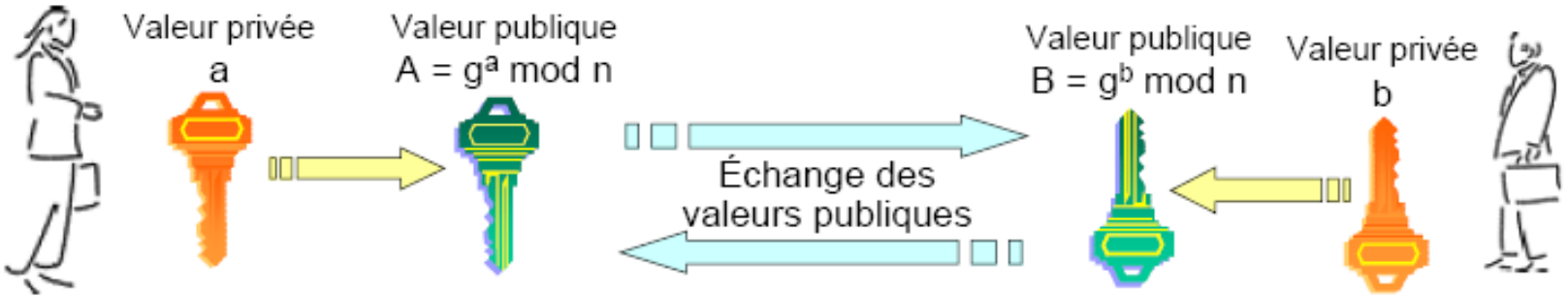


protocole d'une transmission chiffrée par cryptosystème à clé symétrique

- **Avantages :**
  - — la clé secrète est chiffrée et échangée ;
  - — après l'échange on bascule le chiffrement en utilisant un algorithme symétrique plus rapide ;
  - — on démarre l'échange avec l'utilisation d'un algorithme asymétrique qui possède l'avantage d'offrir un moyen d'identifier les interlocuteurs.

Échange sécurisé : la méthode Diffie - Hellman

- Qu'est-ce que le protocole DH ?
  - Protocole cryptographique qui permet à deux tiers de générer un secret partagé sans informations préalables l'un sur l'autre
- Principe
  - Échange de valeurs publiques



  - Permettant de générer un secret partagé

  - Un espion ne peut reconstituer le secret partagé à partir des valeurs publiques

Echange sécurisé : la méthode Diffie - Hellman

Déroulement de l'algorithme

1. Alice et Bob se mettent d'accord sur un grand entier n tel que $(n-1)/2$ soit premier et sur un entier g primitif par rapport à n. Ces deux entiers sont publics.
2. Alice choisit de manière aléatoire un grand nombre entier a, qu'elle garde secret, et calcule sa valeur publique, $A = g^a \bmod n$. Bob fait de même et génère b et $B = g^b \bmod n$.
3. Alice envoie A à Bob ; Bob envoie B à Alice.
4. Alice calcule $K_{AB} = B^a \bmod n$; Bob calcule $K_{BA} = A^b \bmod n$. $K_{AB} = K_{BA} = g^{ab} \bmod n$ est le secret partagé par Alice et Bob.

Une personne qui écoute la communication connaît g, n, $A=g^a \bmod n$ et $B=g^b \bmod n$, ce qui ne lui permet pas de calculer $g^{ab} \bmod n$: il lui faudrait pour cela calculer le logarithme de A ou B pour retrouver a ou b.

Signature Numérique

ElGamel

- L'**algorithme ElGamal** est un algorithme de cryptographie asymétrique basé sur les logarithmes discrets. Il a été créé par Taher Elgamal. Cet algorithme est utilisé par le logiciel libre GNU Privacy Guard, de récentes versions de PGP, et d'autres systèmes de chiffrement, et n'a jamais été sous la protection d'unbrevet contrairement à RSA. Il peut être utilisé pour le chiffrement et la signature électronique. L'algorithme DSA du NIST est basé sur ElGamal.

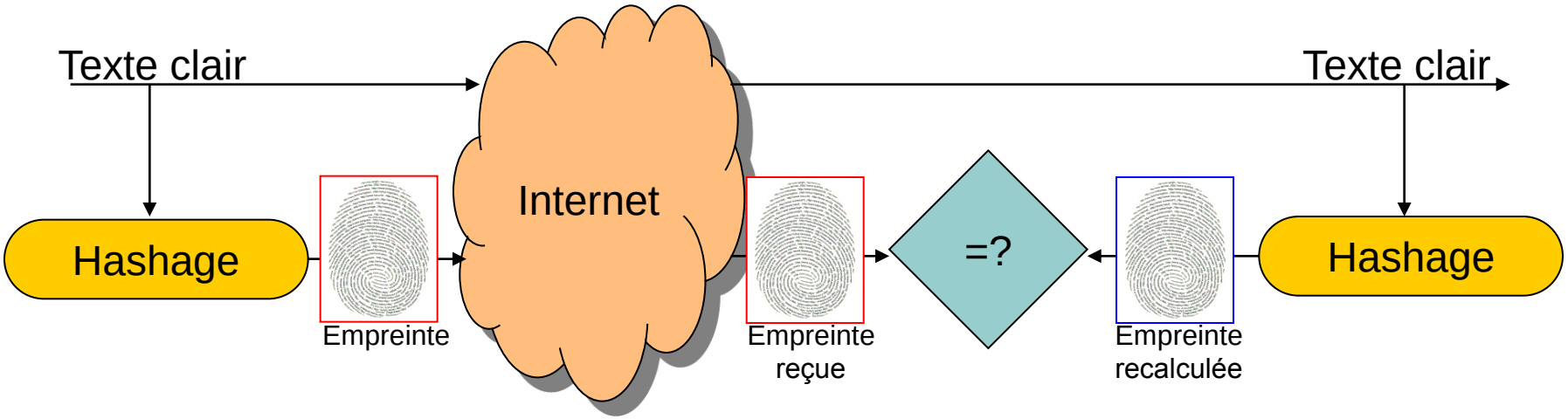
- L'algorithme fonctionne comme suit :
- Alice calcule avec pour un grand nombre premier p, g étant un élément générateur de , et divulgue sa *clé publique* (p,g,h). La valeur x est sa *clé privée*.
- Si Bob veut envoyer un message à Alice, il convertit d'abord son message sous la forme d'un nombre .
- Bob génère un nombre entier aléatoirement et calcule et . Il envoie (c_1,c_2) à Alice.
- Alice peut reconstruire le message initial m en calculant .
- On remarque que :

ECC (Elliptic Curve Cryptography)

Fonction de hachage: Propriété mathématique

- Entrée: message *M* avec contenu et taille arbitraire.
- Sortie: message de taille fixe *h=H(M).*
- *H(M)* est appelé condensât, ou empreinte, ou fingerprint, ou message digest
- Irréversible:
  - Étant donnée *h*, il est difficile de trouver *x* tel que: *h = H(x)*
  - Complexité de l'ordre de 2^n, *n* est le nombre de bits du *digest*.
- Résistance forte à la collision:
  - Étant donné *x*, il est impossible de trouver *y* avec *H(x) = H(y)*
  - Il est impossible de trouver une paire *x, y* tel que *H(x) = H(y)*
- Calcul facile et rapide (plus rapide que le cryptage symétrique).

Fonctions de Hashage : Principes



1)  = 

Empreinte reçue — Empreinte recalculée

Le texte reçu est intègre

2)  ≠ 

Empreinte reçue — Empreinte recalculée

Le texte reçu est altéré

Fonctions de Hashage : Exemples

- MD5 : *Message Digest 5*
  - Développé par Ron Rivest
  - Génère une empreinte de taille 128 bits
- SHA-1 : *Secure Hash Algorithm*
  - Développé par NIST en collaboration avec NSA
  - Génère une empreinte de taille 160 bits
  - Exemple

Introduction à la Cryptographie

```
5aa769e719f153611c3d0dbb4bb02e23
```

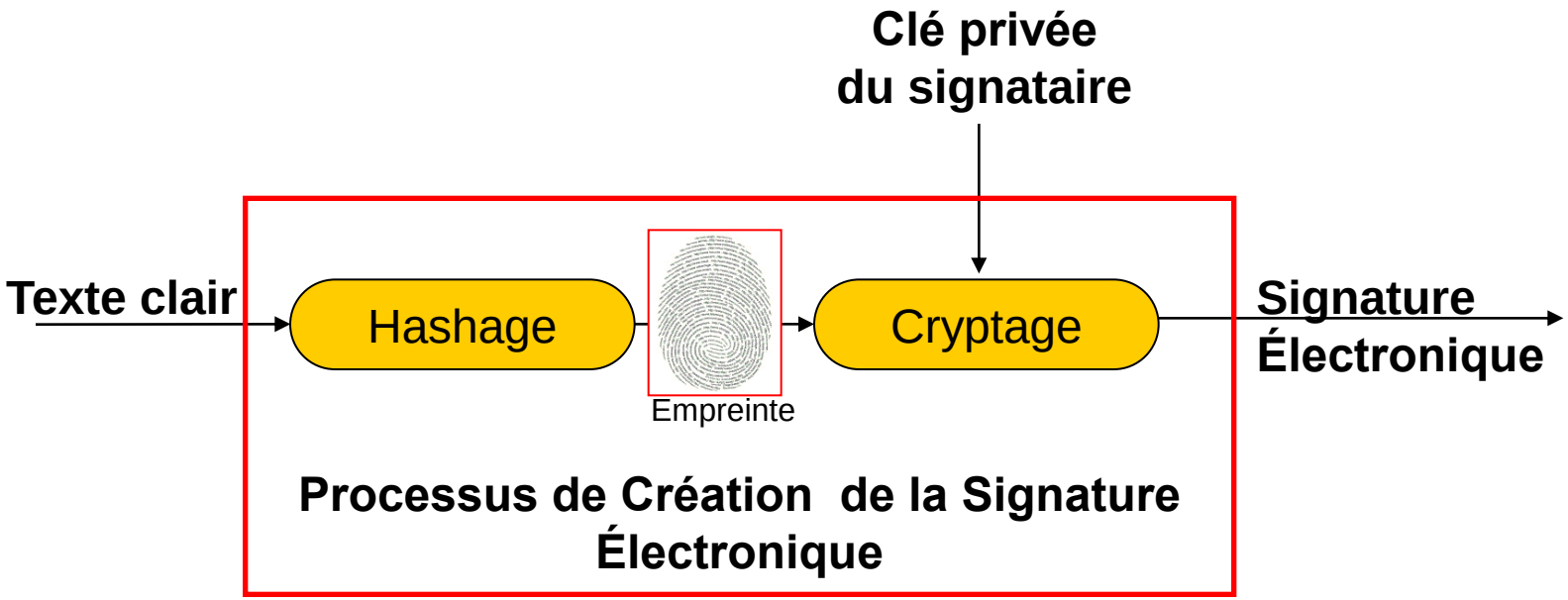
Introduction à la cryptographie

```
af575f3a9216b4158bdcd2c4201d6527
```

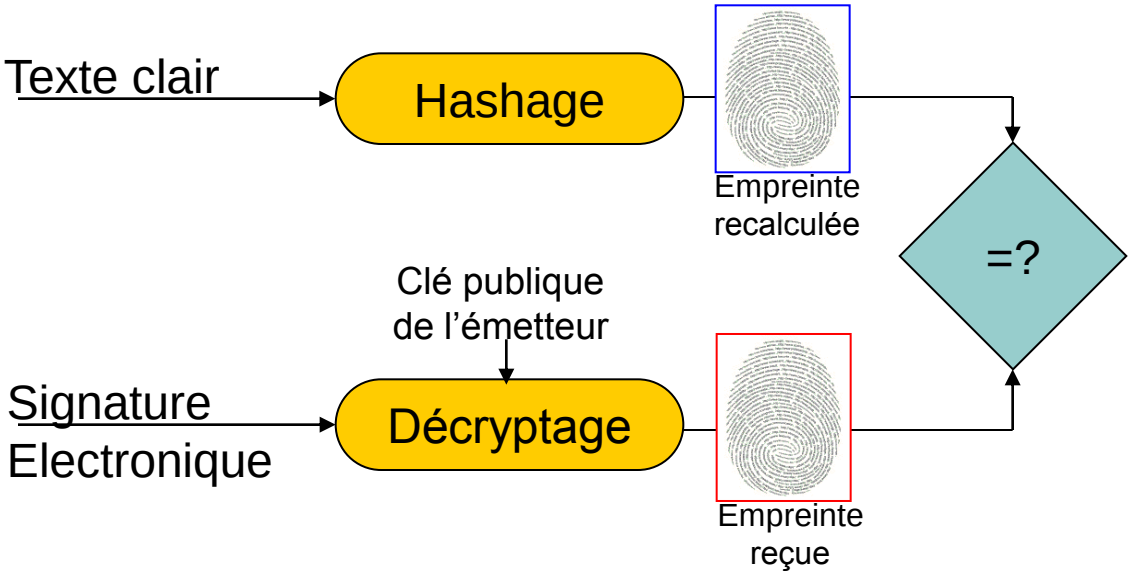
Signature numérique

- Idée clé:
  - Le *Hash* (résultat de la fonction de hachage) d'un message est crypté avec la clé privée de l'émetteur.
  - La clé publique est utilisée pour la vérification de la signature
- Soit:
  - M: message à signer, H: fonction de hachage
  - Kpr, Kpu: paire de clés privée / publique de l'émetteur.
  - E / D: fonction de cryptage / Décryptage en utilisant Kpu / Kpr.
- En recevant (M, $E_{Kpr}(H(M))$), le récepteur vérifie si: $H(M)=D_{Kpu}(E_{Kpr}(H(M)))$

Signature Électronique : Création



Signature Électronique : Vérification



1)  = 

Empreinte reçue — Empreinte recalculée

La signature reçue est correcte

2)  ≠ 

Empreinte reçue — Empreinte recalculée

La signature reçue est incorrecte

Signature numérique

- La signature permet de mettre en œuvre les services:
  - Intégrité du message
  - Authentification
  - Non-répudiation
  - Génération d'une clé de chiffrement symétrique pour le service de Confidentialité

Les standards PKCS

- PKCS : Public Key Cipher Systems
- Standards de la compagnie RSA
  - Format de stockage des clés privées
  - Format et extensions des certificats
  - Requête pour l ’obtention d ’un certificat
  - etc, ...
- PKCS#1 : RSA Encryption Standard
- PKCS#3 : Diffie-Helman Key Agreement Standard
- PKCS#5 : Password-Based Encryption Standard
- PKCS#6 : Extended-Certificate Syntax Standard
- PKCS#7 : Cryptographic Message Syntax Standard
- PKCS#8 : Private-Key Information Syntax Standard
- PKCS#9 : Selected Attribute Types
- PKCS#10 : Certification Request Syntax Standard
- PKCS#11 : Cryptographic Token Interface Standard
- PKCS#12 : Personal Information Exchange Syntax Standard