II-4-2) Chiffrement à clé publique

Dans un système de chiffrements asymétriques, chaque entité possède deux clés distinctes (une privée et une autre publique) avec impossibilité de déduire la clé privée à partir de la clé publique. De ce fait, il est possible de distribuer librement cette dernière.

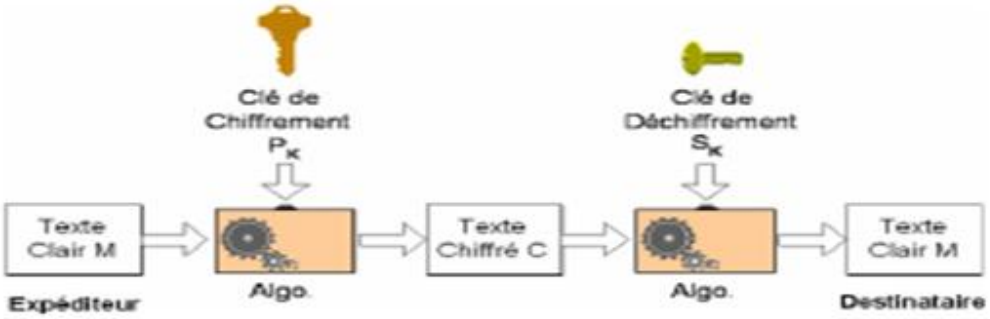


Figure 13 : Chiffrement à clé publique

On peut classer l'utilisation des algorithmes à clé publique en 3 catégories :

- ✓ Chiffrement/déchiffrement : cela fournit le secret.
- ✓ Signatures numériques : cela fournit l'authentification et non répudiation.
- ✓ Échange de clés (ou des clés de session).

Quelques algorithmes conviennent pour tous les usages, d'autres sont spécifiques à un d'eux.
Le concept date officiellement de 1976 de Diffie et Hellman. Officieusement, les bases existent depuis 1969 par Ellis. La première implémentation a eu lieu en 1978 par Rivest, Shamir et Adleman sous la forme de l'algorithme RSA bien que, là aussi, les fondements de ce système datent de 1973, par Cocks.

La sécurité de tels systèmes repose sur des problèmes calculatoires :

- ➢ RSA : factorisation de grands entiers
- ➢ ElGamal : logarithme discret
- ➢ Merkle-Hellman : problème du sac à dos (knapsacks)
- ➢ ...

RSA : Rivest - Shamir - Adleman

Il s'agit du système de chiffrement Asymétrique le plus connu et le plus largement répandu (1978). Il est basé sur le calcul exponentiel. Sa sécurité repose sur la fonction unidirectionnelle suivante : le calcul du produit de deux nombres premiers est aisé. La factorisation d'un nombre en ses deux facteurs premiers est beaucoup plus complexe.

a)Complément mathématique

Définition 1. Soient **a** et **b** deux entiers relatifs ; on dit que **b** divise **a** et on note **b|a** s'il existe un entier **c** tel que **a = b.c**

Définition 2. Un entier positif est premier s'il n'admet pas d'autres diviseurs que 1 et lui-même. 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, . . . sont premiers. 1 n'est pas premier.

Théorème 1. *Tout nombre entier naturel s'écrit comme un produit de nombres premiers ; cette décomposition en facteurs premiers est unique (si l'on range les facteurs par ordre croissant).*

La décomposition en facteurs premiers est aisée pour de petits nombres, mais elle est très **longue pour de grands nombres**, même lorsqu'il n'y a que deux facteurs premiers ; La sécurité du système asymétrique RSA est basée sur la **difficulté de la factorisation**

PGCD
Définition 3. Le PCGD de deux entiers naturels **a** et **b** est le plus grand élément de l'ensemble de leurs diviseurs communs ; on le note PGCD(a,b).

Algorithme de calcul du PGCD (EUCLIDE). Soient **a** et **b** deux entiers naturels avec **a ≥ b**

while b $\neq$ 0 *do*
r ←a mod b ; a←b ; b← r.
end while
*return(*a*)*

Si PGCD(a,b) = 1 alors **a** et **b** sont premiers entre eux.
Le PGCD de deux entiers relatifs est celui de leurs valeurs absolues.
Exemple : PGCD(26,3)=1

Théorème 2 (Identité de BACHET-BÉZOUT*). Soient **a** et **b** deux entiers relatifs. Il existe deux entiers relatifs **u** et **v** tels que **: au + bv = PGCD(a,b)**.*

Cet algorithme peut être étendu pour fournir, outre le PGCD de **a** et **b**, deux entiers premiers entre eux **u** et **v** tels que **au + bv = PGCD(a,b)** (bézout)

Algorithme d'EUCLIDE étendu : Soient **a** et **b** deux entiers tels que **a ≥ b ≥ 1**.

Entrée : **a**, **b** entiers (naturels)
Sortie : **r** entier (naturel) et **u**, **v** entiers relatifs tels que **r = pgcd(a, b)** et **r = a*u+b*v**

Initialisation : **r := a**, **r' := b**, **u := 1**, **v := 0**, **u' := 0**, **v' := 1**
q, **rs**, **us**, **vs** quotient et variables de stockage intermédiaires

les égalités **r = a*u+b*v** et **r' = a*u'+b*v'** sont des invariants de boucle.

while (**r' ≠ 0**) *do*
q := r÷r'
rs := r, us := u, vs := v
r := r', u := u', v := v'
r' := rs -q *r', u' = us - q*u', v' = vs -q*v'
end while
return (**r**, **u**, **v**)

Certains éléments de ***Zn*** n'ont pas d'inverse ; il existe même des éléments non nuls dont le produit est nul.

Théorème 3. Conditions d'inversibilité :
– k est inversible dans Zn implique PGCD(k ,n) = 1.
– Si PGCD(k ,n) ≠ 1 il existe p dans Zn tel que k × p = 0.
– Si n est un nombre premier tous les éléments de Zn sont inversibles sauf 0.
La détermination de l'inverse de k dans Zn s'effectue grâce à l'algorithme d'EUCLIDE étendu.
On aura dans ce cas là : PGCD(k ,n)=u×k+v×n (bézout), et on lit : **u** inverse de **k** modulo **n**

Exemple (euclide étendu)
Calculer l'inverse de 3 en $\mathbf{Z_{26}}$

r									=	**u**	×	**a**	+	**v**	×	**b**
26									=	1	×	26	+	0	×	3
3									=	0	×	26	+	1	×	3
2	=	26	-	8	×	3			=	1	×	26	+	-8	×	3
1	=	3	-	1	×	**2**	=	1×3-1×(1×26 - 8×3)	=	-1	×	26	+	9	×	3

Fonction totient d'Euler
Soit la fonction d'Euler définie par :
ϕ(n) =#{m≤n|PGCD(m,n) =1}
ϕ(n) est l'indicateur d'Euler, c'est-à-dire le cardinal des entiers inversibles de **n**.

Il faut lire : la fonction **ϕ** du nombre n a pour résultat un nombre **m** inférieur ou égal à **n** et tel que le plus grand commun diviseur de **n** et **m** soit 1.

- Si **n** est premier, il vient **ϕ(n) =n−1**. La fonction ϕ(n) donne le nombre d'entiers positifs plus petits ou égaux à n relativement premiers à n.

- De plus, soient **p** et **q** deux nombres premiers et **n=pq**. Il vient

  ϕ(n) =ϕ(pq) = ϕ(p)∗ϕ(q) = (p−1)(q−1)

théorèmes de Fermat
Soit **n** un nombre premier, et PGCD(a,n)=1. Alors : $\mathbf{a^{\phi(n)} = 1}$ mod **n**

b)Principe du RSA

Soit **p** et **q** deux nombres premiers (on utilisera de petits nombres pour faciliter les calculs). En pratique, **p** et **q** ont au moins 100 chiffres décimaux. Le nombre **n** est le produit de **p** et **q**.
On calcule ensuite **ϕ** tel que **ϕ(n) = (p-1) * (q-1).**
Puis on cherche un nombre **e** tel que **e** soit premier avec **ϕ(n).** On cherche ensuite le nombre **d** qui est l'inverse de **e** modulo **ϕ(n)** (e.d≡1mod(Φ(n))).
Le couple clé publique est le couple **(e,n)**. Le couple clé privée est le couple **(d,n).**
En fin on jette **p** et **q**. Le cryptanalyste devant retrouver ces valeurs, il faut les détruire pour éviter les fuites.

Ce cryptosystème utilise deux clés **d** et **e**, interchangeables. Le chiffrement se fait selon
$\mathbf{C = M^e \bmod n}$ et le déchiffrement par $\mathbf{M = C^d \bmod n}$.

Exemple

Soient p = 31, q = 53 c'est-à-dire n=1643.Φ(n) = 1560 (nombre d'éléments relativement premiers à n et < n).
Soit e = 11 (par exemple, et on a bien (e,Φ(n))=1).
On détermine que **d** = 851 (inverse modulaire de **e** sur $Z_{\Phi(n)}$).
La clé publique est donc (11,1643) et la clé privée est (851,1643).
Soit le codage par la position dans l'alphabet du mot «ANEMONE». Il vient
01140513151405
On procède selon deux conditions :
1. Découpage en morceaux de même longueur, ce qui empêche la simple substitution :
_01 140 513 151 405
On ajoute un padding initial, si nécessaire, pour obtenir 001 140 513 151 405

Lors du chiffrement, on a

$001^{11} \bmod 1643$	0001
$140^{11} \bmod 1643$	0109
$513^{11} \bmod 1643$	0890
$151^{11} \bmod 1643$	1453
$405^{11} \bmod 1643$	0374

et pour le déchiffrement,

$0001^{851} \bmod 1643$	001
$0109^{851} \bmod 1643$	140
$0890^{851} \bmod 1643$	513
$1453^{851} \bmod 1643$	151
$0374^{851} \bmod 1643$	405

Lors du déchiffrement, sachant qu'il faut obtenir des blocs de 2 éléments (grâce au codage particulier de l'exemple), on a bien

01	14	05	13	15	14	05
A	N	E	M	O	N	E

Conseils d'utilisation du RSA

Pour garantir une bonne sécurité, il faut respecter certains règles telles que :
– Ne jamais utiliser de valeurs **n** trop petites,
– N'utiliser que des clés fortes (p-1 et q-1 ont un grand facteur premier),
– Ne pas chiffrer de blocs trop courts,
– Ne pas utiliser de n communs à plusieurs clés,
– Si (d,n) est compromise ne plus utiliser n.

Exponentiation rapide

Algorithme square and multiply (bits forts → bits faibles)
Entrée *: x et n*
Sortie *: x^n*
y=1, n=$\sum_0^k b_i 2^i$ (écriture binaire de n)
for *i=k,…….,0* ***do***
  $y=y^2$
  If *b_i=1* ***then***
    y=y.x

Return *y*

Exemple : calculer 9^{107} mod 187

i	b_i	Etapes du calcul
6	1	$1^2 \times 9$
5	1	$9^2 \times 9 = 729 = 168 \mod 187$
4	0	$168^2 = 28224 = 174 \mod 187$
3	1	$174^2 \times 9 = 272484 = 25 \mod 187$
2	0	$25^2 = 625 \mod 187$
1	1	$64^2 \times 9 = 36864 = 25 \mod 187$
0	1	$25^2 \times 9 = 5635 = 15 \mod 187$

c)Sécurité de RSA

Attaques

Il existe trois approches pour attaquer le RSA :

– recherche par force brute de la clé (impossible étant donné la taille des données),

– attaques mathématiques (basées sur la difficulté de calculer Φ(n), la factorisation du module n):

factoriser n=p*q et par conséquent trouver Φ(n) et puis d,

déterminer Φ(n) directement et trouver d,

trouver d directement.

– attaques de synchronisation (sur le fonctionnement du déchiffrement).

A l'heure actuelle, la factorisation connaît de lentes améliorations au cours des années. La meilleure amélioration possible reste l'optimisation des algorithmes. Excepté un changement dramatique, le RSA1024 restera sûr pour les prochaines années. D'après les projections, une clé de 2048 bits est sensée tenir jusque 2079. Mais ces valeurs sont correctes uniquement si on respecte les propriétés de e, d, p et q.

L'attaque de synchronisation est développée dans le milieu des années 90, il s'agit d'exploiter les variations de temps pris pour effectuer certaines opérations (par exemple la multiplication par un petit ou un grand nombre).

Plusieurs contre-mesures existent telles que l'emploi de temps constants d'élévation à une puissance, l'ajout de délais aléatoires, ou le fait de rendre non visibles les valeurs utilisées dans les calculs.

La menace quantique

La physique pourrait accélérer la factorisation par l' utilisation d'un ordinateur quantique. Celui-ci existe d'un point de vue théorique depuis 1994 (algorithme de Shor), et son prototype depuis 1996. Si son évolution se poursuit, il permettrait de réaliser la factorisation d'un nombre en un temps polynomial. Le principe est que les 0 et1 représentés par les portes logiques des transistors sont remplacés par l'orientation du champ magnétique émit par les atomes (que l'on nomme des q-bits).

Chapitre III : Signatures numériques et fonctions de hachage

III-1 Fonctions de hachage

III-1-1) Utilisation des fonctions de hachage

Les fonctions de hachage fournissent l'un des objectifs de sécurité les plus importants. Elles sont utilisées dans le cadre du contrôle de l'intégrité de données et dans le cadre de la signature numérique.

- ✓ L'intégrité est assurée dans la mesure où les fonctions de hachage permettent d'obtenir facilement une empreinte (haché, condensé, résumé) d'une donnée(fichier, message…), cette empreinte est utilisée comme l'identification unique de cette donnée pour contrôler d'éventuelles altérations ou modifications lors de la transmission.
- ✓ Dans la signature numérique, et pour des raisons d'efficacité, on signe l'empreinte de la donnée à transmettre et pas la donnée elle-même.

III-1-2) Notions sur les fonctions de hachage

On désigne un alphabet par le symbole Σ (sigma)

a)Définition

Une fonction de hachage est une application $h : \Sigma^* \dashrightarrow \Sigma^n$ $n \in N$

En clair, une fonction h associe à des chaines de longueurs arbitraires des chaines de caractères de longueur fixée(taille n).

Remarque : les fonctions de hachage ne sont jamais **injectives**.

Exemple :

L'application qui associe b_1 xor b_2 xor b_3 …….. xor b_k au mot binaire $b_1 b_2 b_3$ ……..b_k dans $\{0,1\}^*$ est une fonction de hachage. Pour une chaine b, elle renvoie 1 si le nombre de 1 dans b est impair et en 0 sinon. Exemple pour la chaine 01101 elle renvoie 1.

b)Fonction de compression

Les fonctions de hachage peuvent être fabriquées au moyen de fonctions de compression. Une fonction de compression est une application :

$g : \Sigma^m \dashrightarrow \Sigma^n$ $m,n \in N$ et $m>n$

en d'autres termes, une fonction de compression remplace une chaine de caractère de longueur fixée par une autre chaine de longueur fixée plus courte.

c)Fonction à sens unique

C'est une fonction relativement aisée à calculer mais considérée relativement plus difficile à inverser. En d'autre termes, étant donné un x, il est facile de calculer h(x), mais étant donné h(x) il est très difficile de calculer x. difficile veut dire qu'il faudrait beaucoup trop de temps et d'espace pour arriver à calculer l'inverse de h(x).

d)Collision dans une fonction de hachage

Une fonction de hachage est jamais injective, de ce fait une collision se présente lorsque pour deux entrées différentes **x** et **x'** , on a : **h(x)=h(x')**

Exemple : une collision dans la fonction de l'exemple précédent est un couple formé de deux chaines de bits distinctes, ayant toutes les deux un nombre pair de 1, comme (01101,10101).

III-1-3) Propriétés des fonctions de hachages

Soit h une fonction de hachage, on note y=h(x), alors x est appelé ***préimage*** de y.

Les propriétés à vérifier dans une fonction de hachage h à utiliser pour la cryptographie sont :

1-Propriété de compression et de facilité de calcul ;
2-Résistance à la ***préimage*** : étant donné y, il est calculatoirement difficile de trouver un x tel que y=h(x) (fonction à sens unique) ;
3-résistance à la collision : il est calculatoirement difficile de trouver x et x' tel que h(x)=h(x').

III-1-4) Construction d'une fonction de hachage

Merkel et ***Damgard***, proposent de construire une fonction de hachage à base d'une fonction de compression. Il assurent aussi que si la fonction de compression est résistante aux collisions alors il est de même pour la fonction de hachage résultante. C'est sur ce principe que sont basées les fonctions de hachage les plus utilisées dans la pratique comme **MD5** et **SHA**.

Dans ce qui suit nous utilisons un cryptosystème qui manipule des informations binaires. Le résultat du hachage est dans l'ensemble $\{0,1\}^n$. On prend n>=128, pour éviter l'attaque des anniversaires qu'on va décrire après.

- On dispose d'une fonction de compression **g : $\{0,1\}^m$ ----> $\{0,1\}^n$** pour nϵN et m-n>1
- On va utiliser la fonction **g** pour en définir une fonction de hachage **h()**
  h : $\{0,1\}^*$ ----> $\{0,1\}^n$ pour nϵN

Etapes de construction :
1-Normalisation du message à hacher :

- Soit **Mϵ{0,1}*** le message à hacher. **M** est de longueur arbitraire **L** (|M|=L).
- Soit **r=m-n.**

- Si le nombre de bits du message **M** n'est pas multiple de **r**, on rajoute des **0** au début du message **M** pour qu'il le soit ; on note **$u=0^i.M$,**tel que **$|u|\equiv 0 \bmod r$** .
- Si le nombre de bits de **L** n'est pas multiple de **r-1**, on rajoute des **0** en début de **L**. on note **$y= 0^i.L$**, tel que **$|y|\equiv 0 \bmod r-1$.**
- Découper le mot **y** en blocs de taille **r-1** chacun.
- Rajouter un **1** au début de chacun des blocs obtenu précédemment ; le mot **y** devient un mot de blocs de taille **r** chacun, le nouveau mot obtenu est noté **y'**.
- ✓ En fin, concaténer **u** avec **r** zéro avec **y'** pour obtenir le message normalisé **M'**= **$u0^r y'$;** on obtient un message de **k** blocs **$M'= M'_1\ M'_2\ M'_3 \ldots M'_k$,** chacun des blocs est de taille **r** ($M'_i \in \{0,1\}^r$ $1\leq i \leq k$)

Exemple
r=4, M=11101
L=5=101
$u=0^3.M$=0001 1101
y=101 ; y'=1.y=1101
$M'=u.0^4.y'$=0001 1101 <u>0000</u> 1101 (M'_1 =0001, M'_2 =1101, M'_3 =<u>0000</u>, M'_4=1101)

2-Hachage :
Le hachage de **M** est obtenu en appliquant le schéma dans la figure suivante sur **M'**, avec un vecteur d'initialisation **$IV=0^n$** (**n** bits de valeurs **0**).

$h_0=IV$; $h_i=g(h_{i-1}M'_{i-1})$ $1\leq i \leq k$
$h(M)=h_k$ (résultat du hachage).

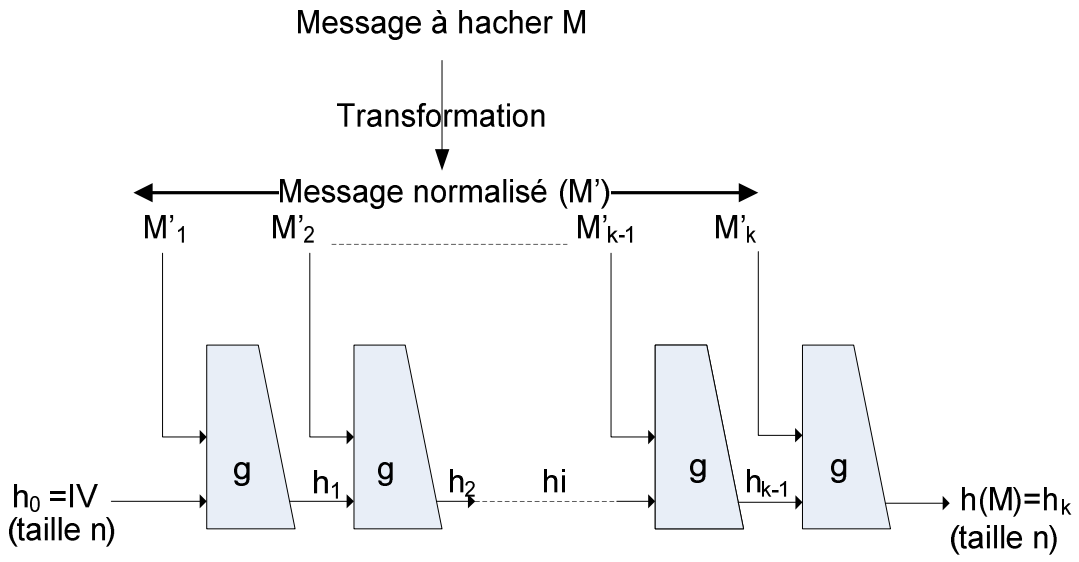


Figure :Etapes de hachage du message M

IV(Initial Value) est une chaine de taille fixée(n) mais publique. Le théorème de Merkle-Damgard nous assure que si **g** est résistante aux collisions, il en est de même pour **h**.

III-1-5) Attaque sur les fonctions de hachage

Les fonctions de hachages sont attaquées en utilisant une technique largement connue, portant le nom de “Paradoxe des anniversaires".
Ce paradoxe s’illustre par la question suivante : combien de personnes faut-il dans un groupe pour que la probabilité d’avoir deux personnes ayant la même date de naissance atteigne 0.5 ?
La réponse est de **23**.

Définition du problème
Pour ce problème, il faut ignorer le 29 février et les années bissextiles. Définissons P(n,k)=Pr[il y a au moins deux éléments communs dans le sous-ensemble de k éléments, et la répartition est uniforme dans l’intervalle 1 à n].
On cherche donc la valeur de k telle que P(365, k) >= 0, 5.
Dans un premier temps, cherchons la probabilité qu’il n’y ait pas d’éléments communs, ce que nous noterons Q(365, k). Pour ce faire, nous notons par N, le nombre de manières de tirer k valeurs sans élément commun(arrangement A^k_n). Plus explicitement, nous pouvons choisir n’importe quel objet sur 365 pour le premier élément, n’importe lequel sur 364 pour le deuxième, etc. Ainsi, il vient $N = A^k_{365} = 365!/(365 - k)!$
Si on enlève la restriction comme quoi il n’y a pas de doublons, il y a 365^k possibilités. Donc, la probabilité de ne pas avoir de doublons est donnée par :
$Q(365, k) = N/365^k = 365!/(365 - k)!(365)^k$
Par conséquent,
$P(365, k) = 1 - Q(365, k) = 1 - 365!(365 - k)!(365)^k$
Pour **k = 23**, on a **P(365, k) = 0.5073**.

Application sur les fonctions de hachage :

Soit une fonction de hachage $\mathbf{h : \{0,1\}^* ----> \{0,1\}^n}$ pour $n \in N$
h peut générer 2^n hachés différents. K le nombre de chaines à choisir pour avoir une probabilité plus de 0,5 de trouver une collision.
En appliquant le Paradoxe des anniversaires on obtient $k=2^{n/2}$ chaines.

Donc, en calculant un peu plus que $2^{n/2}$ valeurs hachées, l’attaque des anniversaires trouve une collision avec une probabilité supérieur à 0.5.

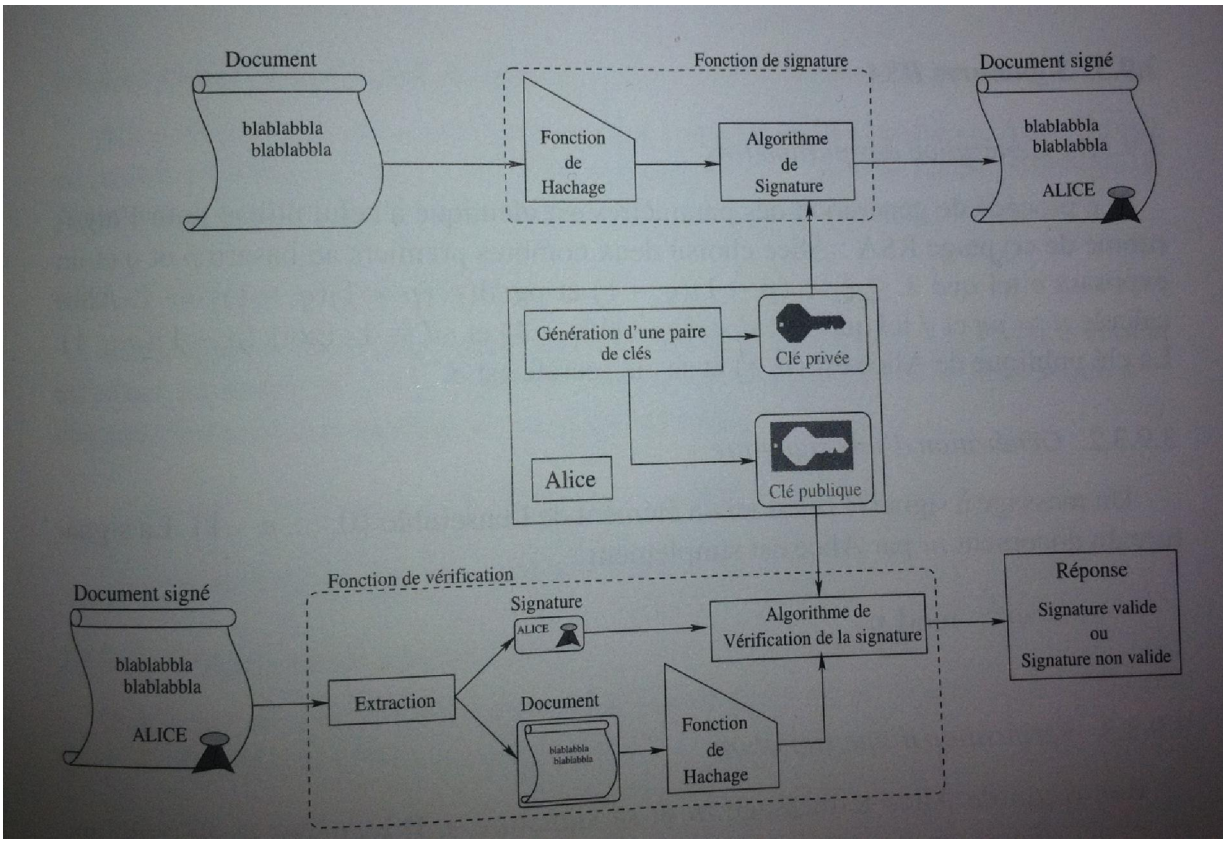
Pour empêcher l’attaque des anniversaires , n doit être choisi de façon que le calcul de $2^{n/2}$ valeurs hachées soit infaisable. Actuellement il est recommander de prendre n>128 ou voir même 160.

III-2 Signature numérique

La signature numérique est utilisée pour prouver l'identité du signataire ou pour lier une donnée à son auteur. Les propriétés d'une signature numérique sont les suivantes:
1-Elle assure l'Authentification : elle permet de convaincre le destinataire de l'auteur du document.
2-Elle assure la Non répudiation : elle est la preuve que le signataire a généré ou bien a accepté le document ;
3-Un document signé est inaltérable : une fois le document signé, il ne peut être modifié
4-une signature n'est pas réutilisable : elle fait partie du document signé et ne peut être déplacé sur un autre document.

III-2-1) Utilisation de la signature numérique

Une des propriétés les plus importantes dans un système de chiffrement à clé publique et la possibilité de signer avec la clé secrète et de pouvoir vérifier cette signature en utilisant la clé publique correspondante.
La signature numérique peut être mise en ouvre par un crypto-système à clé publique combiné avec des fonctions de hachages. Le principe général de la signature d'un document, en combinant ces deux objets, est illustré dans le schéma suivant :



On voit d'abord qu'en pratique , ce n'est pas le document lui-même qui est signé, mais son empreinte. L'intérêt de cette approche tient à l'efficacité des algorithmes de chiffrement à clé publique qui se caractérisent par un chiffrement qui reste globalement lent. Un temps précieux serait donc perdu dans la signature d'un document complet, potentiellement de grande taille. A l'inverse, la signature d'une empreinte garantit une entrée de petite taille(fixe) et donc un temps de calcul de la signature considérablement réduit.

Ensuite, c'est la clé privée d'Alice qui est utilisée pour générer la signature d'un document(à partir de son empreinte). On garantit ainsi que seule Alice(qui possède la clé privée) a pu signer le document. Réciproquement, c'est sa clé publique, accessible par tout le monde(sur une PKI), qui permet la vérification de la signature d'Alice.

III-2-2) Mise en ouvre de la signature numérique

a)Signature RSA

Génération des paramètres

Le procéder de génération des paramètres est identique à ce lui utilisé pour l'algorithme de cryptage RSA : Alice disposera donc d'une clé publique (e,n), et d'une autre clé secrète (d,n).

Génération d'une signature

Un message à signer m est ici un élément de {0,…,n-1}. La signature de m par Alice est simplement : $s=m^d \bmod n$

Vérification d'une signature

Pour vérifier la signature d'Alice, Bob récupère le message m et la signature s. il utilisera la clé publique d'Alice pour vérifier l'identité suivante :

Si $m=s^e \bmod n$ alors **signature valide**
Sinon, signature **non valide**

b) Digital Signature Standard (DSS)

Conçu par le NIST(**National Institute of Standards and Technology**) et la NSA(**National Security Agency**) dans le début des années 90, ce schéma de signature (FIPS-186) fut également approuvé par le gouvernement américain. Il emploie l'algorithme de hachage SHA-1. Comme dans le cas du SHA, on parle de DSS pour la norme, et de DSA pour l'algorithme. Il s'agit en réalité d'une variante d'ElGamal et de Schnorr. La sécurité dépend donc une nouvelle fois de la difficulté de calculer des **logarithmes discrets**.

Paramètres

- p : nombre premier de L bits ($512 < L < 1024$)
- L doit être divisible par 64 et $2^{L-1} < p < 2^L$
- q : facteur premier de p−1, de 160 bits
- $g : g = h^{(p-1)/q} \bmod p$
- h : $1 < h < p-1$ tel que $h^{(p-1)/q} \bmod p > 1$
- x : valeur entière aléatoire t.q. $0 < x < q$ (clé secrète)
- $y : y = g^x \bmod p$ (clé publique)
- H(x) : fonction de hachage à sens unique (SHA)

Signature

Alice engendre tout d'abord un k tel que $0 < k < q$. Ce k doit être aléatoire, secret, détruit après utilisation et jamais réutilisé. Alice engendre ensuite les éléments composant la signature :

- $r = (g^k \bmod p) \bmod q$
- $s = (k^{-1}(H(M) + x.r)) \bmod q$

Alice envoie ensuite (r, s) à Bob.

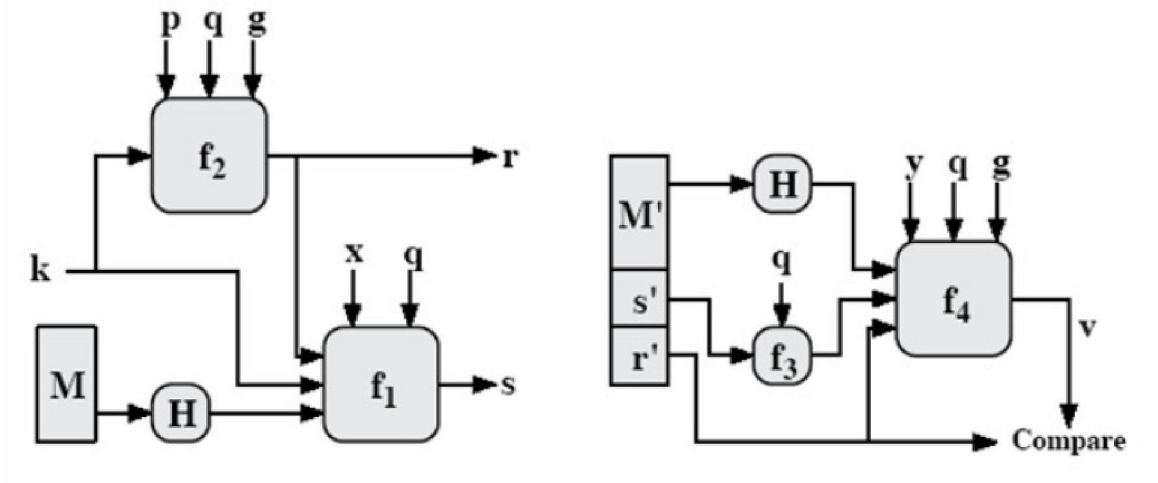
Vérification de la signature

Bob vérifie la signature en calculant :

- $w = s^{-1} \bmod q$

- u1 = (H(M) . w) mod q
- u2 = r.w mod q
- v = ((g^{u1} .y^{u2}) mod p) mod q

Si v = r, alors la **signature est vérifiée**.



$s = f_1(H(M), k, x, r, q) = (k^{-1} (H(M) + xr)) \bmod q$

$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$

$w = f_3(s', q) = (s')^{-1} \bmod q$

$v = f_4(y, q, g, H(M'), w, r')$

$= ((g^{(H(M')w) \bmod q} y^{r'w \bmod q}) \bmod p) \bmod q$

<u>Schéma de signature DSS</u>

A la différence du RSA, le DSS ne peut être utilisé pour chiffrer des messages ou transmettre des clés.
Pour DSS on doit adopter un autre système pour chiffrer (par exemple DSS+RSA) ; alors que dans RSA on chiffre et on signe avec le même système.

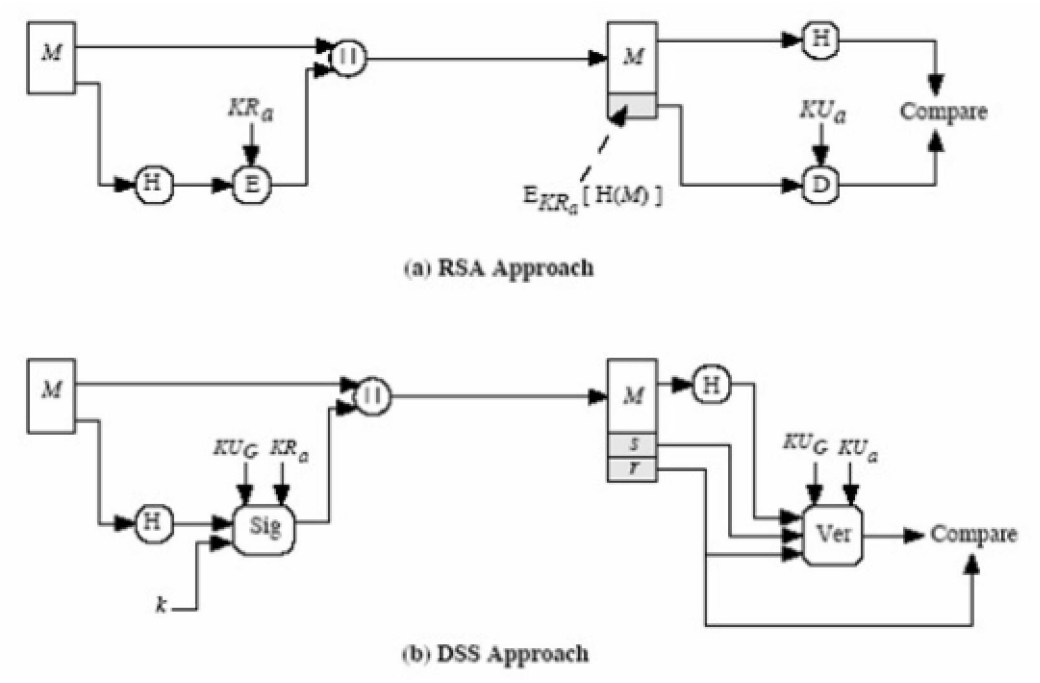


FIG. 9.14 – Signatures par RSA et DSA

Chapitre IV : La Gestion des clés

Nous commençons ce chapitre par le principe de Kerchoff (1883)

La sûreté de tout système de chiffrement, tant à clé publique ou secrète, repose principalement sur le secret de la clé.

Les techniques de gestion des clés viennent pour assurer une bonne utilisation des clés cryptographiques.

IV-1) Cycle de vie des clés de chiffrement

La gestion des clés doit permettre le **génération sûre**, le **stockage** et la **distribution** pour assurer le cycle de vie des clés qui est décrit dans la figure suivante :

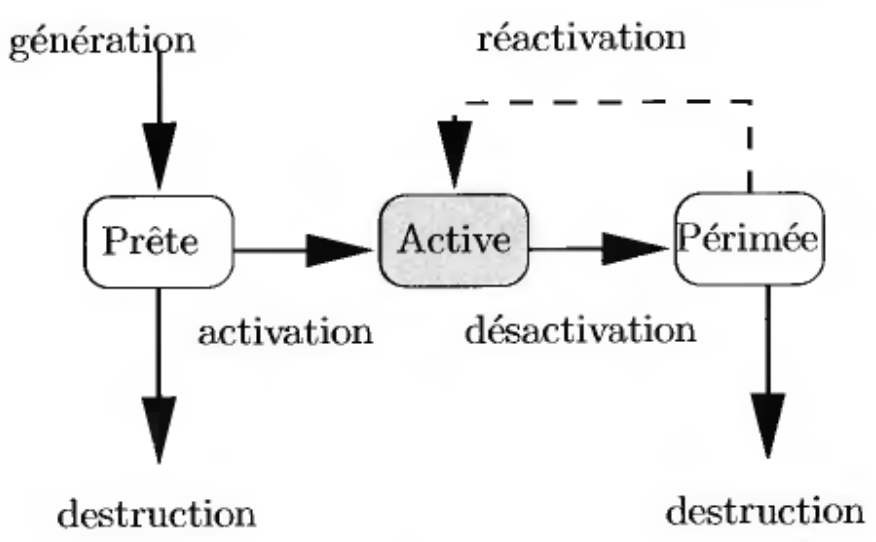


Figure 1 : Cycle de vie d'une clé

La génération : qui doit être effectuée en accord avec les règles de génération de clé ; cela peut demander une phase de test pour vérifier l'adéquation de la clé avec les règles pour éviter, par exemple, des clés faibles.

L'activation : qui rend la clé active pour les opérations de chiffrements. Généralement une clé doit avoir une durée de vie limitée dans le temps et en nombre d'utilisation. Ces contraintes proviennent de considérations cryptanalytiques. En effet, la durée de vie d'une clé est limitée par le temps nécessaire à la cryptanalyse des messages échangés. C'est ce qu'on l'appelle généralement la sécurité calculatoire. Un cryptanalyste peut toujours arriver à cryptanalyser des

messages lorsqu'il dispose d'une quantité d'information convenable et de suffisamment de temps et de ressources informatiques.

Une clé est considérée compromise lorsque son usage non autorisé est connu ou suspecté. Ce risque s'accroit au cours du temps. Les clés compromises doivent être désactivées.

La réactivation : qui permet de réutiliser une clé périmée.

La destruction : qui termine le cycle de vie de la clé. La destruction signifie que l'on doit effacer tout enregistrement de la clé de façon à ce que ne subisse plus aucune information utilisable à des fins cryptanalytiques.

Les techniques de gestion des clés reposent essentiellement sur le chiffrement qui permet de luter contre la divulgation et l'utilisation frauduleuse, les mécanismes d'intégrité de données (hachage et signature) qui évitent la modification abusive et les mécanismes d'authentification (certification) pour contrer la mascarade(lorsqu'un tiers veut se passer pour l'un des communicants).

La fonction de gestion des clés, doit permettre d'éviter aussi :
1-la modification de la clé ;
2-la destruction malintentionnée ;
3-le **rejeu** de la clé ;

IV-2) Distribution des clés

On peut distinguer plusieurs approches de distribuer les clés selon le type de la clé. En effet, s'il s'agit de clés publiques, il n'est à priori pas nécessaire de les chiffrer pour les transmettre. En revanche, ce n'est pas le cas s'il s'agit d'une clé symétrique. Dans ce cas, les clés doivent être chiffrées, soit par un mécanisme de chiffrement à clé publique(enveloppe digitale) soit par un mécanisme à clé secrète.

IV-2-1) Distribution des clés symétriques

La distribution de clés symétriques permet de transmettre une clé engendrée ou obtenue par un sujet à d'autres sujets en la protégeant de façon adéquate. Pour ce faire on peut utiliser des techniques à clé symétrique ou à clé asymétrique avec ou sans recours à un tiers de confiance PKI(public key infrastructure) pour asymétrique et KDC (centre de distribution de clés) dans le cas symétrique.

A) Par mécanisme symétrique

La distribution des clés symétriques au moyen de mécanismes symétriques utilise des clé secrètes partagées au préalable entre l'expéditeur et le destinataire. La norme ISO 11770-2 décrit deuze mécanismes de transport distincts qui font appel ou non à un centre de distribution de clés(KDC). Nous décrivons quelques-uns :

<u>Sans centre de distribution de clés (KDC) :</u>

Le mécanisme de base est illustré dans la figure suivante :

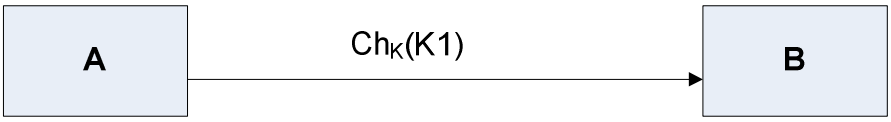


Figure 2 :Échange de clés secrètes par un mécanisme symétrique sans centre de distribution de clé (KDC)

Dans ce cas on suppose qu'une clé K est partagée au préalable entre les communicants A et B et que A est le détenteur de la clé K1 à partager.

Ce mécanisme de base ne procure qu'une authentification implicite de A. Néanmoins il est possible de l'améliorer pour obtenir une authentification de la clé en garantissant sa fraîcheur. Pour ce faire, il suffit d'ajouter quelques données à la clé à échanger :

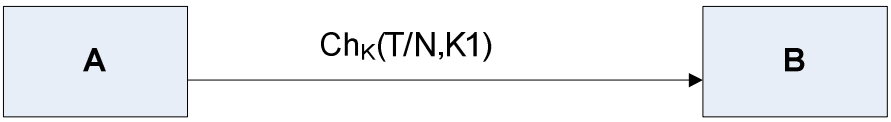


Figure 3: Transport de clés secrètes par un mécanisme symétrique sans KDC avec garantie de fraîcheur.

1- A envoie à B une estampille T ou un numéro de séquence N et la clé à échanger K1 le tout est chiffré par la clé symétrique K préalablement partagée.
2- B déchiffre le message et obtient la clé K1 et vérifies sa fraîcheur grâce à l'estampille T ou le numéro de séquence N.

On observe que cette amélioration permet de prémunir **B** d'une attaque **par rejeu de la clé**

<u>Avec centre de distribution de clés (KDC) :</u>

La distribution de clés secrètes au moyen d'un mécanisme symétrique avec un centre de distribution de clés utilise $\mathbf{K_A}$: come clé secrète partagée entre **A** (demandeur de la communication) et le centre KDC, une clé $\mathbf{K_B}$: comme clé secrète partagée entre **B** (destinataire) et le KDC. L'objectif est de permettre à **A** de retransmettre une clé secrète $\mathbf{K_{AB}}$ engendrée par le KDC.

Etapes du protocole :

1- **A** s'adresse au **KDC** en lui envoyant une requête contenant son identité (ID_A), l'identité du destinataire (ID_B) et un jeton **N** .{ **N,** ID_A**,** ID_B } ;
2- Le **KDC** prépare un message chiffré au moyen de la clé K_A qui contient le jeton de **A** (**N**), une confirmation de l'identité de **B** (ID_B) , la clé secrète à utiliser entre **A** et **B** (K_{AB}) et un certificat à transmettre à **B**, chiffré avec K_B qui contient la clé partagée K_{AB} et l'identité de A. {$Ch_{KA}(N, ID_B, K_{AB}, Ch_{KB}(ID_A,K_{AB}))$]};
3- **A** extrait le certificat issu du **KDC** et le transmet à **B**. { $Ch_{KB}(ID_A,K_{AB})$ } ;
4- **B** reçoit le certificat envoyé par **A**, obtient la clé K_{AB} et l'identité de son correspondant ; il renvoie alors à **A** un accusé de réception (N_B) chiffré avec la clé K_{AB}. { $Ch_{KAB}(N_B)$ } ;
5- **A** accuse bonne réception en effectuant une opération convenue sur le jeton N_B (ici une incrémentation : N_B+1). { $Ch_{KAB}(N_B+1)$}.

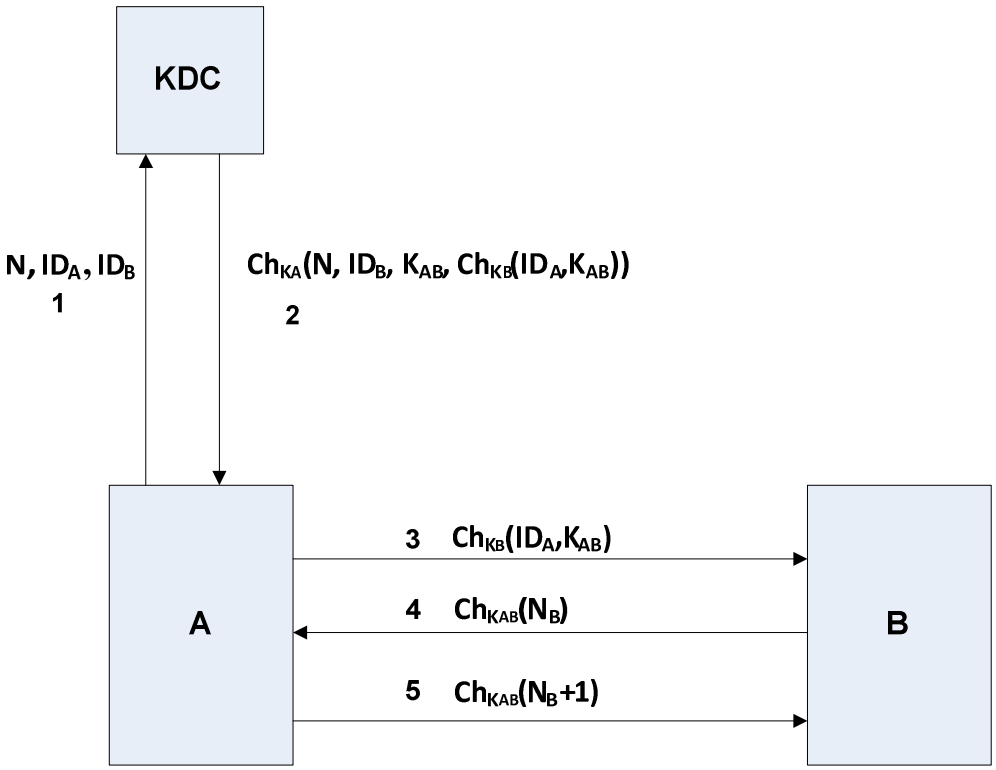


Figure 4: Distribution de clés secrètes par un mécanisme symétrique avec KDC (Needham-Schroeder)

B) Par mécanisme Asymétrique

C'est une mise en ouvre d'un système cryptographique Hybride (Déjà vu).

IV-2-2) Distribution des clés Asymétriques

La gestion des clés **Asymétriques** s'occupe de la génération des paires de clés (publique privée), du stockage de la clé privée, du stockage de la clé publique, de sa protection, de sa révocation. La partie gestion des clés est probablement la partie la plus vulnérable d'un système à clé publique.

A) Approche basée sur une PKI (Public Key Infrastructure)

Dans ce modèle, la gestion des clés est assurée par ce qu'on appelle les **PKI** (Public Key Infrastructure) ou **IGC** (infrastructure de gestion de clés).

La première tache d'une PKI est la génération d'une paire de clés. En fonction de l'organisation de la **PKI**, ceci peut être fait localement ou par un centre de distribution de clés, **KDC**(key distribution Center), qui est un serveur central.

La clé privée est stockée chez l'utilisateur . le mieux et de la chiffrer et de la conserver sur un media non accessible à travers un réseau.

La clé publique doit être enregistrée et validée par une autorité de certification, **CA**(Certificate authority). La CA procure à l'utilisateur un certificat. Le certificat atteste de la réalité d'un lien entre la clé et un utilisateur valide.

Autorité de certification (AC)

c'est un organisme à qui tout le monde fait confiance. L'**AC** fait partie d'une PKI et avec sa signature, elle certifie l'exactitude et l'acceptabilité des clés publiques avec leurs propriétaires. Les utilisateurs connaissent la clé publique de l'AC, elle leur permet de vérifier les signatures faites par l'AC.

Une **AC** assure les taches suivantes :

1-Enregistrement des utilisateurs : Quand BOB devient un nouvel utilisateur d'un système à clés publiques, il est enregistré par l'AC du système. BOB communique son Nom et d'autres informations personnelles utiles. L'AC vérifie ces informations. BOB peut, par exemple, aller en personne à l'AC et présenté sa pièce d'identité. Après l'AC génère, pour BOB, un nom d'utilisateur différent des noms des autres utilisateurs du système.

2-Fabrication des clés : La clé publique et la clé privée de BOB sont fabriquées soit chez lui soit par l'AC. La clé privée est stockée chez BOB. La clé publique est stockée dans un annuaire de l'AC.

3-Certification de la clé publique : L'AC génère un document (certificat) qui associe un clé publique à l'identité réelle d'une entité (personne, serveur….). Le certificat est après signé par la clé privée de l'AC. Un certificat contient au moins les informations suivantes :

- ✓ Le nom d'utilisateur du propriétaire de la clé ;
- ✓ La clé publique ;
- ✓ Le nom de l'algorithme de chiffrement utilisé ;
- ✓ Le numéro de série du certificat ;
- ✓ Date de début et date de fin de validité du certificat ;
- ✓ Le nom de l'AC ;
- ✓ Les restrictions qui s'appliquent à l'usage du certificat.

Le certificat est ensuite stocké, avec le nom d'utilisateur du propriétaire du certificat, dans un annuaire. Seule l'AC peut écrire dans cet annuaire, les autres utilisateurs de l'AC peuvent seulement lire sur l'annuaire.

4-Révocation de certificats : parfois un certificat peut être révoqué avant sa date d'expiration. Une liste des certificats révoqués, **CRL** (Certificate revocation list) est mise à jour par l'autorité de certification. Tout expéditeur voulant utiliser la clé publique déclarée d'un destinataire doit contrôler tout d'abord que le certificat qui atteste de la validité de la clé publique n'a pas été révoqué.

Chaine de certificats

Dans la pratique on trouve généralement une autorité de certification racine et des sous autorités de certifications. Donc, il peut y avoir plus de deux certificats : celui qui garantit la relation entre l'identité de BOB et sa clé publique mais aussi celui qui garantit la relation entre l'identité de l'AC et sa clé publique. Dans ce cas, on parle de Chaine de certification où chaque certificat va garantir l'authenticité du certificat précédent.

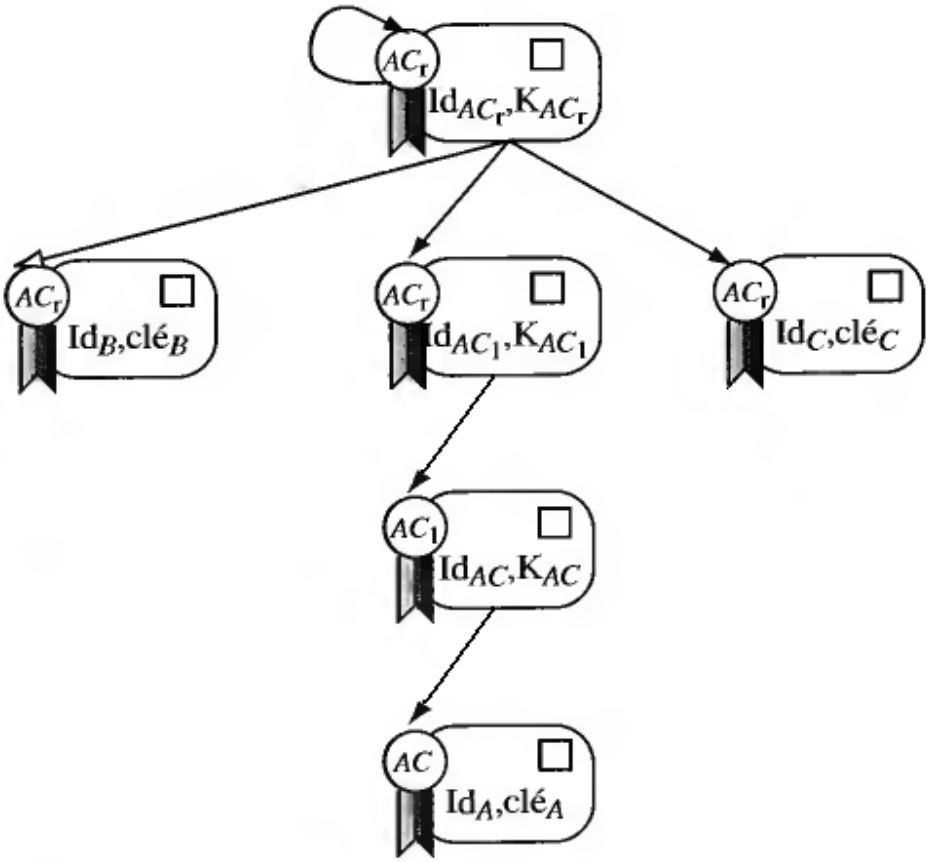


Figure 5 : Chaine de certificats

BOB, qui veut communiquer avec ALICE, lui demande son certificat. Afin de vérifier le certificat, BOB recherche la clé publique de l'autorité qui a signé le certificat d'ALICE. Si la clé et le certificat sont chez la même autorité (autorité racine) , la recherche est terminée. Sinon, BOB demande à l'autorité de certification de contacter l'autre autorité pour obtenir sa clé publique. Pour chaque autorité de certification interrogée par BOB, il doit disposer de la clé publique de l'autorité précédente. Si une chaîne peut être trouvée, aboutissant à l'autorité racine de certification, la recherche est terminée.

B) Approche basée sur les réseaux de confiance

Le réseau **de confiance** est un concept utilisé dans PGP, GnuPG, ainsi que d'autres systèmes compatibles avec OpenPGP. Le réseau de confiance permet de vérifier la relation entre une clé publique et son possesseur.

C'est un modèle de confiance décentralisé, en alternative aux modèles de confiance centralisés de la plupart des autres PKI. Contrairement aux modèles centralisés où la confiance que l'on peut prêter ne passe que par une autorité de certification (CA ou hiérarchie de CA) dont la

plupart ne sait rien, un réseau de confiance est relatif à un individu qui peut choisir lui-même les tiers (en général d'autres personnes physiques) à qui il fait confiance.

De ce fait, il existe de nombreux réseaux de confiances indépendants. N'importe qui peut en faire partie (via son propre certificat), et être un lien entre différents réseaux.

Le concept de réseau de confiance a été élaboré initialement par Phil Zimmermann, créateur de PGP, en 1992.

Fonctionnement

Dans OpenPGP, les certificats d'identité (clés publiques) sont vérifiés par la signature numérique d'autres utilisateurs. Ces utilisateurs, en signant ce certificat, peuvent renforcer (pour d'autres) l'association entre une clé publique et la personne ou l'entité désignée par ce certificat. Dans la pratique, un certificat reçu par BOB est considéré comme valide :

- Si BOB l'a lui même signé
- Ou si le certificat est signé par une personne auquel BOB a donné sa confiance totale
- Ou si le certificat est signé par 3 personnes auquel BOB a donné sa confiance partielle

Ces paramètres sont ajustables par l'utilisateur, en fonction de ses besoins. Ils peuvent même être complètement ignorés.

Ce système assure que c'est bien BOB qui envoie un message. En effet, BOB envoie un message signé avec sa clé privée, et, s'il n'utilise pas de serveur de clés, un certificat contenant des signatures de personnes qui peuvent confirmer que c'est bien BOB qui envoie le message. Il suffit de vérifier les signatures du certificat pour vérifier que c'est bien BOB qui l'a envoyé. Si on ne fait pas confiance aux signataires, on ne peut valider que c'est bien BOB qui possède la clé privée associée à cette signature, et donc que c'est bien lui qui a signé. Dans ce cas la signature peut être bonne (faite par un certain "BOB") mais invalide (on n'est pas sûr que ce "BOB" existe vraiment).