Introduction à la sécurité

Chapitre 2

BELFODIL Aymen

Source 1 : Cours Introduction à la sécurité 2012 – Mohammed Anane

Source 2 : **Guénaël Renault – POLSYS LIP6/UPMC/INRIA 22 janvier 2014**

2013/2014

Partie 0 : Rappel mathématiques

Avant de présenter l'algorithme **RSA** nous présenterons quelque **définitions** et **théorèmes** de l'arithmétique dans $\mathbb{Z}$.

1. Nombre premiers

1.1. Nombre premier :

Un nombre premier $p \in \mathbb{N}$ est un nombre qui ne possède que deux diviseurs : 1 et p. Dans la suite de cours nous noterons l'ensemble des nombres premiers : $\mathbb{P}$

1.2. Nombre semi - premier :

Soit $n \in \mathbb{N}$, n est dit **semi-premier** ssi : $\exists p, q \in \mathbb{P} \mid n = p * q$

2. L'anneau $\mathbb{Z}/n\mathbb{Z}$:

Soit $n \in \mathbb{N}^* - \{1\}$ on définit l'anneau ${}^{\mathbb{Z}}/_{n\mathbb{Z}} = \{\dot{\mathbf{0}}, \dot{\mathbf{1}}, \dots, (\boldsymbol{n} \dot{-} \mathbf{1})\}$**.** C'est l'ensemble quotient ${}^{\mathbb{Z}}/_{R}$ de la relation R définit comme suit :

$$\forall x, y \in \mathbb{N} : x \, R \, y \iff x \equiv y \, [n]$$

Remarque :

1. Les seules éléments x inversible de ${}^{\mathbb{Z}}/_{n\mathbb{Z}}$ sont les éléments **premiers** avec n : $pgcd(x, n) = 1$
2. l'ensemble des éléments inversible de ${}^{\mathbb{Z}}/_{n\mathbb{Z}}$ forme un groupe (muni de l'opération $*$) , on le note $\left({}^{\mathbb{Z}}/_{n\mathbb{Z}}\right)^{\times}$

Théorème 1 :

L'anneau ${}^{\mathbb{Z}}/_{n\mathbb{Z}}$ est un **corps** ssi : $n \in \mathbb{P}$ car tous ses éléments (**sauf $\mathbf{0}$**) sont inversibles puisqu'il sont tous premier avec n.

3. Inverse modulaire :

Soit l'anneau ${}^{\mathbb{Z}}/_{n\mathbb{Z}}$ et soit $x \in \left({}^{\mathbb{Z}}/_{n\mathbb{Z}}\right)^{\times}$, on appelle **inverse modulaire** de x l'élément y telle que :

$$x * y \equiv 1[n]$$

Le calcul de l'inverse modulaire d'un élément $\boldsymbol{x}$ est fait à l'aide de l'algorithme suivant (basé sur théorème de Bézout et l'algorithme d'Euclide pour le calcul du PGCD)

Théorème 2 : Théorème de Bézout

Soient $n, m \in \mathbb{Z}^*$ et soit $d = pgcd(n, m)$ on a :

$$\exists a, b \in \mathbb{Z} \mid a * n + b * m = d$$

Corollaire

Si n est premier avec m alors : $\exists a, b \in \mathbb{Z} \mid a * n + b * m = 1$ (a représente l'**inverse modulaire** de n dans $\mathbb{Z}/_{m\mathbb{Z}}$)

Méthode de calcul d'inverse modulaire (exemple):

Considérons l'exemple suivant $n = 13$ et $x = 8$ trouvons y l'inverse de x dans $\mathbb{Z}/_{n\mathbb{Z}}$:

1. $\mathbf{13} = 1 * 8 + \mathbf{5}$
2. $\mathbf{8} = 1 * \mathbf{5} + \mathbf{3}$
3. $\mathbf{5} = 1 * \mathbf{3} + \mathbf{2}$
4. $\mathbf{3} = 1 * \mathbf{2} + \mathbf{1}$ **(1 représente le $PGCD(n, x)$)**

On remonte pour récupérer le théorème de Bézout :

1. **On a :** $1 = \mathbf{3} - 1 * \mathbf{2}$ $(\mathbf{4})$ or $\mathbf{2} = \mathbf{5} - 1 * \mathbf{3}$ $(\mathbf{3})$

   D'où : $1 = \mathbf{3} - 1 * (\mathbf{5} - 1 * \mathbf{3})$ **ainsi :**

   $$1 = 2 * \mathbf{3} - 1 * \mathbf{5}$$

2. **On a :** $\mathbf{3} = \mathbf{8} - 1 * \mathbf{5}$ $(\mathbf{2})$ Ainsi :

   $$1 = 2 * (\mathbf{8} - 1 * \mathbf{5}) - 1 * \mathbf{5} = \mathbf{2} * \mathbf{8} - \mathbf{3} * \mathbf{5}$$

3. **On a** : $\mathbf{5} = \mathbf{13} - 1 * \mathbf{8}$ $(\mathbf{1})$ Ainsi :

   $$1 = \mathbf{2} * \mathbf{8} - \mathbf{3} * (\mathbf{13} - 1 * \mathbf{8}) = \mathbf{5} * \mathbf{8} - \mathbf{3} * \mathbf{13}$$

Ainsi $y = 5$ est l'inverse modulaire de $x = 8$ dans $\mathbb{Z}/_{13\mathbb{Z}}$

4. Indicatrice d'Euler :

Soit $n \in \mathbb{N}$ l'indicatrice d'Euler noté $\varphi(n)$ est définit comme suit :

$$\varphi(n) = card(\{x \in [\![1, n-1]\!] \mid pgcd(x, n) = 1\})$$

NB. : Le calcule $\varphi(n)$ est très complexe, il nécessite la factorisation de n qui est une opération très coûteuse.

Remarque :

1. si $n \in \mathbb{P}$ alors $\varphi(n) = n - 1$
2. si n semi-premier (avec $p \neq q$) alors $\varphi(n) = (p-1) * (q-1)$

5. Corolaire du petit théorème de Fermat :

Soit $p \in \mathbb{P}$, soit $m \in \mathbb{N}^*$ **non divisible par p** alors :

$$m^{p-1} \equiv 1[p]$$

6. Théorème d'Euler :

Le théorème d'Euler est une généralisation du **théorème de Fermat** (ce théorème est basé sur la notion des **groupes cycliques** et la notion de l'**ordre d'un élément**), soit $m \in \mathbb{N}^*$ et soit $n \in \mathbb{N}$ alors :

$$m^{\varphi(n)} \equiv 1[n]$$

Remarque : Pour un nombre semi-premier (avec $p \neq q$) $n = p * q$ on a $m^{(p-1)*(q-1)} \equiv 1[n]$

Partie 1 : Chiffrement Asymétrique

1. Introduction :

Le **chiffrement symétrique** possède plusieurs inconvénients, les principaux sont :

1. La Gestion des clés
2. L'échange des clés

Le **chiffrement asymétrique** vient pallier aux inconvénients du chiffrement symétrique. Le **chiffrement asymétrique** repose sur la notion de **fonction avec trappe**.

2. Fonction avec trappe (one-way trapdoor function) :

Une fonction : $F : X \longrightarrow Y$ est à **sens unique** avec **trappe**, si :

1. $\boldsymbol{F}$ **est à sens unique :** F est **facile à calculer**, **difficile à inverser**.
2. $\boldsymbol{F}$ **possède une trappe :** la connaissance d'une **information supplémentaire** appelée **trappe**, rend, pour tout $y \in Im(F)$ le calcule de $x \in X$ tel que $F(x) = y$ **réalisable en temps polynomial**.

Remarque :

Ainsi, la **clé publique** est la fonction F (elle permet de **chiffrer**), la **clé secrète** représente **la trappe** et permet de déchiffrer (ceci dit que tout le monde peut chiffrer un message à **destination précise**, mais ce n'est que la destination qui a la capacité de déchiffrer ce message).

3. Définition : Chiffrement à clé publique :

Soit l'espace des messages représentés par le **groupe** G.

1. On chiffre les messages $M \in G$ en utilisant l'algorithme public F et une **clé publique** $\boldsymbol{k}$:
$$C = F(M, k)$$
2. On déchiffre le **texte chiffré** C en utilisant la trappe qui a la **clé privé** $\boldsymbol{d}$ et la fonction G
$$M = G(C, k, d)$$

Ainsi, Le scénario d'échange des messages est :

Bob veut recevoir des messages codés d'**Alice**, il souhaite que ces messages soient indéchiffrables par **oscar** qui a accès à leurs échanges. **Bob** et **Alice** connaissent la fonction unidirectionnelle F.

1. **Bob** fournit à **Alice** sa **clé publique** $\boldsymbol{K}$. (**Oscar** peut connaitre F et K)
2. **Alice** chiffre son message M de la manière suivante $C = F(M, k)$ puis envoie ce dernier à **Bob**
3. **Bob** possède la **trappe** (la **clé privé**e d) qui est absolument secrète. il récupère $M = G(C, k, d)$

Notion de l'Annuaire :

De manière générale, chaque entité publie sa **clé publique** dans un **annuaire** (qui doit être **sûr**). Pour envoyer un message à **Bob** il suffit de trouver sa clé dans l'**annuaire** et de s'en servir pour chiffrer le message avant de lui envoyer. Seul **Bob** pourra **déchiffrer** les messages car il possède la **trappe** (**clé privé**).

4. Chiffrement RSA

4.1. L'algorithme RSA :

La première instance d'un cryptosystème à clé publique est **RSA** proposé en 1977 par Rivest, Shamir et Adleman.

Soit $n \in \mathbb{N}$ un nombre semi-premier $n = p * q$ (p, q sont différents de tailles à peu près égales).
Soit $e \in \left(\mathbb{Z}/_{\varphi(n)\mathbb{Z}}\right)^{\times}$ et soit d son inverse $e * d \equiv 1[(p-1)*(q-1)]$ dans ce groupe.

1. La **clé publique** est la paire (e, n)
2. la **clé privée** est la paire (d, n)

Soit $0 \leq M < n$ le message à chiffrer :

1. Le chiffrement est fait de la manière suivante :

$$C = M^e \ mod \ n$$

2. Le déchiffrement est fait de la manière suivante :

$$M = C^d \ mod \ n$$

Remarque :

Effectivement $M = C^d \ mod \ n$ car $C = M^e \ mod \ n$, Vérifions que : $M = M^{e*d} \ mod \ n$

On a $e * d = K * (p-1) * (q-1) + 1 = K * \varphi(n) + 1 \mid K \in \mathbb{N}$ on a:

$$M^{e*d} = M^{K*\varphi(n)+1}$$

Or par le théorème d'Euler on a :

$$M^{\varphi(n)} \equiv 1[n] \ ainsi \ (M^{\varphi(n)})^K \equiv 1[n] \ d'où : \ M^{K*\varphi(n)+1} \equiv M[n]$$

Or $0 \leq M < n$ ainsi : $M = M^{e*d} \ mod \ n$ **CQFD**

NB. : Lorsque le message à crypter $M > n$ on le découpe de telle façon que les M_i soient de même tailles et inférieur de n (si n est sur $1024 \ bits$ les M_i sont sur $1023 \ bits$), on applique ensuite le chiffrement sur les M_i (C est la concaténation des C_i)

4.2. L'exponentiation modulaire :

La complexité de L'algorithme **RSA** dépend de la complexité du calcul de l'**exponentiel modulaire** ie. : $C = M^e \ mod \ n$

Ainsi il faut qu'on utilise un algorithme optimal pour le calcul d'exponentiel **modulaire.**

La méthode indienne :

Principe :
On a $x * y \ mod \ n = (x \ mod \ n * y \ mod \ n) \ mod \ n$

Soient A, B et $n \in \mathbb{N}$ nous désirons calculer $A^B \ mod \ n$, on a:

$$B = \sum_{i=0}^{k-1} b_i * 2^i \ : représentation \ de \ B \ en \ binaire \ \text{Ainsi} \ A^{\sum_{i=0}^{k-1} b_i * 2^i} = \prod_{i=0}^{k-1} A^{b_i * 2^i}$$

Dans la **méthode indienne** on calcule l'exponentiel la manière suivante :

1. Initialiser $V = 1$
2. pour $i\ allant\ de\ k-1$ à 0 (on commence à partir du poids fort de l'exposant B) :
    a. $V \longleftarrow V^2$
    b. Si $b_i = 1$ alors $V \longleftarrow V * A$

Pour l'**exponentiel modulaire** il suffit de calculer le mod dans chaque **itération**.

Exemple :
Calculons : $22^{10}\ mod\ 17$

On a : $10 = (1010)_b$ ainsi :

1. $V = 1, b_3 = 1 : V = (V^2 * 22)\ mod\ 17 = 22\ mod\ 17 = 5$
2. $V = 5, b_2 = 0 : V = V^2\ mod\ 17 = 25\ mod\ 17 = 8$
3. $V = 8, b_1 = 1 : V = (V^2 * 22) mod\ 17 = 1408\ mod\ 17 = 14$
4. $V = 14, b_0 = 0 : V = V^2\ mod\ 17 = 196\ mod\ 17 = 9$

Remarque :
La **force de chiffrement RSA** dépend de la longueur des clés utilisées (et donc des nombres premiers choisis). Actuellement on utilise des clés de 1024 bits (jusqu'à 2048 bits) $\Longrightarrow$ les nombres premiers p et q sont énormes !

Le chiffrement RSA peut être utilisé pour l'**authentification**, il possède la propriété suivante :

$$Déchiffrer\big(Chiffrer(M)\big) = Chiffrer(Déchiffrer(M))$$

(Voir la partie 3 pour plus de détails)

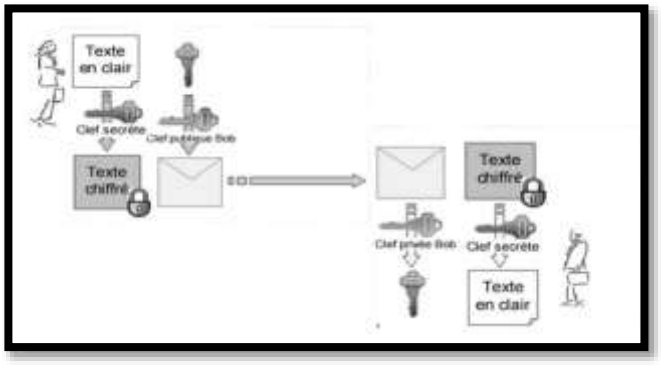
5. Chiffrement symétrique vs Chiffrement asymétrique :

Le **chiffrement asymétrique** est pratique (notion de clé publique) mais nécessite trop de calcule, ainsi le **chiffrement asymétrique** est utilisé pour transférer des petits messages.

Par contre le **chiffrement symétrique** est performant et adaptés pour les gros messages.

L'idée est d'utiliser un **chiffrement hybride** :

1. On utilise le **chiffrement à clé publique** pour transférer la **clé de session** (qui est la clé d'un algorithme de chiffrement symétrique)
2. Cette clé de session sera utiliser pour Chiffrer/déchiffrer les messages



Partie 2 : Autres protocoles cryptographiques

Outre le problème de l'**échange des clés privées**, Le nombre croissant d'échanges augmente le nombre de clés secrètes nécessaires ce qui rajoute un nouveau problème.

En 1976 **Diffie**, **Hellman** et **Merkle** publient le premier schéma d'**échange de clés**.

1. Diffie – Hellman :

1.1. Principe :

Diffie-Hellman est un protocole cryptographique qui permet à deux entités de générer un secret partagé sans informations préalable l'un sur l'autre.

Diffie-Hellman est basée sur la difficulté de calculer des logarithmes discrets sur un corps finie.

1.2. Définition Logarithme discret :

Soit $(G,\circ)$ un **groupe** fini et soient $g, h \in G$. Le **logarithme discret** en base g de h est défini par :

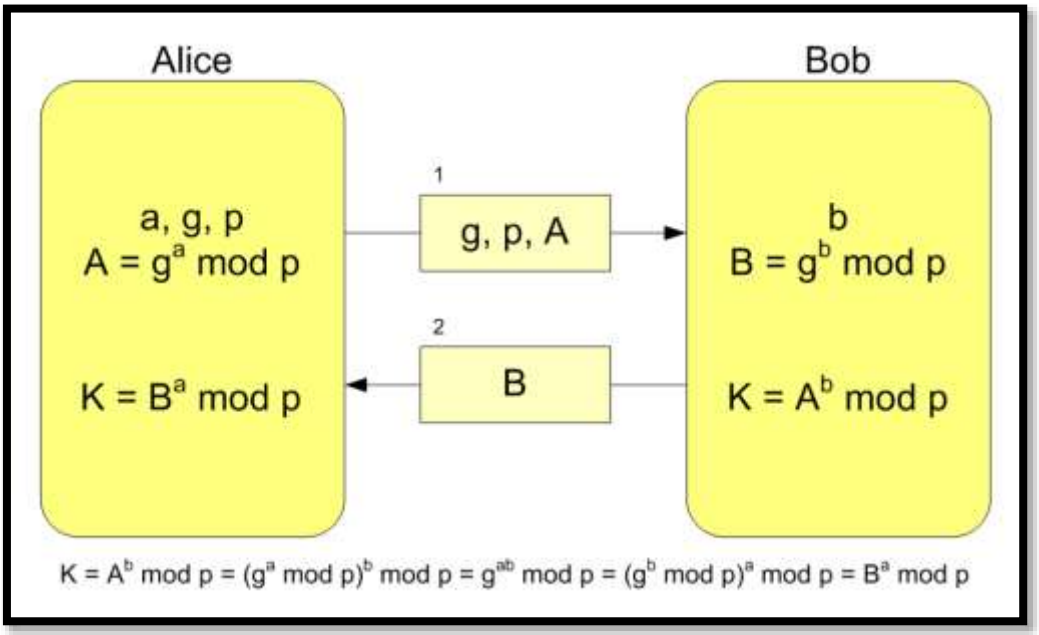
$$k = \log_g(h) \mid h = g^k = g \circ g \circ \ldots \circ g \ (k\ fois)$$

Le calcul de k est très **difficile**. ce problème est appelé DLP : $Discret\ logarithme\ Problem$

1.3. Etablissement de clés secrète :

1. Choix d'un **nombre premier** n et d'un générateur $g \in \left(\mathbb{Z}/_{n\mathbb{Z}}\right)^*$ (n, g sont **publiques**)
2. **Alice** génère un entier aléatoire a et envoie à **Bob** le nombre $A = g^a \ mod\ n$ (il est très difficile de récupérer $a = \log_g A$ dans le groupe $\left(\mathbb{Z}/_{n\mathbb{Z}}\right)^*$)
3. de même, **Bob** génère un entier aléatoire b et envoie à **Alice** le nombre $B = g^b \ mod\ n$
4. **Alice** construit $k = B^a \ mod\ n$ et **Bob** construit $k' = A^b \ mod\ n$. La clé secrète commune est :

$$k = k' = g^{ab} \ mod\ n$$



2. El – Gamal :

El – Gamal inspiré de **Diffie – Hellman** a été proposé en 1985, il est basé sur le **DLP**.

2.1. Communication dans El - Gamal:

1. Le destinataire **Alice** choisit un **nombre premier** n et d'un générateur $g \in (\mathbb{Z}/n\mathbb{Z})^*$
2. **Alice** sélectionne aléatoirement un nombre $a \in \mathbb{Z}/n\mathbb{Z}$
3. **Alice** publie $(n, g, A = g^a \ mod \ n)$

Lorsque quelqu'un (**Bob**) veut communiquer un message $m < n$ avec **Alice** il suit les étapes suivantes :

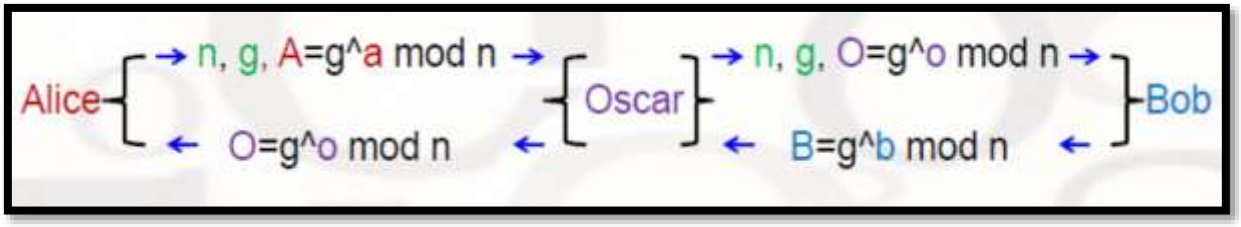
1. **Bob** sélectionne aléatoirement un nombre $b \in \mathbb{Z}/n\mathbb{Z}$
2. **Bob** calcule $c_1 = g^b \ mod \ n$ et $c_2 = m * A^b \ mod \ n = m * (g^{a*b}) \ mod \ n$
3. **Bob** envoie $c = (c_1, c_2)$

Ensuite **Alice** fait les opérations suivantes pour récupérer le message m :

1. **Alice** calcule $d_1 = {c_1}^{n-1-a} \ mod \ n = {c_1}^{-a} \ mod \ n = g^{-ab} \ mod \ n$ | g^{-ab} est l'inverse modulaire de g^{ab} dans $\mathbb{Z}/n\mathbb{Z}$
2. **Alice** calcule $d_2 = d_1 * c_2 \ mod \ n = g^{-ab} * m * g^{ab} \ mod \ n = m$

3. Problème de Men-in-the-middle :

Oscar peut s'insérer entre **Alice** et **Bob** et propose sa valeur **O** en lieu et place de **A** pour Bob et de **B** pour Alice, ce problème est dit le problème de l'homme au milieu (Men In The Middle) :



Oscar échange une clé avec **Alice** et une autre avec **Bob** et se met au milieu.

Pour remédier à ce problème, il faut une phase préliminaire d'**Authentification.**

Partie 3 : Authentification et hachage

Note : Dans les parties précédentes nous nous somme intéressé sur l'aspect **confidentialité** des données, dans cette partie nous nous intéresserons sur l'aspect **intégrité**, **non répudiation** et **authentification**.

1. Authentification

L'**authentification** consiste à vérifier l'identité d'une entité (sans nécessairement connaitre son identité) pour lui donner des autorisations. L'**approche traditionnelle** de l'authentification est d'utiliser une combinaison d'une identification et d'un mot de passe.

1.1. Challenge/Réponse :

L'approche évoluée de l'**authentification** est la notion de **challenge/réponse**, deux cas de figures existent :

1.1.1. Chiffrement à clé secrète

Alice veut vérifier qu'elle communique bien avec **Bob (Alice** et **Bob** partages la même clé secrète) **:**

1. **Alice** envoie à **Bob** un message aléatoire **(challenge)**
2. **Bob** renvoie à **Alice** le message **chiffré** à l'aide de la clé **privé secrète (réponse)**.
3. **Alice** déchiffre le message reçu de **Bob en utilisant la clé secrète** et retombe sur le **challenge** envoyé. elle confirme qu'elle communique avec **Bob**

1.1.2 Chiffrement à clé publique

Alice veut vérifier qu'elle communique bien avec **Bob,** puisque c'est **Alice** qui veut initier la communication elle connait la clé publique de **Bob** récupéré depuis l'**annuaire**, c'est **Bob** qui possède la **clé privé** :

1. **Alice** envoie à **Bob** un message aléatoire **(challenge)**
2. **Bob** renvoie à **Alice** le message **chiffré** à l'aide de la clé **privé (réponse)**.
3. **Alice** déchiffre le message reçu de **Bob en utilisant la clé publique** et retombe sur le **challenge** envoyé. elle confirme qu'elle communique avec **Bob**. (en fait **Bob** a signer le challenge)

L'algorithme **RSA** possède la propriété :

$$M = Chiffrer(Déchiffrer(M))$$

Ainsi il peut être utilisé dans ce cas de figure.

1.2. Authentification par signature :

1.2.1. Définition Signature :

La **signature** est une information **en plus** qu'envoie l'**expéditeur** avec le **message initial**. Elle permet d'authentifier l'expéditeur.

La **Signature** garantit en plus l'**intégrité** du message envoyé.

1.2.2 Signature avec RSA

Soient (n, d) la clé publique et (n, e) la clé privée. Soit m le message envoyé.

1. L'**expéditeur** calcule la signature $S_m = m^e \ mod \ n$ et envoie le couple (m, S_m) au destinataire.
2. Le **destinataire** authentifiera l'expéditeur à partir de la clé publique en vérifiant que : $(S_m)^d \ mod \ n = m$. Si c'est le cas il accepte le message.

De manière plus générale, l'expéditeur envoie le $\boldsymbol{s = Signature_d\ (m)}$ en utilisant la **clé privé**. Le destinataire vérifie la signature $\boldsymbol{Signature_d\ (m)}$ en utilisant la clé publique e avec la fonction $\boldsymbol{verification_e(m, s)}$. Cette dernière renvoie $VRAI$ si s $\boldsymbol{s = Signature_d\ (m)}$

Problème : La signature directe sur les messages longs crée des signatures très longues (la signature RSA est de taille égale à la taille du message initiale), ceci peut être une faille de sécurité !

2. Hachage cryptographique :

2.1. Définition

Une fonction $H : [0,1]^* \longrightarrow [0,2^t - 1]$ est dite de **hachage** si elle vérifie :

1. **Difficile à inverser :** difficile (si c'est n'est pas **impossible**) de reconstruire le message à partir de **l'empreinte** $H(m)$
2. **Difficile à reconstruire un second message de même signature (second préimage)**
3. **Probabilité de collision très faible :** Soit m_1 et m_2 le cas $H(m_1) = H(m_2)$ est **rare**.

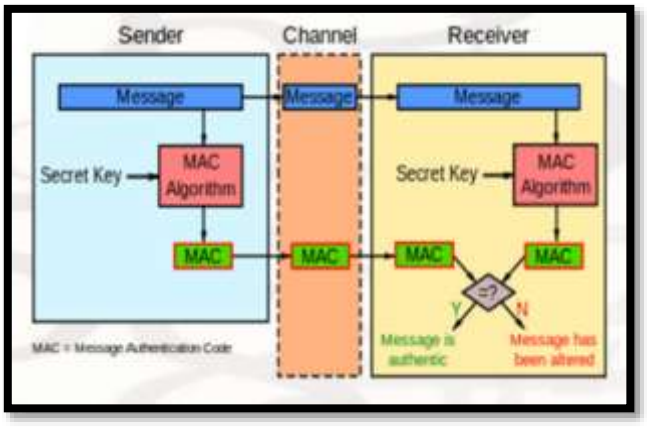
Note : On remarque que l'entrée $\boldsymbol{m}$ de la fonction de **hachage** est de taille variable, par contre, la dimension de la sortie $\boldsymbol{H(m)}$ dite **empreinte** ou **haché** de m doit être de **fixe**.

Exemples : Il existe plusieurs fonctions d'hachage, les principaux sont : **MD5 (Message digest 5)** développé par **Rivest** en **1991, SHA1 (Secure Hash Algorithm)** développé par le **NIST** en **1995.**

3.2. Types de fonctions d'hachage

Il existe deux types de fonctions d'hachage :

1. **MDC (Modification Detection Code) :** fonction de hachage **sans clé**, on peut s'en servir pour s'assurer de **l'intégrité du message** (garantir que le message n'a pas été modifié intentionnellement ou accidentellement).
2. **MAC (Message Authentication Code) :** fonction de hachage **avec clé**. Elle possède un paramètre additionnel (**la clé**) qui permet de vérifier l'**intégrité** et la **provenance du message** (**Authentification avec une petite signature**) en même temps. Si l'intrus modifie le message, Le recepteur détecte cette modification (Le MAC calculé au niveau du destinataire est différent de celui envoyé)



3.3. Utilisation du hachage en sécurité :

Le hachage est utilisé dans plusieurs aspects de sécurité, on cite :

1. Stockage et vérification des mots de passe :

Imaginons qu'**Alice** veut se connecter à un site ou elle devra s'authentifier en utilisant son **mot de passe**. Si **Alice** envoie son **mot de passe** (vers le serveur) **en clair Oscar** peut s'emparer de celui-ci.

L'idée est que Alice envoie le **Haché du mot** de passe au lieu du **mot de passe lui-même**, même si **Oscar** s'empare du **haché** il ne pourra pas déduire le mot de passe (Difficulté de retrouver l'inverse est une propriété de la fonction de hachage).

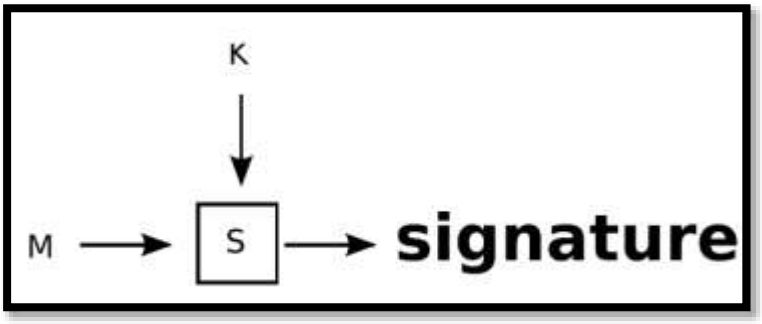
Une fois le mot de passe arrivé au serveur, le serveur vérifie si l'**haché envoyé** existe dans **/etc/passwd** (on sauvegarde que **les Haché des mots de passes**)

Question : le hachage est une fonction non injective **à priori,** ainsi **Oscar** peut trouver un **mot de passe** $\boldsymbol{m_O}$ non nécessairement égale **au mot de passe de Alice** $\boldsymbol{m_A}$ mais : $H(m_o) = H(m_A)$. Il pourra ainsi se connecter au site avec son propre **mot de passe** !

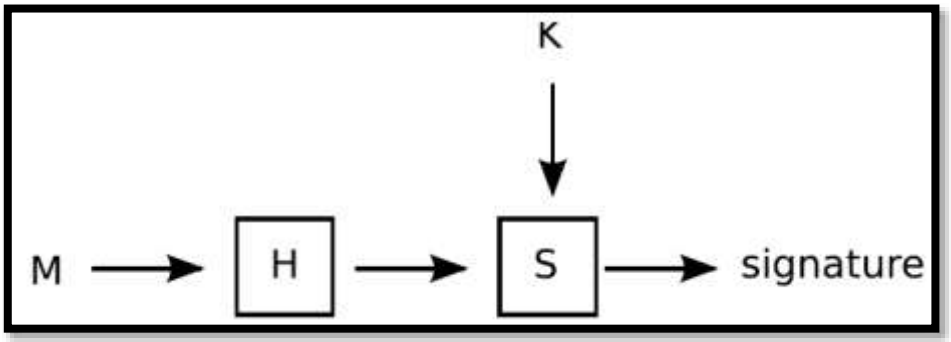
Réponse : Il ne faut pas oublier que l'une des propriétés des fonctions d'hachage est : **Il est difficile de retrouver un autre message** $\boldsymbol{m_O}$ **telle que** $\boldsymbol{H(m_A) = H(m_O)}$

2. Signature et hachage :

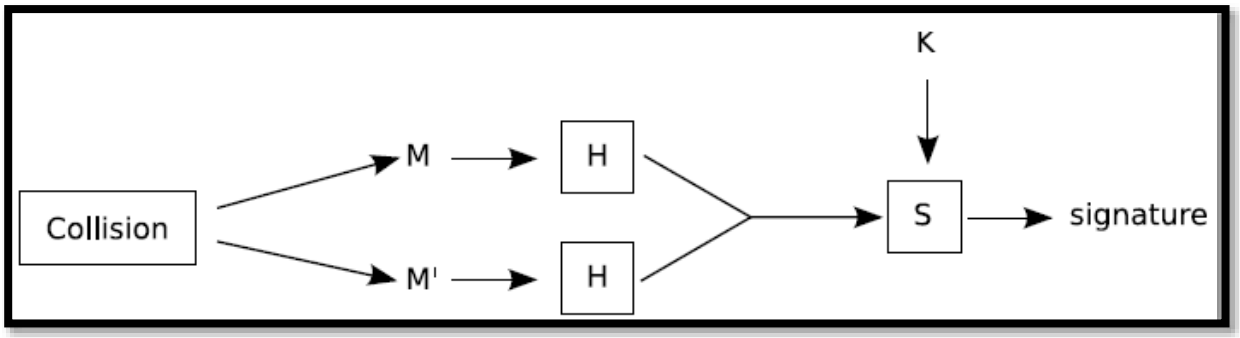
Nous avons vue dans la partie **1.2.2** qu'une signature longue peut poser des problèmes de sécurité, le schéma précédent de la signature été :



Pour obtenir une petite signature, l'idée est de signer le haché du message au lieu du message, on obtient le schéma **Hash-and-Sign :**



Le schéma **Hash-and-Sign** est bien meilleur, mais il peut poser un autre problème de sécurité :



Si **l'émetteur** retrouve un autre message M' différent du message initial M, alors l'**émetteur peut signer le message initial m et prétendre avoir signé m'.**

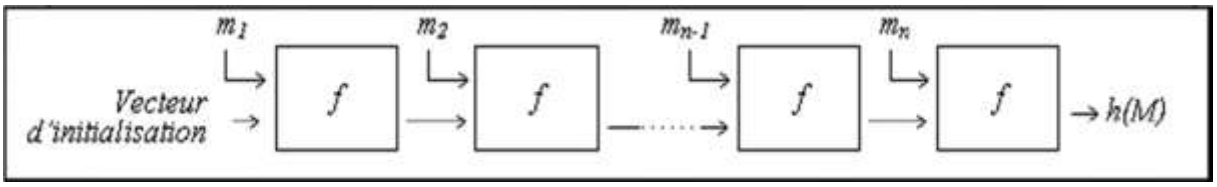
3. Intégrité :

On peut utiliser le hachage comme moyen pour vérifier l'intégrité du message envoyé

3.4. Principe d'une fonction de hachage :

La plupart des fonctions d'hachage sont construites par **itération** d'une fonction de **compression** :

1. Le message M est décomposé en n blocs $m_1, m_2 \ldots . m_n$
2. Pour chaque itération i, on applique la fonction de compression f sur m_i et le résultat de la compression de l'itération précédente ($si\ i = 0\ on\ utilise\ un\ vecteur\ d'initialisation$)
3. L'empreinte $H(m)$ est le résultat de la dernière compression.



3.5. Hachage MD5

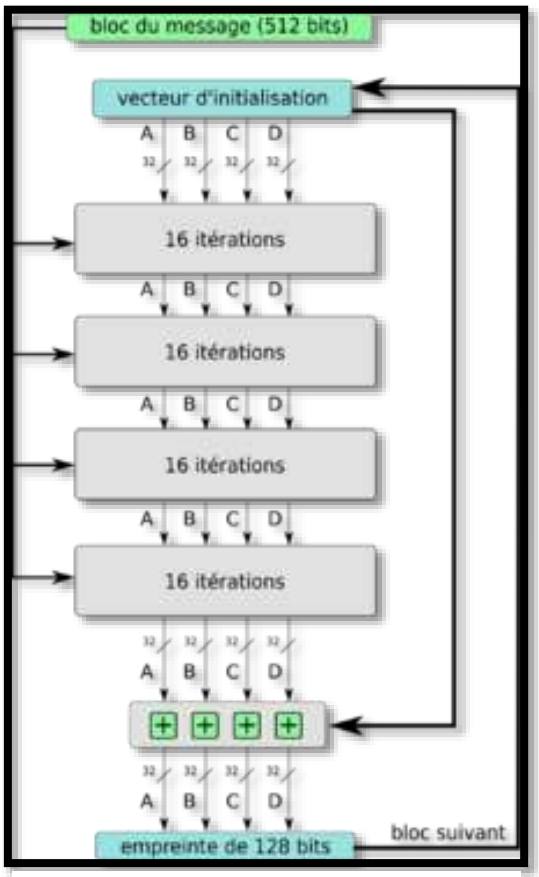
3.5.1. Définition :

MD5 a été développé par Ron Rivest en 1991, elle produit une empreinte de **128** bits à partir d'un texte de taille arbitraire. **MD5** manipule le texte d'entrée par blocs de **512** bits, chaque **bloc** est traité comme 16 **mots** de **32** bits

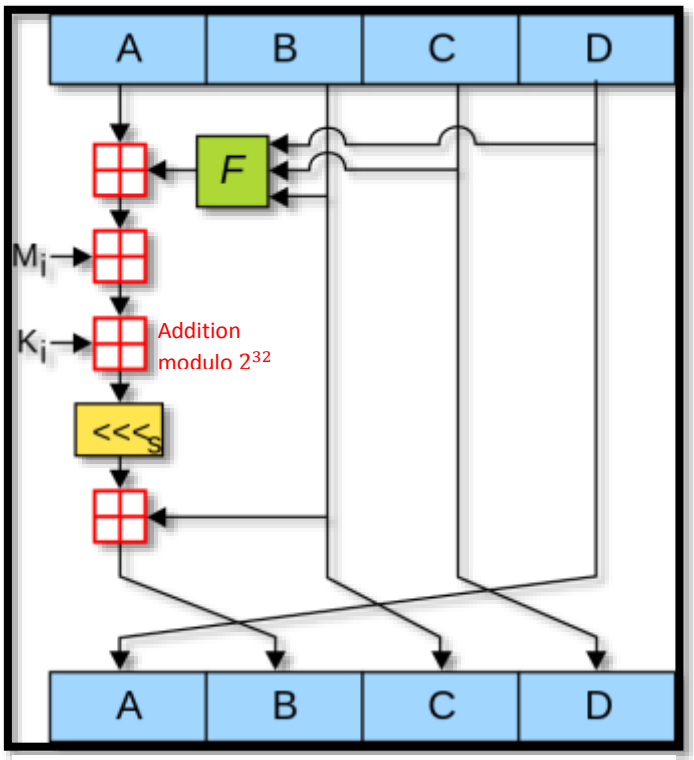
3.5.2. Fonctionnement :

1. La première étape consiste à bourrer le message pour que sa longueur L soit un multiple de **512** bits, le bourrage se fait de la manière suivante :
    a. Ajouter un bit à 1
    b. Ajouter autant de 0 que nécessaire pour arriver à $L\ mod\ 512 = 448\ (512 - 64)$
    c. Ajouter 64 bits représentant **la longueur** du **message sans bourrage** (en little-endian)
2. On travaille sur un état interne de **4*32 bits** $a = h_0, b = h_1, c = h_2, d = h_3$ avec h_0, h_1, h_2, h_3 sont les 4 vecteurs d'initialisation donné par :
    a. $h_0 = 0x01234567$
    b. $h_1 = 0x89ABCDEF$
    c. $h_2 = 0xFEDCBA98$
    d. $h_3 = 0x76543210$

3. On travaille ensuite de la manière suivante :



1 - Déroulement global de MD5, les opérations sur un bloc de message se déroule en 4 rondes



2 - Les rondes sont eux même subdivisées en 16 opérations similaires, basée sur une fonction non linéaire F qui varie selon la ronde, une addition et une rotation vers la gauche

Dans **(2)** :

a. K_i sont des constantes qui dépendent de $i \in [\![0,63]\!]$ $K_i = Int[2^{32} * |\sin(i+1)|]$
b. M_i est un bloc de 32 bits du Bloc (on subdivise M en 16 mots M_i de 32 bits en little-endian, dans chaque **Ronde** on utilise tous les mots du blocs initiale dans un ordre qui dépend du Ronde).
c. la **rotation vers gauche** de s_i bit est noté $<<<_{s_i}$, le nombre de bit dépend de i

La fonction non linéaire dépend de la **ronde** :

1. **Ronde 1 :** $0 \leq i < 15 : F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$
2. **Ronde 2 :** $15 \leq i < 31 : F(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$
3. **Ronde 3 :** $32 \leq i < 47 : F(B,C,D) = B \oplus C \oplus D$
4. **Ronde 4 :** $48 \leq i < 63 : F(B,C,D) = C \oplus (B \vee \neg D)$

*Pour bien comprendre l'algorithme MD5, voir en détaille le **pseudo code** ci-dessous.*

3.5.3. Pseudo Code :

```
//Note: Toutes les variables sont sur 32 bits

//Définir r comme suit :
var entier[64] r, k
r[ 0..15] := {7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22}
r[16..31] := {5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20}
r[32..47] := {4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23}
r[48..63] := {6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21}

//MD5 utilise des sinus d'entiers pour ses constantes:
pour i de 0 à 63 faire
    k[i] := floor(abs(sin(i + 1)) × 2^32)
fin pour

//Préparation des variables:
var entier h0 := 0x67452301
var entier h1 := 0xEFCDAB89
var entier h2 := 0x98BADCFE
var entier h3 := 0x10325476

//Préparation du message (padding) :
ajouter le bit "1" au message
ajouter le bit "0" jusqu'à ce que la taille du message en bits soit égale à 448
(mod 512)
ajouter la taille du message codée en 64-bit little-endian au message

//Découpage en blocs de 512 bits:
pour chaque bloc de 512 bits du message
    subdiviser en 16 mots de 32 bits en little-endian w[i], 0 ≤ i ≤ 15

    //initialiser les valeurs de hachage:
    var entier a := h0
    var entier b := h1
    var entier c := h2
    var entier d := h3

    //Boucle principale:
    pour i de 0 à 63 faire
        si 0 ≤ i ≤ 15 alors
              f := (b et c) ou ((non b) et d)
              g := i
        sinon si 16 ≤ i ≤ 31 alors
              f := (d et b) ou ((non d) et c)
```

```
            g := (5×i + 1) mod 16
        sinon si 32 ≤ i ≤ 47 alors
            f := b xor c xor d
            g := (3×i + 5) mod 16
        sinon si 48 ≤ i ≤ 63 alors
            f := c xor (b ou (non d))
            g := (7×i) mod 16
        fin si
        var entier temp := d
        d := c
        c := b
        b := ((a + f + k[i] + w[g]) leftrotate r[i]) + b
        a := temp
    fin pour

    //ajouter le résultat au bloc précédent:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
fin pour

var entier empreinte := h0 concaténer h1 concaténer h2 concaténer h3 //(en little-endian)
```

3.6. Signature électronique, Non-répudiation et Certificats

Comme vue précédemment, Il est possible de joindre le message envoyé sa **signature** obtenu à l'aide d'une **fonction de hachage** en la chiffrant à l'aide de sa **clé privé** (Dans le cas ou la **confidentialité** du message est nécessaire, le hachage et la signature sont calculé à partir du message clair, le message est crypté ensuite, la signature est rajouté à la fin). Le destinataire utilise la **clé publique** pour déchiffrer la signature pour vérifier l'**intégrité** (en comparant avec l'empreinte du fichier)**, la provenance du message** et **La non répudiation**.

Question : qui confirme que la clé publique est bien associée à l'entité désiré ?

Réponse : Il faut disposer d'un intermédiaire de confiance qui détient et distribue les clés publiques, Cette entité est appelée L'autorité de certification (CA : Certification Authority).

3.6.1. Notion de certificat :

Un **certificat** permet d'associer une clé publique à une entité afin d'en assurer la validité, le certificat est émis par un CA qui chargé de :

1. Délivrer les certificats
2. d'assigner une date de validité aux certificats
3. révoquer éventuellement des certificats avant la date limite en cas de compromission (de la clé (ou du propriétaire).

3.6.2. Format d'un certificat

Le certificat est construit suivant une **norme reconnue internationalement** pour faciliter l'interopérabilité.

Un certificat est un petit fichier séparé en deux parties :

1. Un contenant les informations suivantes (Norme ISO X509)
    a. le nom de l'autorité de certification
    b. le nom du propriétaire du certificat
    c. la date de validité du certificat
    d. l'algorithme de chiffrement utilisé
    e. la clé publique du propriétaire
2. Une autre contenant **La signature de l'autorité de certification** (la confiance s'établit en faisant confiance à une autorité supérieure pour la signature)