Bases de la cryptographie: Authentification, intégrité des données et non-répudiation .

Michaël Quisquater (Maître de Conférences,UVSQ)

Première partie I

Mécanisme asymétrique d'authentification et d'intégrité des données et non-épudiation

Plan de la première partie I

1. Notion de signature numérique
   - Objectifs
   - Sécurité
2. Fonction de hachage
   - Utilité, définition et propriétés de sécurité requises
   - Recherche de collision sur les fonctions de hachage
   - Notion de fonction de compression à sens unique
   - Construction de fonction de compression à sens unique
     - à partir d'algorithme de chiffrement par blocs
     - Constructions dédiées
   - Fonction de hachage à partir de fonction de compression
   - Exemples de fonctions de hachage
3. Algorithmes de signature numérique

Plan de la première partie II

- Algorithme de signature DSA (ou DSS)
- Primitive RSA en signature
- Attaques sur la primitive RSA en signature
- Standard RSA en signature : PKCS#1 v1.5 et v2.1(PSS)

4. Génération des paramètres d'un algorithme cryptographique
   - Test de primalité de Fermat
   - Génération de générateur d'un groupe fini
5. Librairies des grands nombres et crypto.
   - Librairies des grands nombres
   - Librairies cryptographiques

Notion de signature numérique

Objectif de la signature

Réaliser les mêmes fonctionnalités pour un message electronique que celles de la signature d'un document papier. A savoir :

- intégrité des données (un document signé altéré n'a plus de valeur : dans le cas numérique la signature permet de détecter l'alteration d'un document : + fort !).
- authenticité des données (Signature assure la provenance du document)
- non-répudiation (signature à valeur d'engagement)

Notion de signature numérique (suite)

Remarques :

- La signature manuscrite n'est pas parfaite, la signature numérique non plus (attaques !).
- La signature numérique est attaché au document, la signature numérique va devoir établir un lien "mathématique" en le fichier de la signature et le document associé à cette signature.
- Un algorithme de signature numérique est composé d'un algorithme de génération des clés, d'un algorithme de signature et d'un algorithme de vérification

Notion de signature numérique (suite)

Dans le modèle simple, on souhaite que

- uniquement la personne autorisée puisse signer. L'algorithme de signature, noté S_s, dépend de données secrètes s (et éventuellement de données publiques).
- n'importe qui puisse vérifier la signature. L'algorithme de vérification, noté V_v, ne dépend que de la donnée publique v (et éventuellement d'autres données publiques).

Remarques :

- La signature d'un message est un couple $(m, S_s(m))$ où m est le message signé.
- L'algorithme de de vérifie que la signature $(m, S_s(m))$ est bien cohérent, c-à-d que $S_s(m)$ est bien l'image de m par S_s quand on utilise la clé s.

Signature numérique : Notion de sécurité

Objectif de l'attaquant :

- Retrouver la clé de signature (cassage total)
- Signer un message imposé quelconque (forge universelle)
- Signer un message choisi (forge sélective)
- Signer un message quelconque (forge existentielle)

L'attaquant peut disposer de plus ou moins de données (seulement la clé publique de signature, un ensemble de message signé connus, un ensemble de message signé choisi).

remarque : Lorsque l'on est capable de signer un message quelconque (non nécessairement choisi) à partir d'un ou d'un ensemble de message signé, on parle de malléabillité.

Signature numérique et primitive de signature

- En pratique un algorithme de signature est composé d'une primitive de signature qui permet de signer des messages courts.
- Une procédure additionnelles est appliquée à cette primitive pour constituer l'algorithme de signature qui permet de signer des messages de longueur quelconque (paradigme "hash and sign"). Cette procédure additionnelle utilise la notion de fonction de hachage.

Utilité

Définition

Une fonction de hachage est une application h de l'espace des séquences binaires de longueurs arbitraires dans l'espace des séquences de longueur fixée n, i.e. $h : \{0,1\}^* \rightarrow \{0,1\}^n$.

Remarques :

- Dans cette définition aucune notion de sécurité n'est associée.

Utilité

- Dans le cadre de la signature, on calculera d'abord l'image du message par une fonction de hachage (cette image est appelée emprunte) et on signera ensuite cette valeur via la primitive de signature ("Hash and Sign paradigm").
- Les fonctions de hachage ont un usage très général en cryptographie (design de MAC, générateur pseudo-aléatoires etc). Ces sont ces usages qui vont motiver les notions de sécurité des fonctions de hachage.

Sécurité requise : Preimage resistant

Définition

Une fonction de hachage $h : \{0,1\}^* \to \{0,1\}^n$ *est preimage-resistant si pour n'importe quelle valeur* $y \in \{0,1\}^n$*, il est calculatoire difficile (non polynomial) de trouver une preimage* $x \in \{0,1\}^*$ *telle que* $h(x) = y$*.*

Sécurité requise : Second preimage resistant

Définition

Une fonction de hachage $h : \{0,1\}^* \to \{0,1\}^n$ est second preimage resistant si étant donné $x \in \{0,1\}^*$, il est calculatoire difficile (non polynomial) de trouver $x' \in \{0,1\}^*$ avec $x \neq x'$ tel que $h(x) = h(x')$.

Sécurité requise : Collision resistant

Définition

Une fonction de hachage $h : \{0,1\}^* \to \{0,1\}^n$ est collision resistant si il est calculatoire difficile (non polynomial) de trouver $x, x' \in \{0,1\}^*$ avec $x \neq x'$ tels que $h(x) = h(x')$.

Lien entre les notions

Théorème

- *collision resistant* $\Rightarrow$ *second preimage resistant*
- *second preimage resistant* $\Rightarrow$ *preimage-resistant*

Remarque :

- Si on considère que 2^{80} est une complexité de calcul accessible, on peut faire une recherche exhaustive pour chacune des propriétés pour $n \leq 80$. Par conséquent, il faut que n soit supérieur à 80. Est-ce suffisant ?

Paradoxe des anniversaires

Théorème

Considérons une urne contenant N boules de couleurs distinctes. Alors,

- *La probabilité d'avoir au moins deux boules identiques après M tirages avec remise est donnée par*

$$1 - \left(\frac{N-1}{N}\right)\left(\frac{N-2}{N}\right)\cdots\left(\frac{N-M+1}{N}\right).$$

- *Aussi,* $1.17\sqrt{N}$ *tirages avec remise dans l'urne sont nécessaires pour avoir une chance sur deux de tirer au moins deux boules de la même couleur.*

Paradoxe des anniversaires (suite)

Dans un amphi de 23 étudiants, on a une chance sur deux d'avoir deux étudiants avec la même date d'anniversaire. Ce fait peut être expliqué par le théorème précdent, dit "le paradoxe des anniversaires". En effet, pour $N = 365$ il faut $1.17 * \sqrt{365} \approx 23$ personnes (=tirages avec remise) pour avoir une chance sur deux que deux personnes aient leur anniversaire le même jour.

Paradoxe des anniversaires

Preuve : La probabilité P d'avoir des tirages (avec remise) différents dans une urne à N éléments est

- Pour deux tirages : $P = \frac{N-1}{N}$.
- Pour trois tirages : $P = \left(\frac{N-1}{N}\right)\left(\frac{N-2}{N}\right)$.
- Pour quatre tirages : $P = \left(\frac{N-1}{N}\right)\left(\frac{N-2}{N}\right)\left(\frac{N-3}{N}\right)$.
- etc
- Pour Q tirages : $P = \left(\frac{N-1}{N}\right)\left(\frac{N-2}{N}\right)\cdots\left(\frac{N-Q+1}{N}\right)$.

La probabilité d'avoir au moins deux boules de la même couleur vaut $1 - P$, d'où le premier résultat.

Paradoxe des anniversaires

Preuve (suite) :
Considérons l'expansion de $e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} \cdots$. Pour x petit, on peut approximer e^{-x} par $1 - x$. Donc $P = \prod_{i=1}^{Q-1} (1 - \frac{i}{N}) \approx \prod_{i=1}^{Q-1} e^{-\frac{i}{N}} = e^{-\frac{1}{N}\sum_{i=1}^{Q-1} i}$.

Notons que $\sum_{i=1}^{Q-1} i = \frac{Q(Q-1)}{2} \approx \frac{Q^2}{2}$. Donc,

$$P \approx e^{-\frac{Q^2}{2N}}.$$

En imposant que $P = 0.5$, on a $0.5 = e^{-\frac{Q^2}{2N}}$. Donc $ln(0.5) = -\frac{Q^2}{2N}$ ou encore $2ln(2)N = Q^2$. Par conséquent,

$$Q = \sqrt{2ln(2)N} \approx 1.17\sqrt{N}.$$

Application du paradoxe des anniversaires à la recherche de collisions

Soit une fonction de hachage $h : \{0,1\}^* \rightarrow \{0,1\}^n$.

Algorithme pour trouver une collision :

1. Tirer $1.17 \cdot 2^{n/2}$ élements $x_i \in \{0,1\}^*$.
2. Evaluer $h(x_i)$ pour les différents x_i (ces valeurs $h(x_i)$ peuvent être considérées comme aléatoires dans $\{0,1\}^n$)
3. Avec probabilité $1/2$, il existe au moins deux valeurs identiques dans l'ensemble des $h(x_i)$, les deux x_i correspondant constituent une collision.

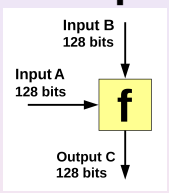
Conséquence : si on considère 2^{80} opérations comme accessible, il faut une taille d'emprunte de 160 bits minimum pour que la fonction de hachage soit résitante aux collisions.

Notion de fonction de compression à sens unique

Définition

Une fonction de compression est une fonction de hachage dont les éléments du domaine ont une longueur fixée.

Exemple :



Remarques :

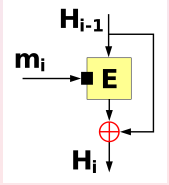
- La stratégie consiste généralement à d'abord construire des fonctions de compression et ensuite de les utiliser pour construire des fonctions de hachage.
- On essaie de réduire la sécurité de hachage à la sécurité de la fonction de compression à partir de laquelle elle est construite.

Davies-Meyer

- Soit $E_\cdot(\cdot)$ un algorithme de chiffrement par bloc dont la longueur des clés est k.
- Considérons un message m dont la longueur est un multiple de k (*padder* le message si nécessaire). Soit m_i les différents blocs de longueur k.
- Soit IV une valeur constante de la taille du bloc de $E_\cdot(\cdot)$.

Alors, la fonction de compression de Davies-Meyer est définie par la récurrence

$$H_i = E_{m_i}(H_{i-1}) \oplus H_{i-1} \text{ avec } H_0 = IV\,.$$



Remarque :

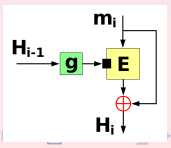
- MD5, SHA-1 and SHA-2 sont construites sur ce modèle-ci.

Matyas-Meyer-Oseas

- Soit $E_{\cdot}(\cdot)$ un algorithme de chiffrement par bloc dont la longueur des clés est k et du bloc est t.
- Considérons un message m dont la longueur est un multiple de t (*padder* le message si nécessaire). Soit m_i les différents blocs de longueur t.
- g une fonction qui envoie des mots de t bits vers des mots de k bits
- Soit IV une valeur constante de la taille du bloc de $E_{\cdot}(\cdot)$.

Alors, la fonction de compression de Matyas-Meyer-Oseas est définie par la récurrence

$$H_i = E_{g(H_{i-1})}(m_i) \oplus m_i \text{ avec } H_0 = IV\,.$$

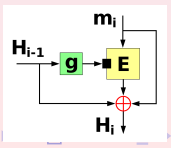


Miyaguchi-Preneel

- Soit $E_{\cdot}(\cdot)$ un algorithme de chiffrement par bloc dont la longueur des clés est k et du bloc est t.
- Considérons un message m dont la longueur est un multiple de t (*padder* le message si nécessaire). Soit m_i les différents blocs de longueur t.
- g une fonction qui envoie des mots de t bits vers des mots de k bits
- Soit IV une valeur constante de la taille du bloc de $E_{\cdot}(\cdot)$.

Alors, la fonction de compression de Matyas-Meyer-Oseas est définie par la récurrence

$$H_i = E_{g(H_{i-1})}(m_i) \oplus H_{i-1} \oplus m_i \text{ avec } H_0 = IV\,.$$



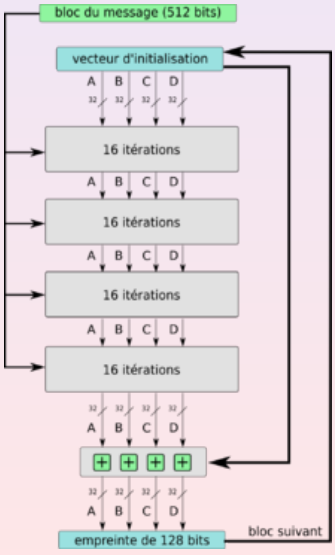
Sécurité

Sans entrer dans le détail, si l'algorithme de chiffrement par bloc présente certaines des faiblesses ci-dessous, la fonction de compression peut être attaquée dans certains cas :

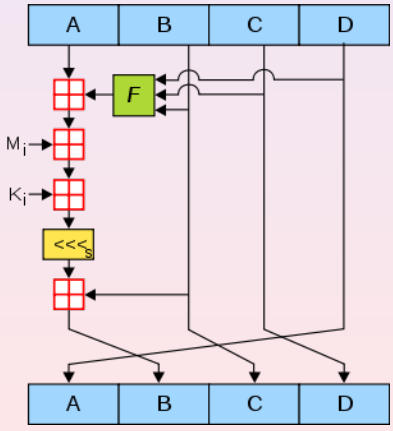
1. Propriété de complementation : $E_K(m) = \overline{E_{\overline{K}}(\overline{m})}$,
2. Clé faible : une clé K pour laquelle $E_K(E_K(m)) = m$ pour tout m,
3. Paire de clés semi-faibles : K_1, K_2 telles que $E_{K_1}(E_{K_2}(m)) = m$ pour tout m,
4. Point fixe : $E_K(m) = m$ pour un certain K et m,
5. Collision de clés : $E_K(m) = E_{K'}(m)$ pour un certain m et K, K'.

Construction de fonctions de compression à sens unique dédiées

Schéma global de MD5 :

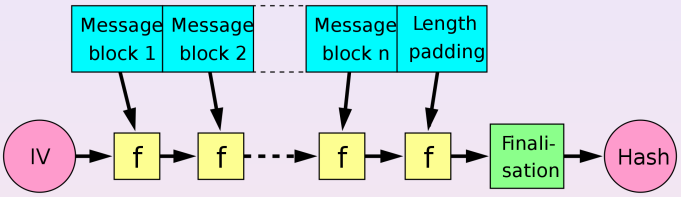


Une itération :



PS : modèle de Davies-Meyer.

Construction de Merkle-Damgård



- Soit m le message à chiffrer et L la longueur de l'argument de la fonction de compression.
- Concaténer le padding $10000\cdots000LG$ où LG est la longueur de en nombre de bits et le nombre de zéros est choisi de telle façon que la longueur du message et du padding est un multiple de L.
- Appliquer la procédure ci-dessus. IV est souvent le vecteur tout à zéro.

Construction de Merkle-Damgård (suite)

Théorème

Si on connait une collision sur la fonction de hachage h, alors on peut fabriquer (en temps polynomial) une collision sur la fonction de compression f.

La signification de ce théorème : si il y a un problème de collision sur la fonction de hachage h alors il y avait déjà un probalème avec la fonction decompression f. On réduit la sécurité de la fonction h à la sécurité de la fonction f.

Cas pratique de collision : Fichiers postcript

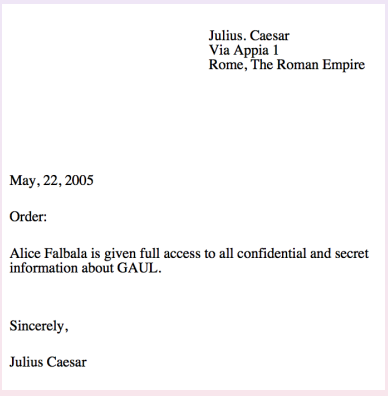
Objectif de l'attaque :

A partir d'une collision connue (qui n'a pas de sens liguistiquement) sur une fonction de hachage utilisée avec la construction de Merkle-Damgård, créer deux fichiers postcript différents dont la visualisation est intelligible (et de sens différents) et qui ont des empruntes identiques.

Cette attaque tire parti d'une faiblesse de la construction de Merkle-Damgård :

Si il existe deux messages m_1 et m_2 tels que $h(m_1) = h(m_2)$ alors $h(m_1||S) = h(m_2||S)$ pour tout S et $||$ est l'opération de concaténation.

Cas pratique de collision : Fichiers postcript



Julius. Caesar
Via Appia 1
Rome, The Roman Empire

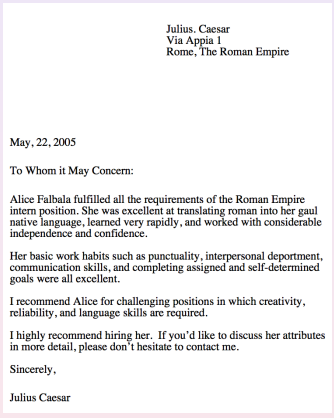
May, 22, 2005

Order:

Alice Falbala is given full access to all confidential and secret information about GAUL.

Sincerely,

Julius Caesar



Julius. Caesar
Via Appia 1
Rome, The Roman Empire

May, 22, 2005

To Whom it May Concern:

Alice Falbala fulfilled all the requirements of the Roman Empire intern position. She was excellent at translating roman into her gaul native language, learned very rapidly, and worked with considerable independence and confidence.

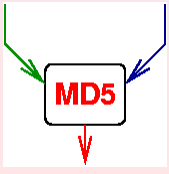
Her basic work habits such as punctuality, interpersonal deportment, communication skills, and completing assigned and self-determined goals were all excellent.

I recommend Alice for challenging positions in which creativity, reliability, and language skills are required.

I highly recommend hiring her. If you'd like to discuss her attributes in more detail, please don't hesitate to contact me.

Sincerely,

Julius Caesar



a25f7f0b 29ee0b39 68c86073 8533a4b9

Cas pratique de collision : Fichiers postcript

Principe de l'attaque :

Les documents cibles intelligibles sont $T1$ et $T2$.

On crée les deux documents $Y1$ et $Y2$ suivant :

$$Y1 = \overbrace{preamble; put(R1);}^{X1} \underbrace{put(R1); if(=) then T1 else T2;}_{S}$$

$$Y2 = \underbrace{preamble; put(R2);}_{X2} \overbrace{put(R1); if(=) then T1 else T2;}^{S}$$

et $X1$ et $X2$ sont tels que $MD5(X1) = MD5(X2)$ (attaque sur la fonction MD5)

Par l'observation sur la faiblesse de la construction de Merkle-Damgård, $MD5(Y1) = MD5(Y2)$.

$Y1$ est affiché comme $T1$ car le test est $R1 = R1$ (toujours vrai)
$Y2$ est affiché comme $T2$ car le test est $R1 = R2$ (jamais vrai)

Tableau récapitulatif de fonctions de hachage (cassées)

- MD4 : Design par R. Rivest (1990), cassée par Dobbertin (1995)
- MD5 : Design par R. Rivest (1991), faille grave en 1996, cassée complètement par Wang *et. al* (2004)
- SHA-0 : Design par la NSA (1993) : cassée par A. Joux (2004)
- SHA-1 : Design par la NSA (1995) : cassée par Wang *et. al* (2005)

Tableau récapitulatif de fonctions de hachage (valides)

- RIPEMD-160 : Design par Hans Dobbertin, Antoon Bosselaers et Bart Preneel (1996). Existe aussi en 128, 256 et 320 bits. Une version originale (non-utilisée) a été cassée par Wang (2004)
- SHA-2 (SHA-256, SHA-384 ou SHA-512 bits au choix) : Design par la NSA (2001). Basée sur la construction de Merkle-Damgård,
- Whirlpool : Design par Vincent Rijmen et Paulo Barreto (2003). Basée sur la construction de Miyaguchi-Preneel (+AES). Emprunte de 512 bits.

Tableau récapitulatif de fonctions de hachage (futures)

Un appel à candidats (SHA-3) pour les fonction de hachage a été réalisé par le NIST (similaire à l'AES dans le cadre des algorithmes de chiffrement par bloc). Le candidat final a été choisi le 2 octobre 2012. L'algorithme SHA-3 n'est donc dérivé de SHA-2. Cet algorithme n'est pas destiné à remplacer SHA-2, qui n'est pas cassé à l'heure actuelle. Cet algorithme est destiné à fournir une alternative à SHA-2.

Algorithme de signature DSA (ou DSS)

Cet algorithme est basé sur la difficulté du logarithme discret (Digital signature algorithm ou standard).

Génération des paramètres :

- Soit q un nombre premier,
- Soit p un nombre premier tel que $q \mid (p-1)$
- Soit α un générateur du sous-groupe cyclique de $\mathbb{Z}_p^*$ d'ordre q (il suffit de choisir un générateur g de $\mathbb{Z}_p^*$ et de poser $\alpha = g^{\frac{p-1}{q}}$),
- Soit $x \in_R \{2, \cdots, q-1\}$ et $y = \alpha^x \bmod p$

La clé publique est (p,q,α,y) et la clé privé est (x)

Algorithme de signature DSA (ou DSS)

Algorithme de signature :

- Soit $m \in \mathbb{Z}_q$ le message à signer,
- Soit $k \in_R \{1, \cdots, q-1\}$ (gardé secret)
- Soit $r = (\alpha^k \bmod p) \bmod q$ (si nul, il faut choisir un autre k)
- Soit $s = k^{-1}(m + x \cdot r) \bmod q$ (si nul, il faut choisir un autre k)

La signature est triplet (m, r, s).

Remarque : L'algorithme est probabiliste (il peut y avoir plusieurs signature pour un même message)

Algorithme de signature DSA (ou DSS)

Algorithme de vérification de signature :

Soit la signature (m, r, s) et la clé publique (p, q, α, y).

- Calculer $w = s^{-1} \bmod q$
- Calculer $u_1 = w \cdot m \bmod q$ et $u_2 = r \cdot w \bmod q$
- Vérifier si $(\alpha^{u_1} \cdot y^{u_2} \bmod p) \bmod q = r$. Si c'est le cas la signature est acceptée, sinon pas.

Exercice : Vérifier que la signature est acceptée quand la signature est correcte (voir TD)

Primitive RSA en signature

$n = p \cdot q$

$v \in \mathbb{Z}^*_{\Phi(n)}$

$s = v^{-1} \bmod \Phi(n)$

n, v

$(m, S(m))$ ⟶ $S(m)^v \stackrel{?}{=} m \bmod n$

avec $S(m) = m^s \bmod n$

- s est l'exposant secret de signature,
- v est l'exposant public de vérification de signature.
- p, q des premiers distincts de même taille.

Attaques sur la primitive RSA en signature

- Cassage total :
  - factoriser n pour trouver $\Phi(n)$ et finalement s
  - Si s est trop petit, l'attaque de Wiener (voir chiffrement) est d'application.
- Forge existentielle : la signature du message $m^v \bmod n, S(m^v \bmod n) = m$ est toujours une signature valide. $\rightarrow$ utilisation d'un padding.
- Malléabilité : si $(m, S(m))$ est une signature valide, $(m^r, S(m)^r \bmod n)$ l'est aussi. (idem si $(m_1, S(m_1))$ et $(m_2, S(m_2))$ sont valides alors $(m_1 \cdot m_2 \bmod n, S(m_1) \cdot S(m_2) \bmod n)$ l'est aussi.) $\rightarrow$ utilisation d'un padding.

PKCS (publi- key cryptography standards)

PKCS (publi- key cryptography standards) ont été définis par RSA Laboratories (fondateurs : Ron Rivest, Adi Shamir et Leonard Adleman ; 1319 employés en 2007). Racheté par EMC Corporation pour 2.1 milliards de $.

Parmi ces standards, on retrouve le standard de signature PKCS#1 v1.5. Ce standard est construit sur le paradigme "Hash and Sign" en utilisant la primitive RSA. Ce standard ne possède pas de preuve de sécurité mais a été conçu en tenant compte des attaques connues.

Standard RSA en signature : PKCS#1 v1.5

Soit n un module RSA et m un message à signer.
Signature :

$$(m, S(m) = (00\ 01\ FF\ FF \cdots FF\ 00 \mid\mid c_k \mid\mid h(m))^s \bmod n)\,.$$

Remarques :

- 00, 01, FF sont des expressions hexadécimales.
- 00 pour que le "message" soit plus petit que le module, 01 pour qu'il ne soit pas petit, c_k est l'identifiant de la fonction de hachage utilisée.

Vérification :

$$(00\ 01\ FF\ FF \cdots FF00 \mid\mid c_k \mid\mid h(m)) = S(m)^v \bmod n\,.$$

Toujours utilisé et n'est pas cassé.

Standard RSA en signature : ISO 9796-2

Signature :

$$(M[2], S(m) = (6A \mid\mid M[1] \cdots \mid\mid h(m) \mid\mid BC)^d \bmod n)\,.$$

où $M[1]$ est la première partie du message et h est SHA-1.
Intérêt : gagner de la bande passante.

Cassé en 2009 par Coron, Naccache, Tibouchi, Weinmann (Luxembourg et France) et ne dépend pas de la fonction de hachage utilisée. Forger une signature coûte 800 $.

Standard RSA en signature : PKCS#1 v2.1(PSS)

PKCS#1 v2.1(PSS) : Preuve de sécurité calculatoire (n doit faire 1536 bits). C'est un schéma probabiliste. Actuellement recommandé.

Test de primalité

Les systèmes à clé publique nécessitent la génération de grands nombres premiers (> 160 bits).

- Méthode déterministe : on est sûr que le nombre généré est premier. On tire un nombre au hasard et on teste de façon déterministe si ce nombre est premier ou pas. Ce problème est dans $\mathcal{P}$ (théoriquement bon mais trop coûteux en pratique).
- Méthode probabiliste : On peut dire qu'un nombre est premier avec une grande probabilité (si un nombre est premier, il réussit le test toujours, par contre si il n'est pas premier cela peut arriver qu'il réussisse le test). Ex. méthode Fermat (simple), Miller/Rabin (+ complexe et pas vu). Ce sont les méthodes probabilistes qui sont utilisées en pratique.

Test de primalité de Fermat

Ce test est basé sur le petit théorème de Fermat.

Théorème

Soit p un nombre premier, alors $a^{p-1} = 1 \bmod p$ pour tout $a \in \mathbb{Z}_p^*$.

Conséquence : Soit $p \in \mathbb{N} \setminus \{0\}$. Si on trouve $a \in \mathbb{Z}_p \setminus \{0\}$ tel que $a^{p-1} \neq 1 \bmod p$ alors p est composé (non premier).

L'idée de l'algorithme permettant de tester la primalité d'un nombre n consiste à tester si $a^{n-1} = 1 \bmod n$ pour un certain nombre de "a".

Test de primalité de Fermat

Algorithme 1 : Test de primalité de Fermat

Données : n (dont on veut tester la primalité) et t un paramètre de sécurité
Résultat : primalité de n avec une certaine probabilité
pour $i = 1, \ldots, t$ **faire**
  $a \in_R \{2, \cdots, n-1\}$
  calculer $r = a^{n-1} \bmod n$
  si $r \neq 1$ **alors**
    retourner n est composé
  fin
fin
retourner "n" est premier

Test de primalité de Fermat

Malheureusement, il existe des nombres n dit pseudo-premier :

Définition

Soit n un naturel impair composé et $a \in \mathbb{N}$ tel que $1 \leq a \leq n-1$. Si $a^{n-1} = 1 \bmod n$ alors n est dit pseudo-premier en base a.

Il existe même des nombres pseudo-premier pour toute base a pour laquelle $pgcd(a, n)$. Pour ces nombre, il y a de grande chance que l'algorithme déclare le nombre premier alors qu'il est premier.

Test de primalité de Fermat

Définition

Un nombre de Carmichaël est un nombre vérifiant $a^{n-1} = 1 \bmod n$ pour tout $a \in \mathbb{Z}_n^*$.

Exemple de nombre de Carmichaël : $561 = 3 \cdot 11 \cdot 17$.

Remarque : Notons que ces nombres sont rares. Le test de Miller/Rabin raffine le test de Fermat et n'échoue pas sur ces nombres.

Génération de générateur d'un groupe fini

Soit G un groupe fini tel que $| G | = \prod_{i \in I} p_i^{\alpha_i}$ avec p_i premiers distincts et $\alpha_i \in \mathbb{N} \setminus \{0\}$.

Observation :

Soit $x \in G$ et e l'élément neutre de G. On sait que l'ordre de x divise l'ordre de G, donc l'ordre de x est de la forme $\prod_{i \in J} p_i^{\beta_i}$ avec $J \subseteq I$ et $\beta_i \leq \alpha_i$ pour tout $i \in J$.

Par conséquent, si $x^{\frac{|G|}{p_i}} \neq e$ pour un certain $i \in I$, cela implique que l'ordre de x est un multiple de $p_i^{\alpha_i}$.

De cette observation suit l'algorithme suivant :

Génération de générateur d'un groupe fini

Algorithme 2 : Génération de générateur d'un groupe fini

Données : G cyclique d'ordre $\prod_{i \in I} p_i^{\alpha_i}$ premiers distincts et $\alpha_i \in \mathbb{N} \setminus \{0\}$.

Résultat : un générateur x de G

pour $x \in G$ **faire**
  pour $i \in I$ **faire**
    $temp = x^{\frac{|G|}{p_i}}$
    si $temp = e$ **alors**
      break
    fin
  fin
  retourner x
fin

Librairies des grands nombres

En pratique, un ordinateur ne gère que des nombres de 32 ou 64 bits la plupart du temps → librairies des grands nombres

Librairies de grands nombres

- (C/C++) : `http://gmplib.org/`
- Java : BigInteger `http://java.sun.com/j2se/1.4.2/docs/api/java/math/BigInteger.html`
- Python : natif `www.python.org/`
- PHP :
  - BC Math : `http://www.php.net/manual/en/book.bc.php`
  - GMP : `http://www.php.net/manual/en/book.gmp.php`

Librairies cryptographiques

Librairies cryptographiques

- (C/C++) : OpenSSL : `www.openssl.org/`
- Java : JCA : `http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html`
- Python :
  - M2CRYPTO `http://freshmeat.net/projects/m2crypto/`
  - PyMe `http://pyme.sourceforge.net/`
- PHP : Hash, Mcrypt, Mhash, OpenSSL : `http://phpbuilder.com/manual/en/refs.crypto.php`

Deuxième partie II

Conclusion

Conclusion

	Confidentialité	Intégrité	Authentification	Non-répudiation	Problèmes
Crypto sym.	chiff. sym. (bloc et flot)	MAC	Pas de solution	Pas de sol.	Gestion des clés
Crypto asym.	chiff. asym. (RSA,..)	Sign. (RSA,..)	Sign.	Sign.	Temps de calcul

Source des images :

- png factory
- tux factory
- wikipedia