Algorithmes de hachage

Algorithmes de hachage

- On constate des similitudes dans l'évolution des fonctions de hachage et les chiffrements symétriques
  - Puissance croissante des attaques par force brute
  - Ce qui pousse à l'évolution dans les algorithmes
  - du DES à l'AES dans des chiffrements symétriques
  - de MD4 et de MD5 à Sha 1 et à Ripemd 160 dans des algorithmes de hachage
- Ces algorithmes ont tendance à employer la même structure itérative (Feistel) que les chiffrements symétriques

MD5

MD5 - présentation

- conçu par Ronald Rivest (le R dans RSA)
- Le dernier d'une série (MD2, MD4)
- produit un condensé de 128 bits
- était jusqu'à récemment l'algorithme de hachage le plus largement répandu
  - La cryptanalyse et l'attaque par force brute l'on affaibli
- Spécification Internet : RFC 1321

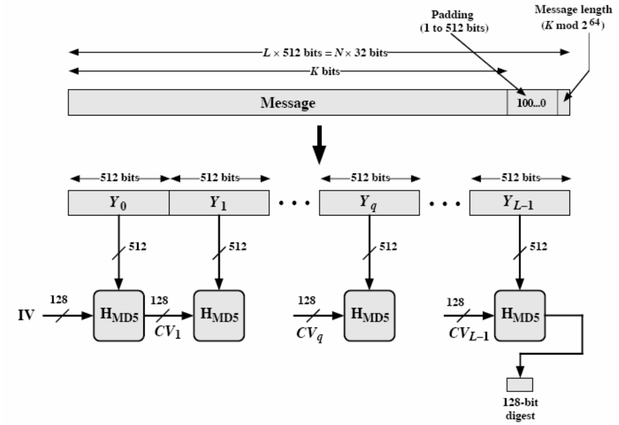
MD5 - Vue d'ensemble

1. Complétion :
   - ajout de padding si nécessaire afin que le message ait une longueur de 448 mod 512
2. Ajout de la longueur :
   - on ajoute la longueur réelle du message (sur 64 bits) après les 448 bits ! 512 bits
3. Initialisation :
   - initialiser 4 buffers de 32 bits chacun (A,B,C,D)

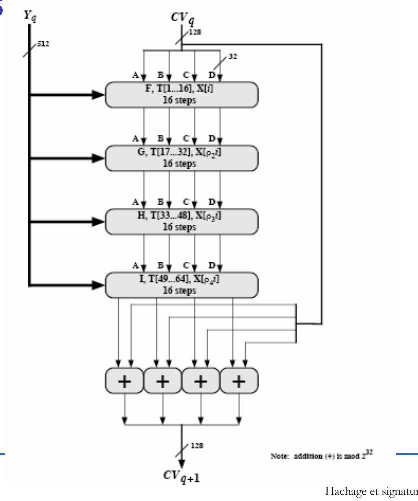
MD5 – Vue d'ensemble

4. Calcul itératif :
   - traiter le message par blocs de 512 bits
   - 4 rondes de 16 opérations fonction du bloc (512), des buffers et de fonctions primitives
   - Le résultat (4*32 bits) est utilisé pour l'initialisation des registres suivants
5. Le résultat final est obtenu en additionnant A,B,C,D

MD5 – Vue d'ensemble



MD5 - Rondes



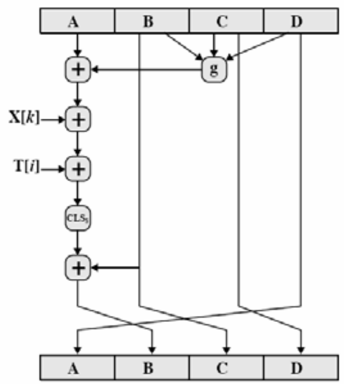
MD5 - fonction de compression

- chaque ronde a 16 itérations de la forme :
  - a = b+((a+g(b, c, d)+X[k]+T[i])<<<s)
  - a, b, c, d se rapportent aux 4 buffers, mais sont utilisés selon des permutations variables
    - Note : cela ne met à jour qu'un seul des buffers (de 32 bits)
    - Après 16 étapes, chaque buffer a été mis à jour 4 fois
  - g(b, c, d) est une fonction non linéaire différente dans chaque ronde (F,G, H, I)
  - T[i] est une valeur constante dérivée de sin
  - <<< s : décalage circulaire à gauche (pour chaque buffer séparément) de s bits

MD5 - Algorithme

- $CV_0 = IV$
- $CV_{q+1}=\Sigma_{32}[CV_q, RF_I (Y_q,RF_H(Y_q, RF_G(Y_q, RF_F(Y_q, CV_q))))]$
- $MD = CV_{L-1}$
- Où :
  - IV : valeur initiale des registres ABCD
  - Y_q : le q^e bloc de 512 bits du message
  - L : le nombre de blocs de 512 bits dans le message
  - CV_q : variable chainée obtenue par la manipulation du q^e bloc
  - RF_x : fonction primitive dépendante de la ronde en cours
  - MD : résultat final
  - Σ_{32} : addition modulo 2^{32}

MD5 - Opération élémentaire



MD5 - Calculs des valeurs

- Valeurs initiales des registres :
  - A : 01 23 45 67
  - B : 89 AB CD EF
  - C : FE DC BA 98
  - D : 76 54 32 10
- Primitives :
  - F(b,c,d) = (b ∧ c) ∨ (¬ b ∧ d)
  - G(b,c,d) = (b ∧ d) ∨ (c ∧ ¬ d)
  - H(b,c,d) = b ⊕ c ⊕ d
  - I(b,c,d) = c ⊕ (b ∨ ¬ d)
- Valeurs pour X[k]
  - i = itération, k=
  - $\rho_1(i) = i$
  - $\rho_2(i) = (1+5i) \bmod 16$
  - $\rho_3(i) = (5+3i) \bmod 16$
  - $\rho_4(i) = 7i \bmod 16$


```
/* Process each 16-word (512-bit) block. */
For q = 0 to (N/16) - 1 do
    /* Copy block q into X. */
    For j = 0 to 15 do
        Set X[j] to M[q*16 + j].
    end /* of loop on j */

    /* Save A as AA, B as BB, C as CC, and
    D as DD. */
    AA = A
    BB = B
    CC = C
    DD = D

    /* Round 1. */
    /* Let [abcd k s i] denote the operation
    a = b + ((a + F(b,c,d) + X[k] + T[i]) <<<s).
    Do the following 16 operations. */
    [ABCD    0   7   1]
    [DABC    1  12   2]
    [CDAB    2  17   3]
    [BCDA    3  22   4]
    [ABCD    4   7   5]
    [DABC    5  12   6]
    [CDAB    6  17   7]
    [BCDA    7  22   8]
    [ABCD    8   7   9]
    [DABC    9  12  10]
    [CDAB   10  17  11]
    [BCDA   11  22  12]
    [ABCD   12   7  13]
    [DABC   13  12  14]
    [CDAB   14  17  15]
    [BCDA   15  22  16]

    /* Round 2. */
    /* Let [abcd k s i] denote the operation
    a = b + ((a + G(b,c,d) + X[k] + T[i]) <<<s).
    Do the following 16 operations. */
    [ABCD    1   5  17]
    [DABC    6   9  18]
    [CDAB   11  14  19]
    [BCDA    0  20  20]
    [ABCD    5   5  21]
    [DABC   10   9  22]
    [CDAB   15  14  23]
    [BCDA    4  20  24]
    [ABCD    9   5  25]
    [DABC   14   9  26]
    [CDAB    3  14  27]
    [BCDA    8  20  28]
    [ABCD   13   5  29]
    [DABC    2   9  30]
    [CDAB    7  14  31]
    [BCDA   12  20  32]

    /* Round 3. */
    /* Let [abcd k s i] denote the operation
    a = b + ((a + H(b,c,d) + X[k] + T[i]) <<<s).
    Do the following 16 operations. */
    [ABCD    5   4  33]
    [DABC    8  11  34]
    [CDAB   11  16  35]
    [BCDA   14  23  36]
    [ABCD    1   4  37]
    [DABC    4  11  38]
    [CDAB    7  16  39]
    [BCDA   10  23  40]
    [ABCD   13   4  41]
    [DABC    0  11  42]
    [CDAB    3  16  43]
    [BCDA    6  23  44]
    [ABCD    9   4  45]
    [DABC   12  11  46]
    [CDAB   15  16  47]
    [BCDA    2  23  48]

    /* Round 4. */
    /* Let [abcd k s i] denote the operation
    a = b + ((a + I(b,c,d) + X[k] + T[i]) <<<s).
    Do the following 16 operations. */
    [ABCD    0   6  49]
    [DABC    7  10  50]
    [CDAB   14  15  51]
    [BCDA    5  21  52]
    [ABCD   12   6  53]
    [DABC    3  10  54]
    [CDAB   10  15  55]
    [BCDA    1  21  56]
    [ABCD    8   6  57]
    [DABC   15  10  58]
    [CDAB    6  15  59]
    [BCDA   13  21  60]
    [ABCD    4   6  61]
    [DABC   11  10  62]
    [CDAB    2  15  63]
    [BCDA    9  21  64]

    /* Then increment each of the four registers by the
    value it had before this block was started. */
    A = A + AA
    B = B + BB
    C = C + CC
    D = D + DD

end /* of loop on q */
```

MD4 - comparaison

- précurseur du MD5
- produit également des condensés de 128 bits
- a 3 rondes de 16 étapes contre 4 dans MD5
- buts de conception :
  - Résistant aux collisions (difficile de trouver des collisions)
  - sécurité directe (aucune dépendance envers des problèmes math.)
  - rapide, simple, compact
  - Favorise les systèmes « little endian » (exple : les PCs)

Force du MD5

- Le condensé MD5 dépend de tous les bits du message $\rightarrow$ avalanche
- Rivest prétend que La sécurité est aussi bonne que possible
- Les attaques connues sont :
  - Berson 92, Den Boer et Bosselaers 93, Dobbertin 96
  - Présence de collision, attaque sur une ronde
  - Pas d'attaque concrète mais de plus en plus de faiblesses trouvée
- Conclusion : MD5 sera bientôt vulnérable

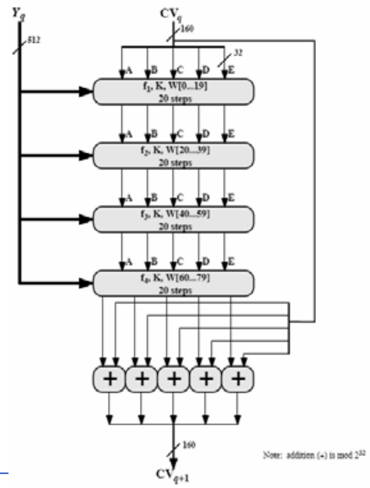
SHA-1 (Secure hash algorithm)

Le Secure Hash Algorithm (SHA-1)

- SHA a été conçu par NIST et NSA en 1993, révisé 1995
- Standard US pour usage avec le schéma de signature DSA
  - norme : FIPS 180-1 (1995), Internet RFC3174
  - l'algorithme est SHA, la norme est SHS (secure hash standard)
- produit des valeurs condensées de 160 bits
- actuellement l'algorithme généralement préféré pour le hachage
- basé sur le design du MD4 avec quelques différences

SHA - Vue d'ensemble

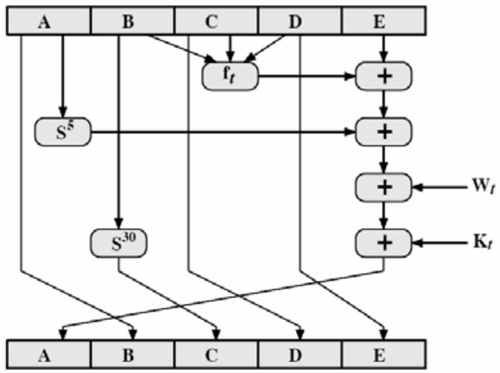
1. Complétion :
   - le message condensé doit être de longueur 448 mod 512
2. Ajout de la longueur
   - une valeur codée sur 64 bits
3. Initialisation
   - Initialiser 5 buffers de 32 bits (= 160 bits) – A,B,C,D,E
4. Calcul itératif :
   - 4 rondes de 20 itérations chacune
   - Les rondes ont une structure similaire mais utilisent des fonction primitives différentes (f_1,f_2,f_3,f_4)
   - Utilisation de constante additives K_t ($0 \leq t \leq 79$)
   - Le résultat est utilisé pour initialiser les buffers du bloc suivant.
5. Le condensé final constitue le condensé attendu



SHA - Fonction de compression

- chaque ronde a 20 étapes qui manipulent ainsi les 5 registres :
- (A,B,C,D,E) $\leftarrow$ (E + f(t,B,C,D) + (A<<5) + W_t + K_t), A, (B<<30), C, D)
- Où :
  - A,B,C,D,E se rapportent aux 5 registres
  - t est le numéro de l'étape
  - f(t, B, C, D) est une fonction primitive non-linéaire de la ronde pour l'étape t
  - W_t est dérivé du bloc de message(32 bits)
  - K_t est une valeur additive

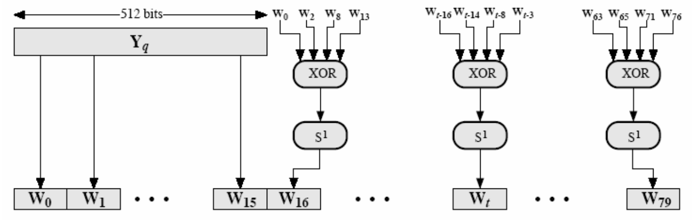
SHA - Fonction de compression



SHA - Fonction de compression

- Fonctions primitives
  - Travaille sur 32 bits et fournit un résultat sur 32 bits
  - $0 \leq t \leq 19$ $f_1=f(t,B,C,D)$ $(B \wedge C) \vee (\neg B \wedge D)$
  - $20 \leq t \leq 39$ $f_2=f(t,B,C,D)$ $B \oplus C \oplus D$
  - $40 \leq t \leq 59$ $f_3=f(t,B,C,D)$ $(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
  - $60 \leq t \leq 79$ $f_4=f(t,B,C,D)$ $B \oplus C \oplus D$
- W_t
  - $0 \leq t \leq 15$: les 16 premières valeurs du bloc
  - $16 \leq t$: $W_t = ((W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}) <<< 1)$

SHA - Calcul de W_t



SHA - Valeurs initiales

- Registre :
  - A = 67452301
  - B = efcdab89
  - C = 98badcfe
  - D = 10325476
  - E = c3d2e1f0
- K_t
  - $0 \leq t \leq 19$ – K_t=5a827999
  - $20 \leq t \leq 39$ – K_t=6ed9eba1
  - $40 \leq t \leq 59$ – K_t=8f1bbcdc
  - $60 \leq t \leq 79$ – K_t=ca62c1d6

SHA-1 vs MD5

- l'attaque par force brute est plus difficile (160 contre 128 bits pour MD5)
- non vulnérable à toutes les attaques connues (comparées à MD4/5)
- un peu plus lent que MD5 (80 contre 64 étapes)
- conçu comme simple et compact
- optimisé pour CPU's big-endian (contre MD5 qui est optimisé pour CPU's little-endian)

SHA-1 vs MD5

	MD5	SHA-1
Digest length	128 bits	160 bits
Basic unit of processing	512 bits	512 bits
Number of steps	64 (4 rounds of 16)	80 (4 rounds of 20)
Maximum message size	∞	$2^{64} - 1$ bits
Primitive logical functions	4	4
Additive constants used	64	4
Endianness	Little-endian	Big-endian

Revised Secure Hash Standard

- Le NIST a publié une révision : FIPS 180-2
- Ajout de 3 algorithmes additionnels de hachage
- Sha-256, Sha-384, Sha-512
- Conçu pour la compatibilité avec la sécurité accrue a fourni par le chiffrement AES
- La structure et le détail est semblable à Sha-1
- Par conséquent l'analyse devrait être semblable

Comparaison des propriétés de SHA

	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size	160	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	80	80	80
Security	80	128	192	256

1. Toutes les tailles sont mesurées en bits.
2. La sécurité se rapporte au fait qu'une attaque d'anniversaire sur un condensé de message de taille *n* produit une collision avec un facteur d'approximativement $2^{n/2}$.

Algorithmes pour les MACs

Fonction de hachage pour le calcul de MAC

- Désire de créer des MACs à partir de fonction de hachage plutot que de chiffrement par bloc
  - Fonctions de hachage généralement plus rapide
  - Non limitées par les contrôles à l'exportation
- Hachage incluant une clef avec le message
- Proposition originale :
  - HashSur = Hash(clé|Message)
  - Quelques faiblesses ont été trouvées dans ce schéma
- A mené au développement de HMAC

Cahier des charges

- Utiliser , sans modifications, des fonctions de hachage existantes
- Permettre un remplacement aisé de la fonction de hachage au cas où des fonctions plus rapides ou plus sures seraient trouvées ou exigées
- Préserver les performances initiales de la fonction de hachage
- Employer et manipuler les clefs de manière simple

Correspondance avec ce CCH

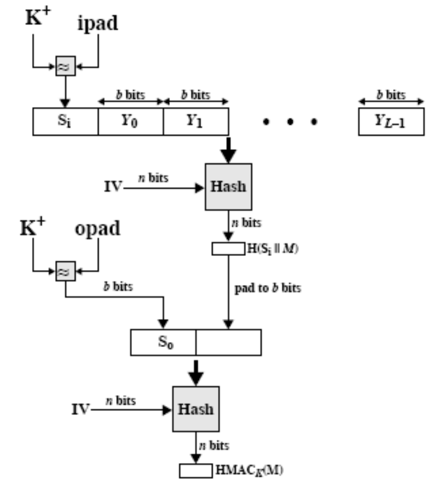
- HMAC traites les fonctions de hachage comme des ‘black box’.
  - La fonction de hachage devient un module
  - Pas de modification à apporter à la fonction

Principe de l’algorithme

- Soient :
  - H : la fonction de hachage
  - IV : un vecteur d’initialisation
  - M : le message (+ padding)
  - Y_i : i^{eme} bloc de M
  - L : le nombre de blocs de M
  - b : le nombre de bits dans un bloc
  - n : la longueur du condensé produit
  - K : la clé
  - K^+ : la clé ‘paddée’ à une longueur b
  - ipad : 0x36 (répété b/8 fois)- opad : 0x5C (répété b/8 fois)

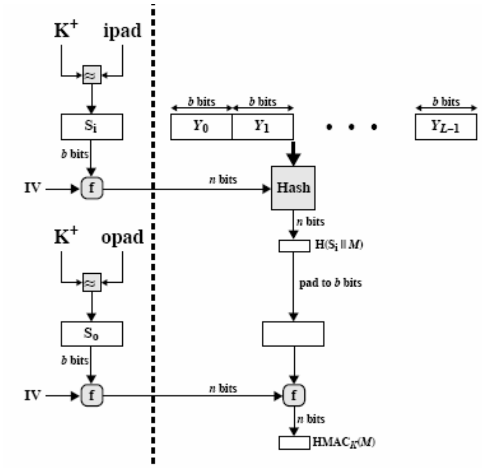
Principe de l'algorithme

- HMAC :
  - $HMAC_k\ (M) = H[(K^+ \oplus opad)\ ||\ H[(K^+ \oplus ipad)\ ||\ M]]$
- En bref :
  1. Padding de la clé :
     Ajout de 0 à gauche de K pour créer un flux de b bits (K^+)
  2. XOR de K^+ et ipad pour produire S_i (longueur b bits)
  3. Ajout de M à S_i
  4. Application de H à ce flux
  5. XOR de K^+ et opad pour produire S_o (longueur b bits)
  6. Ajoute du hash (etape 4) avec S_o
  7. Application de H au résultat de l'étape 6



Optimisations

- Possibilité de préprocessing :
  - f(IV, ($K^+ \oplus$ ipad))
  - f(IV, ($K^+ \oplus$ opad))
  - f
    - fonction de compression utilisée pour hacher
    - Prend en argument une variable chaînée de n bits et un bloc de b bits
    - Produit une variable chaînée de n bits
  - Ces quantités ne sont calculée qu'à l'initialisation et quand la clé change
  - Ces quantités se substituent à IV dans la fonction de hachage



Sécurité de HMAC

- La sécurité de HMAC est directement liée à la fonction de hachage sous-jacente
- Attaquer HMAC exige
  - Soit une attaque de force brutale sur la clef utilisée (2^n essais)
  - Soit une attaque d'anniversaire ($2^{n/2}$mais puisque utilisé avec une clé, on devrait observer un nombre très grand de messages)
- Choisir la fonction de hachage à utiliser en se basant sur des contraintes de sécurité et de vitesse

Signatures digitales

Signatures Digitale

- On a regardé l'authentification de message
  - mais n'aborde pas les questions de manque de confiance
- Les signatures numériques permettent de :
  - vérifier l'auteur, la date et l'heure de la signature
  - authentifier le contenu de message
  - être vérifié par des tiers pour résoudre des conflits
- Par conséquent, elles incluent la fonction d'authentification avec des possibilités additionnelles

Propriétés des signatures digitales

- Doit dépendre du message signé
- Doit employer une information unique propre à l'expéditeur
  - pour empêcher la contrefaçon et le démenti
- Doit être relativement facile produire
- Doit être relativement facile reconnaître et vérifier
- Doit être mathématiquement infaisable à forger
  - avec de nouveau message pour une signature numérique existante
  - avec une signature numérique frauduleuse pour un message donné
- Doit être pratique à stocker

Signatures Digitales Directes

- Implique uniquement l'expéditeur et le récepteur
- Suppose que le récepteur dispose de la clé publique de l'expéditeur
- Signature numérique faite par l'expéditeur signant le message entier ou le condensé avec sa clé privée
- peut être chiffrée en utilisant la clé publique des récepteurs
- Il est important de signer d'abord et de chiffrer ensuite le message et la signature
- La sécurité dépend de la clé privée de l'expéditeur

Signatures digitales arbitrées

- comporte l'utilisation d'un arbitre A
  - Il valide n'importe quel message signé
  - Le date et l'envoie au destinataire
- exige un niveau approprié de confiance en l'arbitre
- peut être mis en application avec des algorithmes symétriques ou à clés publiques
- l'arbitre peut ou ne peut pas voir le message

Techniques

(a) Conventional Encryption, Arbiter Sees Message
(1) $X \rightarrow A$: $M \parallel E_{K_{xa}}[ID_X \parallel H(M)]$
(2) $A \rightarrow Y$: $E_{K_{ay}}[ID_X \parallel M \parallel E_{K_{xa}}[ID_X \parallel H(M)] \parallel T]$
(b) Conventional Encryption, Arbiter Does Not See Message
(1) $X \rightarrow A$: $ID_X \parallel E_{K_{xy}}[M] \parallel E_{K_{xa}}[ID_X \parallel H(E_{K_{xy}}[M])]$
(2) $A \rightarrow Y$: $E_{K_{ay}}[ID_X \parallel E_{K_{xy}}[M] \parallel E_{K_{xa}}[ID_X \parallel H(E_{K_{xy}}[M])] \parallel T]$
(c) Public-Key Encryption, Arbiter Does Not See Message
(1) $X \rightarrow A$: $ID_X \parallel E_{KR_x}[ID_X \parallel E_{KU_y}(E_{KR_x}[M])]$
(2) $A \rightarrow Y$: $E_{KR_a}[ID_X \parallel E_{KU_y}[E_{KR_x}[M]] \parallel T]$

Notation:
- X = sender
- Y = recipient
- A = Arbiter
- M = message
- T = timestamp

Algorithmes de signatures

El Gamal : principe

- Clé publique :
  - p premier, g < p
  - $y = g^x \bmod p \rightarrow (y,g,p)$
- Clé privée
  - x < p
- Signature
  - K tel que (k,p-1)=1
  - $a = g^k \bmod p$
  - b tel que M = (xa + kb) mod (p-1)
- Vérification :
  - La signature est valide si $y^a a^b \bmod p = g^M \bmod p$

Un nouveau k à chaque signature ou chiffrement

Si plusieurs fois le même k : retrouver x aisément

El Gamal : exemple

- p = 11, g = 2, x = 8
- $y = g^x \bmod p = 2^8 \bmod 11 = 3$ PK = (3,2,11)
- Authentification : M = 5 , k=9 (9,10)=1 (ok)
  - $a = g^k \bmod p = 2^9 \bmod 11 = 6$
  - Par Euclide :
    - M = (ax + bk) mod (p-1)
    - 5 = (8*6 + 9*b) mod 10
    - b = 3   → signature = (a,b) = (6,3)
- Vérification :
  - $y^a a^b \bmod p = g^M \bmod p$
  - $3^6 6^3 \bmod 11 = 2^5 \bmod 11$

Digital Signature Standard (DSS)

- Schéma de signature FIPS-186 approuvé par le govt US
- Emploie l'algorithme de hachage SHA
- Conçu par le NIST et la NSA dans le début des années 90
- Le DSS est la norme, DSA est l'algorithme
- Variante d'ElGamal et de Schnorr
- Crée une signature de 320 bits, mais avec une sécurité équivalentes à un chiffrement 512-1024 bits
- La sécurité dépend de la difficulté de calculer des logarithmes discrets

DSA – description de l'algorithme

- p : p = 2^L : nbre premier de L bits (512<L<1024)
- q : facteur premier de p-1, 160 bits
- g : g = $h^{(p-1)}/q$ mod p
  - h : h < p-1 et $h^{(p-1)}/q > 1$
- x : x < q
- y : y = g^x mod p
- H(x) : fonction de hachage à sens unique (SHA)
- Pk : y et Sk = x

p doit également être un multiple de 64

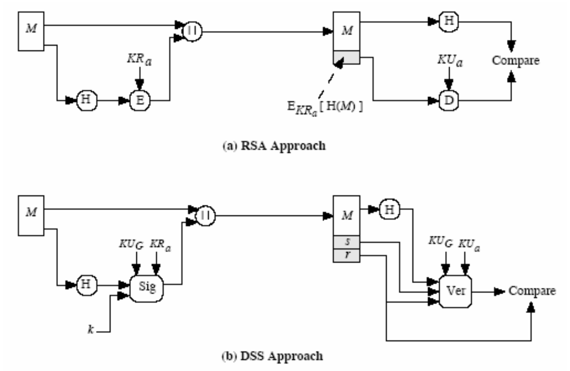
DSS - signature

- Alice engendre k tq k < q
  - k doit être aléatoire, détruit après utilisation, jamais réutilisé
- Alice engendre
  - $r = (g^k \bmod p) \bmod q$
  - $s = (k^{-1}(H(M) + xr)) \bmod q$
- r et s forment la signature
- Alice envoie (r, s) à Bob

DSS - vérification

- Bob vérifie la signature en calculant
  - $w = s^{-1} \bmod q$
  - $u_1 = (H(M) * w) \bmod q$
  - $_{u2} = rw \bmod q$
  - $v = ((g^{u1} * g^{u2}) \bmod p) \bmod q$
- Si v = r, alors la signature est vérifiée

Comparaison des approches



Questions

- Expliquer
  - MD5 et/ou SHA + comparaisons
  - HMAC
  - Signatures digitales : principes
  - Signatures ElGamal et/ou DSS

Références

- http://www.secure-hash-algorithm-md5-sha-1.co.uk/
- http://www.bibmath.net/crypto/moderne/sigelec.php3
- http://www.securiteinfo.com/crypto/hash.shtml
- http://www.ssh.fi/support/cryptography/introduction/signatures.html
- [schneier] – ch 2, 18, 20
- [stallings] – ch 12,13