Sécurité des systèmes d'information

(initiation à la cryptographie)

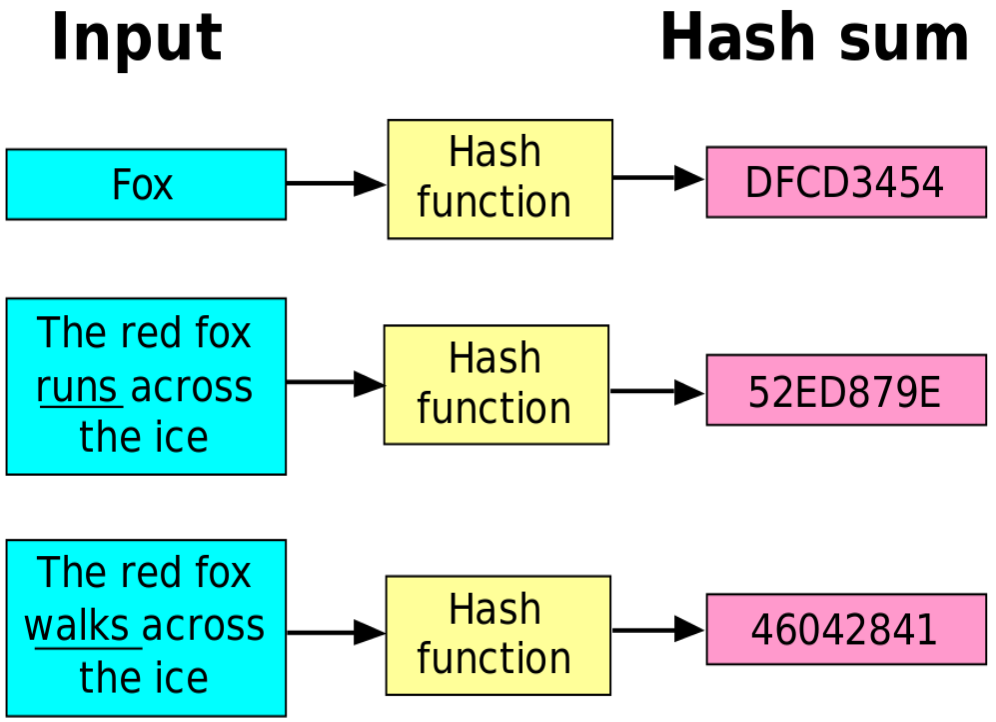
Partie 4: fonctions d'hachage et signature numérique

université d'Alger 1 -
Benyoucef Benkhedda

Fonctions d’hachage

Fonction d’hachage

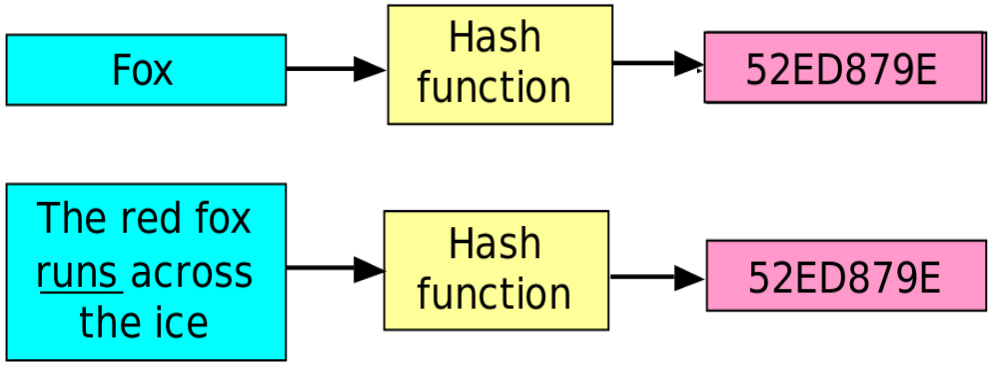
- C’est une fonction facilement calculable permettant de transformer une message claire m de taille quelconque à un message y de taille fixe n => y = H(m)
- Les fonctions d’hachage sont des fonctions a sens unique mais **sans trapdoor** c’est à dire qu’il n’y a aucun moyen de revenir au texte clair à partir du code haché



Fonction d’hachage

Problème de collision:

- On appelle problème de collision lorsqu’il existe deux textes m et m’ qui possèdent le même code haché y => y = H(m) = H(m’) tel que m ≠ m’



- Le problème de collision est souvent connu en mathématique par le **paradoxe d’anniversaire**: « Dans une assemblée de 23 personnes, la probabilité qu'au-moins 2 d'entre-elles aient leur anniversaire le même jour est égale a 0.5 (en ne tenant pas compte de l'année de naissance.) »

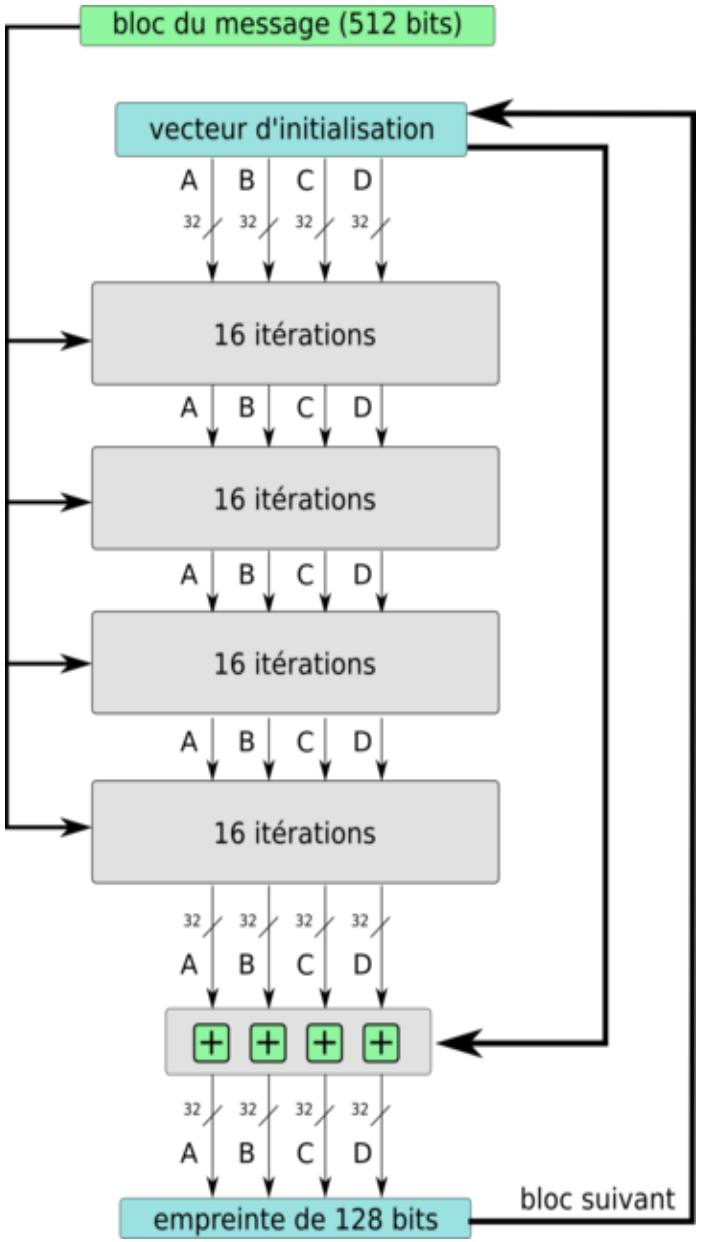
Quelques algorithmes d'hachage

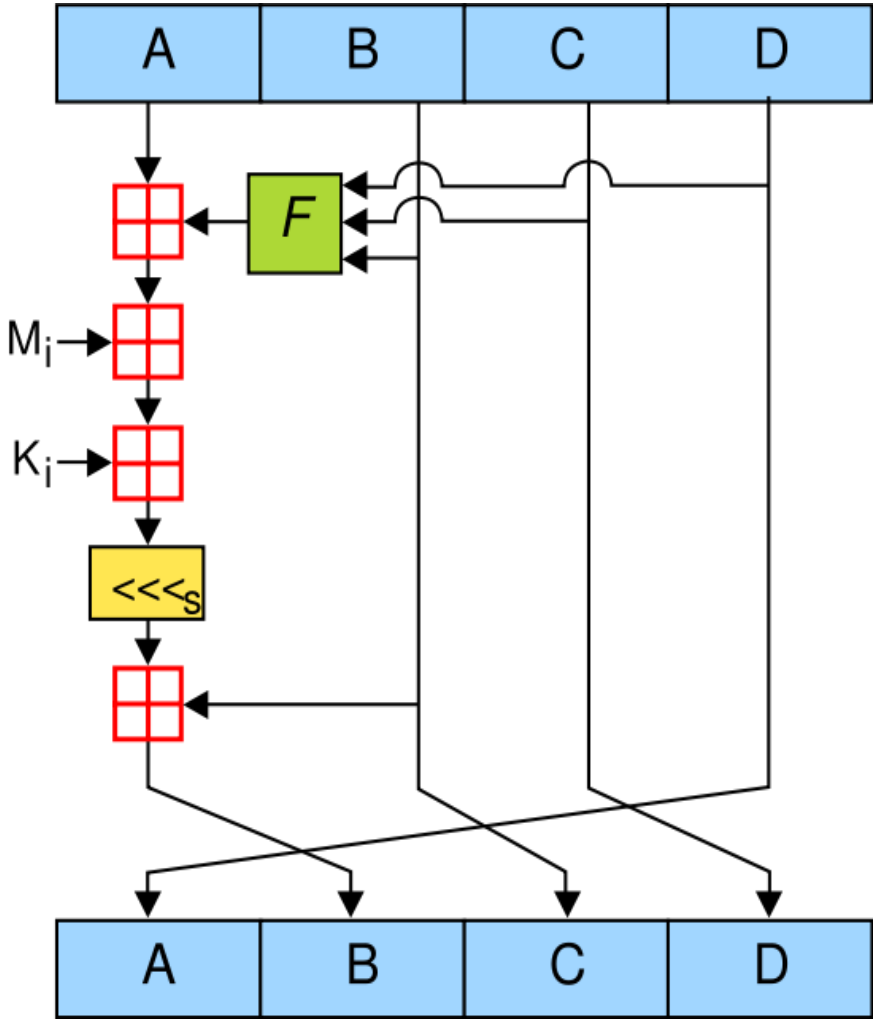
Algorithme MD5:

- Transforme une chaine de taille variable a une empreinte de 128 bits.
- N'est plus considéré comme sûr depuis 2004
- Divise le message en blocs de 512 bits, utilise un état de 128 bits composé de 4 mots de 32 bits: **A**= 0x67452301, **B**= 0xEFCDAB89, **C**= 0x98BADCFE et **D**= 0x10325476, et 64 constantes K_i de 32 bits
- Les opérations appliquées sur chacun des blocs se décomposent en 4 rounds dont chacun des rounds se décompose en 16 opérations pour un sous-bloc de 32 bits
- Les 16 opérations sont basées sur: une fonction non-linéaire F qui varie selon le round et une rotation vers la gauche avec une addition:

$$F = (B \text{ AND } C) \text{ OR } (\overline{B} \text{ AND } D)$$
$$F = (D \text{ AND } B) \text{ OR } (\overline{D} \text{ AND } C)$$
$$F = B \oplus C \oplus D$$
$$F = C \oplus (B \text{ OR } \overline{D}))$$

Quelques algorithmes d'hachage



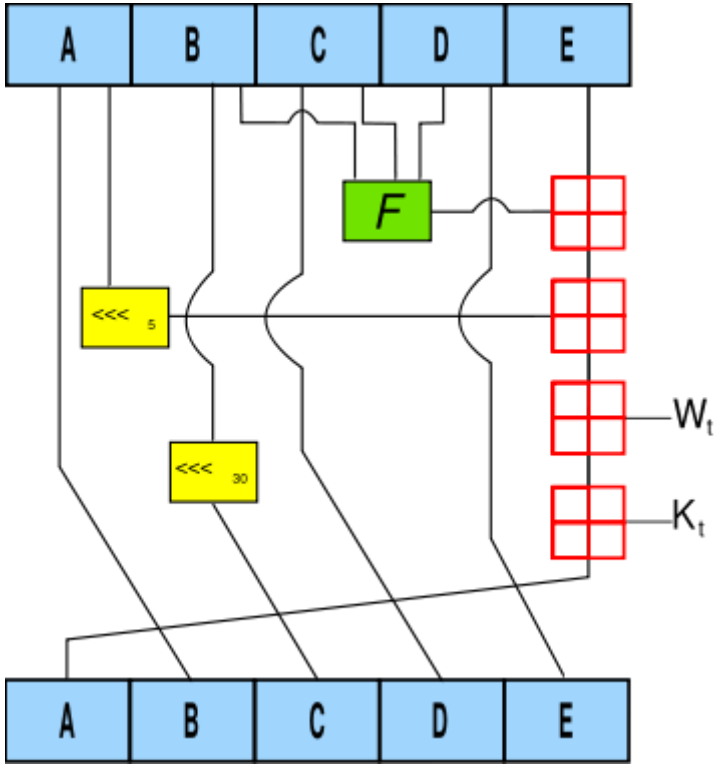


Quelques algorithmes d'hachage

Algorithme SHA-1:

- Transforme une chaine de taille variable a une empreinte de 160 bits.
- Travaille sur des blocs de 512 bits
  - Chacun des blocs est divisé en 16 mots de 32 bits
  - Etendus en 80 nouveaux blocs
  - 4 constantes fixé K_i
  - 80 rounds sur 5 mots de 32 bits: A, B, C, D, et E
  - Une fonction non-linéaire F:

$$F = (B \text{ AND } C) \text{ OR } (\overline{B} \text{ AND } D)$$
$$F = B \oplus C \oplus D$$
$$F = (B \text{ AND } C) \oplus (B \text{ AND } D) \oplus (C \text{ AND } D)$$
$$F = B \oplus C \oplus D$$



Quelques algorithmes d'hachage

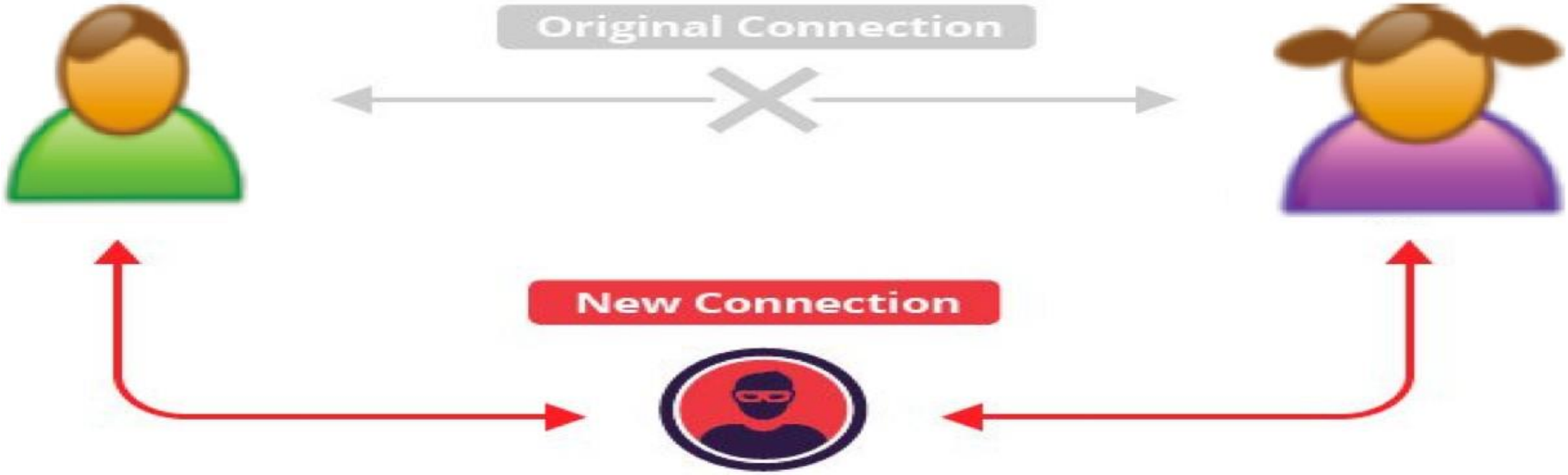
Bilan d'algorithmes :

Fonction	Empreinte	Compl. requise	Rés. aux coll.	Comp. attaque
MD5	128 bits	$\mathcal{O}(2^{64})$	Cassé [1]	$\mathcal{O}(2^{30})$
SHA-1	160 bits	$\mathcal{O}(2^{80})$	Cassé [2]	$\mathcal{O}(2^{63})$
HAVAL	256 bits	$\mathcal{O}(2^{128})$	Cassé [3]	$\mathcal{O}(2^{10})$
SHA-256	256 bits	$\mathcal{O}(2^{128})$	Sûr	
Whirlpool	512 bits	$\mathcal{O}(2^{256})$	Sûr	

Signature numérique

Rappel

- Chiffrement asymétrique utilise deux clés, une pour chiffrement qui reste **publique** et l'autre pour le déchiffrement qui doit être conservée **secrète**
- Lourd en terme de calcule (besoin de calculer plusieurs clés)
- Problèmes de non-répudiation des clés publiques



- Est-ce qu'on peut faire confiance aux clés publiques

Signature numérique

- Est un mécanisme de protection de l'intégrité et la non-répudiation des messages

- Ce mécanisme utilise:
  - Le hachage du message
  - La cryptographie asymétrique

- Conditions : la signature doit être authentique, infalsifiable, non réutilisable, inaltérable et irrévocable

Signature numérique

Principe général

- Calculer le hachage de message m => h = H(m)
- Déchiffrer h en utilisant sa clé privé Sk (S(m) = Dec_{Sk}(h)

(m, S(m))

- Utiliser la clé publique de Bob pour chiffrer la Signature ENC_{Pk}(S(m)) = h
- Hacher le message m => H(m) = h'
- Comparer h et h' s'il sont égaux alors le message est bien reçu de la part de Bob

Quelques algorithmes de signature numérique

DSA (Digital Signature Algorithm)

Génération de clés:

- Choisir un entier **p** : nombre premier p de L-bit avec 512 ≤ L ≤ 1024 et L est divisible par 64. et un entier **q**: nombre premier de 160 bits tel que **$p - 1 = q * z$** avec z un entier

  Choisir aléatoirement un entier **h** tel que **$g = h^z \bmod p > 1$**

  Clé secrète : un entier aléatoire **x** tel que 0 < x < q

  Clé publique : **(p, q, g, y)** tel que **$y = g^x \bmod p$**

Signature:

- Choisir un nombre aléatoire **k**, 1 < k < q. Calculer **$r = (g^k \bmod p) \bmod q$**
- Calculer **$s = (H(m) + r * x) * k^{-1} \bmod q$** (si r=0 ou s=0 choisir un autre k).

•La signature du message : est **(r, s)**

Vérification:

- Vérifier si 0<r <q ou 0<s <q. Calculer **$u1 = H(m) * s^{-1} \bmod q$** et **$u2 = r * s^{-1} \bmod q$**
- Calculer **$v = [g^{u1} * y^{u2} \bmod p] \bmod q$**.
- La signature est valide si **v = r**

Quelques algorithmes de signature numérique

DSS (Digital Signature Standard)

Paramètres:

- Choisir un entier premier **p** et **g** générateur du groupe Z_p^*.

Génération de clés:

- Clé secrète : **a** tel que $1 < a < p-1$
- Clé publique : **(p, g, A)** tel que $A = g^a \bmod p$

Signature:

- Choisir un nombre aléatoire **k** | $0 < k < p-1$;
- Calculer $r = g^k \bmod p$, Calculer $s = (H(m) - a * r) * k^{-1} \bmod p-1$ (si s=0 choisir un autre k).

- La signature du message est : **(r, s)**.

Vérification:

- Vérifier si $0<r<p$ et $0<s<p-1$
- La signature est valide si $g^{H(m)} = A^r * r^s \bmod p$

Signature numérique

Utilisation des signatures numériques

- Dans les infrastructuresà clé publique (PKI)
  - Génération des autorités de certifications qui assure que **les bonnes clés publiques valides sont utilisées dans le chiffrement**

- Vérification de l'intégrité des messages dans les communications

- Assurance de la non-répudiation et l'identité de l'émetteur des messages

Hachage en pratique

Openssl

- Open source
- Préinstallé dans toute les distributionsde Linux
- Simple et pratique
- Contient aussi une bibliothèque en c « openssl.h »