Fonctions de hachage

Fouque Pierre-Alain

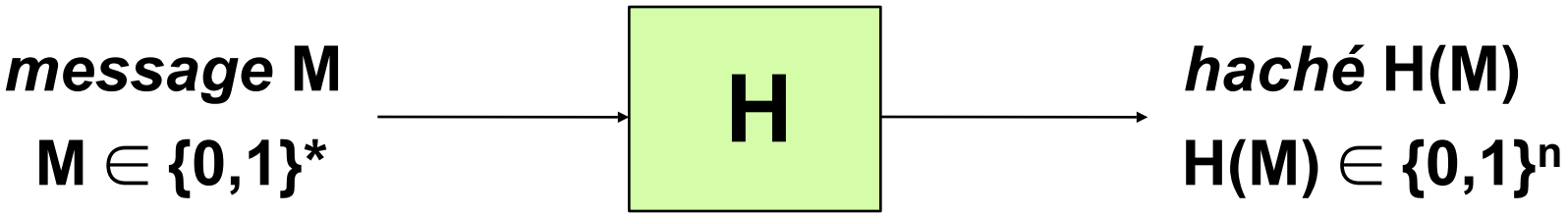
Equipe de Cryptographie

Ecole normale supérieure

Introduction

- Problématiques en crypto
  - Convertir des entrées de taille variable vers une taille fixe (ex : signature, intégrité)
  - Calcul d'empreintes uniques
  - Fonctions « à sens unique »
- Utilisation de fonctions de hachage:
  - stockage mots de passe,
  - codes d'authentification de messages (MAC),
  - signatures, chiffrement, "fonction aléatoire"...

Définition



Une fonction de hachage H calcule un haché de n bits à partir d'un message arbitraire M

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

Exemple

Cryptographic hash functions that compute a fixed size message digest from arbitrary size messages are widely used for many purposes in cryptography, including digital signatures. NIST was recently informed that researchers had discovered a way to "break" the current Federal Information Processing Standard SHA-1 algorithm, which has been in effect since 1994. The researchers have not yet published their complete results, so NIST has not confirmed these findings. However, the researchers are a reputable research team with expertise in this area. Previously, a brute force attack would expect to find a collision in 2^{80} hash operations.

SHA-1 →

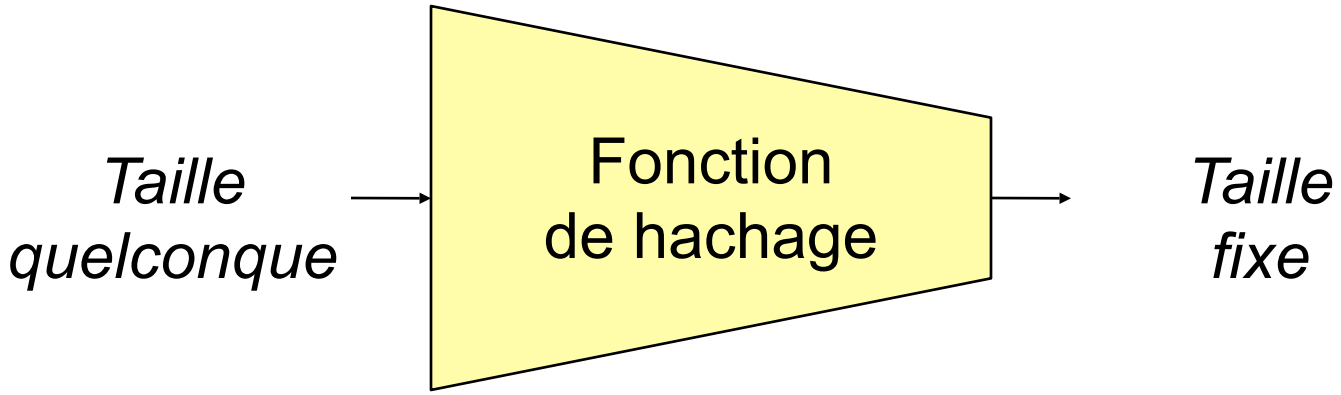
Valeur hachée
de n=160 bits :

A51F 07BB 62EC 44A3 F118

Critères de conception

La fonction de hachage doit

- réduire la taille des entrées à n bits (compression)
- être facile à calculer



Fonctions classiques

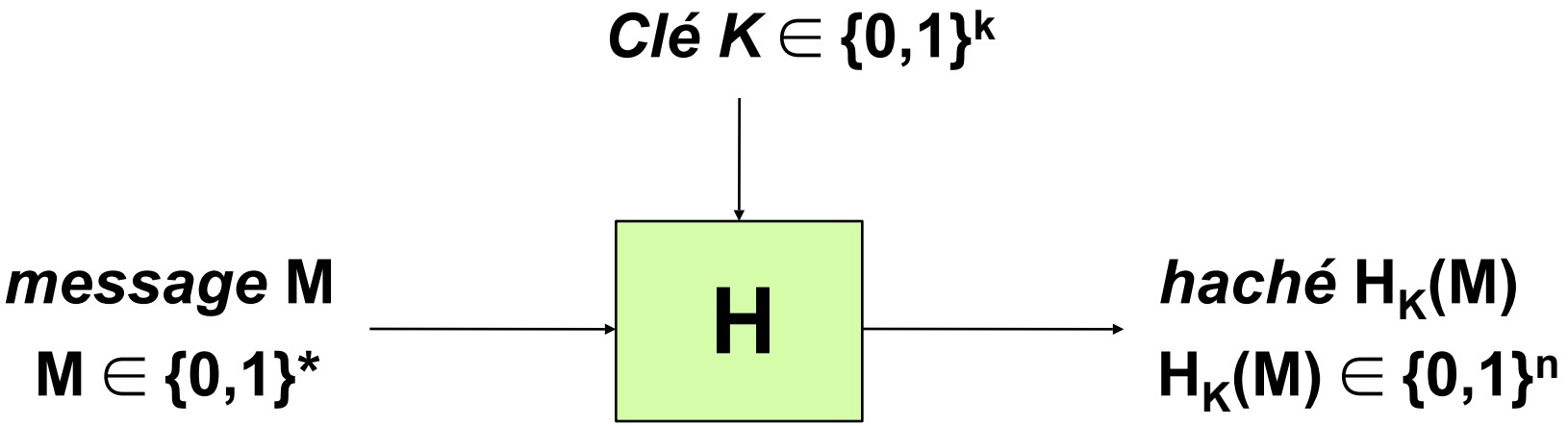
Nom	n	Auteur
MD2	128	RSA
MD4	128	RSA
MD5	128	RSA
SHA-1	160	NIST
SHA-256	256	NIST
RIPEMD-*	128,160,…	
HAVAL-*	128,160,…	

Remarque

- Une fonction de hachage n'est pas spécifiquement une primitive à clé secrète (il n'y a pas de clé)
- Toutefois, les méthodes de construction sont souvent similaires au block ciphers

Présence d'une clé

Toutefois, dans certaines situations, on peut être amené à utiliser des fonctions de hachage avec clé :



Classification

- Fonctions de hachage sans clé
- Fonctions de hachage avec clé
  - « Famille » de fonctions de hachage (plusieurs instances sont disponibles, possibilité de randomiser les hachés)
  - Vérification d'intégrité (MAC)

Classification

- Fonctions de hachage sans clé
- Fonctions de hachage avec clé
  - « Famille » de fonctions de hachage (plusieurs instances sont disponibles, possibilité de randomiser les hachés)
  - Vérification d'intégrité (MAC)

Propriété attendue

- Intuition = La fonction de hachage doit se comporter de façon aléatoire
- Qu'est-ce que cela signifie en pratique ? (Absence de relation entre haché et message qui ferait qu'on puisse prédire la valeur du hacé sans la calculer explicitement)
- En général, on spécifie plus précisément certaines propriétés de sécurité (ex : collisions)
  - formalisation du problème des collisions ?

Utilisation

1. Intégrité de fichier
2. Stockage de mots de passe
3. Intégrité de communications
4. Signature numérique

1 - Intégrité de fichier

Idée : on veut vérifier qu'un fichier n'a pas été modifié
→ On calcule son empreinte

```
// Fichier code.c

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv)
{
  if (argc <2)
    {
      …
    }
}
```

SHA-1 →

Valeur hachée
de 160 bits :

SHA-1 (`code.c`) =
A51F 07BB 62EC 44A3 F118

Sécurité

- Limitation :
  - Possibilité de re-calculer l'empreinte
  - L'empreinte doit être transmise de façon sécurisée (intégrité)
- Résistance mathématique
  - Difficulté de fabriquer un 2ème message donnant le même haché que `code.c`

2 - Mots de passe

- Au lieu de stocker un mot de passe « en clair » sur une machine, on stocke son haché (ex : systèmes d'exploitation)

$$h = H(\textit{password})$$

- Pour s'authentifier, un utilisateur envoie h
- Cela rend la base de données moins sensible

Sécurité

- Limitation :
  - On peut toujours énumérer les mots de passe, s'ils ont peu d'entropie (attaque par dictionnaire)
- Résistance mathématique
  - A partir du haché h, difficulté de trouver un antécédent x tel que

    $$h = H(x) = H(\textit{password})$$

  - Est-il nécessaire que x == *password* ?

3 – Communications

- Garantir qu'un message n'a pas été modifié en cours de transmission (intégrité)
  - Fonction de hachage avec clé
  - L'empreinte est aussi appelée MAC
- Solutions :
  - MAC basé sur un block cipher (ex : CBC-MAC)
  - MAC basé sur une fonction de hachage sans clé

Sécurité

- Idée naturelle

  MAC_K (*message*) = H(K || *message*)

  → Problèmes liés à la structure itérative de H

- Méthode standard : HMAC

  $HMAC_{K1,K2}$ (*message*)

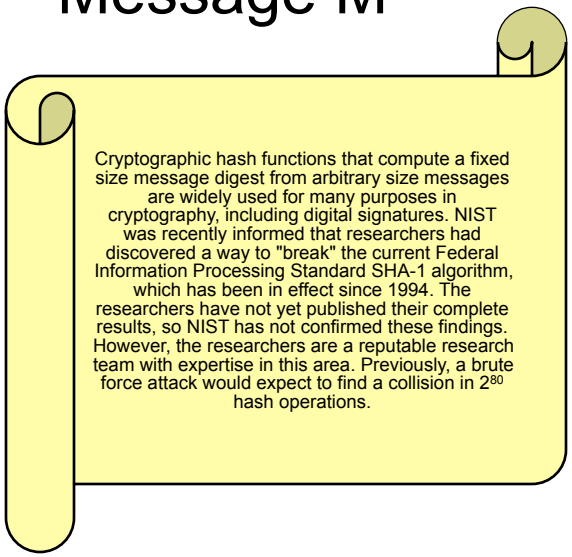
  = H(K1 || H (K2 || *message*))

4 – Signatures

- Pour des raisons d'efficacité et de sécurité, on ne signe jamais directement un message M mais toujours son haché H(M)
  - Enlever certaines propriétés mathématiques des primitives de signature (ex : RSA)
  - On ne signe que des données de taille fixée (et petite)

Signatures

Message M



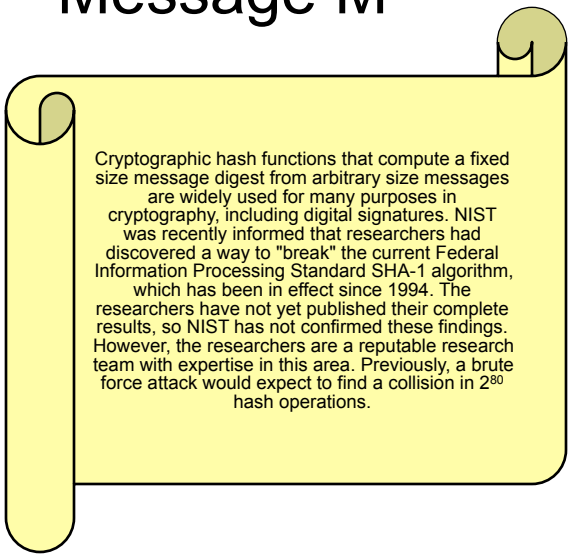
Clé de signature K_s

Algorithme de signature $Sign_{KS}(.)$

Fonction de hachage H(.)

Signatures

Message M



Clé de signature K_s

Algorithme de signature $Sign_{KS}(.)$

Fonction de hachage H(.)

Signature du message M :

$$\sigma = Sign_{KS}(H(M))$$

Sécurité

- On ne doit pas pouvoir substituer un message M' ayant le même haché à la place de M
- Dans le cas contraire,

$\sigma = Sign_{K_S}(H(M))$

$\sigma' = Sign_{K_S}(H(M'))$

} Donc $\sigma = \sigma'$

Exemple

Message M

Cryptographic hash functions that compute a fixed size message digest from arbitrary size messages are widely used for many purposes in cryptography, including digital signatures. NIST was recently informed that researchers had discovered a way to "break" the current Federal Information Processing Standard SHA-1 algorithm, which has been in effect since 1994. The researchers have not yet published their complete results, so NIST has not confirmed these findings. However, the researchers are a reputable research team with expertise in this area. Previously, a brute force attack would expect to find a collision in 2^{80} hash operations.

→ H(M) = 7b…08 → Signature σ

Message M'

Cryptographic hash functions that compute a fixed size message digest from arbitrary size messages are widely used for many purposes in cryptography, including digital signatures. NIST was recently informed that researchers had discovered a way to "break" the current Federal Information Processing Standard SHA-**0** algorithm, which **is no longer** in effect since **1995**. The researchers have not yet published their complete results, so NIST has not confirmed these findings. However, the researchers are a reputable research team with expertise in this area. Previously, a brute force attack would expect to find a collision in 2^{80} hash operations.

→ H(M') = 7b…08 → Signature $\sigma' = \sigma$

Sécurité

Les notions suivantes sont apparues :

- Résistance aux collisions

  trouver M_1 et M_2 tel que $H(M_1) = H(M_2)$

- Résistance aux secondes préimages

  avec M_1 donné, trouver M_2 tel que $H(M_1) = H(M_2)$

- Résistance aux préimages

  avec x donné, trouver M tel que $H(M) = x$

Attaques « génériques »

- Résistance aux collisions

  une attaque naïve coûte $2^{n/2}$

- Résistance aux secondes préimages

  une attaque naïve coûte $2^{n/2} < ? < 2^{n}$

- Résistance aux préimages

  une attaque naïve coûte 2^{n}

Principales notions

- Suivant le contexte, on a besoin d'une notion de sécurité plus ou moins forte

  exemple : Fabriquer un faux certificat à partir d'un certificat valide → Attaque en Seconde Préimage

- Notion la plus forte = Préimages
- Notion la plus faible = Collisions

Attaques en collision

- Choisir $N = 2^{n/2}$ messages quelconques

$$x_1, \ldots, x_N$$

- Calculer les hachés correspondants

$$y_i = H(x_i)$$

- Chercher (a,b) tel que

$$y_a = y_b$$

- Paradoxe des anniversaires

Problème

- Pour trouver une collision, il faut :
  - Stocker tous les y_i dans un tableau
  - Trier ce tableau
- Peu efficace (mémoire de $2^{n/2}$)
- Difficile de paralléliser ce calcul
- Il existe des solutions (méthode ρ de Pollard)
- Dans tous les cas, la complexité en temps est $2^{n/2}$

Attaques en seconde préimage

- Entrée : message M_1 et son haché $x = H(M_1)$

- Calculer $h_i = H(m_i)$ pour des messages aléatoires
- $h_i = x$ avec probabilité 2^{-n}

- Donc, après 2^n messages, on s'attend à trouver une seconde préimage m_i telle que

$$H(m_i) = H(M_1)$$

- Certains compromis sont possibles (prendre M_1 très long)

Attaques en préimage

- Entrée : haché h

- Calculer $h_i = H(m_i)$ pour des messages aléatoires
- $h_i = x$ avec probabilité 2^{-n}

- Donc, après 2^n messages, on s'attend à trouver une préimage m_i telle que

$$h = H(m_i)$$

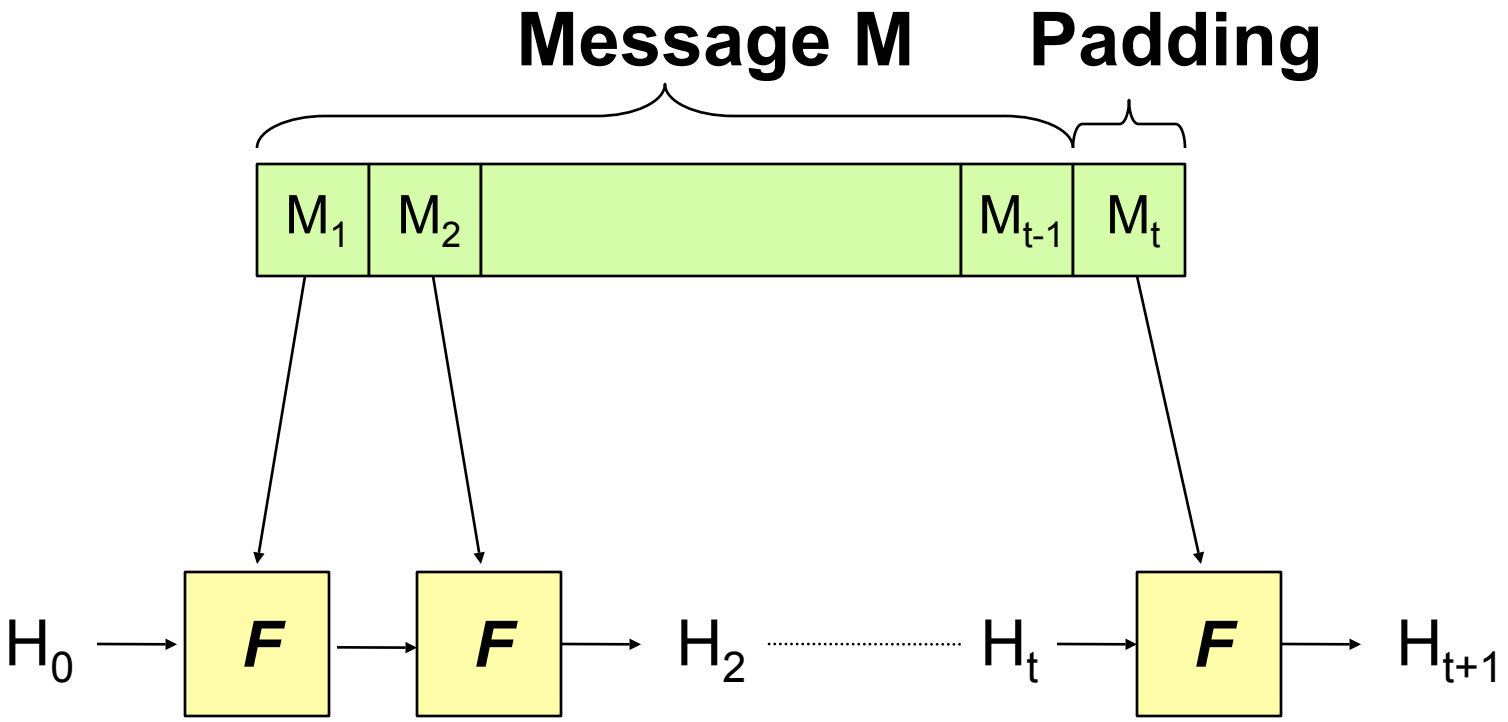
Fonctions à sens unique

- Existence de « fonctions à sens unique » ?
  - « faciles » à calculer
  - « difficiles » à inverse
- Lié au problème de savoir si P ≠ NP
- En pratique, on conjecture que construire une telle fonction est possible

Construction

- Comment construire une fonction de hachage qui traite une entrée de taille arbitraire ?
  - On coupe l'entrée en blocs de taille fixée
  - On utilise une primitive appelée fonction de compression
  - On utilise un mode opératoire

Fonctions itératives



Les H_i sont appelés hachés intermédiaires

Merkle-Damgaard

- C'est le mode opératoire habituel
  - Le padding contient la longueur du message à hacher
  - Les hachés intermédiaires H_i font tous n bits
  - Le haché de M est le dernier haché intermédiaire H_{t+1}
- Il y a certains problèmes de sécurité (liés au chaînage, notamment)

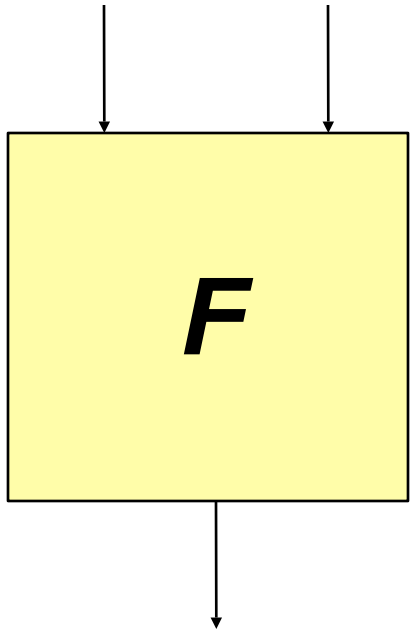
Théorème

- Th: Si la fonction de compression f est résistante aux collisions, alors la construction H^f de MD avec strengthening l'est également
- Preuve: ?

Fonctions classiques

Nom	n	h	Auteur
MD2	128	128	RSA
MD4	512	128	RSA
MD5	512	128	RSA
SHA-1	512	160	NIST
SHA-256	512	256	NIST
RIPEMD-*	512	128,160,…	
HAVAL-*	1024	128,160,…	

Fonction de compression

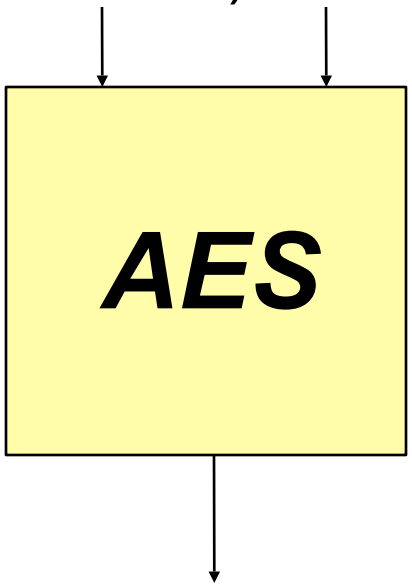


$F : \{0,1\}^m \times \{0,1\}^n \rightarrow \{0,1\}^n$

Idée naturelle

Haché intermédiaire
== Plaintext (128 bits)

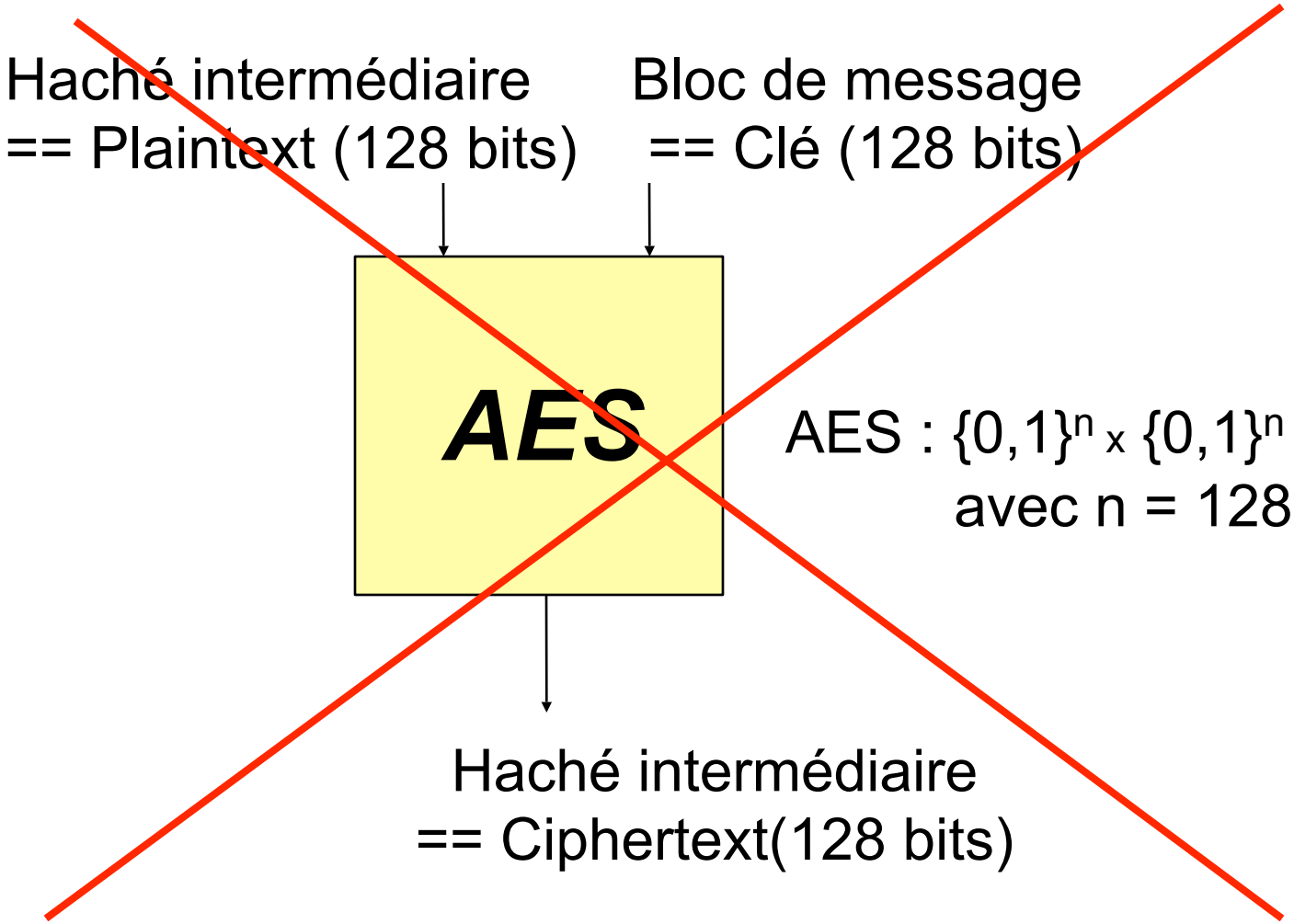
Bloc de message
== Clé (128 bits)



AES : $\{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$
avec n = 128

Haché intermédiaire
== Ciphertext(128 bits)

Idée naturelle

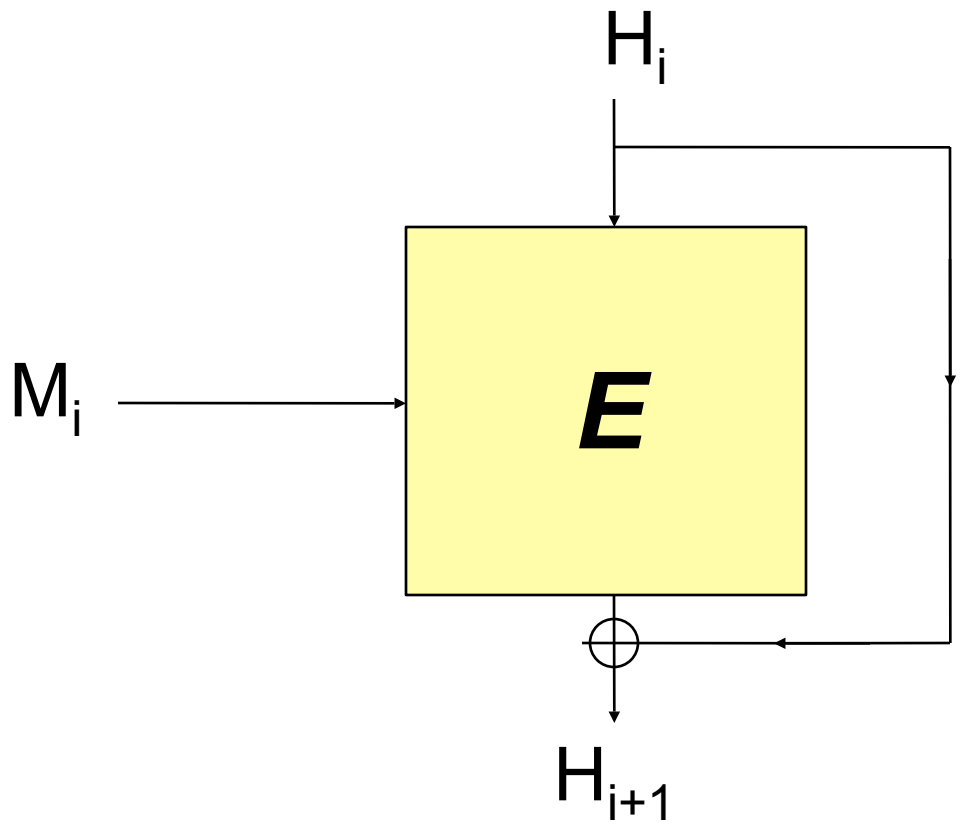


AES : $\{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$
avec n = 128

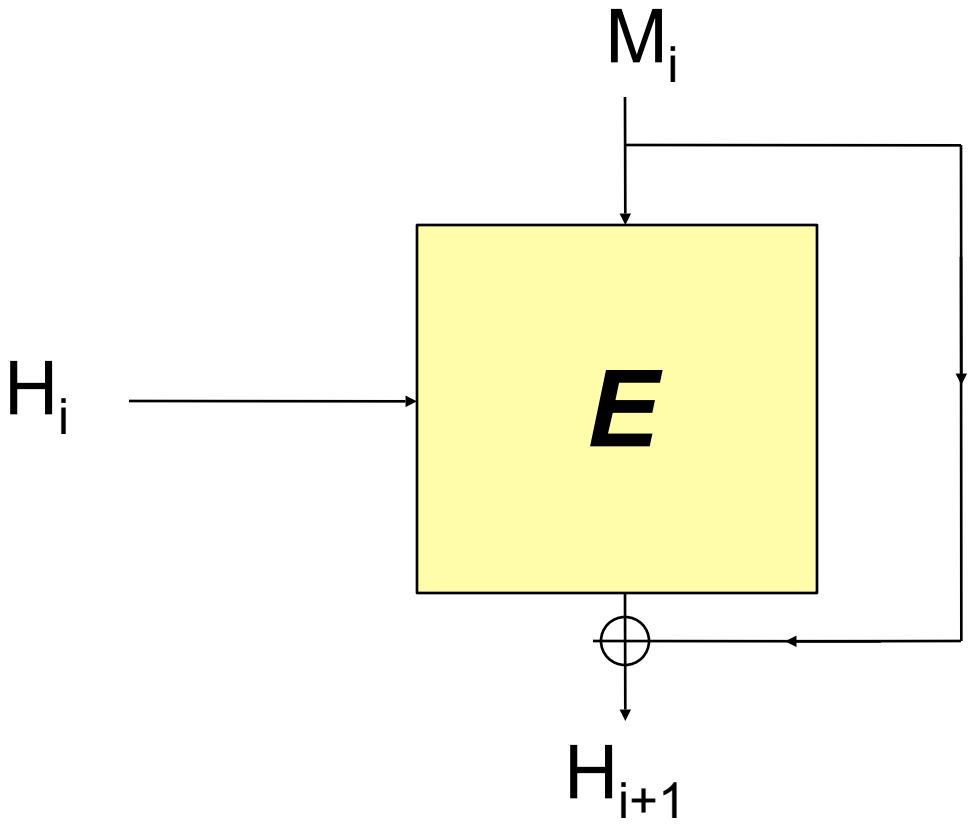
Construction

- Les fonctions de compression sont souvent obtenues à partir d'un algorithme de chiffrement par bloc (noté E)
- MAIS elles doivent être difficiles à inverser
- Nombreuses constructions possibles
  - Davies-Meyer
  - Matyas-Meyer-Oseas
  - Miyaguchi-Preneel

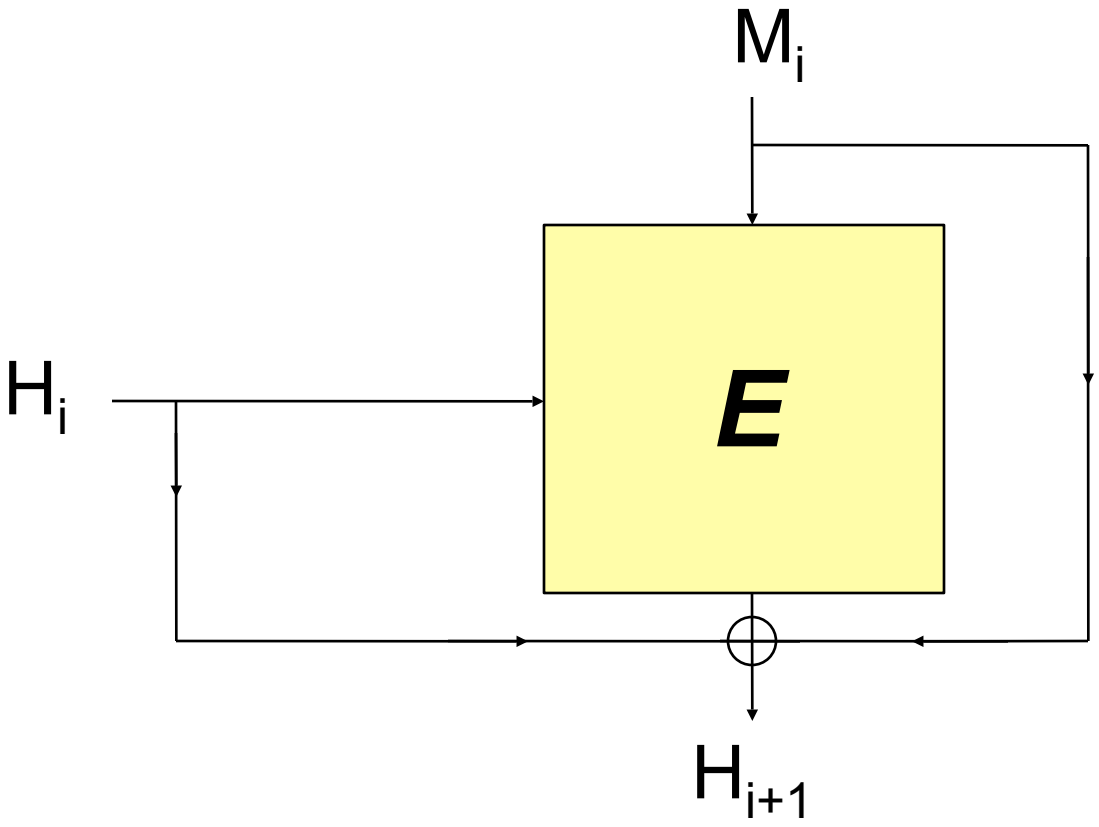
Davies-Meyer



Matyas-Meyer-Oseas



Miyaguchi-Preneel



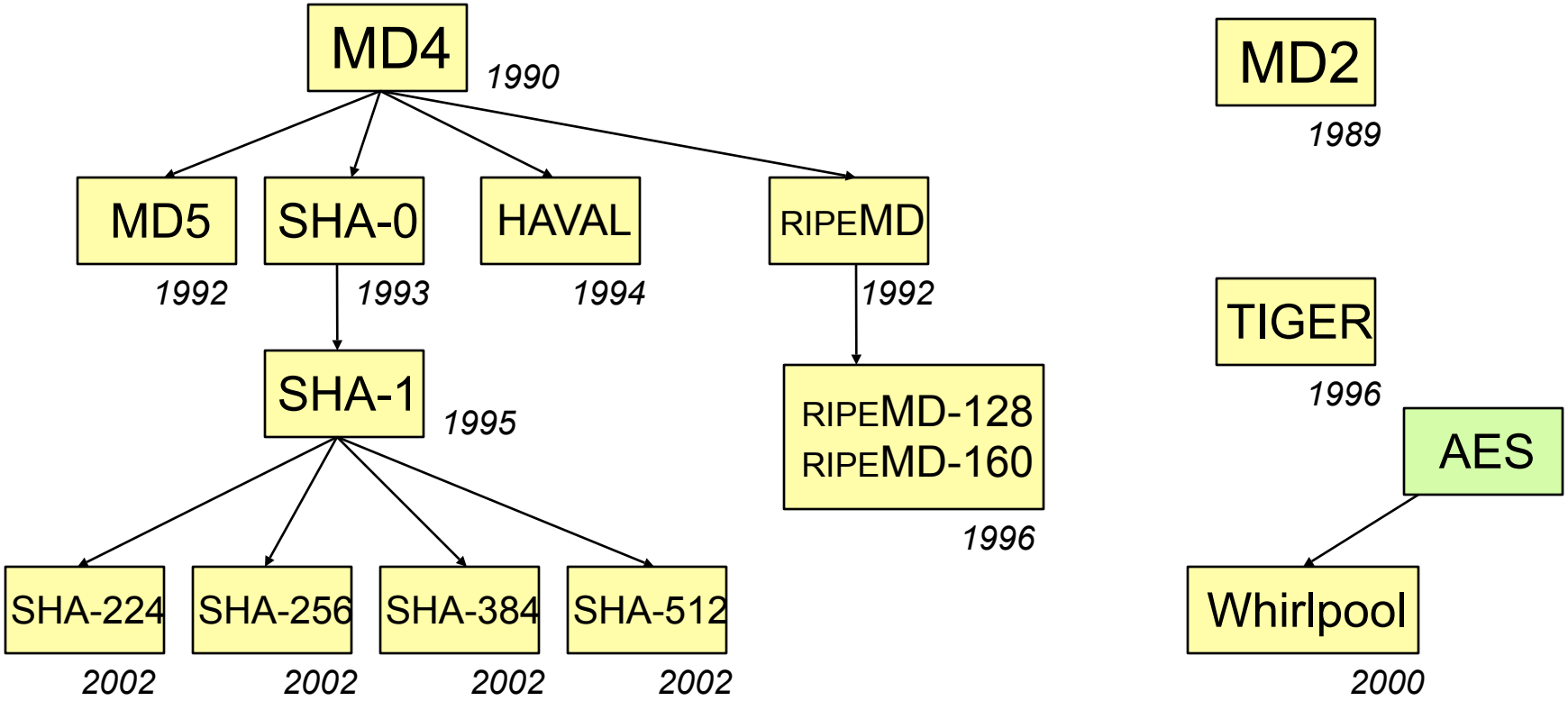
En pratique ...

- La taille du bloc est parfois trop faible (ex : DES = 64 bits; AES = 128 bits)
- Problème d'efficacité
- On utilise donc souvent des fonctions de compression dédiées qui « ressemblent » à la construction de Davies-Meyer

Fonctions classiques

- La plupart des fonctions usuelles utilisent cette idée :
  - Construction itérative
  - Fonction de compression : Davies-Meyer
  - L'algorithme de chiffrement par bloc sous-jacent a été spécialement conçu
  - Modèle = MD4

Historique


MD4
1990
MD5
1992
SHA-0
1993
HAVAL
1994
RIPEMD
1992
SHA-1
1995
RIPEMD-128
RIPEMD-160
1996
SHA-224
2002
SHA-256
2002
SHA-384
2002
SHA-512
2002
MD2
1989
TIGER
1996
AES
Whirlpool
2000

Famille SHA

- SHA-0 publié par le NIST en 1993
- Modifié légèrement en 1995 : version SHA-1
- Nouvelle famille d'algorithmes publiée en 2002, SHA-2
- Mode opératoire = Merkle-Damgaard
- Fonction de compression = Davies-Meyer avec un block cipher dédié

Remarques

- SHA-0 et SHA-1 utilisent presque la même fonction de compression
  - Le block cipher sous-jacent (SHACAL) semble sûr
  - Seule la dérivation des sous-clé change entre SHA-0 et SHA-1
- La famille SHA-2 utilise une fonction de compression modifiée

SHA-0

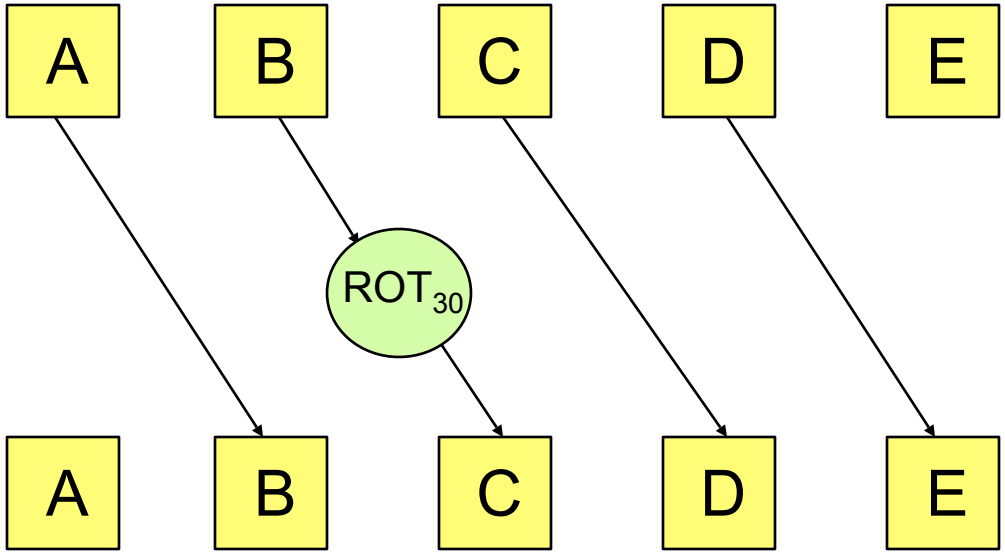
- La fonction de compression F prend en entrée
  - un haché intermédiaire de 160 bits
    vu comme 5 mots de 32 bits = (A B C D E)
  - un bloc de message de 512 bits
- Un tour élémentaire est itéré 80 fois dans F (comme dans un block cipher)

SHA-0

Tour élémentaire de SHA-0

Bloc de Message (512 bits)

« Sous-clé » de 32 bits

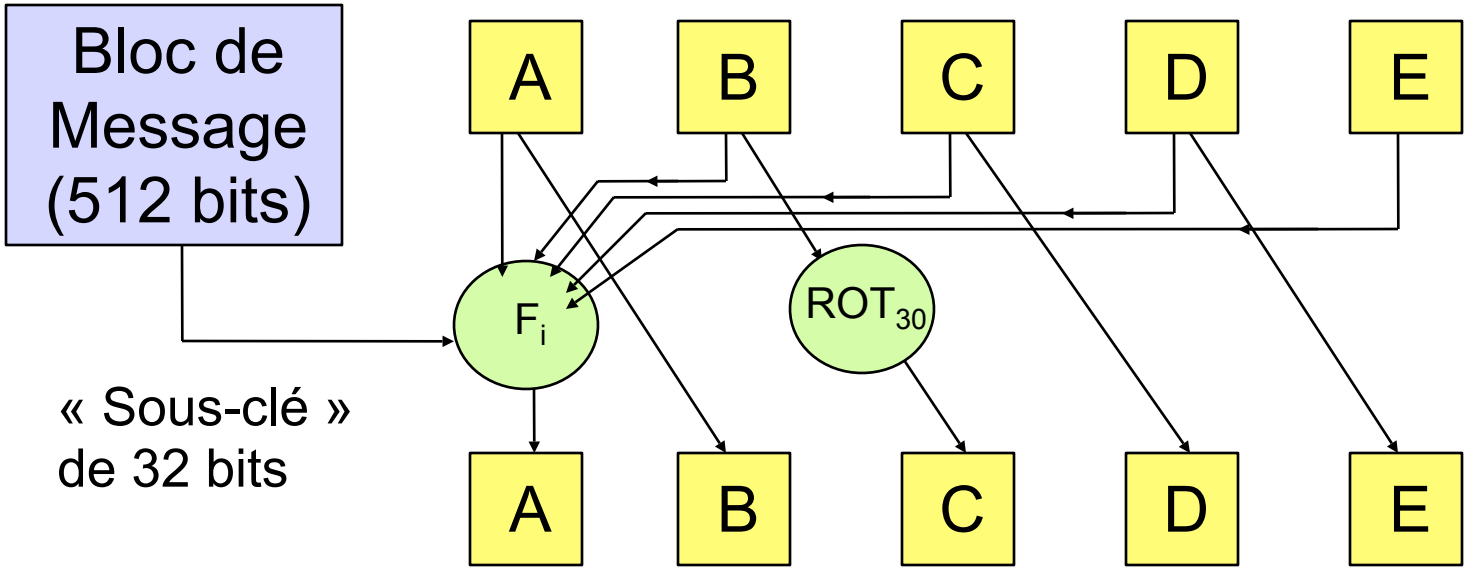


Tous les opérations se font sur 32 bits (efficacité)

Schéma de Feistel généralisé

SHA-0

Tour élémentaire de SHA-0



Tous les opérations se font sur 32 bits (efficacité)

Schéma de Feistel généralisé

SHA-0

$E' := D$

$D' := C$

$C' := ROT_{30}(B)$

$B' := A$

$A' := ROT_5(A) + f_i(B,C,D) + E + Cst_i + W_i$

Tour numéro $i \in \{1,80\}$

Fonction booléenne simple

Constante de tour

Sous-clé dérivée du message

SHA-0

Valeurs initiales de (A B C D E) =

67452301 efcdab89 98badcfe 10325476 c3d2e1f0

Cst_i = (suivant la valeur de i)

5a827999 ou 6ed9eba1 ou 8f1bbcdc ou ca62c1d6

Fonction f_i =(suivant la valeur de i)

XOR(x,y,z) ou IF(x,y,z) ou MAJORITE(x,y,z)

SHA-0

- Les sous-clés de 32 bits W_i sont dérivées des mots du bloc de message $(M_i)_{i=0,\ldots,15}$
- Expansion linéaire :

$$W_i = M_i \text{ pour } i = 1 \ldots 16$$

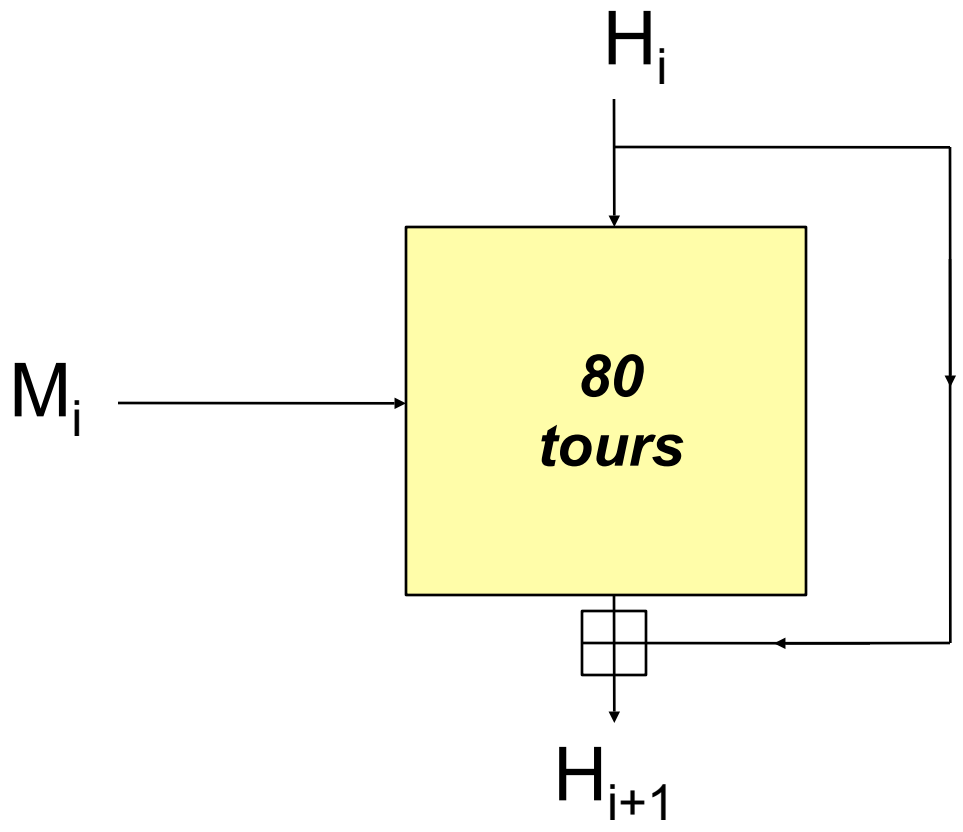
$$W_i = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16})_{<<<1}$$

$$\text{pour } i = 17 \ldots 80$$

Ajouté dans SHA-1

SHA

Le tour élémentaire est inversible
(Feistel généralisé) ! On applique Davies - Meyer

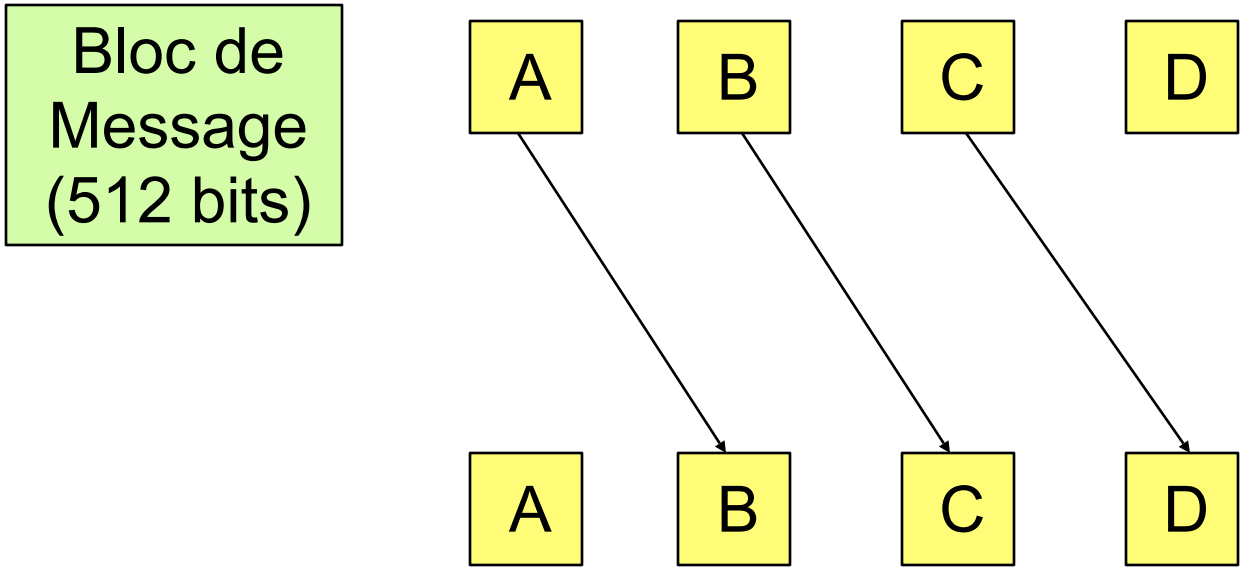


MD5

- Version « renforcée » de MD4 (plus de tours)
- MD4 et MD5 développés par Rivest (RSA Labs)
- MD4 proposé en 1990
- MD5 proposé en 1992
- Très similaire à SHA-1

MD5

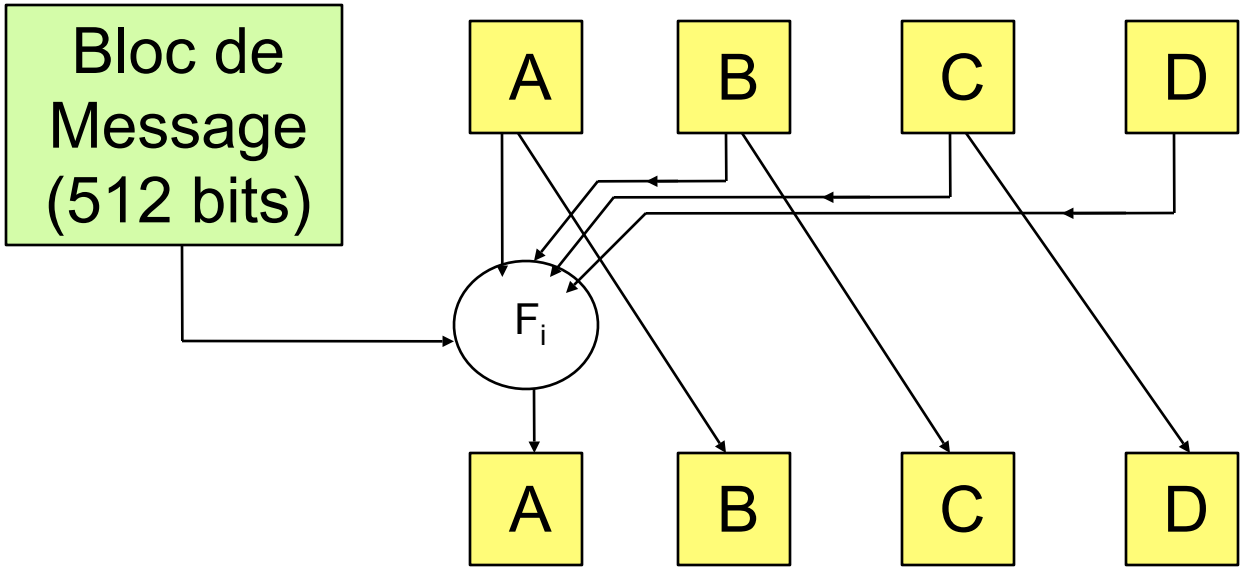
La fonction de compression est constituée de 64 tours :



Tous les objets manipulés font 32 bits

MD5

La fonction de compression est constituée de 64 tours :



Tous les objets manipulés font 32 bits

MD5

- Fonction F_i de la forme

  $$F_i(A,B,C,D) = ROT_{s_i}(A + (D + f_i(A,B,C) + Cst_i + W_i))$$

- s_i et Cst_i sont donnés dans les spécifications

- $W_i = M_{\pi(i)}$

  Les mots du message initial sont simplement mélangés

Sécurité des fonctions

- MD2 : attaque en préimages
- MD4 : attaque en collisions
- MD5 : attaque en collisions
- SHA-0 : attaque en collisions
- SHA-1 : attaque en collisions
- SHA-256 : pas d'attaque

Synthèse

- Remous dans le domaine des fonctions de hachage
  - Attaques en collision contre MD4, MD5, SHA-0
  - Attaques théoriques contre SHA-1 en 2^{63}
- En pratique, seul SHA-256 résiste encore
- Besoin de nouvelles primitives ? SHA-3
- Autres constructions à base de problèmes issus de la théorie des nombres avec preuve