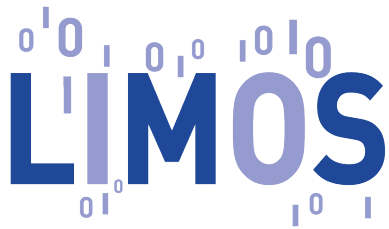
Hachage et Signature

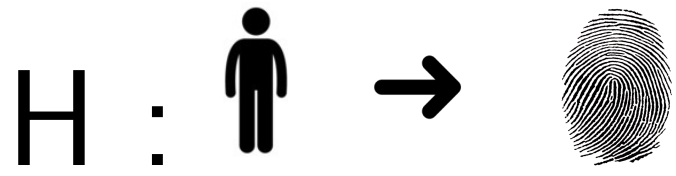
Pascal Lafourcade



2022-2023

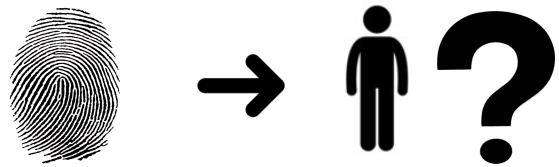
Plan

Hash function (SHA-1, SHA-3)

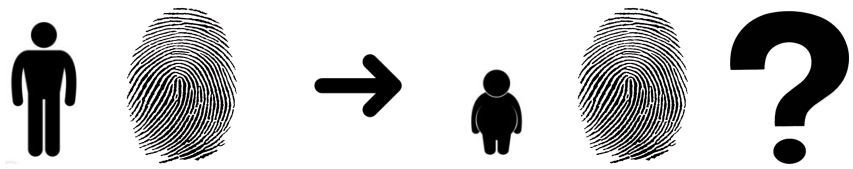


Properties

- First Pré-image

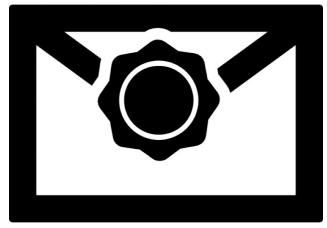


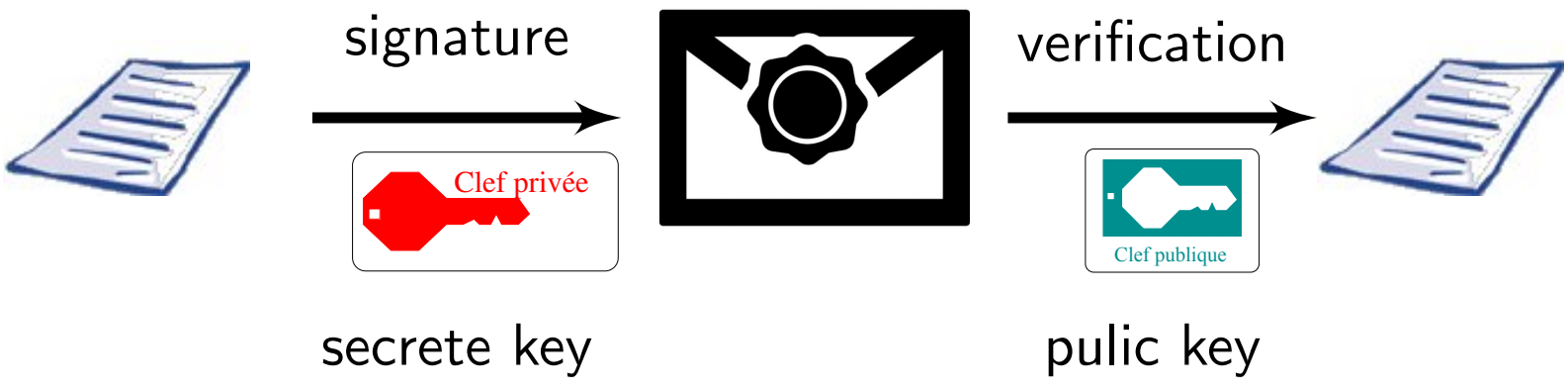
- Second Pré-image



- Collision

Signature





RSA: $m^d \mod n$

Plan

Deux types de fonctions de hachage

Sans clé : fonction de hachage

- Intégrité des donnée
- Empreintes des messages

Avec clé : Message Authentication Code (MAC)

- Vérification de mots de passe
- Confirmation ou établissement de clé (TLS 1.3)
- Horodatage

Fonction de Hachage

Une fonction de hachage H prend en entrée un bit-string de n'importe quelle taille et retourne un haché ('digest') ou empreinte de taille fixe n.

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

$$H(Alice) \quad = \quad$$ 

Definition (Pre-image resistance (One-Way) OWHF)

Soit y, il est difficile de retrouver x tel que :

$$h(x) = y$$

Properties des fonctions de hachage

2nd Pre-image resistance (weak-collision resistant CRHF)

Soit x, il est difficile de trouver $x' \neq x$ tel que

$$h(x') = h(x)$$

Collision resistance (strong-collision resistant)

Il est difficile de trouver x et x' tels que $x \neq x'$ et

$$h(x) = h(x')$$

Construction de Merkle-Damgård

Soit f une fonction de compression

$$f : \{0,1\}^m \to \{0,1\}^n$$

1. Couper le message x en blocs de taille $m - n$:

$$x = x_1 x_2 \dots x_t$$

2. Ajouter des zéros si nécessaire à x_t
3. Initialiser $H_0 = 0^n$
4. Itérer sur les blocs : $H_i = f(H_{i-1} || x_i)$
5. pour obtenir $h(x) = H_t$

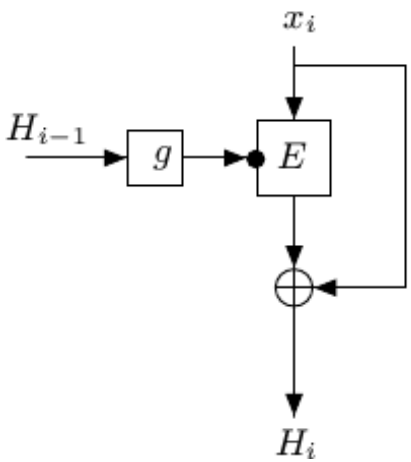
Théorème

Theorem

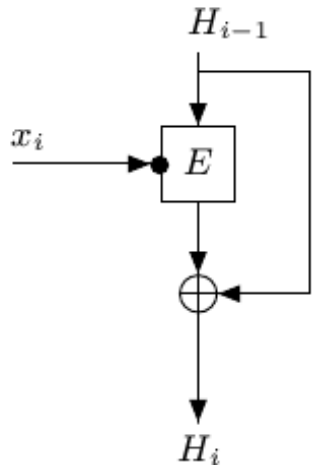
Si la fonction de compression f est collision resistante, alors la fonction h de Merkle-Damgård est collision resistante.

En utilisant des chiffrements par blocs

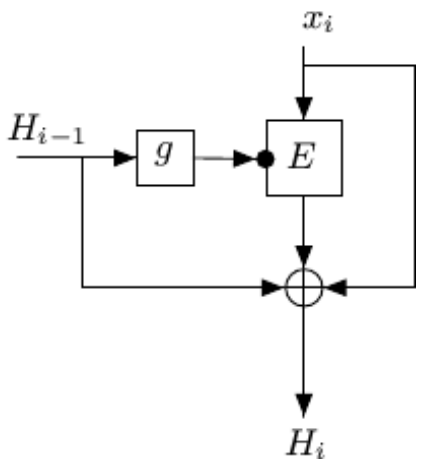
Matyas-Meyer-Oseas



Davies-Meyer

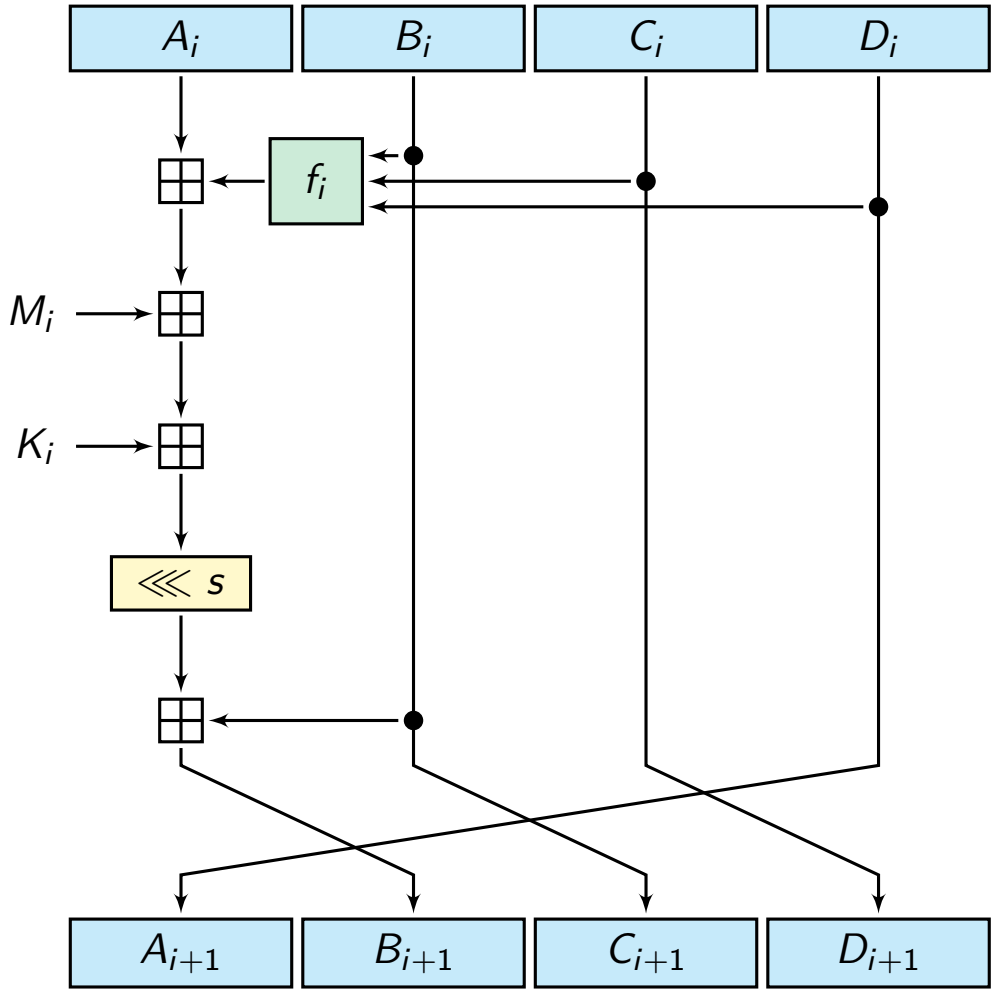


Miyaguchi-Preneel



MD5 par Ron Rivest en 1991

Pour chaque bloc de 512-bit du message à hacher



Ki constante de 32-bit telle que $K_i^{\{256\}} = \lfloor |2^{32} \times \sin(i+1)| \rfloor$.
$\boxplus$ représente l'addition modulo 2^{32} et $\lll s$ un décalage à gauche de s bits variant à chaque tour.

MD5 Détails

Il y a 4 possibles fonctions F, qui change à chaque tour.

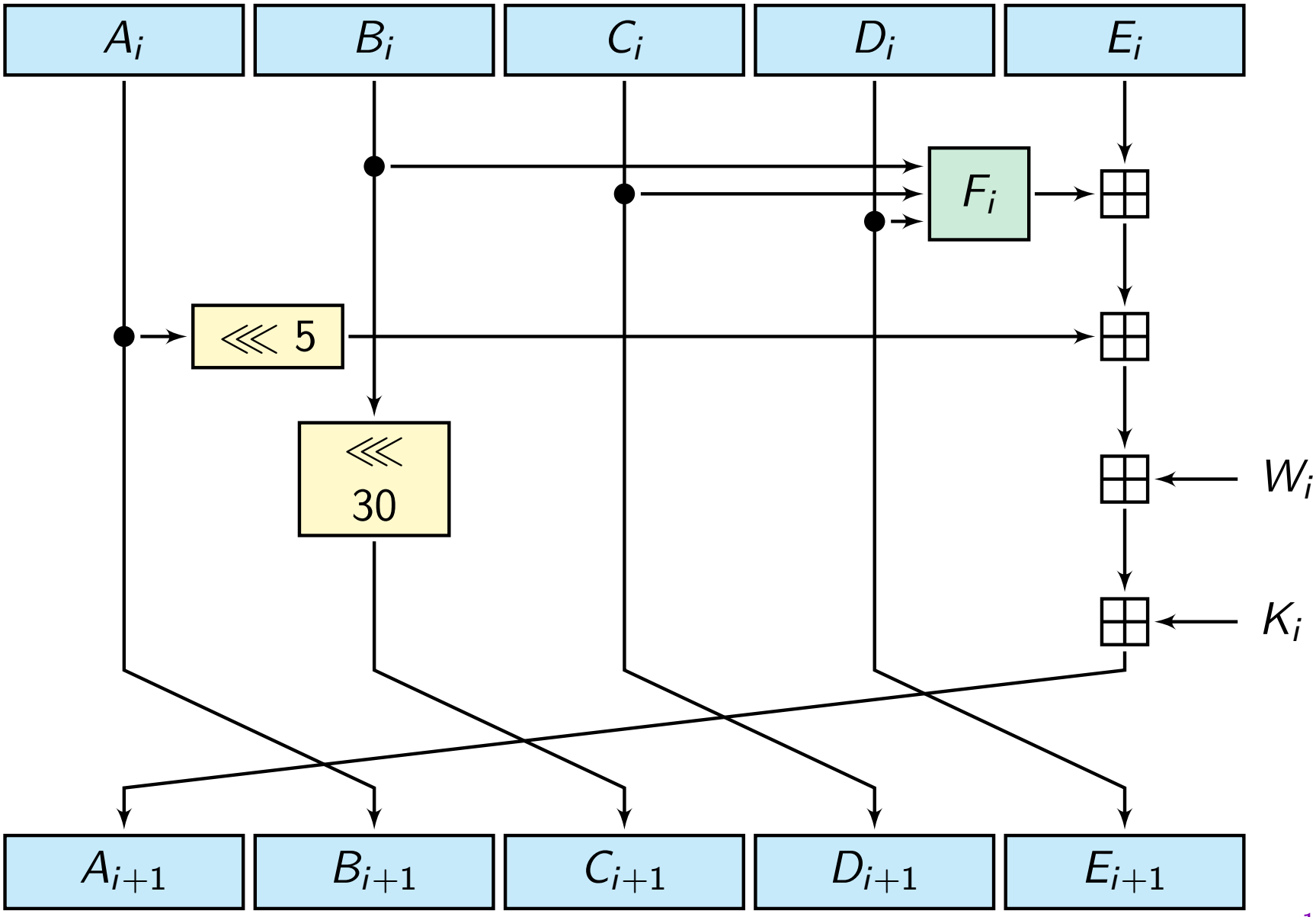
- $F(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$
- $G(B, C, D) = (B \wedge D) \vee (C \wedge \neg D)$
- $H(B, C, D) = B \oplus C \oplus D$
- $I(B, C, D) = C \oplus (B \vee \neg D)$

```
s[ 0..15] := { 7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22 }
s[16..31] := { 5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20 }
s[32..47] := { 4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23 }
s[48..63] := { 6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21 }
```

Cryptanalyses de MD5

- En 1993, Den Boer et Bosselaers ont trouvé une "pseudo-collision".
- En 1996, Dobbertin propose une collision de MD5.
- Le 17 aout 2004, Collisions pour MD5 par Xiaoyun Wang, Dengguo Feng, Xuejia Lai, et Hongbo Yu.
- Le 1 mars 2005, Arjen Lenstra, Xiaoyun Wang, et Benne de Weger produisent deux certificats X.509 qui ont le même MD5.
- En 2005, Vlastimil Klima construit des collisions MD5 en quelques heures avec un PC.
- Le 18 mars 2006, Klima propose un algorithme pour trouver une collision en moins d'une minute sur un PC.
- Le 24 décembre 2010, Tao Xie et Dengguo Feng annoncent le premier (512 bit) MD5 collision.

SHA-1 (Secure Hash Algorithm), par la NSA en 1995



Détails de SHA-1 : $f_0, f_1, \ldots, f_{79}$

$$f_t = \begin{cases} Choix & \text{si} \quad 0 \leq t \leq 19 \\ Parite & \text{si} \quad 20 \leq t \leq 39 \\ Majorite & \text{si} \quad 40 \leq t \leq 59 \\ Parite & \text{si} \quad 60 \leq t \leq 79 \end{cases}$$

$$\begin{aligned} Choix(b,c,d) &= (b \wedge c) \vee ((\neg b) \wedge d) \\ &= (b \wedge c) \oplus ((\neg b) \wedge d) \\ &= (b \wedge c) + ((\neg b) \wedge d) \\ &= d \oplus (b \wedge (c \oplus d)) \end{aligned}$$

$$\begin{aligned} Majorite(b,c,d) &= (b \wedge c) \vee (b \wedge d) \vee (c \wedge d) \\ &= (b \wedge c) \oplus (b \wedge d) \oplus (c \wedge d) \\ &= (b \wedge c) \vee (d \wedge (b \vee c)) \\ &= (b \wedge c) \vee (d \wedge (b \oplus c)) \\ &= (b \wedge c) \oplus (d \wedge (b \oplus c)) \\ &= (b \wedge c) + (d \wedge (b \oplus c)) \\ &= Choix(c \oplus d, b, c) \end{aligned}$$

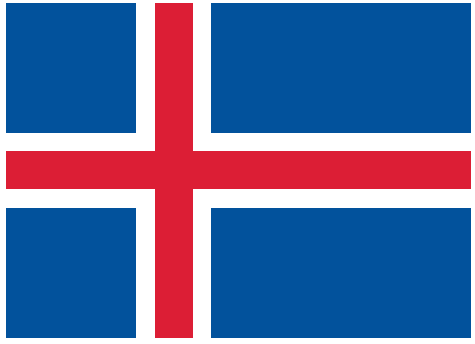
$$Parite(b,c,d) = b \oplus c \oplus d$$

Constantes de SHA-1

SHA-1 utilise quatre valeurs réparties dans les 80 constantes $K_0, K_1, \ldots, K_{79}$

$$K_t = \begin{cases} \text{0x5a827999}, & \text{si } 0 \leq t \leq 19 \\ \text{0x6ed9eba1}, & \text{si } 20 \leq t \leq 39 \\ \text{0x8f1bbcdc}, & \text{si } 40 \leq t \leq 59 \\ \text{0xca62c1d6}, & \text{si } 60 \leq t \leq 79 \end{cases}$$

Collision pour deux PDF



SHA1(TP/SHA1/a.pdf)=
a5a678701d8b2ab07c96d101b3331fb4992f0980
SHA1(TP/SHA1/b.pdf)=
a5a678701d8b2ab07c96d101b3331fb4992f0980

`https://shattered.io/`

List of Hash Functions

Algorithm	Output size	Internal state size	Block size	Length size	Word size	Collision
HAVAL	256/.../128	256	1024	64	32	Yes
MD2	128	384	128	No	8	Almost
MD4	128	128	512	64	32	Yes
MD5	128	128	512	64	32	Yes
PANAMA	256	8736	256	No	32	Yes
RadioGatún	Arbitrarily long	58 words	3 words	No	1-64	No
RIPEMD	128	128	512	64	32	Yes
RIPEMD	128/256	128/256	512	64	32	No
RIPEMD	160/320	160/320	512	64	32	No
SHA-0	160	160	512	64	32	Yes
SHA-1	160	160	512	64	32	With flaws
SHA-256/224	256/224	256	512	64	32	No
SHA-512/384	512/384	512	1024	128	64	No
Tiger(2)	192/160/128	192	512	64	64	No
WHIRLPOOL	512	512	512	256	8	No

Historique

$$h : \{1,0\}^* \to \{1,0\}^n$$

- MD5: $n = 128$ (Ron Rivest, 1992)
- SHA-1: $n = 160$ (NSA, NIST, 1995)
- SHA-2: $n \in \{224, 256, 384, 512\}$ (NSA, NIST, 2001)
- SHA-3: n is arbitrary (NSA, NIST, 2012)

Compétition : SHA-3 Zoo (11/2/2007 – 10/2/2012)

64 Submissions, 54 selectionnés,

1. * BLAKE Jean-Philippe Aumasson
2. Blue Midnight Wish Svein Johan Knapskog
3. CubeHash Daniel J. Bernstein preimage
4. ECHO Henri Gilbert
5. Fugue Charanjit S. Jutla
6. * Grøstl Lars R. Knudsen
7. Hamsi Özgül Küçk
8. * JH Hongjun Wu preimage
9. * Keccak The Keccak Team
10. Luffa Dai Watanabe
11. Shabal Jean-François Misarsky
12. SHAvite-3 Orr Dunkelman
13. SIMD Gaëtan Leurent
14. * Skein Bruce Schneier

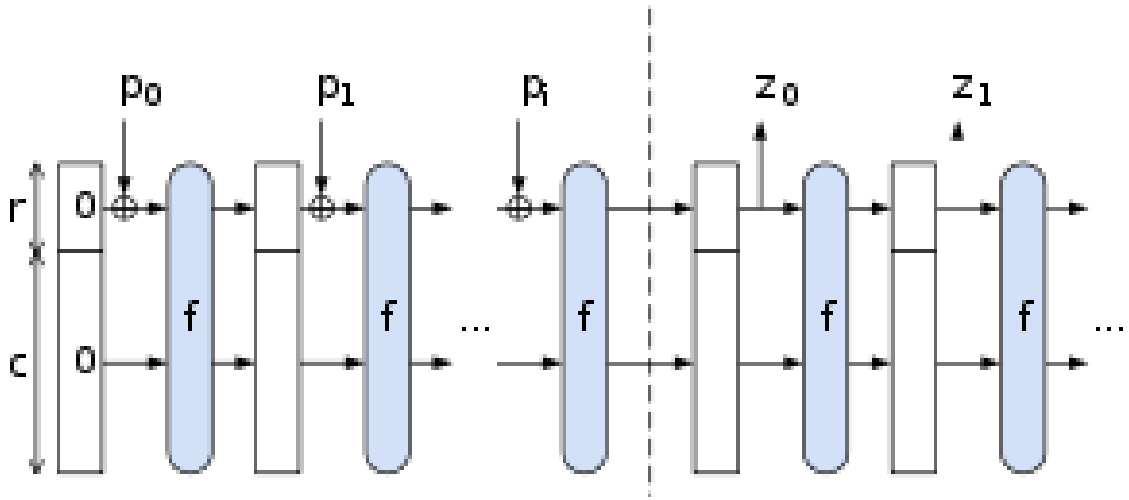
SHA-3 = Keccak (éponge + compression)

Auteurs

- Guido Bertoni (Italy) STMicroelectronics,
- Joan Daemen (Belgium) STMicroelectronics,
- Michaël Peeters (Belgium) NXP Semiconductors,
- Gilles Van Assche (Belgium) STMicroelectronics.

SHA-3 = Keccak

$$H(P_0|P_1|\ldots|P_i) = Z_0|Z_1|\ldots|Z_l$$



$b = r + c$

- r bits de taux (rate)
- c bits de capacité

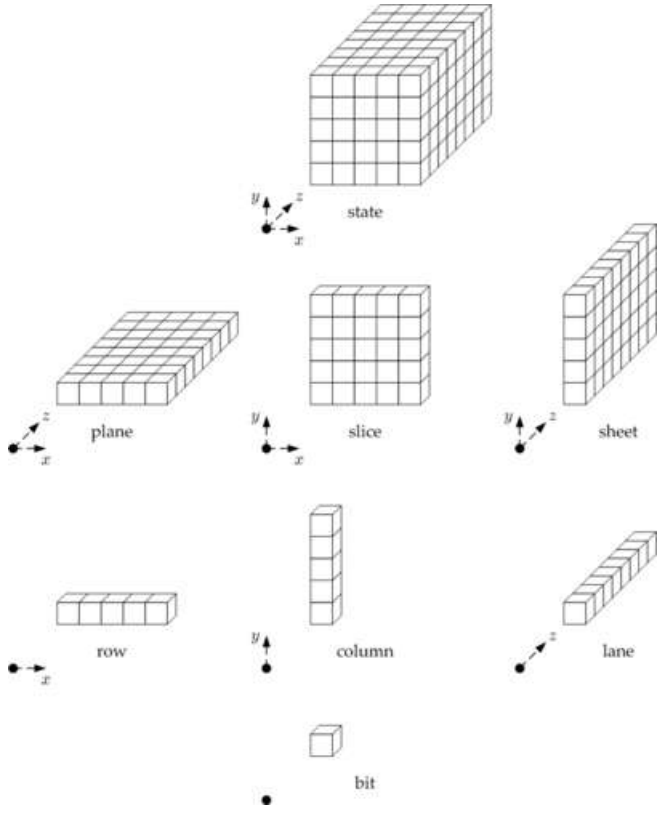
Détails de Keccak

- 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
- SHA-3 avec r = 1088 et c = 512
  - permutation : 1600
  - securité de 256: post-quantum suffisant
- SHA-3 Lightweight avec r = 40 et c = 160
  - permutation : 200
  - securité de 80 : comme SHA-1

SHA-3 = Keccak détails

Defini pour des mots de taille, $w = 2^\ell$ bits (si $\ell = 6$ mots de 64-bit)

État (State) est un tableau de bits de taille $5 \times 5 \times w$ (a[i][j][k])



- État = 5×5 "lanes", chacune contenant 2^ℓ bits
- (5×5)-bit slices, il y en a 2^ℓ

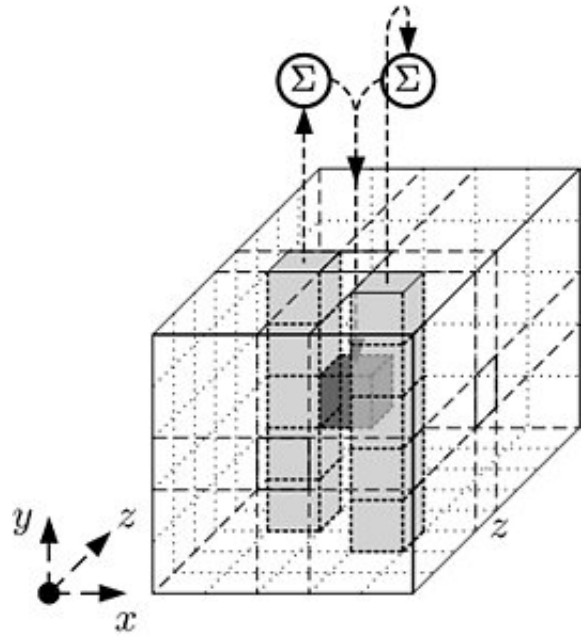
SHA-3 = Keccak

On fait $12 + 2 \times \ell$ iterations de ces étapes :

1. Θ
2. ρ
3. π
4. χ
5. ι

Keccak Θ

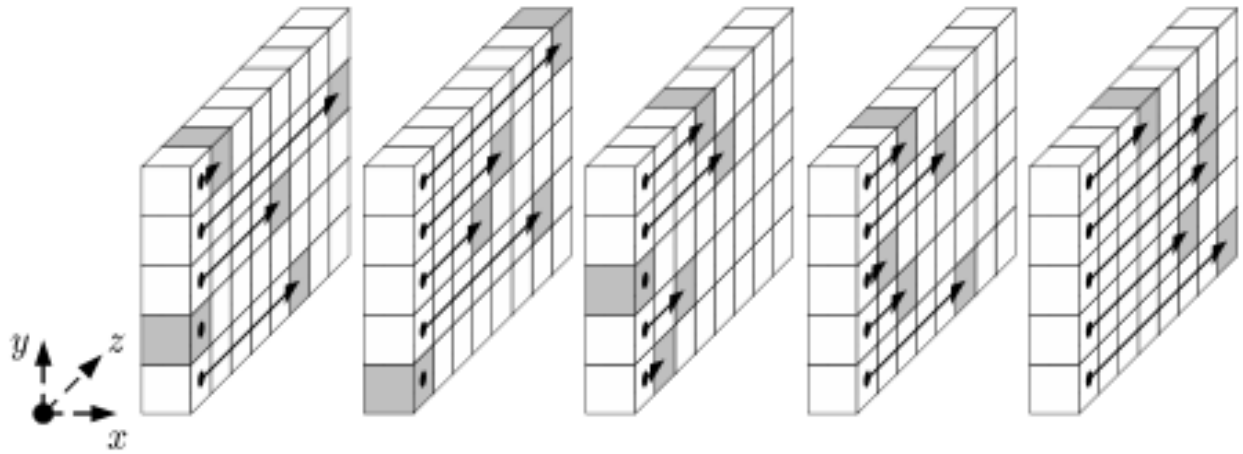
1. Calculer la parité de chacune des colones de 5-bit
2. $\oplus$ la somme de a[x-1][][z] et de a[x+1][][z-1] dans a[x][y][z].



$$a[i][j][k] \oplus = parity(a[0..4][j-1][k]) \oplus parity(a[0..4][j+1][k-1])$$

Keccak ρ

Rotation de bit pour chacun des 25 mots par une rotation différente.

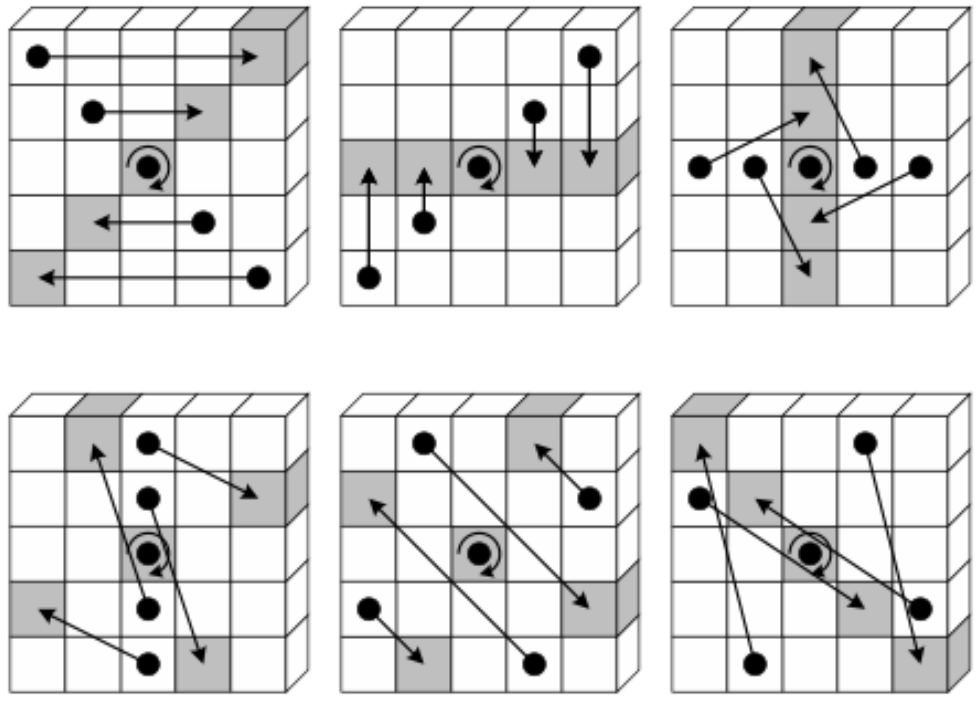


$a[0][0]$ n'est pas tourné, et pour tout $0 \leq t < 24$

$a[i][j][k] = a[i][j][k-(t+1)(t+2)/2]$, où $\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 1 & 0 \end{pmatrix}^t \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

Keccak π

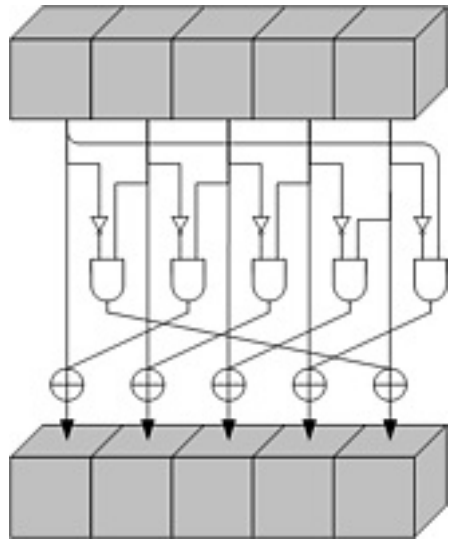
Permutation des 25 mots comme suit :



$$a[i][j] = a[j][2i + 3j]$$

Keccak χ

Calcul pour chaque colonne de $a = a \oplus (\neg b \& c)$.



$$a[i][j][k] \oplus = \neg a[i][j+1][k] \& a[i][j+2][k]$$

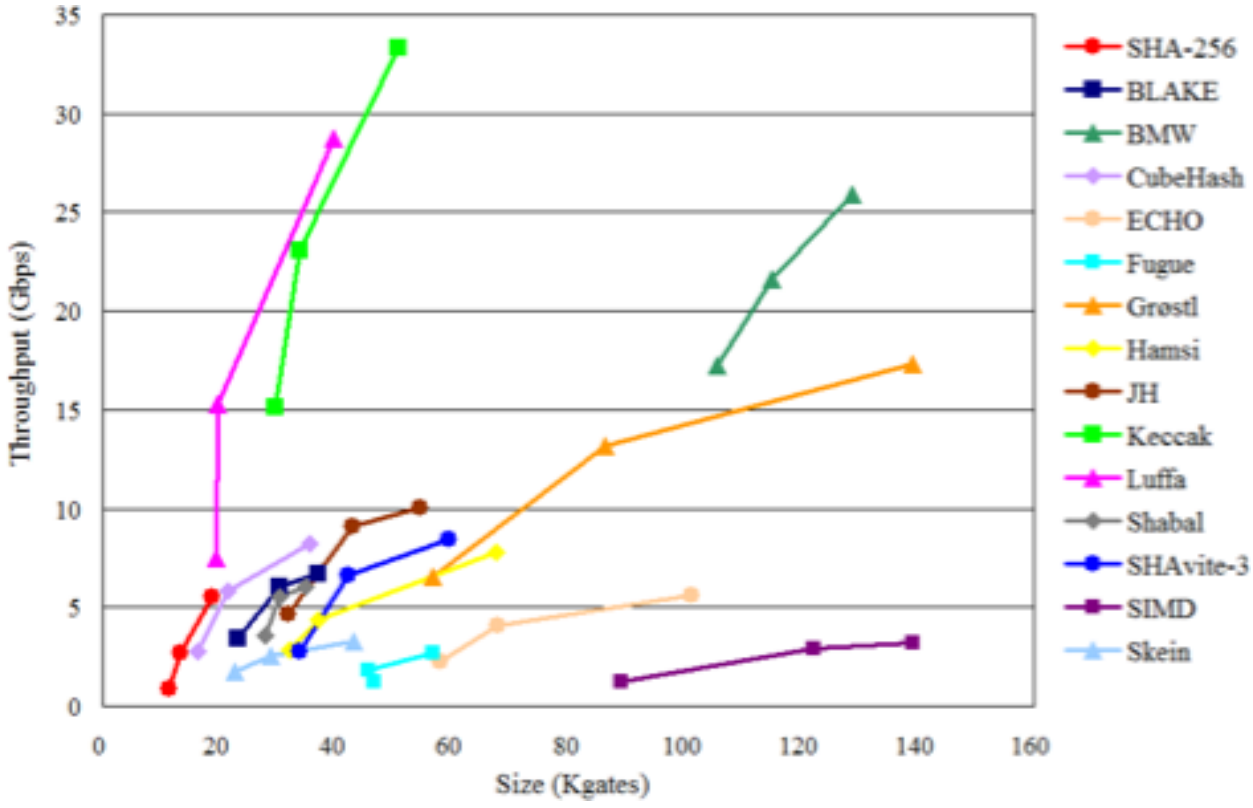
Seule opération non-lineare de SHA-3.

Keccak ι

- Dans le tour n, pour $0 \leq m \leq \ell$, faire $a[0][0][2m-1]$ exclusive avec le bit $m+7n$ de degré-8 LFSR (Linear Feedback Shift Register).

Cela casse la symmétrie qui est préservée par les autres sous-tours.

Keccak performances



DMAC (CBC-MAC variant)

Avec k et k' pour les chiffrements symétriques E et E' respectivement.

$c_1 := m_1$;
for $i = 2$ to n do:
    $z_i := c_{i-1} \oplus m_i$
    $c_i := E(z_i)$;
$tag := E'(c_n)$;

HMAC

Avec k et k'

$z_1 := k \| m_1$;

$c_1 := \mathcal{H}(z_1)$;

for $i = 2$ to n do:;

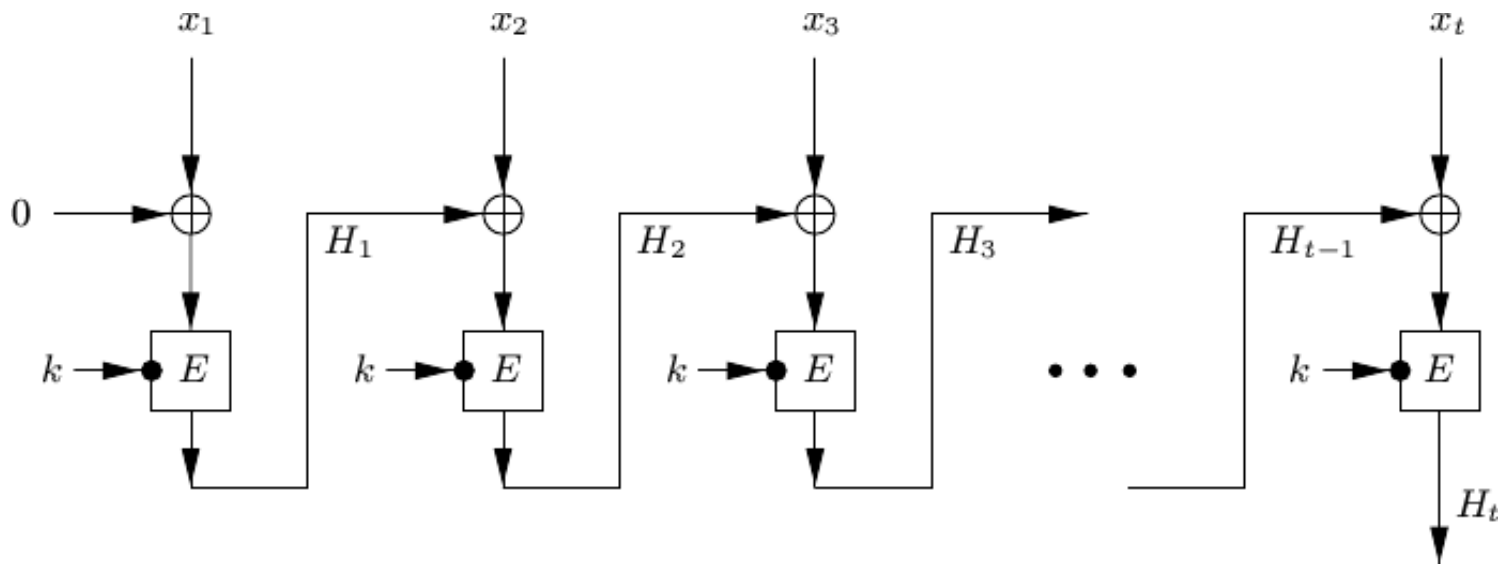
$z_i := c_{i-1} \| m_i$

$c_i := \mathcal{H}(z_i)$

$z' := k' || c_n$;

$tag := \mathcal{H}(z')$;

MAC basés sur un chiffrement par blocs



Plan

Propriété de sécurité

Forge Existentielle (Existential UnForgeability, EUF):

BUT : Forger au moins UN couple (m, σ) est difficule.

Forge Selective (Selective UnForgeability, SUF):

Soit m choisi par le challenger avant l'attaque.
BUT : Forger au moins UN couple (m, σ) est difficule.

Forge Universelle (Universal UnForgeability, UUF):

BUT : **Pour tout message** m, forger (m, σ) est difficile.

RSA Signature

Rappel : Chiffrement RSA

- $pk = (n, e)$ et $sk = d$ tel que $ed = 1 \mod \phi(n)$
- Chiffrement : $m^e \mod n$
- Déchiffrement : $c^d \mod n$

RSA Signature

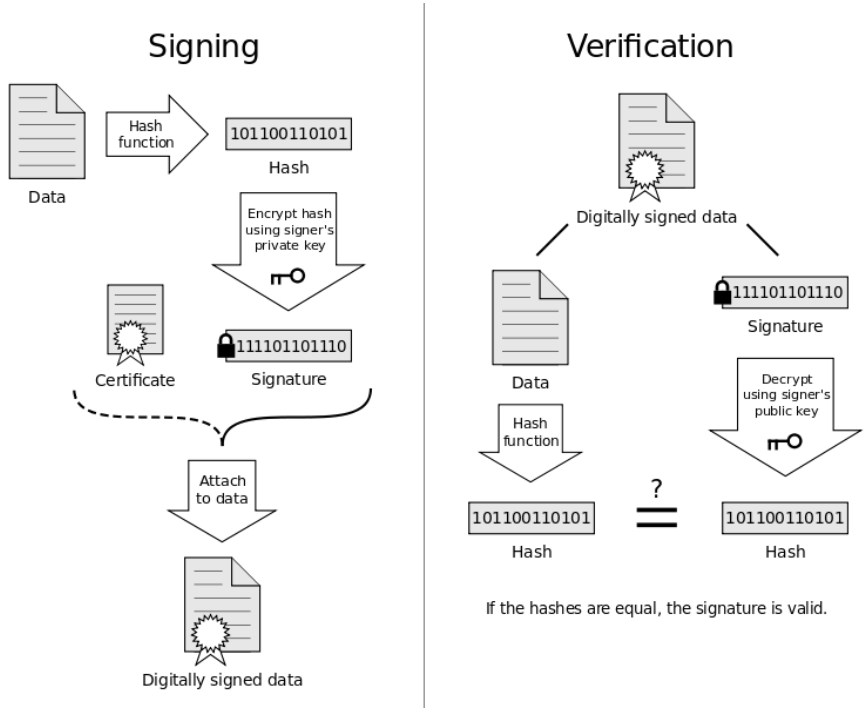
- $pk = (n, e)$ et $sk = d$ tel que $ed = 1 \mod \phi(n)$
- Signature : $\sigma = m^d \mod n$
- Verification : $\sigma^e = m \mod n$

Exercice : Signature RSA

Montrer que la signature RSA n'est pas EUF sûre ?:

Signature en pratique

Signer de gros fichier n'est pas éfficace : HASH-and-SIGN



Standards

- PKCS#1 v1.5: pas de preuve de sécurité
- PKCS#1 v2.1: PSS proposée en 1996 par Bellare et Rogaway

Elgamal Signature

Génération de clés

- Choisir une clé secrète x telle que $1 < x < p - 1$
- Calculer $y = g^x \mod p$
- $pk = (p, g, y)$
- $sk = x$

Elgamal Signature

Signature de m avec la clé $sk = x$

- Choisir un nombre aléatoire k tel que $1 < k < p - 1$ et $pgcd(k, p - 1) = 1$
- Calculer $r \equiv g^k \pmod{p}$
- Calculer $s \equiv (H(m) - xr)k^{-1} \pmod{p - 1}$

La paire (r, s) est la signature de m.

Elgamal Signature

Verification de la signature (r, s) du message m

- Vérifier que $0 < r < p$ et $0 < s < p - 1$.
- Vérifier que $g^{H(m)} \equiv y^r r^s \pmod{p}$

Elgamal Signature Correctness

Par construction : $s \equiv (H(m) - xr)k^{-1} \pmod{p-1}$
Ce qui implique $H(m) \equiv xr + sk \pmod{p-1}$

$$\begin{aligned} g^{H(m)} &\equiv g^{xr} g^{ks} \\ &\equiv (g^x)^r (g^k)^s \\ &\equiv (y)^r (r)^s \pmod{p}. \end{aligned}$$

Schnorr Signature

Génération des clés

- Choisir une clé privée, x.
- $pk = g^x$.

Signer M

- Choisir un nombre aléatoire k
- Calculer $r = g^k$.
- Soit $e = H(r \parallel M)$, $\parallel$ dénote la concatenation.
- Soit $s = k - xe \bmod (p-1)$.

La signature est $\sigma = (s, e)$

Vérification of $\sigma = (s, e)$

- $r_v = g^s y^e = g^{k-xe} g^{xe}$
- $e_v = H(r_v \parallel M)$
- Si $e_v = e = H(g^k \parallel M)$ alors la signature est vérifiée.

Plan

Rappel : Théorème des restes Chinois

Soient $n_1, \ldots, n_k$ des entiers deux à deux premiers entre eux, c'est-à-dire que $PGCD(n_i, n_j) = 1$ lorsque $i \neq j$. Alors pour tous entiers $a_1, \ldots, a_k$, il existe un entier x, unique modulo $n = \prod_{i=1}^{k} n_i$, tel que :

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ &\ldots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

$$x = \sum_{i=1}^{i=n} a_i \times e_i$$

Où, $e_i = \frac{n}{n_i} \times ((\frac{n}{n_i})^{-1}(\mod n_i))$
Utilisé pour RSA et dans l'algorithme de Silver-Pohlig-Hellman pour le calcul du logarithme discret.

Signature RSA-CRT

Rappel Signature RSA : $\sigma = m^d \mod n$ où $sk = d$, $pk = (e, n)$, $n = pq$

Signature RSA-CRT

- Cacluler, $s_1 = m^d \mod p$ et $s_2 = m^d \mod q$
- Pré-calculer $a = q \times (q^{-1} \mod p)$ et $b = p \times (p^{-1} \mod q)$
- $m^d \mod n = a \times s_1 + b \times s_2$

Grâce au théorème des restes Chinois : $a \times s_1 + b \times s_2 = q \times (q^{-1} \mod p) \times m^d \mod p + p \times (p^{-1} \mod q) \times m^d \mod q = m^d \mod (p \times q) = m^d \mod n$

Injection de fautes sur la signatue RSA-CRT

Un attaquant demande la signature σ de message m.

Injection de la faute sur s_1

Il peut aussi redemander la signature σ de message m et injecter une faute dans s_1 pour obtenir $\sigma^* = a \times s_1^* + b \times s_2$

Première observation : $\sigma \mod q = b \times s_2 = \sigma^* \mod q$, ainsi $\sigma - \sigma^* = 0 \mod q$, donc q divise $\sigma - \sigma^*$.
Par contre $\sigma \mod p = a \times s_1 \neq \sigma^* \mod p = a \times s_1^*$, donc, $\sigma - \sigma^* \neq 0 \mod p$, donc p ne divise pas $\sigma - \sigma^*$.
Ainsi : $pgcd(\sigma - \sigma^*, n = p \times q) = q$

Plan

Aujourd'hui

1. Fonctions de hachage
2. MAC
3. Signatures
4. Injection de fautes

Merci pour votre attention.

Questions ?

Phil Zimmermann en 1991

“If privacy is outlawed, only outlaws will have privacy”

https://privacytests.org/