Cryptographie – Chiffrement à clé publique et Fonction de hachage

Claire Delaplace & Léo Robert

Principes de Kerckhoffs

Auguste Kerckhoff : *"La méthode de chiffrement ne doit pas être secrète, et elle doit pouvoir tomber entre les mains de l'ennemi sans provoquer de problème"*

Principes de Kerckhoffs

1. Plus facile de garder secret une petite clé que tout un algorithme.

Principes de Kerckhoffs

1. Plus facile de garder secret une petite clé que tout un algorithme.
2. Si k n'est plus secret, il est plus simple de changer la clé que tout un algorithme.

Principes de Kerckhoffs

1. Plus facile de garder secret une petite clé que tout un algorithme.
2. Si k n'est plus secret, il est plus simple de changer la clé que tout un algorithme.

Aujourd'hui, les algorithmes sont *publiques*:

- Beaucoup de personnes vont regarder/améliorer les algos.

Principes de Kerckhoffs

1. Plus facile de garder secret une petite clé que tout un algorithme.
2. Si k n'est plus secret, il est plus simple de changer la clé que tout un algorithme.

Aujourd'hui, les algorithmes sont *publiques*:

- Beaucoup de personnes vont regarder/améliorer les algos.
- Il vaut mieux révéler les failles de sécurité par des *"ethical hackers"* que par des entités malicieuses.

Principes de Kerckhoffs

1. Plus facile de garder secret une petite clé que tout un algorithme.
2. Si k n'est plus secret, il est plus simple de changer la clé que tout un algorithme.

Aujourd'hui, les algorithmes sont *publiques*:

- Beaucoup de personnes vont regarder/améliorer les algos.
- Il vaut mieux révéler les failles de sécurité par des *"ethical hackers"* que par des entités malicieuses.
- Le code d'un algo peut être cassé (reverse engineering). La clé (nombre aléatoire) ne fait pas partie du code.

Principes de Kerckhoffs

1. Plus facile de garder secret une petite clé que tout un algorithme.
2. Si k n'est plus secret, il est plus simple de changer la clé que tout un algorithme.

Aujourd'hui, les algorithmes sont *publiques*:

- Beaucoup de personnes vont regarder/améliorer les algos.
- Il vaut mieux révéler les failles de sécurité par des *"ethical hackers"* que par des entités malicieuses.
- Le code d'un algo peut être cassé (reverse engineering). La clé (nombre aléatoire) ne fait pas partie du code.
- Un design publique rend possible l'établissement de *standard*.

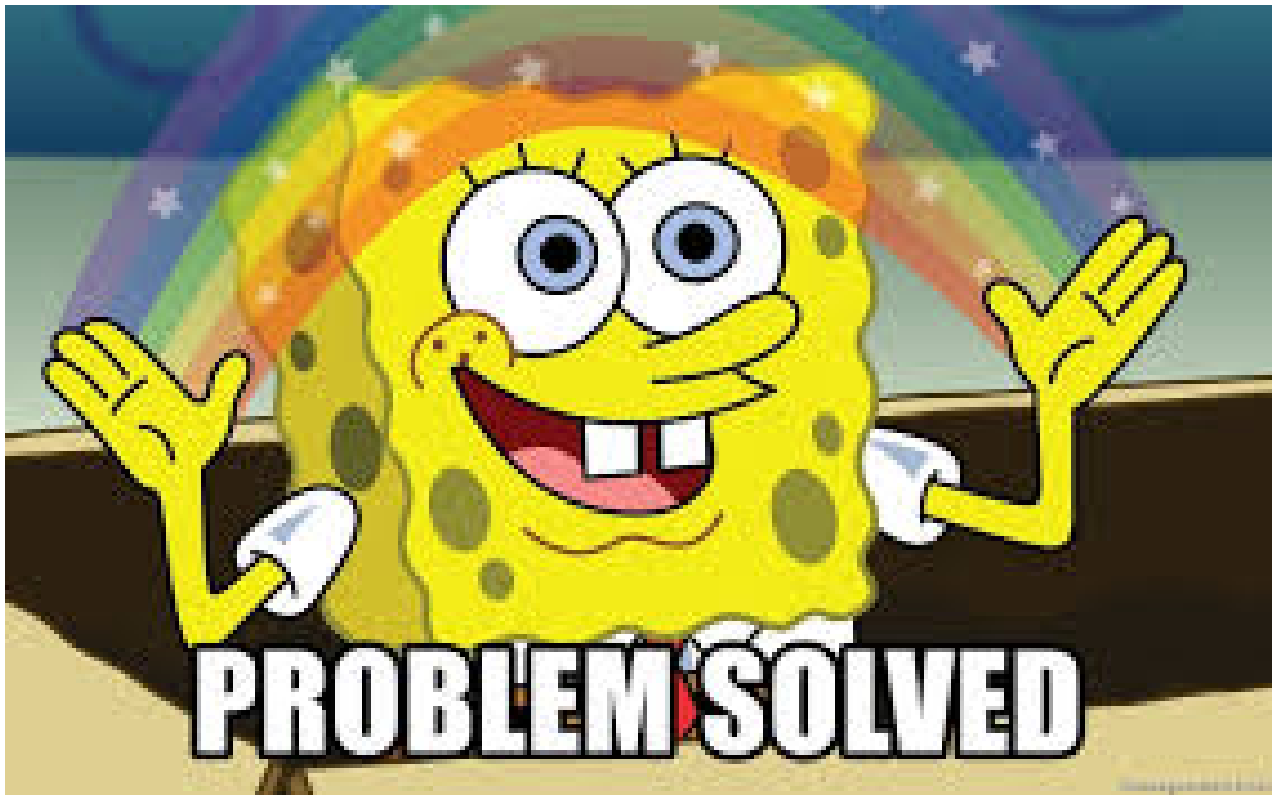
Exemples : A5/0 et A5/1 pour GSM, DVD CSS (1996) et I-message :)

Chiffrement parfait

Vernam ou OTP (One-Time Pad) : seul chiffrement connu prouvé inconditionnellement sûr.

$$C = M \oplus K$$

On prouve $P[M = m|C = c] = P[M = m]$: le chiffré C ne donne aucune information sur M.


PROBLEM SOLVED

Ou pas...

Deux gros problèmes :

1. Les clés : jetables, uniformément aléatoires et de même taille que le message;
2. L'échange de clé !!!

Brève histoire de la crypto asymétrique

Premier schéma par Merkle en 1974 (publié in 1978)
Un autre par Diffie and Hellman en 1976, encore utilisé de nos jours. Basé sur le problème du logarithme discret.
Un autre: Rivest, Shamir and Adleman (RSA) en 1978. Dépend du problème de factorisation.
D'autres systèmes: McEliece (1978, error-correcting codes), Merkle-Hellman (1978, knapsack problem, cassé en 1984), ElGamal (1985, discrete log), NTRUEncrypt (1996, lattices), Naccache-Stern (1998, factorisation)...

Chiffrement asymétrique

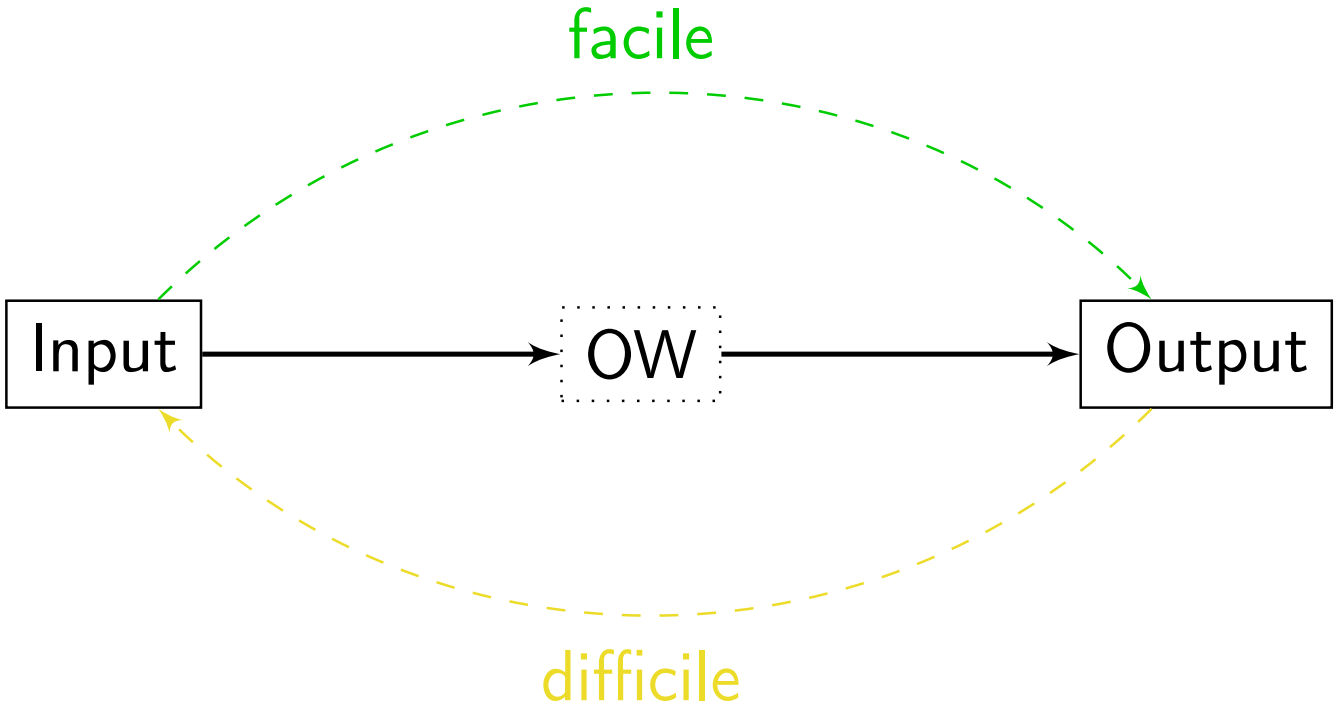
Caractéristiques :

- Une clé publique, partagée avec tout le monde
- Une clé secrète, sans la rélèver à personne

On peut voir le chiffrement symétrique comme un coffre et le chiffrement asymétrique comme un cadenas : n'importe qui peut le fermer mais une seule personne peut l'ouvrir.

Sécurité de la crypto asymétrique

La crypto asymétrique est liée au concept de fonctions *one-way*.



Fun fact: nous ne savons pas si les fonctions one-way existent $\Rightarrow$ cela impliquerait $P \neq NP$.

Fonction à sens unique (One-way function)

Définition

Une fonction f est à sens unique si :

- elle est facile à calculer : il existe un algorithme polynomial qui, en entrée x, donne $f(x)$ en sortie; et
- son inverse est difficile à calculer :

$$\Pr[m \xleftarrow{\$} \{0,1\}^*; y := f(m) : f(A(y, f)) = y]$$

est negligeable.

Trappe (Trapdoor):

- L'inverse devient facile à calculer avec la connaissance d'une information additionnelle.

Exemple: factorisation

Soient deux nombres premiers, il est “facile” de calculer leur produit. Mais en considérant un nombre composite, il est “difficile” de calculer ses deux facteurs premiers.

Exercice: factoriser

- 15

Exemple: factorisation

Soient deux nombres premiers, il est “facile” de calculer leur produit. Mais en considérant un nombre composite, il est “difficile” de calculer ses deux facteurs premiers.

Exercice: factoriser

- 15
- 143

Exemple: factorisation

Soient deux nombres premiers, il est “facile” de calculer leur produit. Mais en considérant un nombre composite, il est “difficile” de calculer ses deux facteurs premiers.

Exercice: factoriser

- 15
- 143
- une vrai clé RSA-2048 : **25 195 908 475 657 893 494 027 183 240 048 398 571 429 282 126 204 032 027 777 137 836 043 662 020 707 595 556 264 018 525 880 784 406 918 290 641 249 515 082 189 298 559 149 176 184 502 808 489 120 072 844 992 687 392 807 287 776 735 971 418 347 270 261 896 375 014 971 824 691 165 077 613 379 859 095 700 097 330 459 748 808 428 401 797 429 100 642 458 691 817 195 118 746 121 515 172 654 632 282 216 869 987 549 182 422 433 637 259 085 141 865 462 043 576 798 423 387 184 774 447 920 739 934 236 584 823 824 281 198 163 815 010 674 810 451 660 377 306 056 201 619 676 256 133 844 143 603 833 904 414 952 634 432 190 114 657 544 454 178 424 020 924 616 515 723 350 778 707 749 817 125 772 467 962 926 386 356 373 289 912 154 831 438 167 899 885 040 445 364 023 527 381 951 378 636 564 391 212 010 397 122 822 120 720 357**

Connaissances actuelles

Un algorithme naïf calcule en $\mathcal{O}(\sqrt{n})$

Le meilleur algo générique de factorisation calcule en $\mathcal{O}\left(e^{1.92(\ln n)^{1/3}(\ln \ln n)^{2/3}}\right)$: on dit que la complexité sub-exponential.

Mieux que exponentiel, mais pire que polynomiale $\Rightarrow$ OWF correcte.

Textbook RSA (unsecure)

Soient deux premiers p, q. Calculer $n = pq, \varphi(n) = (p-1)(q-1)$.
Choisir $e < \varphi(n)$ premier avec $\varphi(n)$, calculer $d = e^{-1} \mod \varphi(n)$.
Alors il existe k tel que $ed = k\varphi(n) + 1$.
La clé publique est $\mathsf{pk} = (n, e)$, la clé secrète est $\mathsf{sk} = d$.

Textbook RSA (unsecure)

Soient deux premiers p, q. Calculer $n = pq, \varphi(n) = (p-1)(q-1)$.
Choisir $e < \varphi(n)$ premier avec $\varphi(n)$, calculer $d = e^{-1} \mod \varphi(n)$.
Alors il existe k tel que $ed = k\varphi(n) + 1$.
La clé publique est $\mathrm{pk} = (n, e)$, la clé secrète est $\mathrm{sk} = d$.

RSA one-way function: $Enc(m, \mathrm{pk}) = m^e \mod n$
RSA assumption: On suppose qu'inverser la fonction est difficile.
(i.e., trouver une fonction $(c = m^e) \mapsto m$)

Textbook RSA (unsecure)

Soient deux premiers p, q. Calculer $n = pq, \varphi(n) = (p-1)(q-1)$.
Choisir $e < \varphi(n)$ premier avec $\varphi(n)$, calculer $d = e^{-1} \mod \varphi(n)$.
Alors il existe k tel que $ed = k\varphi(n) + 1$.
La clé publique est $\mathsf{pk} = (n, e)$, la clé secrète est $\mathsf{sk} = d$.

RSA one-way function: $Enc(m, \mathsf{pk}) = m^e \mod n$
RSA assumption: On suppose qu'inverser la fonction est difficile.
(i.e., trouver une fonction $(c = m^e) \mapsto m$)
RSA trapdoor: $Dec(c, \mathsf{sk}) = c^d \mod n$.
Cela fonctionne car $c = m^e$, on calcule
$(m^e)^d = m^{ed} = m^{k\varphi(n)+1} = m \mod n$, avec le théorème d'Euler.

Logarithme Discret

Soit G un groupe cyclique engendré par g d'ordre n :

$$G = \{1, g, g^2, \ldots, g^{n-1}\}$$

Logarithme discret

- $g, n, x \mapsto y = g^x \mod n$ facile (quadratique)
- $g, n, y = g^x \mod n \mapsto x$ difficile

Exemple :
$n = 7, g = 3$

$$3^5 \mod 7 \equiv 243 \mod 7 \equiv 5$$

ElGamal (1984)

Soit p un *grand* nombre premier sûr, et g un générateur du groupe $\mathbb{Z}_p^*$.

GenKey

- clé secrète $sk = x : x \in \{1, \ldots, p-1\}$
- clé publique $pk = (g, p, h) : h \equiv g^x \mod p$

Enc

Soit M un message et r un élément aléatoire de $\{1, \ldots, p-1\}$, on a :

$$C = (C_1, C_2) = (g^r \mod p, M \cdot h^r \mod p)$$

Dec

Soit C un chiffré, alors $M \equiv C_2 \cdot C_1^{-x} \mod p$

Rappel : Petit théorème de Fermat

Petit théorème de Fermat

Soit p premier et a entier. Alors $a^p \equiv a \mod p$.

Théorème d'Euler (généralisation)

Pour tout entier $n > 0$ et tout entier a premier avec n

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

où φ est la fonction indicatrice d'Euler.

Pour tout entier naturel n non nul, la fonction φ associe le nombre d'entiers compris entre 1 et n (inclus) et premiers avec n.
Par exemple, si p et q premiers alors $\varphi(p \times q) = (p-1)(q-1)$

Rivest Shamir Adelmann (1978)

Soit $n = pq$, p et q deux nombres premiers.

KeyGen

- Clé Publique : (e, n)
- Clé Secrète : d où $d = e^{-1} \mod \phi(n)$

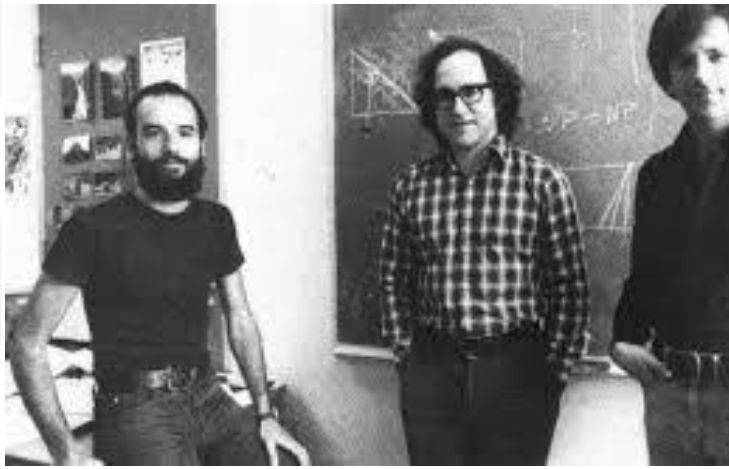
et $pgcd(e, \phi(n)) = 1$. Un choix populaire pour e est $e = 3, e = 2^{16} + 1$.

Enc

Soit $m \in (\mathbb{Z}/n\mathbb{Z})^*$, on a : $c = m^e \mod n$

Dec

Soit $c \in (\mathbb{Z}/n\mathbb{Z})^*$, on a : $m = c^d \mod n$



Square and Multiply

Soit p un nombre premier and g un entier. On veut calculer $g^d \pmod p$.

Nous prenons $d = d_{k-1}2^{k-1} + d_{k-2}2^{k-2} + ... + d_1 2 + d_0$.

Input g, d
Output $y = g^d$
$y := 1$
for $i = k - 1$ down to 0 **do**
  $y := y^2$
  if $d_i = 1$ **then** $y := y \times g$
end for
Return y

Optimal Asymmetric Encryption Padding : OAEP-RSA

Soit deux fonctions de hachage :

$$G : \{0,1\}^{k_0} \rightarrow \{0,1\}^{k-k_0}$$

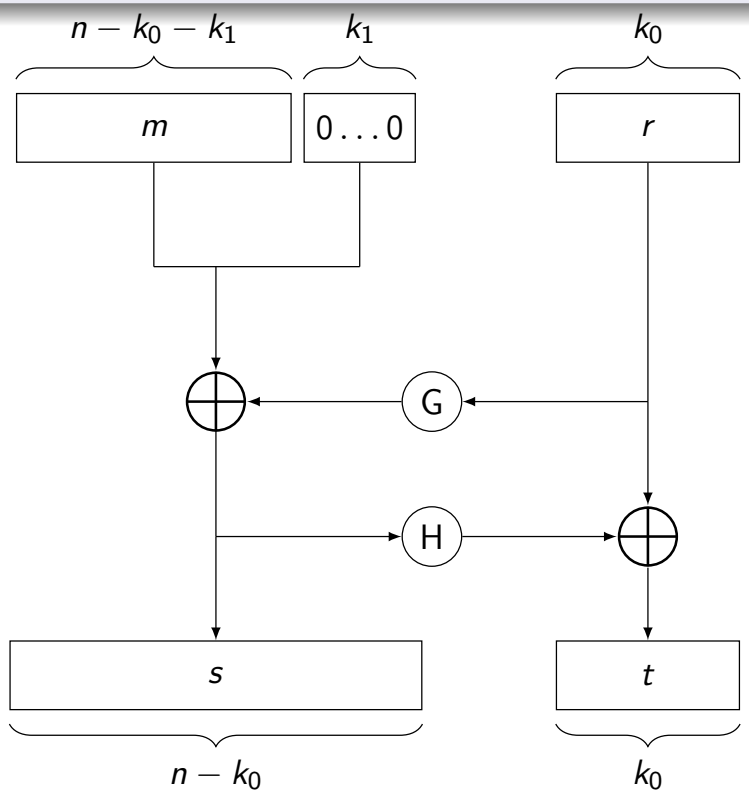
$$H : \{0,1\}^{k-k_0} \rightarrow \{0,1\}^{k_0}$$

Le chiffrement c de m avec la clef RSA pk est noté $RSA_{pk}(m)$
Le déchiffrement de c avec la clé privée sk est noté $RSA^{-1}_{pk}(m)$

OAEP: Chiffrement

$E_{pk}(m, r) = c = RSA_{pk}(s, t)$ avec $m \in \{0,1\}^n$, et $r \leftarrow \{0,1\}^{k_0}$

$s = (m||0^{k_1}) \oplus G(r), t = r \oplus H(s)$



OAEP: Déchiffrement

$D_{sk}(c)$

$$RSA_{sk}^{-1}(c) = (s, t)$$

$$r = t \oplus H(s)$$

$$M = s \oplus G(r)$$

Si $[M]_{k_1} = 0^{k_1}$, alors on renvoie $[M]^n$, sinon "Reject"

- $[M]_{k_1}$ dénote les k_1 dernier bits de M
- $[M]^n$ dénote les n premiers bits of M

Coût en calcul

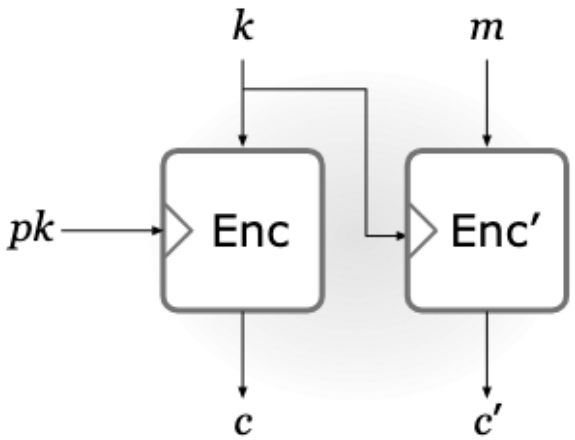
2 heures de video (avec 3Ghz CPU) :

	DVD 4,7 G.B		Blu-Ray 25 GB	
Schémas	chiffrer	déchiffrer	chiffrer	déchiffrer
RSA 2048	22 min	24 h	115 min	130 h
RSA 1024	21 min	10 h	111 min	53 h
AES CTR	20 sec	20 sec	105 sec	105 sec

Chiffrement Hybride (hybrid encryption)

Très repandue en pratique, l'idée est d'utiliser l'efficacité du symétrique en tandem avec l'expressivité de l'asymétrique.
Pour chiffrer un message m, il faut :

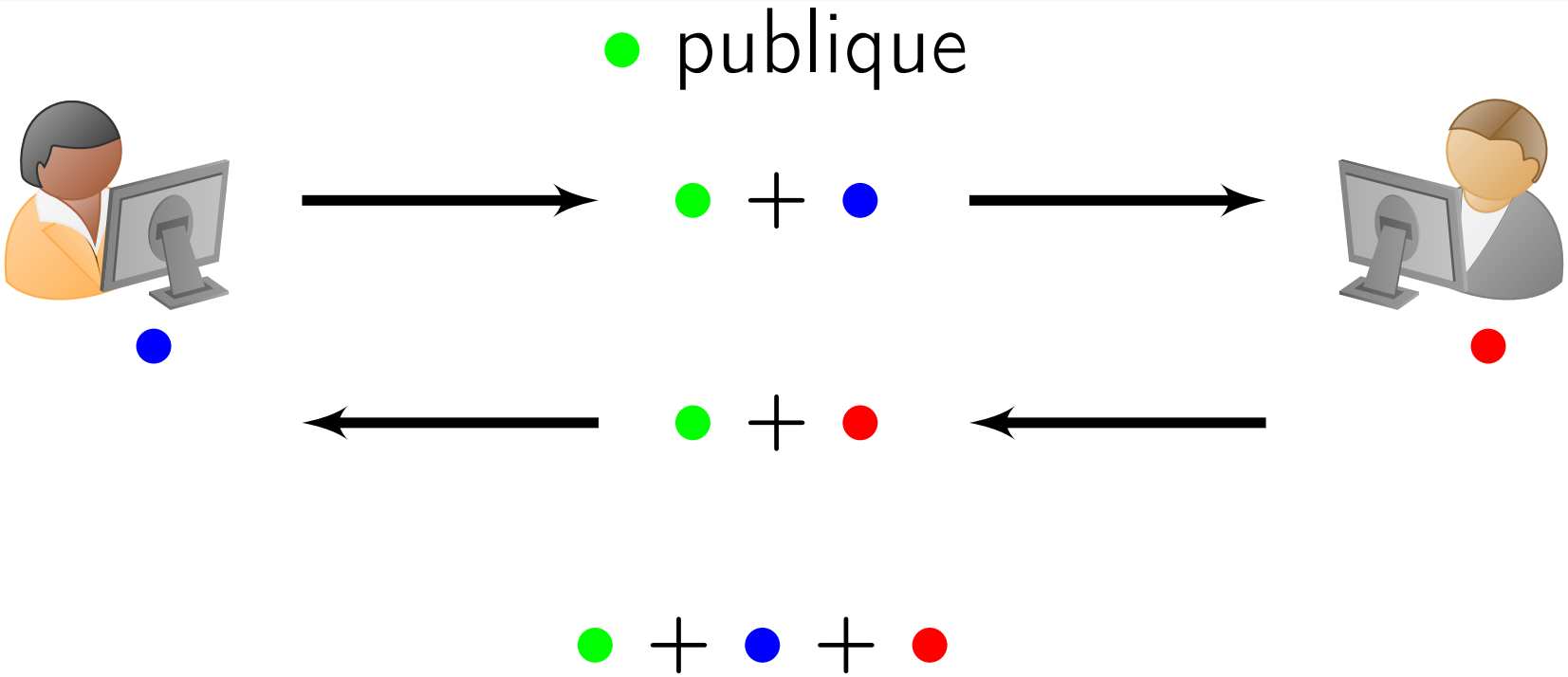
1. Choisir une clé aléatoire privée k et la chiffrer avec la clé publique du destinataire.
2. Chiffrer le message m avec la clé k avec un schéma symétrique.



[1]

[1] Livre : Jonathan Katz, Yehuda Lindell: Introduction to Modern Cryptography

Diffie-Hellman



- $g =$
- $a =$
- $b =$

$$(g^b)^a = g^{ab} = (g^a)^b \bmod n$$

Chiffrement Partiellement Homomorphe

$$E(m_1) \otimes E(m_2) \equiv E(m_1 \oplus m_2)$$

Applications

- Electronic voting
- Secure Fonction Evaluation
- Private Multi-Party Trust Computation
- Private Information Retrieval
- Private Searching
- Outsourcing of Computations (e.g., Secure Cloud Computing)
- Private Smart Metering and Smart Billing
- Privacy-Preserving Face Recognition
- ...

Exemple : textbook RSA

$$\mathcal{E}(m) = m^e \mod n$$

$$\begin{aligned} \mathcal{E}(m_1) \cdot \mathcal{E}(m_2) &= m_1^e m_2^e \mod n \\ &= (m_1 m_2)^e \mod n \\ &= \mathcal{E}(m_1 \cdot m_2) \end{aligned}$$

Montrer que le schéma d'ElGamal est homomorphe.

Fully Homomorphic Encryption

$$Enc(a,k) * Enc(b,k) = Enc(a * b, k)$$
$$Enc(a,k) + Enc(b,k) = Enc(a + b, k)$$
$$f(Enc(a,k), Enc(b,k)) = Enc(f(a,b), k)$$

Fully Homomorphic encryption

- Craig Gentry (STOC 2009) utilisant les lattices
- Marten van Dijk; Craig Gentry, Shai Halevi, et Vinod Vaikuntanathan utilisant des entiers.
- Craig Gentry; Shai Halevi. "A Working Implementation of Fully Homomorphic Encryption"

Idée : bootstraping (“déchiffrement dans les chiffrés”).

Autres chiffrements ...

- Attribute-Based
- Identity-Based
- Matchmaking
- Broadcast
- Functional
- ...

Fonctions de Hachages

Qu'est-ce que c'est?

Une fonction de hachage calcule une empreinte de n bits pour un message **M** de taille arbitraire.

- Cas d'usage : integrité des messages, stockage des mots de passe, signatures, crypto-monnaie
- Exemples: SHA-1 (160 bits), MD5 (128 bits), SHA-2, SHA-3...

Application: Integrité des messages

Pour détecter si un fichier a été modifié il suffit de recalculer son empreinte.

```
// Fichier code.c

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

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

$\Rightarrow$ Hash de 160 bits
SHA-1(code.c)=
A51F 07BB 62EC 44A3 F118

Les mots-de-passe

- Au lieu de stocker le mot-de-passe, on stocke son hash $h = H(\textit{password})$.
- Pour s'authentifier, l'utilisateur envoie son mot-de-passe *password*.
- Côté serveur, on vérifie si $H(\textit{password})$ est égal au h stocké dans la base.

Les mots-de-passe

- Au lieu de stocker le mot-de-passe, on stocke son hash $h = H(\textit{password})$.
- Pour s'authentifier, l'utilisateur envoie son mot-de-passe *password*.
- Côté serveur, on vérifie si $H(\textit{password})$ est égal au h stocké dans la base.

Sécurité

Si l'attaquant récupere la base de donées, les mots-de-passe sont "chiffrés": il a les valeurs des hashs, mais ne peut pas calculer les mot-de-passes.

Notions de sécurité

Une bonne fonction de hachage doit satisfaire les propriétés suivantes.

Résistance à la préimage

Étant donné x, il est difficile de trouver M t.q. $H(M) = x$.

Résistance aux collisions

Il est difficile de trouver M_1 et M_2 t.q. $H(M_1) = H(M_2)$.

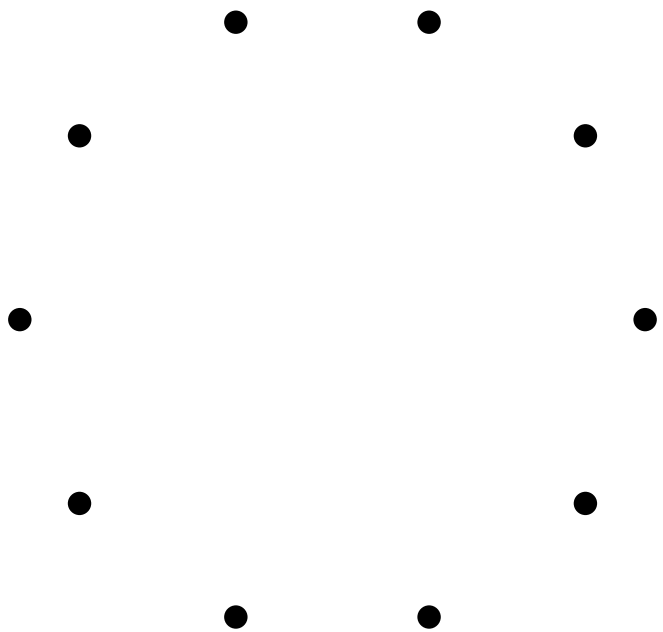
Résistance à la seconde préimage

Étant donné M_1, il est difficile de trouver M_2 t.q. $H(M_1) = H(M_2)$.

H résistante aux collisions $\Rightarrow$ H résistante à la seconde préimage

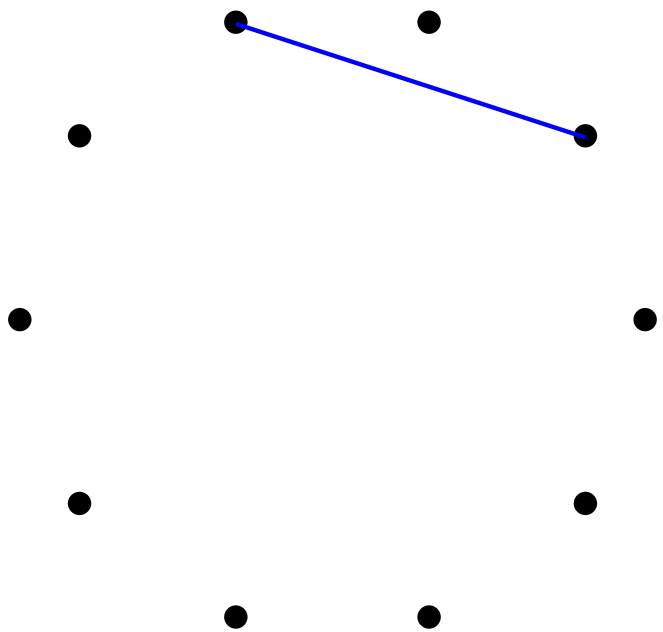
Le paradoxe des anniversaires

Combien faut-il réunir de personnes pour que deux anniversaires tombent le même jour?



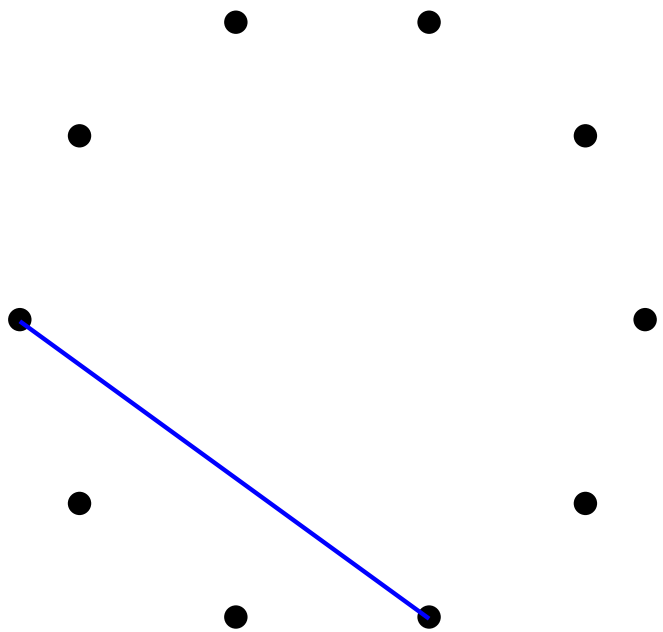
Le paradoxe des anniversaires

Combien faut-il réunir de personnes pour que deux anniversaires tombent le même jour?



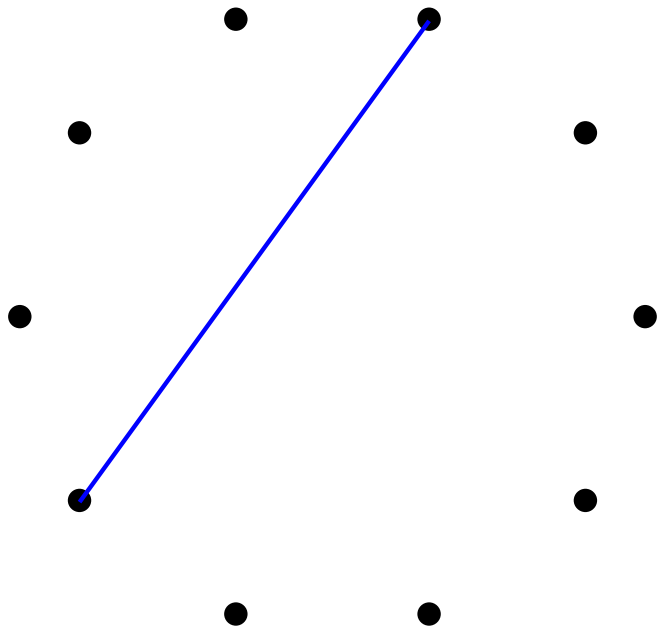
Le paradoxe des anniversaires

Combien faut-il réunir de personnes pour que deux anniversaires tombent le même jour?



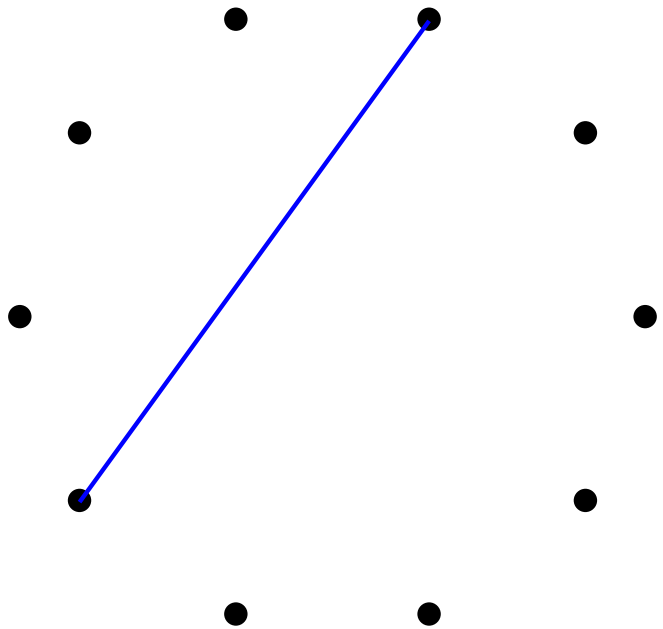
Le paradoxe des anniversaires

Combien faut-il réunir de personnes pour que deux anniversaires tombent le même jour?



Le paradoxe des anniversaires

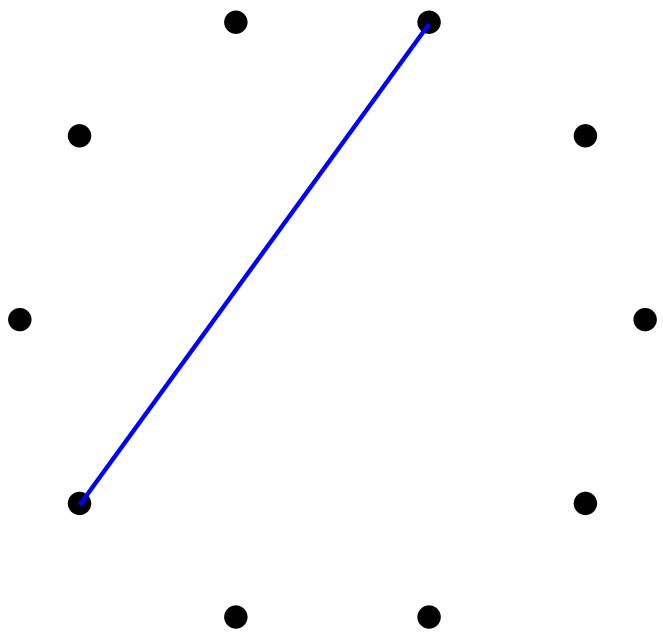
Combien faut-il réunir de personnes pour que deux anniversaires tombent le même jour?



Combien de possibilités?

Le paradoxe des anniversaires

Combien faut-il réunir de personnes pour que deux anniversaires tombent le même jour?



Combien de possibilités? $\sum_{i=1}^{N-1} i = \frac{N(N-1)}{2}$

Le paradoxe des anniversaires

Theorem

Si on choisit k balles parmi N balles de manière aléatoire et independante, alors si $k \geq 1.2\sqrt{N}$, la probabilité que deux soient identiques est $> \frac{1}{2}$.

Exemple : Si $N = 365$, alors la probabilité que parmi k personnes se trouvant dans une salle à un moment donné il y ait 2 avec le même jour d'anniversaire est $> \frac{1}{2}$ si $k > 23$.

Attaque par collision

SHA-1 est une fonction de hachage qui produit des sorties de 160 bits.

En exploitant le paradoxe des anniversaires, quelle est la complexité d'un calcul de collision sur SHA-1?

Attaque par collision

SHA-1 est une fonction de hachage qui produit des sorties de 160 bits.

En exploitant le paradoxe des anniversaires, quelle est la complexité d'un calcul de collision sur SHA-1?

2^{80} évaluations

Attaque par collision

SHA-1 est une fonction de hachage qui produit des sorties de 160 bits.
En exploitant le paradoxe des anniversaires, quelle est la complexité d'un calcul de collision sur SHA-1?

2^{80} évaluations

En réalité, depuis 2017, SHA-1 est cassée : une collision a été trouvée en 2^{63} opérations.

`https://shattered.io`

Chronologie de la famille SHA

1993 SHA-0 proposée par le NIST (empreinte de 160 bits)

Chronologie de la famille SHA

1993 SHA-0 proposée par le NIST (empreinte de 160 bits)

1995 SHA-0 est retirée par le gouvernement Américain remplacée par une variante SHA-1.

Chronologie de la famille SHA

1993 SHA-0 proposée par le NIST (empreinte de 160 bits)

1995 SHA-0 est retirée par le gouvernement Américain remplacée par une variante SHA-1.

2000 Nouvelles version proposées SHA-2 (SHA-256, 384, 512)

Chronologie de la famille SHA

1993 SHA-0 proposée par le NIST (empreinte de 160 bits)

1995 SHA-0 est retirée par le gouvernement Américain remplacée par une variante SHA-1.

2000 Nouvelles version proposées SHA-2 (SHA-256, 384, 512)

2004 Collision trouvée sur MD5

2005 Attaque théorique sur SHA-1: collisions en 2^{69} opérations

Chronologie de la famille SHA

1993 SHA-0 proposée par le NIST (empreinte de 160 bits)

1995 SHA-0 est retirée par le gouvernement Américain remplacée par une variante SHA-1.

2000 Nouvelles version proposées SHA-2 (SHA-256, 384, 512)

2004 Collision trouvée sur MD5

2005 Attaque théorique sur SHA-1: collisions en 2^{69} opérations

2006 Le NIST lance la compétition SHA-3

Chronologie de la famille SHA

1993 SHA-0 proposée par le NIST (empreinte de 160 bits)

1995 SHA-0 est retirée par le gouvernement Américain remplacée par une variante SHA-1.

2000 Nouvelles version proposées SHA-2 (SHA-256, 384, 512)

2004 Collision trouvée sur MD5

2005 Attaque théorique sur SHA-1: collisions en 2^{69} opérations

2006 Le NIST lance la compétition SHA-3

2012 Keccak remporte la compétition SHA-3

Chronologie de la famille SHA

1993 SHA-0 proposée par le NIST (empreinte de 160 bits)

1995 SHA-0 est retirée par le gouvernement Américain remplacée par une variante SHA-1.

2000 Nouvelles version proposées SHA-2 (SHA-256, 384, 512)

2004 Collision trouvée sur MD5

2005 Attaque théorique sur SHA-1: collisions en 2^{69} opérations

2006 Le NIST lance la compétition SHA-3

2012 Keccak remporte la compétition SHA-3

2015 Nouvelle attaque théorique contre SHA-1 (2^{63} opérations)

Chronologie de la famille SHA

1993 SHA-0 proposée par le NIST (empreinte de 160 bits)

1995 SHA-0 est retirée par le gouvernement Américain remplacée par une variante SHA-1.

2000 Nouvelles version proposées SHA-2 (SHA-256, 384, 512)

2004 Collision trouvée sur MD5

2005 Attaque théorique sur SHA-1: collisions en 2^{69} opérations

2006 Le NIST lance la compétition SHA-3

2012 Keccak remporte la compétition SHA-3

2015 Nouvelle attaque théorique contre SHA-1 (2^{63} opérations)

2017 Collision trouvée contre SHA-1 en pratique

La famille SHA

- (SHA-0,) SHA-1 et SHA-2 reposent toutes sur le même type de construction: la construction de Merkle Damgård.
- C'est également sur ce type de construction que reposaient MD4 et MD5
- SHA-3 repose sur une construction bien différente: celui des fonctions éponges

Fonctions pseudo-aléatoires booléennes

- **Fonction de Compression**

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

Fonctions pseudo-aléatoires booléennes

- **Fonction de Compression**

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

- **Fonction d'Expension**

$$f : \{0,1\}^n \to \{0,1\}^{n+m} \quad \text{injective}$$

Fonctions pseudo-aléatoires booléennes

- **Fonction de Compression**

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

- **Fonction d'Expension**

$$f : \{0,1\}^n \to \{0,1\}^{n+m} \quad \text{injective}$$

- **Permutation pseudo-aléatoire**

$$f : \{0,1\}^n \to \{0,1\}^n \quad \text{bijective}$$

f est bijective.

Fonctions pseudo-aléatoires booléennes

- **Fonction de Compression**

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

- **Fonction d'Expension**

$$f : \{0,1\}^n \to \{0,1\}^{n+m} \quad \text{injective}$$

- **Permutation pseudo-aléatoire**

$$f : \{0,1\}^n \to \{0,1\}^n \quad \text{bijective}$$

f est bijective.

Permutation pseudo-aléatoire $\neq$ Permutation dans le cadre des SPN

Fonction de compression

Fonction de hachage: H prend des *entrées* de taille variable ressort une *sortie* de *taille fixée* n.

$$H : \{0,1\}^* \to \{0,1\}^n$$
$$H : M \mapsto H(M)$$

Fonction de compression: f prend des *entrées* de *taille fixée* $n+m$ ressort une *sortie* de taille fixée n.

$$f : \{0,1\}^{n+m} \to \{0,1\}^n$$
$$f : x \mapsto f(x)$$

Exemple

$$f : \{0,1\}^{n+m} \to \{0,1\}^n$$
$$f : x \mapsto LSB_n(x)$$

f est une fonction de compression.
Est-ce que f est sécurisée?

Exemple

$$f : \{0,1\}^{n+m} \to \{0,1\}^n$$
$$f : x \mapsto LSB_n(x)$$

f est une fonction de compression.
Est-ce que f est sécurisée?

NON!

Exemple

$$f : \{0,1\}^{n+m} \to \{0,1\}^n$$
$$f : x \mapsto LSB_n(x)$$

f est une fonction de compression.
Est-ce que f est sécurisée?

NON!

- Collision: $x = (x_1|x_2)$, $x' = (x_1|x_2')$ $\to f(x) = f(x') = x_1$

Exemple

$$f : \{0,1\}^{n+m} \to \{0,1\}^n$$
$$f : x \mapsto LSB_n(x)$$

f est une fonction de compression.
Est-ce que f est sécurisée?

NON!

- Collision: $x = (x_1|x_2)$, $x' = (x_1|x_2')$ $\to f(x) = f(x') = x_1$
- Pré-image: $x = (y|0)$ est solution de $f(x) = y$

Exemple

$$f : \{0,1\}^{n+m} \to \{0,1\}^n$$
$$f : x \mapsto LSB_n(x)$$

f est une fonction de compression.
Est-ce que f est sécurisée?

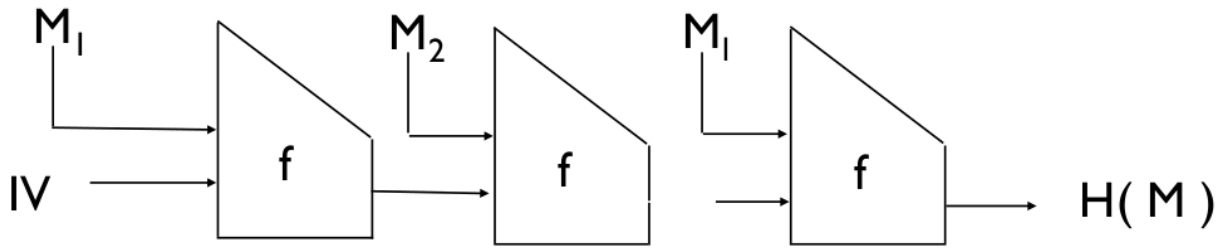
NON!

- Collision: $x = (x_1|x_2)$, $x' = (x_1|x_2') \to f(x) = f(x') = x_1$
- Pré-image: $x = (y|0)$ est solution de $f(x) = y$
- Seconde Préimage: Soit $x = (x_1|x_2)$ tel que $f(x) = x_1$, alors $x' = (x_1|x_2')$ est une seconde préimage de x_1.

La construction de Merkle-Damgård

- Soit $f : \{0,1\}^{m+n} \to \{0,1\}^n$, une fonction de compression.
- Soit $M = M_1||M_2 \ldots ||M_\ell$ le message au quel on doit appliquer H (ℓ blocs de m bits). Le dernier bloc est éventuellement complété avec des zéros.

$$h_1 = f(IV, M_1), H_2 = f(h_1, M_2), \ldots H_\ell = f(H_{\ell-1}, M_l)$$



SHA-1

- Proposée par le NIST en 1995
- Repose sur la construction de Merkle-Damgård
- La fonction de compression f prend en entrée des blocs de 512+160 bits et retourne une sortie de 160 bit.
- La fonction f est une variante d'un schéma de Feistel à 80 tours.
- Depuis 2017, ne doit plus être utilisée à des fins cryptographique.
- La variante SHA-2 est toujours considérée comme sûre.

SHA-3

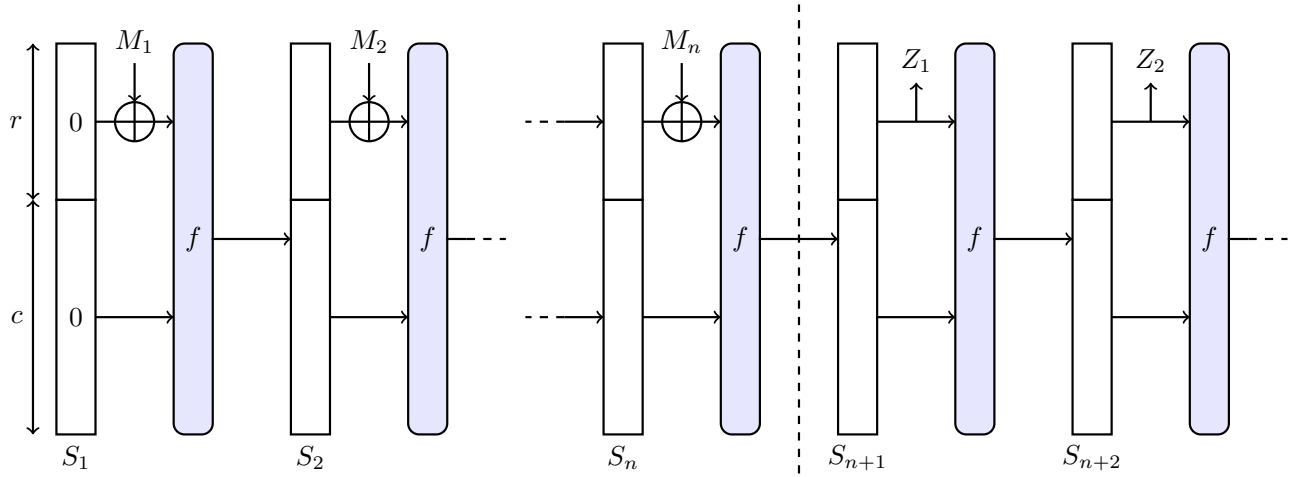
- Compétition organisée par le NIST en 2006
- 64 participants.
- Le vainqueur: Keccak (Bertoni, Daemen, Peters et Van Assche)
- Keccak ne repose pas sur le principe de Merkle-Damgård mais sur celui des fonctions éponges
- Retourne des empreintes de taille 224, 256, 384 ou 512.
- En pratique, SHA-3 est moins utilisée que SHA-2 car plus lente.

Fonction Éponge

- On applique un padding qui transforme l'entrée en blocs de r bits.
  - Dans le cas de SHA-3 le padding est 10...01.
- Soit une permutation pseudo-aléatoire f sur $r + c$ bits.
- L'algorithme consiste en deux étapes: l'absorption et l'essorage

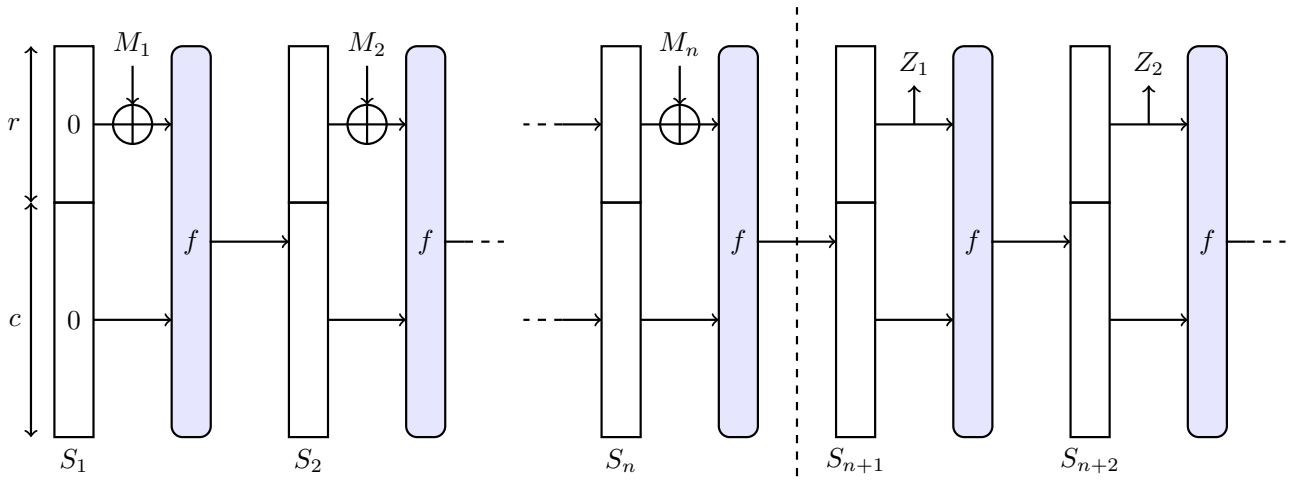
Fonction Éponge

Calcul d'une empreinte de $M = M_1|\dots|M_n$. Les M_i font r bits.



- État interne $S_1 = (R_1|C_1)$, initialisé à 0.
- Absorption: Pendant n étapes: $S_i = f(R_{i-1} \oplus M_i|C_i)$.
- Essorage: Étapes $i + n$: retourner $Z_i = R_{i+n}$.
  Calculer $S_{n+i+1} = f(R_{n+i}|C_{n+i})$.

Fonction Éponge



Le nombre d'étape de l'essorage est à déterminer selon l'application. Dans le cas de SHA-3 c'est 1. La sortie est ensuite tronquée pour obtenir la bonne taille d'empreintes.

Ce genre de construction peut servir pour d'autres primitives cryptographiques, par exemple pour le chiffrement par flot.

Vrai ou Faux?

- On peut inverser une fonction de hachage, si et seulement si on connaît la clef secrète.

Vrai ou Faux?

- On ne peut pas inverser une fonction de hachage!

Vrai ou Faux?

- On ne peut pas inverser une fonction de hachage!
- Une fonction de compression prend des entrées de taille variables et retourne des sorties de taille fixe n.

Vrai ou Faux?

- On ne peut pas inverser une fonction de hachage!
- Une fonction de compression prend des entrées de taille fixe $n + r$ et retourne des sorties de taille fixe n.

Vrai ou Faux?

- On ne peut pas inverser une fonction de hachage!
- Une fonction de compression prend des entrées de taille fixe $n + r$ et retourne des sorties de taille fixe n.
- Dans une fonction éponge, la fonction f qui modifie l'état interne est une permutation pseudo-aléatoire.

Vrai ou Faux?

- On ne peut pas inverser une fonction de hachage!
- Une fonction de compression prend des entrées de taille fixe $n + r$ et retourne des sorties de taille fixe n.
- Dans une fonction éponge, la fonction f qui modifie l'état interne est une permutation pseudo-aléatoire.
- Dans une construction de Merkle Damgård, la fonction f qui modifie l'état interne est une fonction d'expension.

Vrai ou Faux?

- On ne peut pas inverser une fonction de hachage!
- Une fonction de compression prend des entrées de taille fixe $n + r$ et retourne des sorties de taille fixe n.
- Dans une fonction éponge, la fonction f qui modifie l'état interne est une permutation pseudo-aléatoire.
- Dans une construction de Merkle Damgård, la fonction f qui modifie l'état interne est une fonction de compression.

Vrai ou Faux?

- On ne peut pas inverser une fonction de hachage!
- Une fonction de compression prend des entrées de taille fixe $n + r$ et retourne des sorties de taille fixe n.
- Dans une fonction éponge, la fonction f qui modifie l'état interne est une permutation pseudo-aléatoire.
- Dans une construction de Merkle Damgård, la fonction f qui modifie l'état interne est une fonction de compression.
- Étant donné un message x, si on peut trouver un message y tel que $h(x) = h(y)$, alors h n'est pas résistante aux collisions.

Vrai ou Faux?

- On ne peut pas inverser une fonction de hachage!
- Une fonction de compression prend des entrées de taille fixe $n + r$ et retourne des sorties de taille fixe n.
- Dans une fonction éponge, la fonction f qui modifie l'état interne est une permutation pseudo-aléatoire.
- Dans une construction de Merkle Damgård, la fonction f qui modifie l'état interne est une fonction de compression.
- Étant donné un message x, si on peut trouver un message y tel que $h(x) = h(y)$, alors h n'est pas résistante aux collisions.
- Étant donné un message x, si on peut trouver un message y tel que $h(x) = h(y)$, alors h n'est pas résistante aux secondes préimages.