

Outils Cryptographiques

Mathieu Cunche¹

¹INSA-Lyon

1 Introduction

2 Chiffrement par bloc

3 Fonctions de hachage cryptographiques

4 Cryptographie asymétrique

1 Introduction

2 Chiffrement par bloc

3 Fonctions de hachage cryptographiques

4 Cryptographie asymétrique

Blockcipher (définition générale)

- Un *Blockcipher* est défini par deux fonctions :
 - Une fonction de chiffrement E et une fonction de déchiffrement D
 - E permet de calculer le chiffré C à partir du clair P et de la clef K
 - D permet de calculer le clair P à partir du chiffré C et de la clef K

$$\boxed{\begin{array}{ccc} \{0,1\}^k \times \{0,1\}^n & \rightarrow & \{0,1\}^n \\ (K, P) & \rightarrow & C = E(K, P) \end{array}}$$

$$\boxed{\begin{array}{ccc} \{0,1\}^k \times \{0,1\}^n & \rightarrow & \{0,1\}^n \\ (K, P) & \rightarrow & M = D(K, P) \end{array}}$$

avec n et k fixés



- Telles que $\forall (K, M) \in \{0,1\}^k \times \{0,1\}^n, D(K, E(K, M)) = M$
 - Noté également : $D_K(E_K(M)) = M$

Blockcipher : propriétés de sécurité

Permutation pseudoaléatoire

Un *bon blockcipher* doit être une permutation pseudoaléatoire. Si K est inconnue, un attaquant ne doit pas être capable de prédire la sortie de E (ou D) à partir de l'entrée.

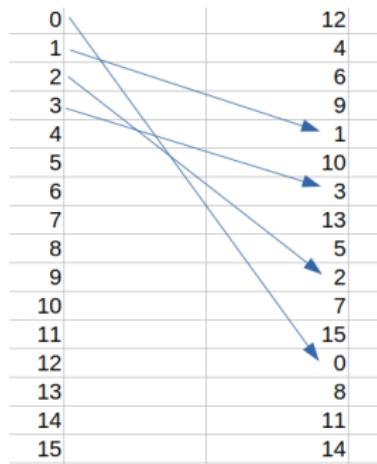


Figure: Permutation des valeurs sur un octet

Blockcipher historique : DES

FIPS PUB 46

DES : Data Encryption Standard

- Standardisé par le FIPS en 1977
- Taille de clef : 56 bits
- Taille de bloc : 64 bits
- Obsolète : ne pas utiliser !

FIPS PUB 46



Federal Information
Processing Standards Publication 46
1977 January 15
SPECIFICATIONS FOR THE
DATA ENCRYPTION STANDARD

The Data Encryption Standard (DES) shall consist of the following Data Encryption Algorithm to be implemented in special purpose electronic devices. These devices shall be designed in such a way that they may be used in a computer system or network to provide cryptographic protection to binary coded data. The method of implementation will depend on the application and environment. The devices shall be implemented in such a way that they may be tested and validated as accurately performing the transformations specified in the following algorithm.

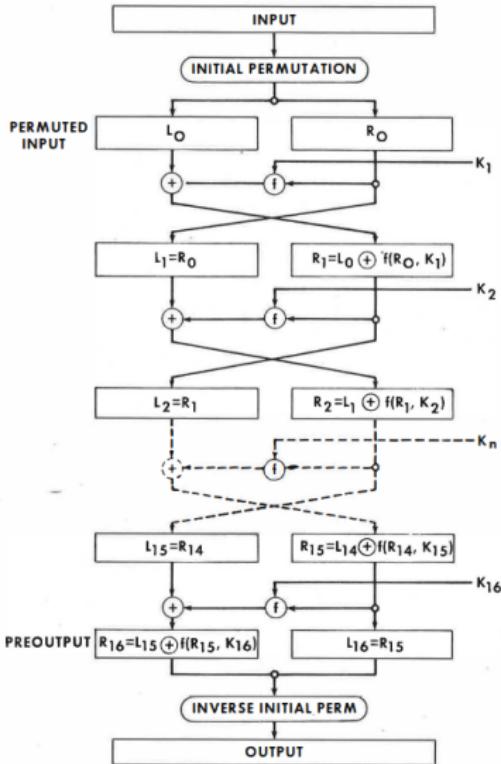


FIGURE 1. Enciphering computation.

Advanced Encryption Standard

AES : Advanced Encryption Standard

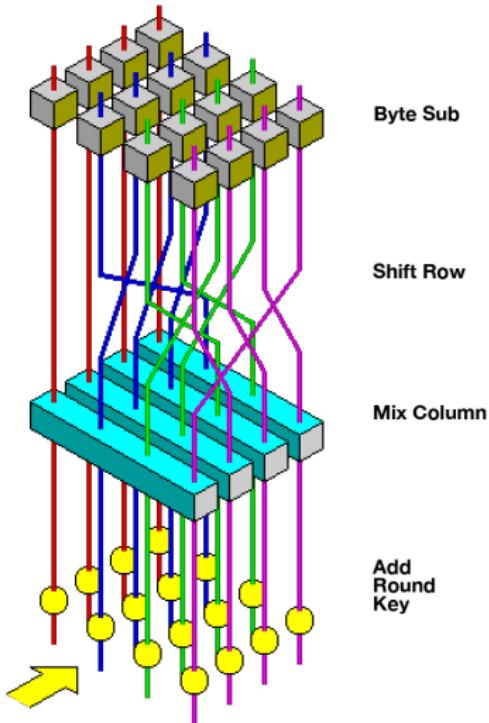
- Standardisé par le NIST en 2001
- Taille de clef : 128 bits (ou 192, ou 256)
- Taille de bloc : 128 bits
- Pas d'attaque connue
- Le standard actuel !

Federal Information
Processing Standards Publication 197

November 26, 2001

Specification for the

ADVANCED ENCRYPTION STANDARD (AES)

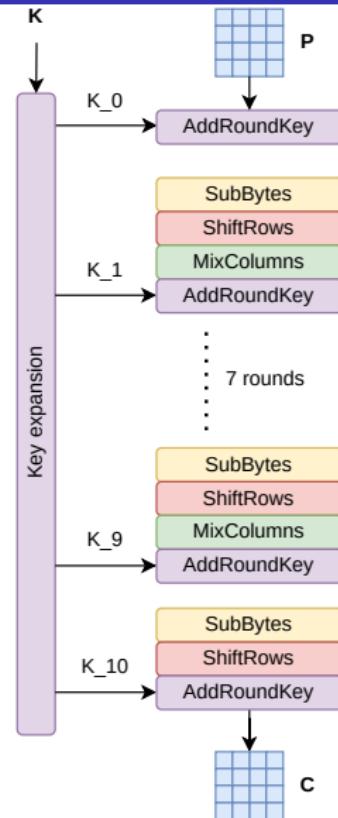


a

^aSource : Wikipedia

AES : fonctionnement interne

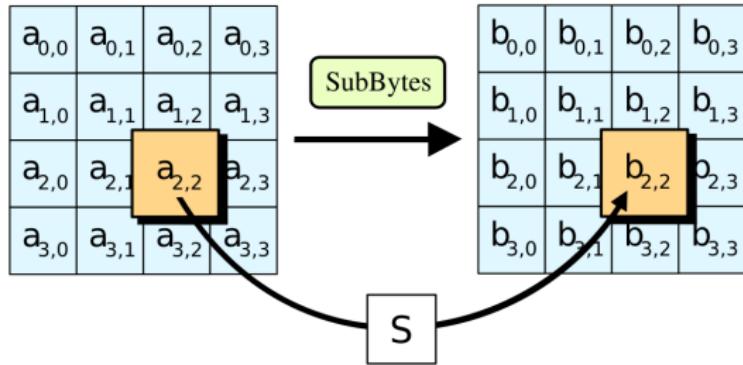
- Données traitées sous forme d'un tableau de 4×4 octets
- Transformations via 10 rondes successives (versions 128 bits)
- Une ronde : 4 opérations successives
 - **SubBytes** : substitution
 - **ShiftRows** : permutation
 - **MixColumns** : transformation linéaire
 - **AddRoundKey** : XOR avec la clef de ronde



Animation : <https://legacy.crypto.org/en/cto/aes-animation>

AES : SubBytes

Substitution de la valeur des octets

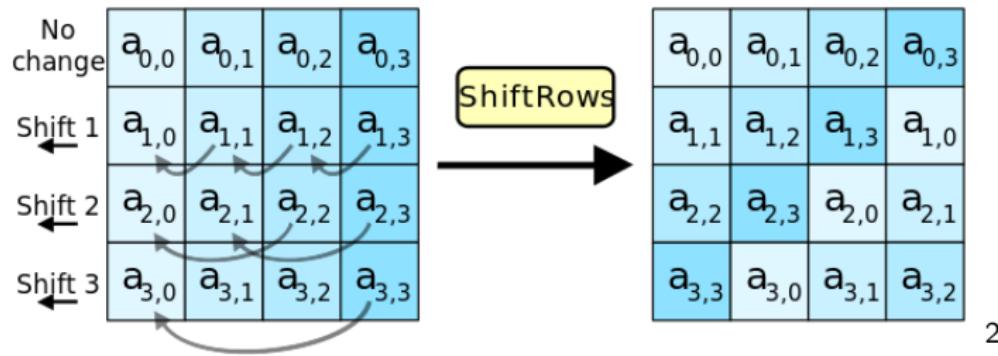


AES S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73

AES : ShiftRows

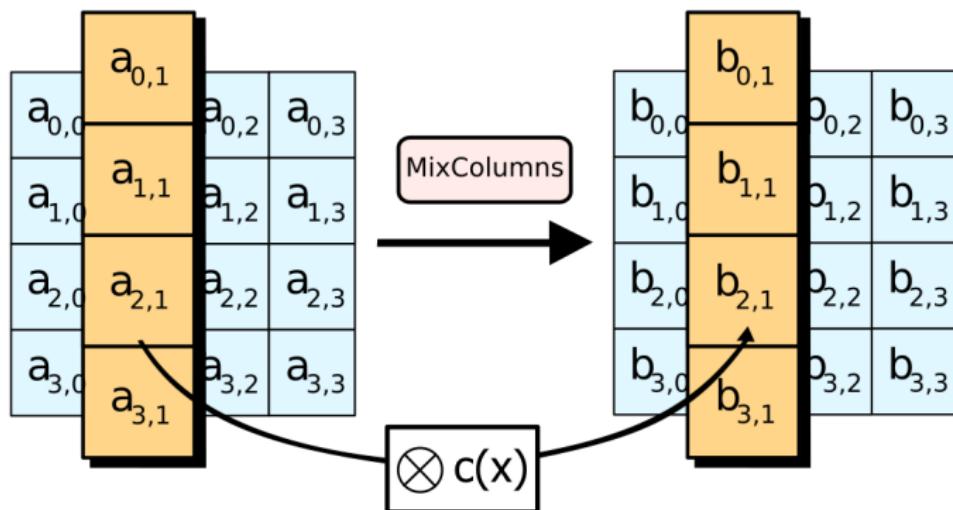
ShiftRows : décallage des lignes



²Source : Wikipedia

AES : MixColumns

MixColumns : Transformation linéaire (Mult. Matrice-Vecteur)

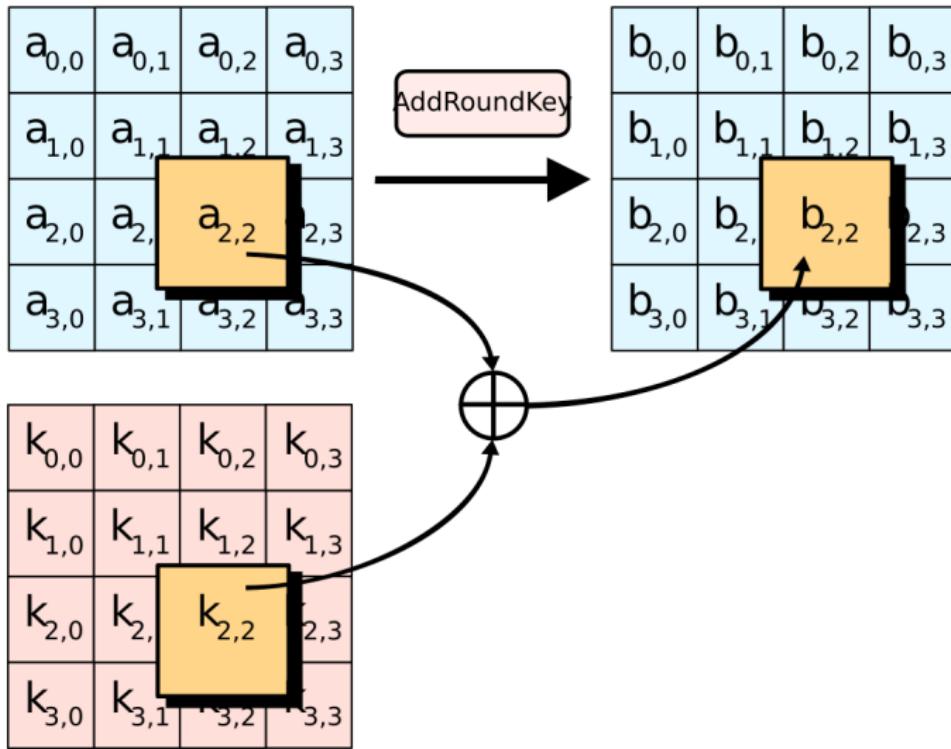


3

³Source : Wikipedia

AES : AddRoundKey

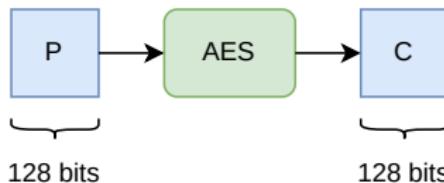
AddRoundKey : XoR avec la clef de ronde



4

⁴Source : Wikipedia

Blockcipher : modes opératoires

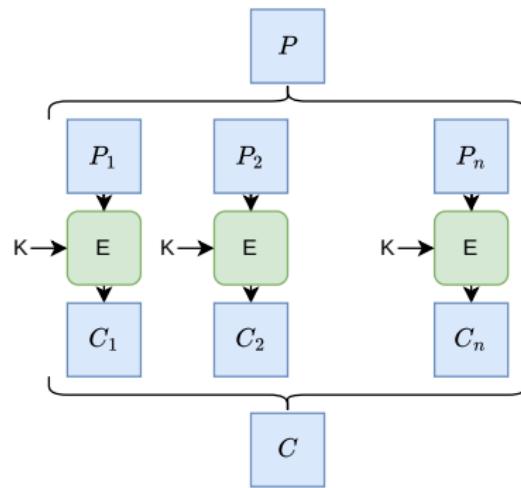


- Un *Blockcipher* a une taille de bloc fixe (128 bits pour AES)
 - Comment faire pour chiffrer des données plus grandes que la taille du bloc ?
- ⇒ La question des **modes opératoires** des *Blockcipher* ...

Blockcipher : mode ECB

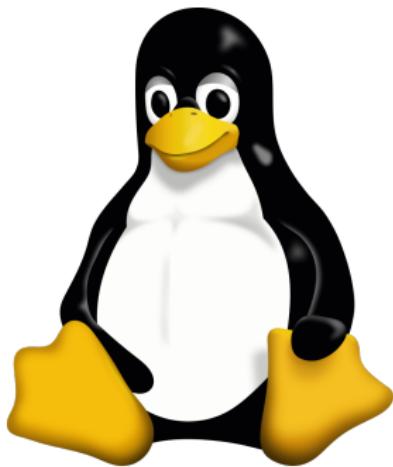
Le Mode ECB : **Electronic CodeBook**

- Donnée P découpée en blocs P_1, P_2, \dots, P_n
- Chaque bloc est chiffré indépendamment avec la clef K



Blockcipher : mode ECB II

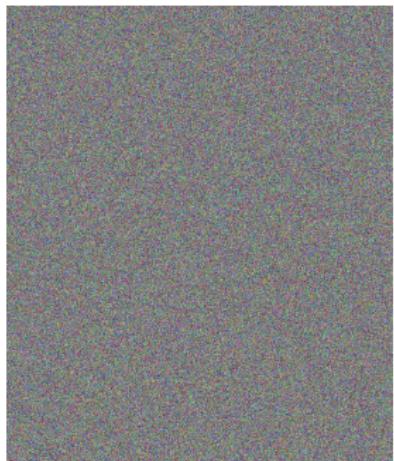
Problème : le mode ECB conserve la structure



(a) Tux



(b) Tux chiffré en mode
ECB

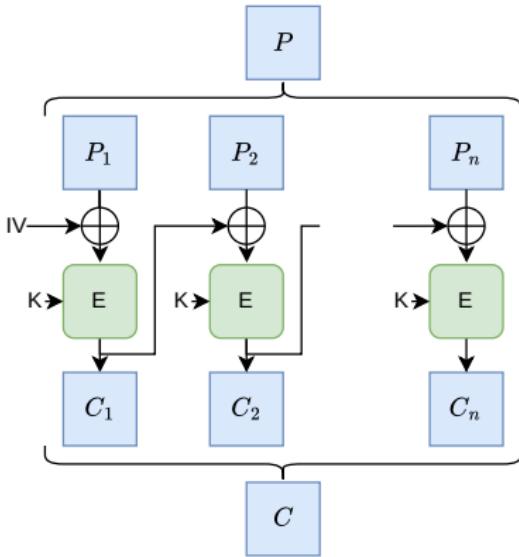


(c) Tux chiffré en mode
CTR

Blockcipher : mode CBC

Mode CBC : Cipher Block Chaining

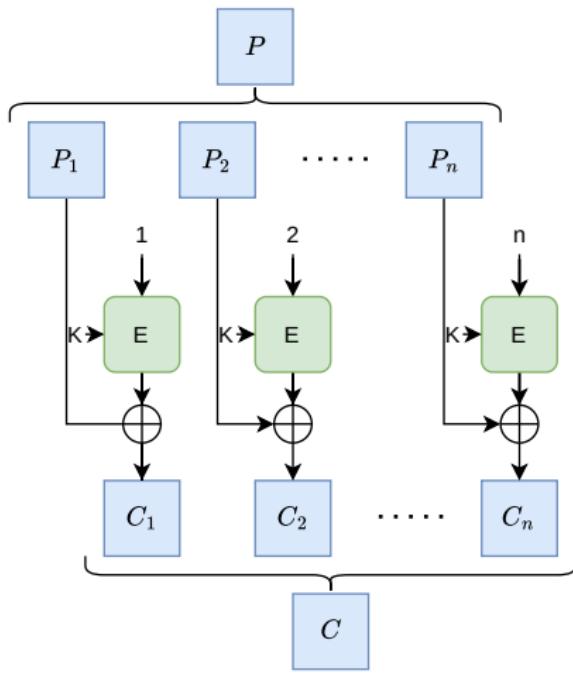
- Chaînage du chiffrement des blocs
- Le chiffré d'un bloc dépend du chiffré du bloc précédent



Blockcipher : mode CTR

Mode CTR : Counter

- Un compteur permet de chiffrer chaque bloc de manière différente et indépendante
- Le chiffré d'un bloc dépend de son compteur (sa position)



1 Introduction

2 Chiffrement par bloc

3 Fonctions de hachage cryptographiques

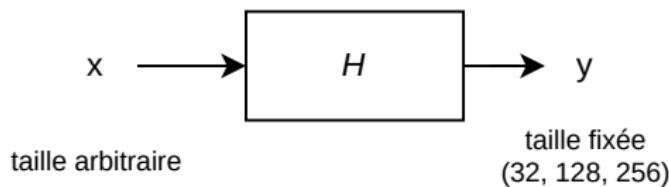
4 Cryptographie asymétrique

Fonctions de hachage (définition générale)

- Fonction de hachage H :

$$\begin{array}{ccc} \{0,1\}^n & \rightarrow & \{0,1\}^s \\ x & \rightarrow & H(x) \end{array}$$

- avec n arbitrairement grand et s fixé (ex. : 32, 128, 256 ...)



- Usage général (non cryptographique) des fonctions de hachage :
 - contrôle d'intégrité (*checksum CRC*)
 - tables de hachage

Fonctions de hachage cryptographiques

- Les fonction de hachage cryptographiques
 - Des fonctions de hachage
 - Entrée : message m de taille arbitraire en entrée
 - Sortie : haché h de taille fixée
 - $$\begin{array}{ccc} \{0,1\}^n & \rightarrow & \{0,1\}^s \\ m & \rightarrow & H(m) = h \end{array}$$
 - Avec en plus des propriétés de sécurité
 - Au coeur de nombreux schémas cryptographiques (ex. signature)
 - Utilisées pour garantir l'**intégrité** d'un message
- Exemples de fonction de hachage cryptographique
 - Secure Hash Algorithm : SHA-1, SHA-2, SHA-3 ...
 - Message Digest 5 : MD5

Hachage : propriétés de sécurité

Une fonction de hachage cryptographique sert généralement à vérifier l'**intégrité** d'un message. C'est à dire qu'elle va permettre de détecter une modification du message (même la plus petite).

Propriétés de sécurité d'une fonction de hachage cryptographique :

- Resistance aux pré-images (1ere et 2nde)
- Resistance aux collisions
- Bonus : Imprédictibilité



Hachage : imprédicibilité

Imprédictibilité

La sortie d'une fonction de hachage cryptographique est imprédicible, elle est (pseudo)-aléatoire

Effet avalanche

Même une petite modification de la valeur d'entrée doit résulter dans une modification significative de la valeur de sortie. La modification d'un bit en entrée doit résulter dans la modification de 50% des bits de sortie en moyenne.

Exemple :

- sha-256(**a**bcdef) = 8094a029b5b48ec0fac156b27490cc30bcb7f3aefb10b8d3dfb87c36decad70e
- sha-256(**b**bcdef) = 51981e2dadd154aeacd4bc28a2f353abe49bb4440c78aadbbb9ef01c1271377

Hachage : resistance aux préimages

Résistance à la 1ere préimage

Etant donné un haché h , il doit être *difficile* de trouver un m tel que $H(m) = h$.

- Les fonctions de hachage cryptographiques (FHC) sont parfois appelées des fonctions à sens unique (*One-way functions*)
 - Il est facile d'aller dans un sens : étant donné m , calculer $h = H(m)$
 - mais il est **impossible** difficile d'aller dans l'autre sens : étant donné h , calculer m tel que $H(m) = h$
- Difficile comment ?
 - Il ne doit pas exister de méthode de recherche de préimage plus efficace que la recherche par *force brute*

Hachage : recherche d'une 1ere pré-image

Recherche de 1ere préimage : étant donné h trouver un m tel que $H(m) = h$.

- La fonction de hachage : $H(.) = SHA - 1(.)$ (taille de la sortie n = 160 bits)
- La valeur du haché

$h = d78f8bb992a56a597f6c7a1fb918bb78271367eb$

Comment faire en pratique pour retrouver m ?

Hachage : recherche d'une 1ere pré-image

Recherche de 1ere préimage : étant donné h trouver un m tel que $H(m) = h$.

- La fonction de hachage : $H(.) = SHA - 1(.)$ (taille de la sortie $n = 160$ bits)
- La valeur du haché

$h = d78f8bb992a56a597f6c7a1fb918bb78271367eb$

Comment faire en pratique pour retrouver m ?

- Tester toutes les valeurs en partant de 0 jusqu'à trouver un m / $H(m) = h$ (force brute)

Hachage : recherche d'une 1ere pré-image

Recherche de 1ere préimage : étant donné h trouver un m tel que $H(m) = h$.

- La fonction de hachage : $H(.) = SHA - 1(.)$ (taille de la sortie $n = 160$ bits)
- La valeur du haché

$$h = d78f8bb992a56a597f6c7a1fb918bb78271367eb$$

Comment faire en pratique pour retrouver m ?

- Tester toutes les valeurs en partant de 0 jusqu'à trouver un m / $H(m) = h$ (force brute)
- Nombre moyen de valeurs à tester : $2^n = 2^{160}$
 - A titre de comparaison : nbr atomes dans l'univers = 2^{286}

Résistance à la 2nde préimage

Etant donné un message m , il doit être *difficile* de trouver un m' tel que $H(m) = H(m')$.

- Notion plus faible que la résistance à la première préimage
 - Le problème de la 2nde préimage peut être réduit à celui de la 1ere préimage
- Toujours la même notion de difficulté
 - Il ne doit pas être possible de faire mieux que la force brute

Hachage : resistance aux collisions

Résistance aux collisions

Il doit être *difficile* de trouver deux messages m et m' tels que $m \neq m'$ et $H(m) = H(m')$.

- Notion plus faible que la résistance à la 2nde et 1ere préimage
 - Le problème de la collision peut être réduit à celui de la 2nde préimage (et donc de la 1ere)
- Toujours la même notion de difficulté
 - Il ne doit pas être possible de faire mieux que la force brute

Hachage : recherche de collision

Recherche de collision : trouver m et m' tel que $H(m) = H(m')$ ($m \neq m'$).

- La fonction de hachage : $H(.) = MD5(.)$ (taille de la sortie $n = 160$ bits)

Comment faire en pratique pour retrouver m et m' ?

Hachage : recherche de collision

Recherche de collision : trouver m et m' tel que $H(m) = H(m')$ ($m \neq m'$).

- La fonction de hachage : $H(.) = MD5(.)$ (taille de la sortie $n = 160$ bits)

Comment faire en pratique pour retrouver m et m' ?

- Idée 1 : fixer m et tester toutes les valeurs en partant de 0 jusqu'à trouver un m' / $H(m) = H(m')$ (force brute)

Hachage : recherche de collision

Recherche de collision : trouver m et m' tel que $H(m) = H(m')$ ($m \neq m'$).

- La fonction de hachage : $H(.) = MD5(.)$ (taille de la sortie $n = 160$ bits)

Comment faire en pratique pour retrouver m et m' ?

- Idée 1 : fixer m et tester toutes les valeurs en partant de 0 jusqu'à trouver un m' / $H(m) = H(m')$ (force brute)
 - Nombre moyen de valeurs à tester : $2^n = 2^{128}$

Hachage : recherche de collision

Recherche de collision : trouver m et m' tel que $H(m) = H(m')$ ($m \neq m'$).

- La fonction de hachage : $H(.) = MD5(.)$ (taille de la sortie $n = 160$ bits)

Comment faire en pratique pour retrouver m et m' ?

- Idée 1 : fixer m et tester toutes les valeurs en partant de 0 jusqu'à trouver un $m' / H(m) = H(m')$ (force brute)
 - Nombre moyen de valeurs à tester : $2^n = 2^{128}$
- Idée 2 : prendre une liste vide $L = ;$ Calculer itérativement $h = H(x)$; si $h \in L$ alors on a trouvé une collision; sinon ajouter à L le tuple (x, h)
 - Nombre moyen de valeurs à tester : $2^{n/2} = 2^{64}$! (Paradoxe des anniversaires)

Hachage : collisions et paradoxe des anniversaires



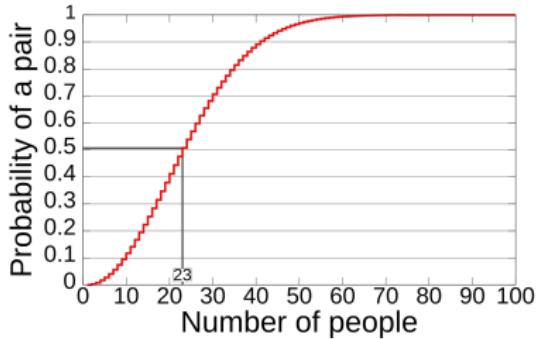
- Probabilité que Alice et Bob soient nés le même jour ? (1/365)

⁵Source: wikipedia https://en.wikipedia.org/wiki/Birthday_problem

Hachage : collisions et paradoxe des anniversaires



- Probabilité que Alice et Bob soient nés le même jour ? ($1/365$)
- Probabilité que, dans un groupe de 23 élèves, au moins 2 élèves soient nés le même jour ? (≈ 0.5)⁵



⁵Source: wikipedia https://en.wikipedia.org/wiki/Birthday_problem

Hachage : exemple de collision MD5

$M_1 =$

d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70

$M_2 =$

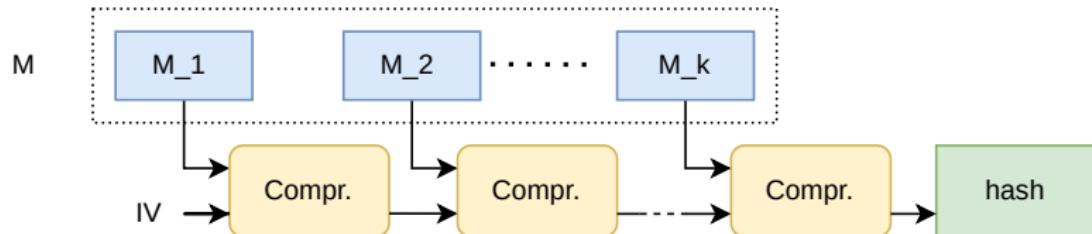
d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965ab6ff72a70

$\text{MD5}(M_1) = \text{MD5}(M_2) = 79054025255fb1a26e4bc422aef54eb4$

Construction d'une fonction de hachage

Construction de Merkle-Damgard

- Message découpé en blocs (e.g. de 512 bits)
 - Complété avec du padding⁶ si nécessaire.
- Utilisation d'une fonction de compression
 - Fonction à sens unique qui transforme n bits en m bits ($n > m$)
- Chaînage des fonctions de compression
 - Prend en entrée un bloc et la précédente sortie
 - Constante IV (Initialisation Vector) pour la 1ere itération
 - La dernière sortie est le haché



⁶Le padding doit dépendre du message.

Construction : illustration avec *SHA-1*

SHA-1 (Secure Hash Algorithm -1)

- Fonction de hachage publiée par la NSA en 1995
- Taille de bloc : 512 bits; Taille du haché : 160 bits
- Considérée comme robuste jusqu'en 2005
- Basée sur une construction Merkle-Damgard

Federal Information
Processing Standards Publication 180-1

1995 April 17

Specifications for the

SECURE HASH STANDARD

1. INTRODUCTION

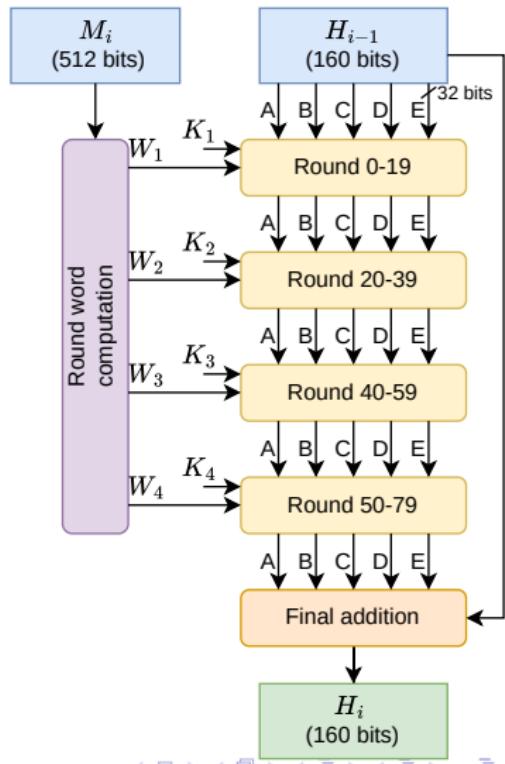
The Secure Hash Algorithm (SHA-1) is required for use with the Digital Signature Algorithm (DSA) as specified in the Digital Signature Standard (DSS) and whenever a secure hash algorithm is required for federal applications. For a message of length $< 2^{64}$ bits, the SHA-1 produces a 160-bit condensed representation of the message called a message digest. The message digest is used during generation of a signature for the message. The SHA-1 is also used to compute a message digest for the received version of the message during the process of verifying the signature. Any change to the message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify.

The SHA-1 is designed to have the following properties: it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest.

SHA-1 : fonctionnement interne

Fonction de compression de SHA-1

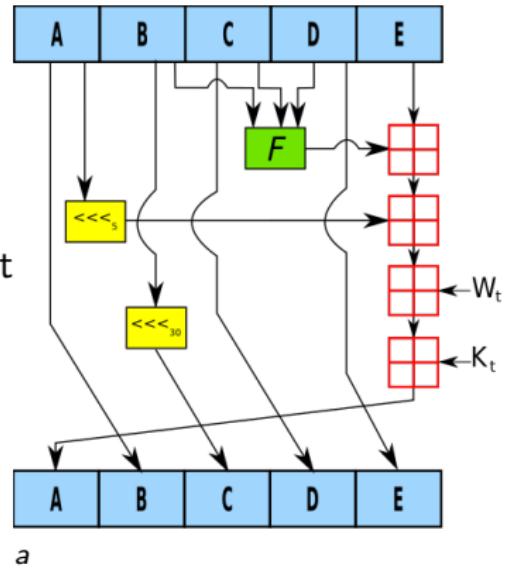
- Entrée : un bloc message M_i de 512 bits et H_{i-1} de 160 bits
- Sortie : un haché H_i de 160 bits
- Des *rondes* répétées 80 fois
 - 5 mots de 32 bits constituant l'état interne (A, B, C, D, E)
 - 4 clefs K_i
 - 4 *round words* dérivés du bloc d'entrée



SHA-1 : fonctionnement d'une ronde

Ronde de SHA-1:

- A, B, C, D, E : des mots de 32 bits
- W_t un mot dérivé du bloc d'entrée
- F : une fonction non linéaire dépendant de la ronde
- K_t : un vecteur (clef) dépendant le la ronde
- Ex : Pour les rondes 0-19
 - $F=(B \text{ and } C) \text{ or } ((\text{not } B) \text{ and } D)$
 - $K_1 = 0x5A827999$



^aSource : Wikipedia, SHA-1

SHA-1 : collision

Table 1: Colliding message blocks for SHA-1.

CV_0	4e a9 62 69 7c 87 6e 26 74 d1 07 f0 fe c6 79 84 14 f5 bf 45
$M_1^{(1)}$	<u>7f</u> 46 dc <u>93</u> <u>a6</u> b6 7e <u>01</u> <u>3b</u> 02 9a <u>aa</u> <u>1d</u> b2 56 <u>0b</u> <u>45</u> ca 67 <u>d6</u> <u>88</u> c7 f8 <u>4b</u> <u>8c</u> 4c 79 <u>1f</u> <u>e0</u> 2b 3d <u>f6</u> <u>14</u> f8 6d <u>b1</u> <u>69</u> 09 01 <u>c5</u> <u>6b</u> 45 c1 <u>53</u> <u>0a</u> fe df <u>b7</u> <u>60</u> 38 e9 <u>72</u> <u>72</u> 2f e7 <u>ad</u> 72 8f 0e <u>49</u> <u>04</u> e0 46 <u>c2</u>
$CV_1^{(1)}$	8d 64 <u>d6</u> <u>17</u> ff ed <u>53</u> <u>52</u> eb c8 59 15 5e c7 eb <u>34</u> <u>f3</u> 8a 5a 7b
$M_2^{(1)}$	<u>30</u> 57 0f <u>e9</u> <u>d4</u> 13 98 <u>ab</u> <u>e1</u> 2e f5 <u>bc</u> <u>94</u> 2b e3 <u>35</u> <u>42</u> a4 80 <u>2d</u> <u>98</u> b5 d7 <u>0f</u> <u>2a</u> 33 2e <u>c3</u> <u>7f</u> ac 35 <u>14</u> <u>e7</u> 4d dc <u>0f</u> <u>2c</u> c1 a8 <u>74</u> <u>cd</u> 0c 78 <u>30</u> <u>5a</u> 21 56 <u>64</u> <u>61</u> 30 97 <u>89</u> <u>60</u> 6b d0 <u>bf</u> 3f 98 cd <u>a8</u> <u>04</u> 46 29 <u>a1</u>
CV_2	1e ac b2 5e d5 97 0d 10 f1 73 69 63 57 71 bc 3a 17 b4 8a c5
CV_0	4e a9 62 69 7c 87 6e 26 74 d1 07 f0 fe c6 79 84 14 f5 bf 45
$M_1^{(2)}$	<u>73</u> 46 dc <u>91</u> <u>66</u> b6 7e <u>11</u> <u>8f</u> 02 9a <u>b6</u> <u>21</u> b2 56 <u>0f</u> <u>f9</u> ca 67 <u>cc</u> <u>a8</u> c7 f8 <u>5b</u> <u>a8</u> 4c 79 <u>03</u> <u>0c</u> 2b 3d <u>e2</u> <u>18</u> f8 6d <u>b3</u> <u>a9</u> 09 01 <u>d5</u> <u>df</u> 45 c1 <u>4f</u> <u>26</u> fe df <u>b3</u> <u>dc</u> 38 e9 <u>6a</u> <u>c2</u> 2f e7 <u>bd</u> 72 8f 0e <u>45</u> <u>bc</u> e0 46 <u>d2</u>
$CV_1^{(2)}$	8d 64 <u>c8</u> <u>21</u> ff ed <u>52</u> <u>e2</u> eb c8 59 15 5e c7 eb <u>36</u> <u>73</u> 8a 5a 7b
$M_2^{(2)}$	<u>3c</u> 57 0f <u>eb</u> <u>14</u> 13 98 <u>bb</u> <u>55</u> 2e f5 <u>a0</u> <u>a8</u> 2b e3 <u>31</u> <u>fe</u> a4 80 <u>37</u> <u>b8</u> b5 d7 <u>1f</u> <u>0e</u> 33 2e <u>df</u> <u>93</u> ac 35 <u>00</u> <u>eb</u> 4d dc <u>0d</u> <u>ec</u> c1 a8 <u>64</u> <u>79</u> 0c 78 <u>2c</u> <u>76</u> 21 56 <u>60</u> <u>dd</u> 30 97 <u>91</u> <u>d0</u> 6b d0 <u>af</u> 3f 98 cd <u>a4</u> <u>bc</u> 46 29 <u>b1</u>
CV_2	1e ac b2 5e d5 97 0d 10 f1 73 69 63 57 71 bc 3a 17 b4 8a c5

- Collision sur SHA-1 découverte en 2017^a par Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov

^a2017 sha1 collision.

Les fonctions SHA

- SHA-0, SHA-1 : collisions identifiées
- SHA-2 (SHA-256, SHA-512 ...) : faiblesses identifiées
- SHA-3 : nouveau standard (2015)

Application d'une fonction de hachage : verif. intégrité

Series	Filename	Size	Release	End-of-Life	Checksums
3.5 [LTS]	openssl-3.5.0.tar.gz	50.7MiB	08 Apr 2025	08 Apr 2030	(PGP sign) (SHA1) (SHA256)
3.4	openssl-3.4.1.tar.gz	17.5MiB	11 Feb 2025	22 Oct 2026	(PGP sign) (SHA1) (SHA256)
3.3	openssl-3.3.3.tar.gz	17.3MiB	11 Feb 2025	09 Apr 2026	(PGP sign) (SHA1) (SHA256)
3.2	openssl-3.2.4.tar.gz	17.0MiB	11 Feb 2025	23 Nov 2025	(PGP sign) (SHA1) (SHA256)
3.0 [LTS]	openssl-3.0.16.tar.gz	14.6MiB	11 Feb 2025	07 Sep 2026	(PGP sign) (SHA1) (SHA256)

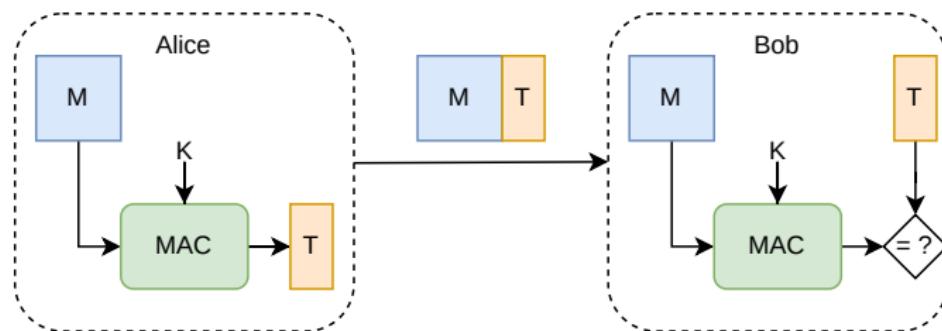
- Utiliser une fonction de hachage pour vérifier l'intégrité d'un fichier
 - Télécharger le fichier f et son haché h
 - Calculer $H(f)$ et comparer à h

```
$ shasum openssl-3.5.0.tar.gz
01ba9f9cc97125eab08bbe7206607e404244cf3c  openssl-3.5.0.tar.gz
$ cat openssl-3.5.0.tar.gz.sha1
01ba9f9cc97125eab08bbe7206607e404244cf3c *openssl-3.5.0.tar.gz
```

Application d'une fonction de hachage : authentification

MAC (Message Authentication Code)⁷

- Protège l'**Intégrité** et l'**Authenticité** d'un message M grâce à une clef secrète K
 - ① Calcul d'un tag $T = \text{MAC}(K, M)$
 - ② Intégrité : le message n'a pas été modifié
 - ③ Authenticité : l'émetteur connaît le secret K



⁷Parfois appelé MIC (*Mess. Integrity Code*) pour éviter une confusion avec *Medium Access Control*.

MAC avec une fonction de hachage

Construire un Message Authentication Code à partir d'une fonction de hachage

- Préfixe secret : $H(K||M)$
 - Construction naïve / simple avec des faiblesses
 - Ex.: possible de calculer $H(K||M_1||M_2)$ à partir de $H(K||M_1)$ sans connaître K (Attaque par extension de longueur)
- HMAC : $H((K \oplus \text{opad})||H(K \oplus \text{ipad})||M)$
 - opad et ipad sont des constantes de la taille du bloc de H
 - Construction sûre tant que H résiste aux collisions



Fonctions de hachage cryptographique : récap.

- Calcule un **haché de taille fixe** ..
- à partir d'une **entrée de taille quelconque**
- Sécurité : doit résister aux attaques de 1ere/2nde préimage et collisions
- Fonction de hachage recommandée : SHA-3

Crypto symétriques : fonctions recommandées

Fonctions de Hachage	MD5 SHA-1 SHA-2 SHA-3
<i>Blockcipher</i>	DES AES

Table: Recommandations relatives aux fonctions cryptographiques symétriques.

Sources :

- ANSSI : Guide de sélection d'algorithmes cryptographiques ⁸
- NIST : *Cryptographic Standards and Guidelines* ⁹

⁸<https://cyber.gouv.fr/publications/mecanismes-cryptographiques>

⁹<https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines>

1 Introduction

2 Chiffrement par bloc

3 Fonctions de hachage cryptographiques

4 Cryptographie asymétrique

Cryptographie à clef publique

- La cryptographie classique (symétrique) nécessite un secret partagé
 - Une clef que seuls Alice et Bob connaissent
 - Difficile à mettre en pratique
- La cryptographie asymétrique permet de nous passer d'un secret partagé
 - Cryptographie à clef publique
 - Alice a une paire de clefs
 - La clef privée K_{priv}^{Alice} seulement connue de Alice (secret)
 - La clef publique K_{pub}^{Alice} connue de tous



Problème de factorisation de grands nombres

Factorisation d'un nombre

Etant donné un entier $n = p \times q$, retrouver ses facteurs p et q .

Exemples :

- $n = 21$: $p = ?$ $q = ?$

Problème de factorisation de grands nombres

Factorisation d'un nombre

Etant donné un entier $n = p \times q$, retrouver ses facteurs p et q .

Exemples :

- $n = 21 : p = ? q = ?$
 - : $21 = 3 \times 7$ (Facile !)
- $n = 692693 : p = ? q = ?$

Problème de factorisation de grands nombres

Factorisation d'un nombre

Etant donné un entier $n = p \times q$, retrouver ses facteurs p et q .

Exemples :

- $n = 21 : p = ? q = ?$
 - : $21 = 3 \times 7$ (Facile !)
- $n = 692693 : p = ? q = ?$
 - $692693 = 977 \times 709$ (Difficile à la main, mais facile à programmer)
- $n = 3213876088517980551083924185487283336189331657515992206038949$

Problème de factorisation de grands nombres

Factorisation d'un nombre

Etant donné un entier $n = p \times q$, retrouver ses facteurs p et q .

Exemples :

- $n = 21 : p = ? q = ?$
 - : $21 = 3 \times 7$ (Facile !)
- $n = 692693 : p = ? q = ?$
 - $692693 = 977 \times 709$ (Difficile à la main, mais facile à programmer)
- $n = 3213876088517980551083924185487283336189331657515992206038949$
 - =
1267650600228229401496703205653 \times 2535301200456458802993406410833
(p et q de l'ordre de 2^{100})
 - Possible à programmer ?

Problème de factorisation de grands nombres

Factoriser un entier N (de taille n bits) :

Problème de factorisation de grands nombres

Factoriser un entier N (de taille n bits) :

- Recherche exhaustive : Pour i dans $[2..\sqrt{N}]$: i divise N ?
 - Coût : $\mathcal{O}(2^{n/2})$

Problème de factorisation de grands nombres

Factoriser un entier N (de taille n bits) :

- Recherche exhaustive : Pour i dans $[2.. \sqrt{N}]$: i divise N ?
 - Coût : $\mathcal{O}(2^{n/2})$
- On ne connaît pas d'algo efficace (complexité polynomiale : $\mathcal{O}(n^c)$)

Si on choisit des entiers suffisamment grand, alors il devient **impossible en pratique** de factoriser $N = n \times p$ (n et p premiers) .

RSA (Rivest–Shamir–Adleman)

- Publié en 1977
 - Système équivalent développé par le GCHQ dès 1973
- Cryptosystème à clef publique
- Sécurité basé sur le problème de factorisation de grands entiers

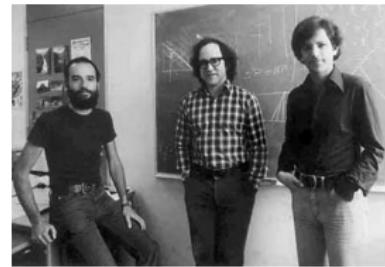


Figure: Ron Rivest, Adi Shamir and Leonard Adleman

RSA : génération des clefs

- Génération d'une **paire de clefs**
 - Clef publique K_{pub} et une clef privée K_{priv}
- Processus de création d'une paire de clef RSA

¹⁰voir Wikipedia pour la preuve

RSA : génération des clefs

- Génération d'une **paire de clefs**
 - Clef publique K_{pub} et une clef privée K_{priv}
- Processus de création d'une paire de clef RSA
 - ① Choisir deux entiers premiers p et q et calculer $n = pq$
ex.: $7 \times 19 = 133$

¹⁰voir Wikipedia pour la preuve

RSA : génération des clefs

- Génération d'une **paire de clefs**
 - Clef publique K_{pub} et une clef privée K_{priv}
- Processus de création d'une paire de clef RSA
 - ① Choisir deux entiers premiers p et q et calculer $n = pq$
ex.: $7 \times 19 = 133$
 - ② Calculer $\phi(n) = (p - 1)(q - 1)$
ex.: $(7 - 1) \times (19 - 1) = 108$

¹⁰voir Wikipedia pour la preuve

RSA : génération des clefs

- Génération d'une **paire de clefs**
 - Clef publique K_{pub} et une clef privée K_{priv}
- Processus de création d'une paire de clef RSA
 - ① Choisir deux entiers premiers p et q et calculer $n = pq$
ex.: $7 \times 19 = 133$
 - ② Calculer $\phi(n) = (p - 1)(q - 1)$
ex.: $(7 - 1) \times (19 - 1) = 108$
 - ③ Choisir un entier premier e avec $e < \phi(n)$
ex.: 29

¹⁰voir Wikipedia pour la preuve

RSA : génération des clefs

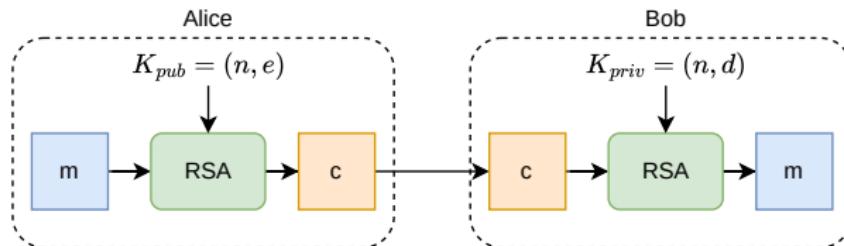
- Génération d'une **paire de clefs**
 - Clef publique K_{pub} et une clef privée K_{priv}
- Processus de création d'une paire de clef RSA
 - ① Choisir deux entiers premiers p et q et calculer $n = pq$
ex.: $7 \times 19 = 133$
 - ② Calculer $\phi(n) = (p - 1)(q - 1)$
ex.: $(7 - 1) \times (19 - 1) = 108$
 - ③ Choisir un entier premier e avec $e < \phi(n)$
ex.: 29
 - ④ Trouver d tel que $ed = 1 \bmod \phi(n)$
ex.: 41
- La paire de clefs : $K_{\text{pub}} = (n, e)$ et $K_{\text{priv}} = (n, d)$
- Et on a $\forall 0 \leq x < n$, $(x^e)^d \bmod n = x^{10}$

¹⁰voir Wikipedia pour la preuve

RSA : chiffrement / déchiffrement

Fonctionnement de RSA (méthode basique¹¹)

- Chiffrement de m avec la clef publique (n, e) : $c = m^e \text{ mod } n$
- Déchiffrement avec la clef privée (n, d) : $m = c^d \text{ mod } n$
 - En effet, $c^d \text{ mod } n = (m^e)^d \text{ mod } n = m^{ed} \text{ mod } n = m$



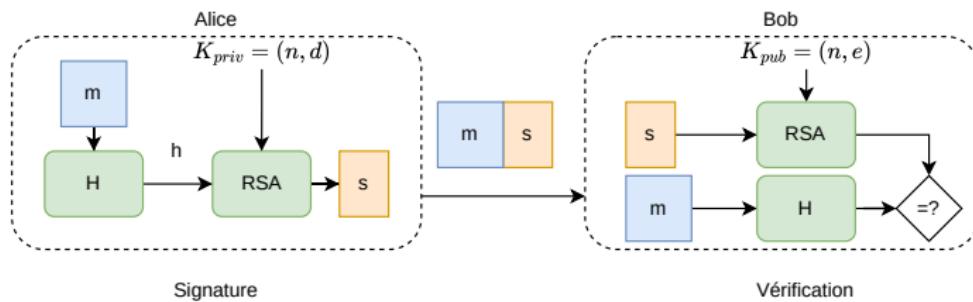
- Seule la clef publique permet de déchiffrer le message
 - Il n'est pas possible de retrouver m à partir de c et (n, e) (sauf à factoriser n)

¹¹Cette méthode naïve est vulnérable à des attaques.

RSA : signature

Signer un message avec RSA

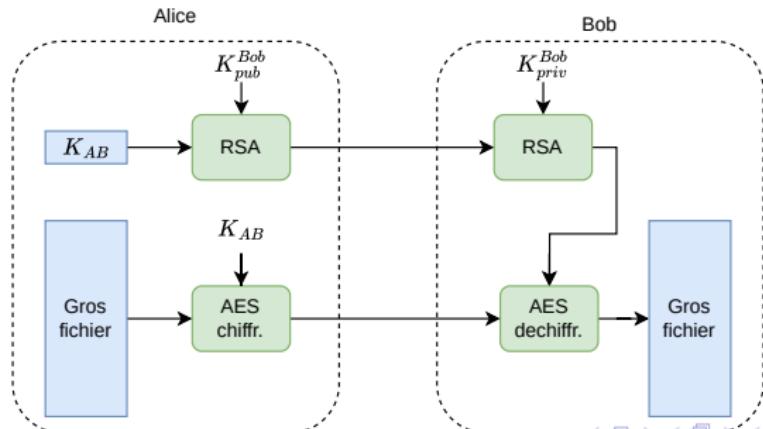
- ① Hacher le message m , $h = H(m)$
- ② Générer la signature avec la clef privée : $s = h^d \text{ mod } n$
- ③ Vérifier la signature avec la clef publique : $H(m) \stackrel{?}{=} s^e \text{ mod } n$



- Seule la clef privée permet de générer une signature telle que $H(m) = s^e \text{ mod } n$
 - L'émetteur du message est bien Alice, qui est la seule à connaître la clef privée

Cryptographie hybride

- Cryptographie symétrique (AES, SHA-3 ...)
 - Rapide / peu coûteux en calcul mais nécessite un secret commun
- Cryptographie asymétrique (RSA ...)
 - Pas besoin de secret commun mais lent / coûteux en calcul
- Cryptographie hybride
 - Crypto asymétrique pour échanger une clef de session (K_{AB})
 - Crypto symétrique + clef de session le reste des échanges
 - Le meilleur des deux mondes : rapide et pas besoin de secret commun !



Résumé

- Cryptographie symétrique
 - Chiffrement à bloc avec une clef secrète commune
 - Fonction de hachage pour vérifier l'intégrité
- Cryptographie asymétrique
 - Paire de clefs Publique/Privée
 - Permet chiffrement et signature