

Module 3: Cryptographie

Méthodes de la Sécurité des Systèmes

Université Claude Bernard Lyon 1

Hugo Blanc
CC BY-SA 4.0

2026 - 2027

1. Xor
2. Chiffrement par bloc
3. Modes d'opération
4. Échange de clés
5. Cryptographie à clé publique
6. Fonctions de hachage
7. L'aléatoire

1. Xor

2. Chiffrement par bloc

3. Modes d'opération

4. Échange de clés

5. Cryptographie à clé publique

6. Fonctions de hachage

7. L'aléatoire

1. XOR

Le OU exclusif (*exclusive OR*, XOR) est un **opérateur booléen binaire** qui est:

- **vrai** quand la première ou la deuxième entrée est à vrai, **mais pas les deux**;
- **faux** sinon.

En mathématiques et cryptographie, le XOR est généralement représenté par le symbole $\oplus$.

1. XOR

Les propriétés du XOR sont donc:

$$0 \oplus 0 = 0 \quad 0 \oplus 1 = 1$$

$$1 \oplus 0 = 1 \quad 1 \oplus 1 = 0$$

1. XOR

On peut également montrer que $a \oplus b \oplus a = b$:

$$\begin{aligned} a \oplus b \oplus a &= a \oplus a \oplus b \\ &= 0 \oplus b \\ &= b \end{aligned}$$

Cette propriété est **très importante pour le chiffrement** (on peut imaginer que le premier XOR chiffre, et l'autre déchiffre).

1. XOR



Exercices

Quel est le résultat en base 2 de l'opération binaire suivante ? En base 16 ?

$$\begin{array}{r} 110011 \\ \oplus 101010 \\ \hline \end{array}$$

1.1 Chiffre de Vernam

Le XOR peut sembler simple en apparence, mais permet la mise en œuvre de la méthode de chiffrement la plus robuste qui soit: **le chiffre de Vernam** (*one-time pad* en anglais).

1.1 Chiffre de Vernam

Le chiffrement par chiffre de Vernam repose sur 3 principes:

- la clé doit être une suite de bits **au moins aussi longue** que le message à chiffrer;
- les bits composant la clé doivent être choisis de manière **totalement aléatoire**;
- chaque clé ne doit être utilisée **qu'une seule fois**.

1.1 Chiffre de Vernam

Si ces 3 règles sont appliquées, le cryptosystème offre **une sécurité absolue**, selon la théorie du chiffrement de Shannon:

«Étant donné une clé réellement aléatoire et utilisée qu'une seule fois, un texte chiffré peut être traduit en n'importe quel texte en clair de même longueur, et tous ont la même probabilité.»

1.1 Chiffre de Vernam

Cependant, cette méthode de chiffrement est rarement employée, car complexe à mettre en place:

- pour chiffrer un fichier de 10GB, il faut une clé de 10GB;
- l'échange de la clé nécessite un canal sûr (dont potentiellement déjà chiffré);
- etc.

1. Xor

2. Chiffrement par bloc

3. Modes d'opération

4. Échange de clés

5. Cryptographie à clé publique

6. Fonctions de hachage

7. L'aléatoire

2. Chiffrement par bloc

Le chiffrement par bloc (*block cipher*) est une des deux grandes catégories de chiffrement en **cryptographie symétrique**, avec le chiffrement par flux.

2. Chiffrement par bloc

La modélisation mathématique des algorithmes de chiffrement par bloc est la suivante:

$$C = E(k, P)$$

où la fonction E transforme les blocs de texte clair P en blocs chiffrés C en utilisant une clé secrète k .

2. Chiffrement par bloc

Afin de simplifier la mémorisation des symboles et la lecture des équations, on peut garder en tête que E est pour «Encrypt», P pour «Plain text» (texte en clair), C pour «Cipher text» (texte chiffré) et k pour «key», la clé secrète.

2. Chiffrement par bloc

Une fois chiffrés, les blocs peuvent être déchiffrés en utilisant la même clé k avec une fonction de déchiffrement D :

$$P = D(k, C)$$

La taille des blocs varie de 64 à 512 bits en fonction des algorithmes.

2. Chiffrement par bloc

On peut modéliser ces deux opérations comme suit:

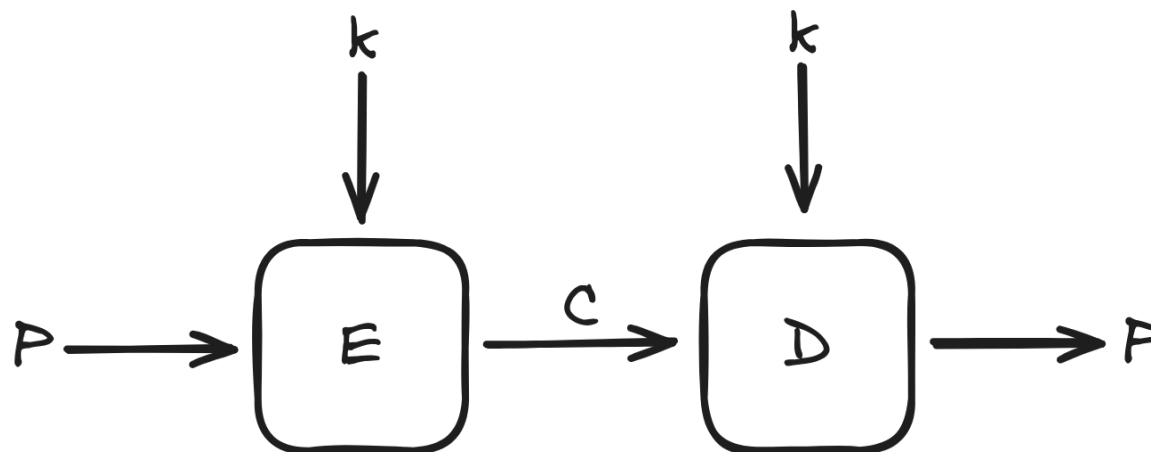


Fig. 1. – Schématisation du fonctionnement de base des algorithmes de chiffrement par bloc.

2. Chiffrement par bloc

L'algorithme de chiffrement par bloc le plus connu est **AES** (*Advanced Encryption Standard*, auparavant appelé **Rijndael**, un dérivé des noms des deux cryptographes belges qui l'ont inventé).

Cet algorithme a été désigné puis choisi comme standard à la suite d'un concours public et *peer-reviewed* organisé par le NIST¹ en 2001.

¹*National Institute of Standards and Technology.*

2. Chiffrement par bloc

Il est le successeur de **DES** (*Data Encryption Standard*), datant du milieu des années 1970 (standardisé en 1977), et qui comporte des vulnérabilités et surtout une taille de clé limitée (56 bits).

2. Chiffrement par bloc

AES fonctionne avec des clés de taille 128, 192 et 256 bits, et utilise des blocs d'une taille de 128 bits.

Il n'existe à ce jour aucune attaque pratique connue contre cet algorithme.

Bien qu'il y ait eu quelques tentatives au cours des dernières années, la plupart d'entre elles impliquent des attaques sur les clés elles-mêmes ou sur des versions réduites d'AES¹.

¹Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, et Adi Shamir, *Key recovery attacks of practical complexity on AES variants with up to 10 rounds*.

L'algorithme AES est un réseau de **substitution-permutation**:

Design générique pour les algorithmes de chiffrement par bloc où les blocs sont chiffrés par une répétition de substitutions et de permutations.

Ces opérations de substitution et permutation sont réparties sur plusieurs étapes indépendantes: **Key schedule**, **SubBytes**, **ShiftRows**, **MixColumns**, **AddRoundKey**.



Exercices

Un chiffrement par bloc par lui-même fonctionne parfaitement pour chiffrer, comme son nom l'indique, un seul bloc. Comment pourrions-nous chiffrer un message qui en contient plusieurs ?

1. Xor
2. Chiffrement par bloc
- 3. Modes d'opération**
4. Échange de clés
5. Cryptographie à clé publique
6. Fonctions de hachage
7. L'aléatoire

3. Modes d'opération

Comme nous l'avons vu à la fin du chapitre précédent, afin de chiffrer des données plus longues qu'un bloc, nous devons transformer un flux continu en *chunks* plus petits d'une taille fixe, puis opérer sur chacun de ces *chunks*. La façon de les découper et de les traiter est appelée le **mode d'opération**.

Définition Le mode d'opération décrit comment appliquer de manière répétée l'opération monobloc d'un chiffrement pour transformer de manière sécurisée des quantités de données supérieures à un bloc.

L'*Electronic Codebook Block* (ECB) est le mode d'opération le plus simple, mais également le moins robuste. Le message à chiffrer est découpé en plusieurs blocs qui sont chiffrés séparément, sans avoir d'influence les uns sur les autres.

3.1 ECB

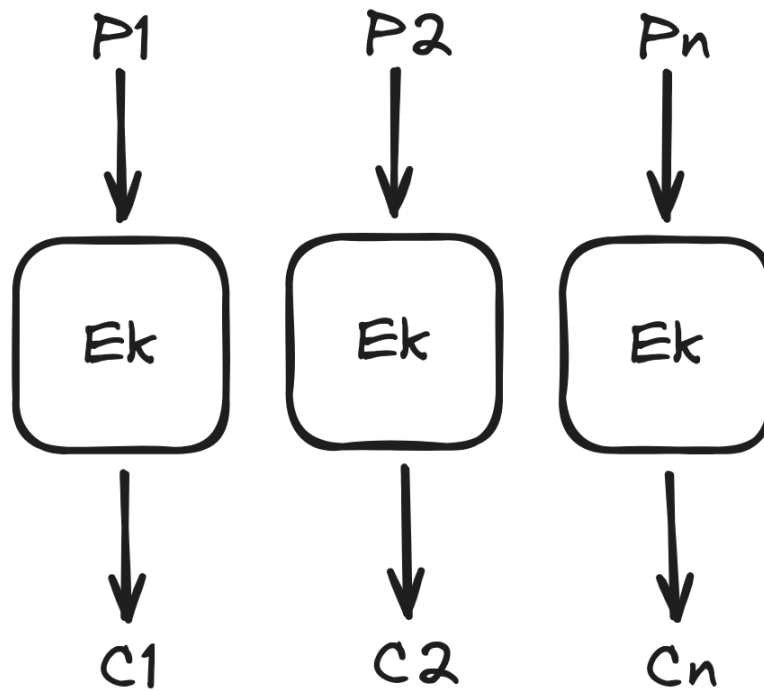


Fig. 2. – Schématisation du fonctionnement du mode d'opération ECB.



Exercices

Quels sont les avantages d'un tel mode d'opération ? Le principal défaut ?

Prenons comme exemple les deux chaînes de caractères, qui représentent le propriétaire d'une maison et le prix de cette dernière:

JOHN__105000

JACK__500000

Si l'on chiffre le premier message suivant le mode d'opération ECB et une taille de bloc de deux octets (soit deux caractères), on obtient par exemple:

J0|HN|__|10|50|00
Q9|2D|FP|VX|C9|10

3.1 ECB

Si l'on répète l'opération avec le second message et (évidemment) la même clé, on obtient:

```
JA|CK|__|50|00|00  
LD|AS|FP|C9|I0|I0
```

On remarque que des paires de caractères identiques apparaissent dans les deux messages chiffrés:

1: Q9|2D|FP|VX|C9|I0

2: LD|AS|FP|C9|I0|I0

Du point de vue de l'attaquant·e, c'est très utile: si l'on connaît quelle entrée donne quelle sortie, nous pouvons finir par construire une table de toutes les entrées possibles et leurs sorties correspondantes, et ainsi pouvoir déchiffrer n'importe quel texte chiffré sans même connaître la clé !

Voici un exemple un peu plus visuel avec une image, en affichant respectivement l'image en clair, l'image chiffrée avec le mode ECB et des blocs de 4 pixels, puis avec le mode CBC et des blocs de 4 pixels:

3.1 ECB

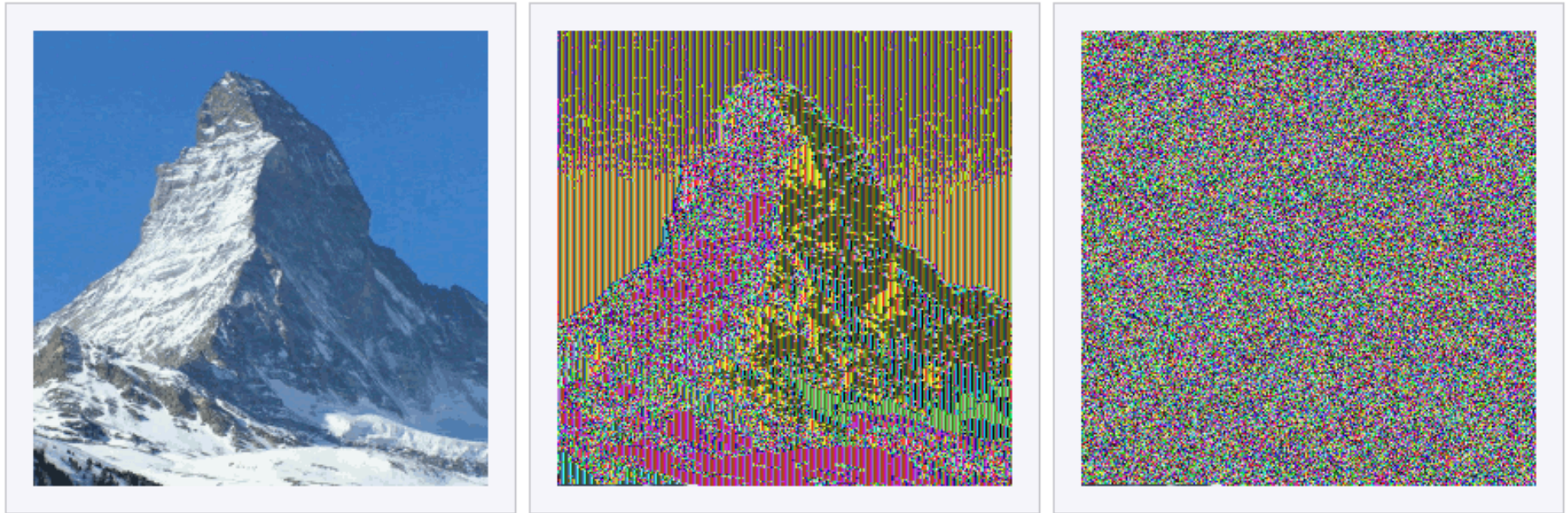


Fig. 3. – Comparaison du chiffrement d'une image avec les modes ECB et CBC.¹

¹[https://fr.wikipedia.org/wiki/Mode_d%27op%C3%A9ration_\(cryptographie\)](https://fr.wikipedia.org/wiki/Mode_d%27op%C3%A9ration_(cryptographie))

3.1.1 CBC

Le *Cipher Block Chaining* (CBC), est un mode qui repose sur un **vecteur d'initialisation** (IV).

Définition Le vecteur d'initialisation, aussi appelé IV, est un bloc de bits utilisé pour « randomiser » le chiffrement, et donc produire des textes chiffrés distincts même si le même texte en clair est chiffré plusieurs fois.

Le vecteur d'initialisation est comparable à un *nonce* : public et unique, contrairement à une clé secrète.

En mode CBC, on applique sur chaque bloc un XOR avec le chiffrement du bloc précédent avant qu'il soit lui-même chiffré. L'IV, quant à lui, est utilisé uniquement sur le premier bloc. Mais comme les résultats des blocs sont interdépendants, il influence tout le reste de la chaîne.

3.1 ECB

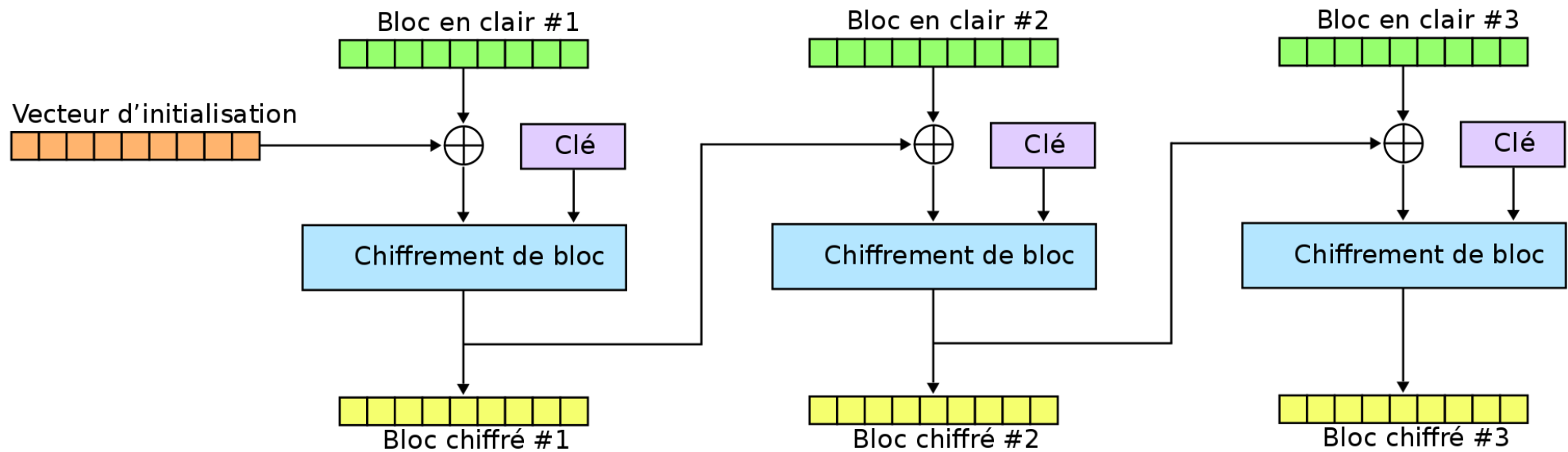


Fig. 4. – Schématisation du fonctionnement du mode d'opération CBC.

3.1.2 Exemple d'attaque sur CBC

Le mode d'opération CBC résout une partie des problèmes que l'on retrouve avec ECB, mais vient avec ses défauts. Il est notamment possible, dans certains cas, de retrouver la clé k , par exemple lorsque le vecteur d'initialisation est mal choisi.

Imaginons une situation où les personnes en charge de la sécurité utilisent un algorithme de chiffrement en mode CBC. De nombreux systèmes utilisent la clé k en tant que vecteur d'initialisation: après tout, il nous faut un secret et avec k , nous en avons déjà un ! De plus, cela améliore les performances car l'expéditeur et le destinataire n'ont pas à s'échanger l'IV explicitement car ils connaissent déjà la clé.

Cette configuration est vulnérable en cas d'interception par un·e acteur·rice malveillant·e: si Alice envoie un message à Bob, et que Charlie peut l'intercepter et le modifier, il peut alors réussir à trouver la clé:

- Alice transforme son texte clair P en trois blocs P_1 , P_2 et P_3 , et les chiffre en mode CBC avec une clé k (clé servant également en tant qu'IV).
- Elle obtient donc le texte chiffré $C = C_1 C_2 C_3$ qu'elle envoie à Bob.
- Avant que le message atteigne Bob, Charlie l'intercepte et le modifie pour qu'il devienne $C' = C_1 Z C_1$, où Z est un bloc rempli de *null bytes* (0x00).

3.1 ECB

Ensuite Bob déchiffre C' et a donc trois blocs de texte en clair P'_1, P'_2, P'_3 :

$$\begin{aligned}P'_1 &= D(k, C_1) \oplus IV \\&= D(k, C_1) \oplus k \\&= P_1\end{aligned}$$

$$\begin{aligned}P'_2 &= D(k, Z) \oplus C_1 \\&= R\end{aligned}$$

$$\begin{aligned}P'_3 &= D(k, C_1) \oplus Z \\&= D(k, C_1) \oplus 0 \\&= D(k, C_1) \\&= P_1 \oplus IV\end{aligned}$$

R est un bloc aléatoire, son contenu ne nous intéresse pas.

À partir de maintenant, nous partons du principe que nous sommes dans un contexte d'attaque à texte chiffré choisi¹, ce qui signifie que le cryptanalyste a accès aux blocs déchiffrés.

¹https://en.wikipedia.org/wiki/Chosen-ciphertext_attack

Seuls $P'_1 = P_1$ et $P'_3 = P_1 \oplus IV$ nous intéressent dorénavant. En appliquant une des règles vues dans le chapitre XOR, nous retrouvons l'IV:

$$(P1 \oplus IV) \oplus P1 = IV$$

Nous sommes partis du postulat que l'IV est égal à k , nous venons alors de retrouver la clé de chiffrement.

Cet exemple n'est qu'une des nombreuses attaques possibles sur le mode CBC (*Padding Oracle*, ...).

Pour AES, le **mode d'opération recommandé est GCM** (*Galois/Counter Mode*) qui, à cause de sa complexité, ne sera pas détaillé dans ce cours.



Aide aux révisions

- ECB chiffre chaque bloc indépendamment : déterministe, il laisse apparaître les motifs du texte clair.
- CBC enchaîne les blocs via un XOR et un **vecteur d'initialisation** (IV) unique par message.
- Un IV mal utilisé, ou une erreur de bourrage révélée (**padding oracle**), permet de déchiffrer sans la clé.
- En pratique, on préfère des modes authentifiés comme GCM (confidentialité + intégrité).

1. Xor
2. Chiffrement par bloc
3. Modes d'opération
- 4. Échange de clés**
5. Cryptographie à clé publique
6. Fonctions de hachage
7. L'aléatoire

4. Échange de clés

L'échange de clés, et plus généralement l'échange de secrets, est une **composante essentielle de tout cryptosystème** utilisant du chiffrement symétrique.

4. Échange de clés

En effet, pour qu'un·e destinataire·rice puisse déchiffrer un message chiffré avec la clé k , il faut au préalable qu'iel connaisse la clé: elle doit donc transiter d'un parti à l'autre à un moment donné.

4. Échange de clés

Il existe des méthodes extrêmement fiables pour partager une clé, comme par exemple...

4. Échange de clés

Il existe des méthodes extrêmement fiables pour partager une clé, comme par exemple... se donner rendez-vous dans un endroit gardé secret et ne pas prendre d'appareil numérique qui puisse enregistrer, ainsi que de ne prendre aucune note.

Cependant, ce genre de méthodes sont 1) extrêmement contraignantes et 2) longues à implémenter (de plusieurs heures à plusieurs jours).

4. Échange de clés

Il a donc fallu mettre en place de nouvelles techniques, et une des plus utilisées est l'échange de clé **Diffie-Hellman**.

4. Échange de clés

L'échange de clés Diffie-Hellman est une méthode permettant à deux agents d'établir un secret commun de manière **publique**. Cette méthode a été publiée en 1976 et a valu à ses deux concepteurs (Whitfield Diffie et Martin Hellman) le prix Turing en 2015.

4. Échange de clés

Diffie-Hellman est utilisé absolument partout: dès que vous allez sur Internet, dès qu'il y a établissement d'une connexion TLS...

Son fonctionnement est assez simple à comprendre en utilisant un système de couleurs.

4. Échange de clés

1. Première étape: Alice et Bob choisissent chacun un secret qu'ils gardent pour eux (a et b) et se mettent d'accord sur un autre secret commun (g).

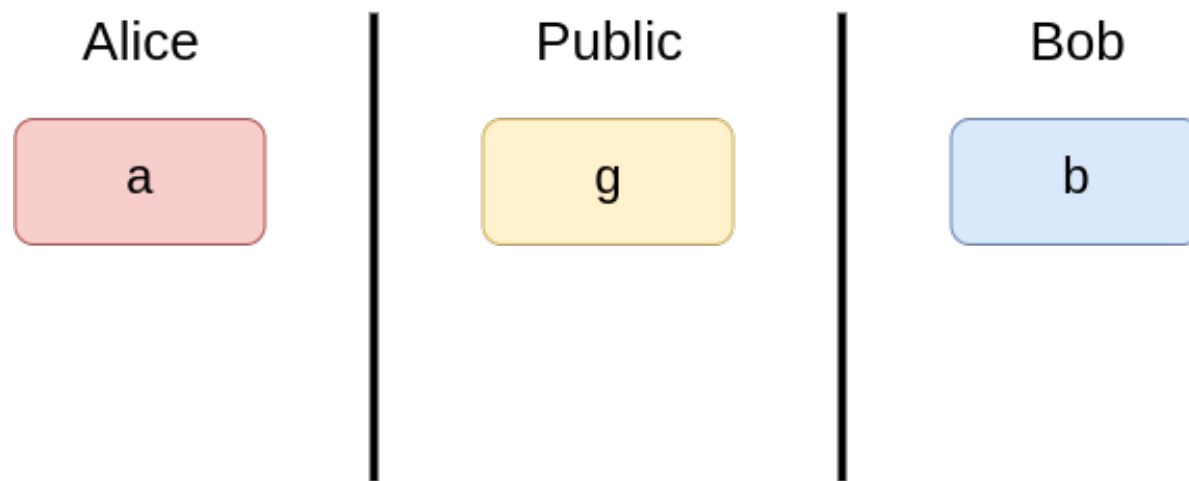


Fig. 5. – Création des secrets initiaux par les partis.

4. Échange de clés

2. Seconde étape: Alice et Bob « mélangent » de manière **privée** leurs secrets avec le secret commun. Cette opération n'est **pas réversible**, c'est-à-dire qu'à partir de ag , on ne peut mathématiquement pas retrouver a (dans un temps raisonnable).

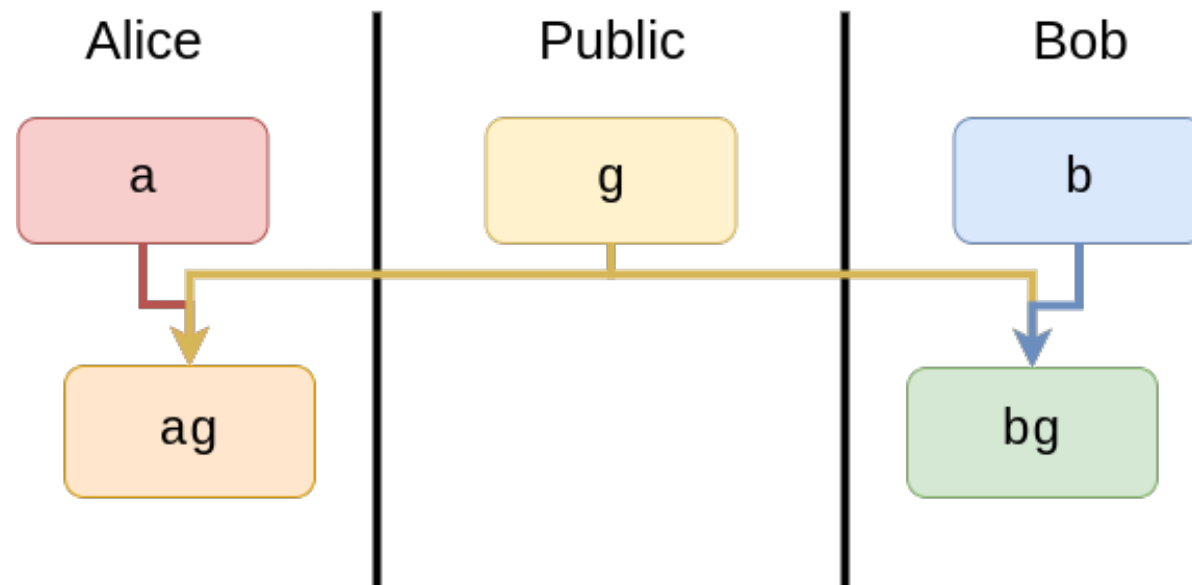


Fig. 6. – Échange et mélange de secrets.

4. Échange de clés

3. Troisième et dernière étape: Alice et Bob envoient publiquement leur nouveau secret l'un à l'autre, qu'ils «mélangeront» avec leur secret initial.

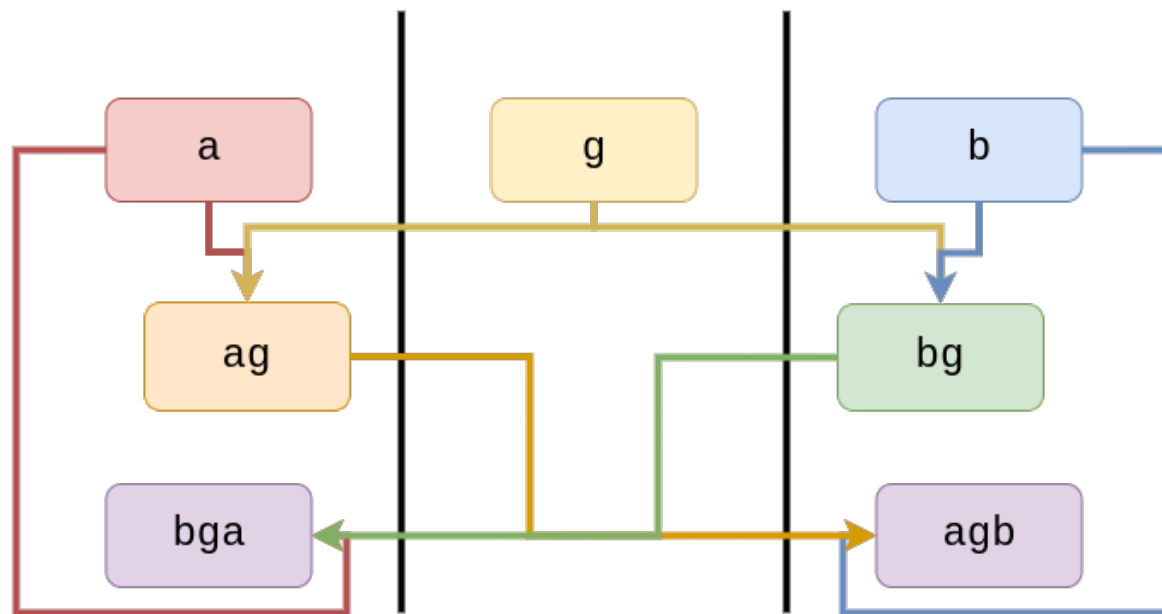


Fig. 7. – Création du secret final.

4. Échange de clés

Un secret commun (abg) vient d'être établi **publiquement**, sans pour autant qu'un·e attaquant·e puisse le retrouver !

4. Échange de clés

Cette méthode est cependant vulnérable à une attaque en particulier: le *Man-in-the-Middle*. Si Alice ne s'assure pas qu'elle parle bien avec Bob (et inversement), un·e attaquant·e peut s'interposer entre eux: iel établit un secret partagé avec Alice, puis un autre avec Bob, et relaie les messages en les déchiffrant d'un côté et les rechiffrant de l'autre.

4. Échange de clés

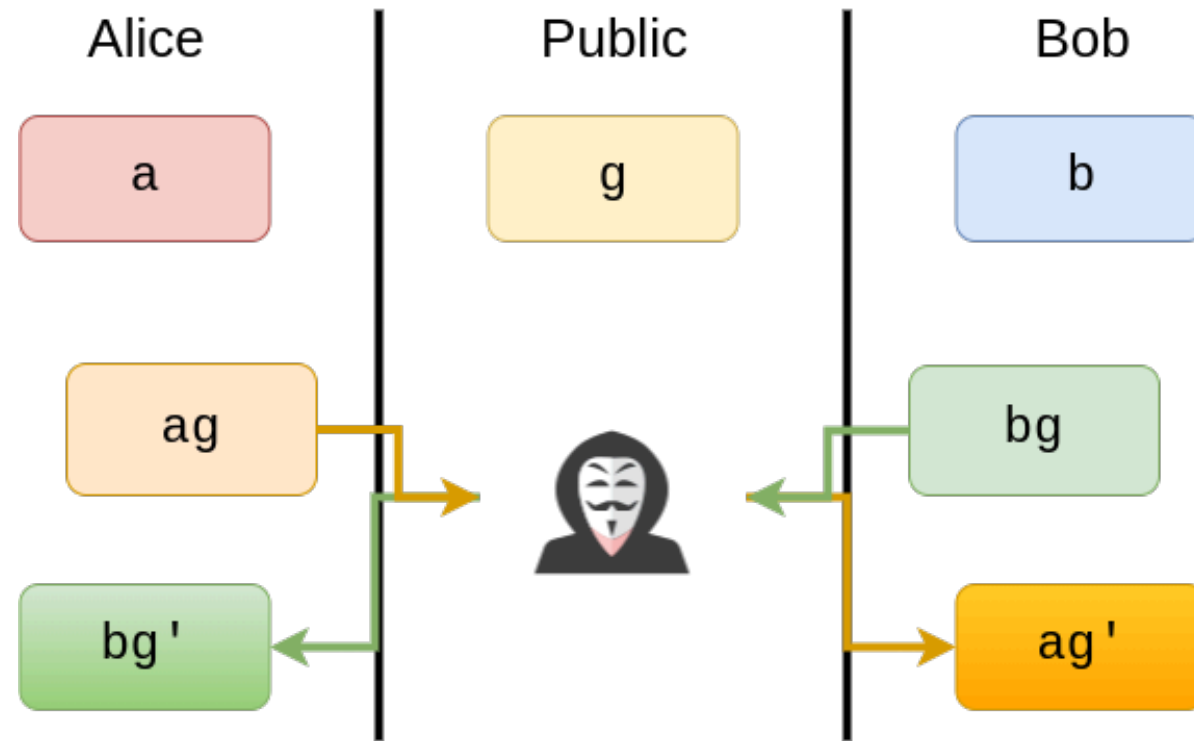


Fig. 8. – Altération des secrets.

4. Échange de clés

Il faut donc utiliser des méthodes d'authentification en plus de Diffie-Hellman pour s'assurer que les messages ne soient pas altérés.

4. Échange de clés



Aide aux révisions

- Diffie-Hellman permet de construire un secret commun sur un canal public, sans jamais le transmettre.
- Sa sécurité repose sur la difficulté du **logarithme discret**.
- Il n'authentifie personne : sans vérification d'identité, l'échange est vulnérable au **Man-in-the-Middle**.
- Il faut donc le combiner à un mécanisme d'authentification des partis.

1. Xor
2. Chiffrement par bloc
3. Modes d'opération
4. Échange de clés
- 5. Cryptographie à clé publique**
6. Fonctions de hachage
7. L'aléatoire

5. Cryptographie à clé publique

Jusqu'à présent, nous avons seulement vu la cryptographie à clé privée, également appelée symétrique: un secret était systématiquement partagé entre les partis.

5. Cryptographie à clé publique

Bien que les cryptosystèmes symétriques soient plus simples à mettre en place, ils viennent aussi avec un risque majeur: si la clé secrète venait à fuiter, alors c'est *game-over*.

5. Cryptographie à clé publique

C'est pour cela que des cryptosystèmes qui ne dépendent pas que d'une seule clé ont été mis en place. Au lieu d'un seul secret, une paire de clés est utilisée: une **clé publique** et une **clé privée**.

Chacune de ces clés a un rôle et une confidentialité particulier·ère:

la **clé publique sert à chiffrer, et peut être partagée**; la **clé privée sert à déchiffrer, et doit rester confidentielle**.

5. Cryptographie à clé publique

En pratique, **les gens chiffrent** les messages qu'ils veulent vous communiquer avec **votre clé publique**, et vous êtes seul à pouvoir **déchiffrer** le message avec **votre clé privée**. L'information est indéchiffrable sans votre clé privée.

5. Cryptographie à clé publique

Avertissement sur le partage de clés

Malheureusement, on retrouve encore et toujours des personnes qui soit ne connaissent pas la différence entre la clé publique et la clé privée, soit les confondent. Cela mène régulièrement au partage en ligne de la mauvaise clé qui, si ce n'est pas correctement géré et à temps, peut causer de **gros problèmes de sécurité**.

5. Cryptographie à clé publique

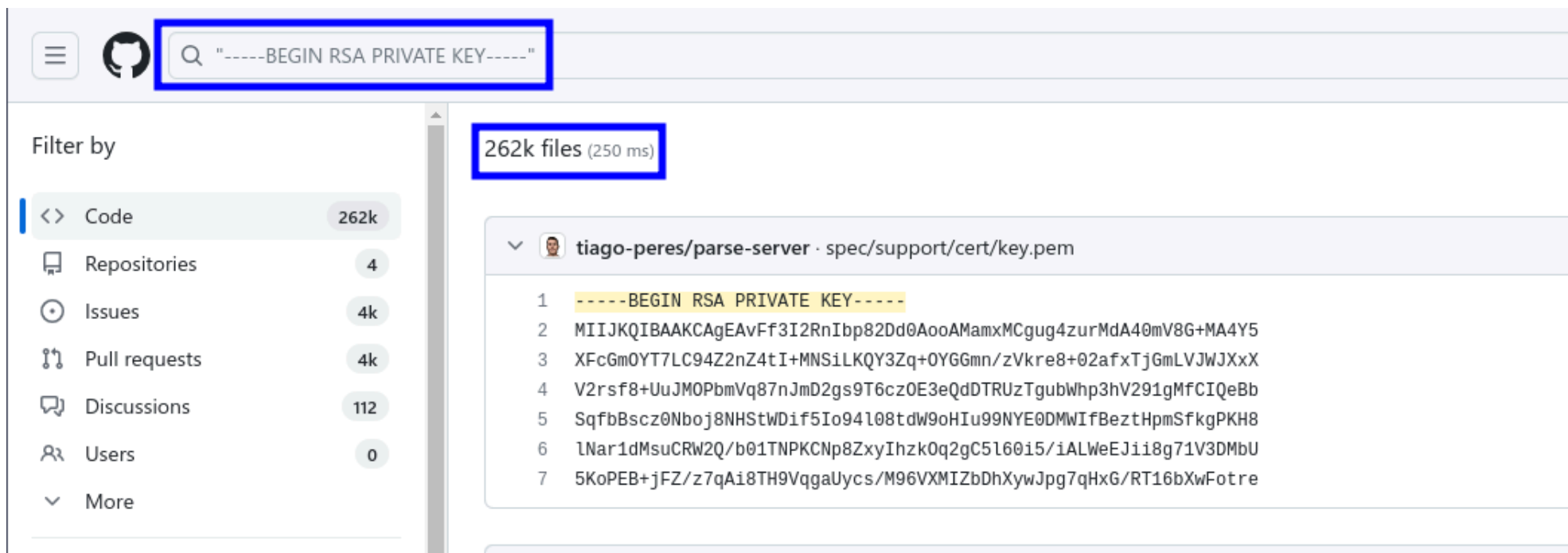


Fig. 9. – Ces clés ne sont évidemment pas toutes valides ou sensibles, mais cela indique la quantité de clés présentes en ligne, certaines par mégarde.

L'algorithme de chiffrement asymétrique le plus connu est **RSA**, du nom de ses trois inventeurs (Rivest, Shamir et Adleman, 1977).

Sa sécurité repose sur la difficulté de **factoriser** un grand nombre en un produit de deux nombres premiers.

La génération d'une paire de clés se fait en quatre étapes:

1. choisir deux grands nombres premiers p et q ;
2. calculer $n = p \times q$ et $\varphi(n) = (p - 1)(q - 1)$;
3. choisir e premier avec $\varphi(n)$;
4. calculer d tel que $e \times d \equiv 1 \pmod{\varphi(n)}$.

La **clé publique** est la paire (e, n) , la **clé privée** la paire (d, n) .

Pour chiffrer un message P (un entier inférieur à n), on calcule $C = P^e \bmod n$; pour déchiffrer, on retrouve $P = C^d \bmod n$.

Les tailles de clés usuelles aujourd'hui sont de 2048, 3072 ou 4096 bits: plus n est grand, plus la factorisation est coûteuse.



Exercices

Soit $p = 3$ et $q = 11$. Vérifier que $n = 33$ et $\varphi(n) = 20$, puis que le couple $e = 3$, $d = 7$ fonctionne. Chiffrer ensuite le message $P = 7$, et vérifier qu'on le retrouve après déchiffrement.

Comme le chiffrement asymétrique est **bien plus lent** que le symétrique, on ne l'utilise en pratique que pour échanger une clé symétrique ou pour signer. C'est le principe du **chiffrement hybride**, utilisé entre autres par TLS.



Aide aux révisions

À retenir: la clé publique sert à chiffrer, la clé privée à déchiffrer. La sécurité de RSA repose sur la difficulté à factoriser $n = p \times q$.

1. Xor
2. Chiffrement par bloc
3. Modes d'opération
4. Échange de clés
5. Cryptographie à clé publique
- 6. Fonctions de hachage**
7. L'aléatoire

6. Fonctions de hachage

Imaginons que vous vouliez partager un gros fichier avec un·e ami·e.

Une fois le partage réalisé (envoi par mail, par *peer-to-peer*, IPoAC...), vous voulez vous assurer que vous avez tous deux la même version: que le fichier n'a pas été altéré durant sa transmission.

Y a-t-il une façon simple de le vérifier ?

6. Fonctions de hachage

Appelons votre version du fichier x (une chaîne de caractères), et la version de votre ami·e y . Le but est de déterminer si $x = y$.

6. Fonctions de hachage

Appelons votre version du fichier x (une chaîne de caractères), et la version de votre ami·e y . Le but est de déterminer si $x = y$.

Une approche naturelle serait de s'accorder sur une fonction déterministe H , calculer $H(x)$ et envoyer le résultat à votre ami·e.

6. Fonctions de hachage

Appelons votre version du fichier x (une chaîne de caractères), et la version de votre ami·e y . Le but est de déterminer si $x = y$.

Une approche naturelle serait de s'accorder sur une fonction déterministe H , calculer $H(x)$ et envoyer le résultat à votre ami·e.

Iel fera alors la même opération avec $H(y)$ et vous pourrez ensuite comparer les résultats.

6. Fonctions de hachage

Pour que cette méthode soit infaillible, il faudrait que chaque entrée unique corresponde toujours à une sortie unique – en d’autres termes, que H soit **injective**. Nous verrons que c’est impossible pour des entrées de longueur arbitraire, mais que l’on peut s’en approcher.

6. Fonctions de hachage

Définition Une fonction de hachage est une fonction qui fait correspondre des chaînes de données arbitraires à une sortie de longueur fixe (appelé *hash* ou empreinte numérique) de manière déterministe, publique et « aléatoire ».

6. Fonctions de hachage

Dans cette définition, les points importants sont:

- chaînes de données arbitraires;
- sortie de longueur fixe (d);
- de manière déterministe: la même entrée donnera toujours la même sortie;
- de manière publique: cette fonction ne nécessite aucun partage de secret;
- «aléatoire»: le véritable aléatoire est très compliqué à obtenir.

6. Fonctions de hachage

La représentation mathématique d'une fonction de hachage est la suivante:

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

où $\{0, 1\}^*$ représente une chaîne de données (des 0 ou des 1) de longueur arbitraire, et $\{0, 1\}^d$ une chaîne de données (0 ou 1) de longueur d . Cette opération est **non-réversible**, ce qui signifie qu'il est impossible de retrouver la donnée originale à partir du hash.

6. Fonctions de hachage

On dit qu'il y a une **collision** dans H lorsque pour une paire d'entrées (x, y) , $x \neq y$, $H(x) = H(y)$.

6. Fonctions de hachage

Les collisions sont généralement considérées comme indésirables mais sont très difficiles à éviter, en raison de la différence de taille entre l'ensemble d'entrée (des chaînes de données de n'importe quelle taille) et l'ensemble de sortie, de taille fixe d (par exemple $d = 256$ bits, soit 32 octets, pour SHA-256).

6. Fonctions de hachage

Les collisions sont néanmoins considérées comme rares grâce à la complexité mathématique des algorithmes de hachage. C'est cette propriété qui rend les collisions suffisamment improbables pour que l'empreinte d'un mot de passe ou d'un fichier soit considérée comme unique.

6.1 Mots de passe

Nous avons vu qu'il est impossible de retrouver les données qui ont permis de générer un hash. Cette propriété rend le hachage idéal pour stocker les mots de passe dans une base de données.

6.1 Mots de passe

En effet, si les développeur·euses ont fait leur travail consciencieusement, en cas de fuite de données les mots de passe ne sont pas en clair mais bel et bien hachés.

6.1 Mots de passe

Cependant, cette protection ne sauve pas si le mot de passe utilisé est un mot de passe faible.

Effectivement, un mot de passe du style password123 peut facilement être récupéré par ingénierie sociale ou OSINT (*Open Source Intelligence*), ou exister dans des bases de données de mots de passe pré-hachés telles que CrackStation¹.

¹<https://crackstation.net/>

6.1 Mots de passe

Les *rainbow tables* sont des structures de données qui permettent de retrouver un mot de passe à partir de son hash, de manière optimisée, en se basant sur des tables pré-calculées (processus de compromis temps-mémoire¹).

¹[https://fr.wikipedia.org/wiki/Compromis_temps-mémoire](https://fr.wikipedia.org/wiki/Compromis_temps-m%C3%A9moire)

6.1 Mots de passe

Afin d'éviter les attaques par *rainbow tables*, les schémas de stockage de mots de passe utilisent un *salt*.

6.1 Mots de passe

Afin d'éviter les attaques par *rainbow tables*, les schémas de stockage de mots de passe utilisent un *salt*.

Il est important de garder à l'esprit qu'un *salt* n'est pas un secret. Il sert simplement à perturber le calcul du hachage, de sorte que la même entrée avec un sel différent donnera deux empreintes digitales différentes, ce qui rend les *rainbow tables* complètement inutiles.

6.1 Mots de passe

Afin d'éviter les attaques par *rainbow tables*, les schémas de stockage de mots de passe utilisent un *salt*.

Il est important de garder à l'esprit qu'un *salt* n'est pas un secret. Il sert simplement à perturber le calcul du hachage, de sorte que la même entrée avec un sel différent donnera deux empreintes digitales différentes, ce qui rend les *rainbow tables* complètement inutiles.

Le salt est concaténé à la fin du mot de passe **en clair**, avant le hachage.



Exercices

Pour illustrer que les attaques par *bruteforce* sont très coûteuses, même contre un algorithme aussi rapide que SHA-256¹, nous pouvons réaliser l'exercice qui suit. Nous avons sous la main le dictionnaire français, comportant 346200 entrées:

```
$ wc -l /usr/share/dict/french
346200 /usr/share/dict/french
```

Nous pouvons écrire un petit script qui, pour chaque entrée du dictionnaire, affichera son hash. On peut rediriger la sortie vers `/dev/null` car elle ne nous servira à rien. Ce qui nous intéresse ici est la durée d'exécution du programme. Le but de l'exercice est de réaliser ce script.

6.1 Mots de passe

¹En pratique, on n'utilise pas SHA-256 pour les mots de passe: on préfère des fonctions lentes et à coût mémoire (bcrypt, scrypt, Argon2).

6.1 Mots de passe

Une fois le script réalisé, lancez-le avec la commande `time` pour mesurer sa durée d'exécution, et avec un outil de notification pour être alerté·e de la fin de son exécution:

```
$ time ./script.sh && dunstify -u critical "script has ended"
```

6.2 Vérification d'intégrité

Comme vu lors de l'introduction du chapitre, une application très importante des algorithmes de hachage est la vérification d'intégrité.

En raison de leurs propriétés déterministes et uniques, les fonctions de hachage sont largement utilisées pour vérifier l'intégrité des fichiers lors de leur partage.

6.2 Vérification d'intégrité

Souvent, les personnes qui partagent des logiciels ou des documents en ligne partagent également le hash du document au moment où il a été publié, et précisent l'algorithme utilisé pour le générer.

Cela permet à toute personne souhaitant récupérer le document de vérifier qu'il s'agit bien du bon. Ce hash est couramment appelé *checksum*.

6.3 Hash tables

Les tables de hachage (*hash tables* ou *hash maps*) constituent une des applications les plus importantes des fonctions de hachage en informatique. Elles permettent de stocker et récupérer des données de manière extrêmement efficace.

6.3 Hash tables

Une table de hachage utilise une fonction de hachage pour transformer une clé (par exemple une chaîne de caractères) en un index dans un tableau. Cet index détermine où stocker ou chercher la valeur associée à cette clé.

Par exemple, pour stocker des informations d'étudiants par leur nom:

Fonction hash: nom $\rightarrow$ index dans le tableau

"Alice" $\rightarrow$ hash("Alice") $\rightarrow$ 3

"Bob" $\rightarrow$ hash("Bob") $\rightarrow$ 7

"Charlie" $\rightarrow$ hash("Charlie") $\rightarrow$ 1

6.3 Hash tables

Avantages :

- **Accès rapide** : recherche, insertion et suppression en $O(1)$ en moyenne;
- **Passage à l'échelle** : les performances restent stables quand le nombre d'éléments augmente;
- **Flexibilité** : peut utiliser n'importe quel type de données comme clé.

6.3 Hash tables

Comme nous l'avons vu, les fonctions de hachage peuvent produire des collisions (même hash pour des entrées différentes). Les tables de hachage doivent gérer ces situations:

- **Chaînage** (*chaining*): Chaque case du tableau contient une liste des éléments ayant le même hash.
- **Adressage ouvert** (*open addressing*): En cas de collision, on cherche la prochaine case libre selon une stratégie définie (sondage linéaire, quadratique, etc.).

6.3 Hash tables

Les tables de hachage sont omniprésentes:

- **Bases de données** : index pour accès rapide aux enregistrements
- **Caches web** : stockage des pages fréquemment consultées
- **Compilateurs** : tables de symboles pour les variables
- **Systèmes de fichiers** : localisation rapide des fichiers
- **Dictionnaires** dans les langages de programmation (Python dict, JavaScript Object, etc.)

6.4 Sécurité des algorithmes

Des institutions officielles telles que la Cour Pénale Internationale utilisent toujours des algorithmes obsolètes¹ pour signer et valider l'authenticité des preuves, comme par exemple MD5², malgré le fait que ces algorithmes soient connus pour être vulnérables depuis plusieurs années.

Dès 1996, des vulnérabilités de collision ont été découvertes dans MD5³ et il est depuis recommandé d'utiliser des algorithmes plus résistants tels que SHA-2 ou SHA-3.

¹<https://www.icc-cpi.int/sites/default/files/RelatedRecords/0902ebd18037cb09.pdf>

²<https://katelynsills.com/law/the-curious-case-of-md5/>

³<https://en.wikipedia.org/wiki/MD5>



Exercices

Vous venez d'outre-passer la DMZ d'une startup du milieu bancaire. En explorant le serveur, vous trouvez le code source de l'application, et un bout de code en particulier vous intéresse: `register.php`.

```
function super_hash($password)
{
    $a = "=ow0pQmcvd3czFGckgSNk1GIuJXd0Vmc";
    $b = strrev($a);
    eval(base64_decode($b));
}
```

Que fait cette fonction ?

6.4 Sécurité des algorithmes



Exercices

Vous parvenez ensuite à dumper la table `admin_accounts` de la base de données:

name	username	hashed_password
Ada	boss	b75bc76e44a3666eadc6aaa6f5e49eff
Margaret	lead_dev	d8704242529ed8335002faca459b0062
Alan	sysadmin	ee27fcee39334f365c1f383bc619487e

Retrouvez les mots de passe de chaque utilisateur·rice.¹

¹Désobfusquer `strrev` puis `base64_decode` permet d'identifier md5. Les empreintes font 32 caractères hexadécimaux; une attaque par dictionnaire (hashcat, john, CrackStation) suffit.



Aide aux révisions

- Une fonction de hachage est déterministe, de sortie de taille fixe, et à sens unique.
- Elle doit résister aux collisions et à la préimage ; pour les mots de passe, on y ajoute un **salt**.
- SHA-256 est rapide : pour les mots de passe, on préfère des fonctions lentes (bcrypt, Argon2).
- Les empreintes servent aussi à vérifier l'intégrité et à indexer des hash tables.

1. Xor
2. Chiffrement par bloc
3. Modes d'opération
4. Échange de clés
5. Cryptographie à clé publique
6. Fonctions de hachage
- 7. L'aléatoire**

7. L'aléatoire

Comme nous avons pu le constater lors de nos découvertes de différents cryptosystèmes, beaucoup ont besoin de nombres aléatoires, ce qui nécessite un processus complexe.

«Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin. »

7.1 Générateurs de nombres aléatoires

En effet, nous ne pouvons pas espérer produire des nombres aléatoires en utilisant une arithmétique prévisible et déterministe. Nous avons besoin d'une source d'aléatoire qui n'est pas une conséquence de règles déterministes.

7.1 Générateurs de nombres aléatoires

Nous allons voir 3 catégories de générateurs de nombres aléatoires:

- les générateurs de nombres aléatoires réels;
- les générateurs de nombres pseudo-aléatoires cryptographiquement sûrs;
- les générateurs de nombres pseudo-aléatoires.

7.1 Générateurs de nombres aléatoires

7.1.1 Générateurs de nombres aléatoires réels

Les vrais générateurs de nombres aléatoires tirent leur caractère aléatoire à partir de processus physiques. Ceux principalement utilisés aujourd'hui sont les:

- processus quantiques;
- processus thermiques;
- dérives des oscillateurs;
- évènement temporels.

7.1 Générateurs de nombres aléatoires

La **désintégration radioactive** est un exemple de processus physique **quantique** utilisé pour produire des nombres aléatoires. Les substances radioactives se désintègrent lentement avec le temps et il est impossible de savoir quand le prochain atome va se désintégrer, ce qui rend ce processus entièrement aléatoire.

7.1 Générateurs de nombres aléatoires

Détecter quand une telle désintégration s'est produite, cependant, est assez facile. En mesurant le temps entre les désintégrations individuelles, nous pouvons produire des nombres aléatoires.

7.1 Générateurs de nombres aléatoires

Le **bruit de grenaille** (*shot noise*) est un autre processus physique **quantique**, basé sur le fait que la lumière et l'électricité sont causées par le mouvement de petits paquets indivisibles: les photons dans le cas de la lumière, et les électrons dans le cas de l'électricité.

7.1 Générateurs de nombres aléatoires

Le **bruit de Nyquist** est un exemple de processus **thermique** utilisé pour produire des nombres aléatoires.

C'est le bruit qui se produit à partir de porteurs de charge (généralement des électrons) se déplaçant à travers un milieu présentant une certaine résistance. Cela provoque un courant minuscule à travers la résistance (ou alternativement, une différence de tension aux bornes de la résistance).

7.1 Générateurs de nombres aléatoires

$$i = \sqrt{\frac{4k_B T \Delta f}{R}}$$
$$v = \sqrt{4k_B T R \Delta f}$$

où:

- Δf est la bande passante;
- T est la température;
- R est la résistance;
- k_B est la constante de Boltzmann.

7.1 Générateurs de nombres aléatoires

On voit que le bruit de Nyquist est assez faible. À température ambiante, avec des hypothèses raisonnables (bande passante de 10 kHz et une résistance de $1\text{k}\Omega$), la tension de Nyquist est de l'ordre de plusieurs centaines de nanovolts.

En arrondissant généreusement à un microvolt (un millier de nanovolts), cela n'est toujours qu'un millième de millième de volt.

7.1 Générateurs de nombres aléatoires

7.1.2 Générateurs de nombres pseudo-aléatoires sûrs

Il existe de nombreux algorithmes et programmes permettant de générer des nombres pseudo-aléatoires cryptographiquement sûrs, mais il est toujours préférable d'utiliser ceux mis à disposition par l'OS:

- `/dev/urandom` sur une machine UNIX;
- `BCryptGenRandom` sous Windows (anciennement `CryptGenRandom`, aujourd'hui obsolète).

Attention toutefois, sous certaines conditions même `/dev/urandom` peut ne pas être idéal tel quel (voir `man urandom`).

7.1 Générateurs de nombres aléatoires

7.1.3 Générateurs de nombres pseudo-aléatoires

Enfin, il existe des algorithmes qui permettent de générer des nombres pseudo-aléatoires, mais qui eux ne sont pas cryptographiquement sûrs. Par exemple **Mersenne Twister**. Pour qu'un algorithme soit considéré comme sûr, il ne faut pas que l'on puisse:

- prédire les prochaines valeurs;
- retrouver les anciennes valeurs.

7.1 Générateurs de nombres aléatoires



Aide aux révisions

- Un bon aléatoire est indispensable : clés, IV, salts, jetons.
- Les générateurs **pseudo**-aléatoires sont déterministes : s'ils sont cassés, tout l'aléatoire l'est.
- On les amorce avec une source **réelle** (bruit physique, désintégration radioactive, /dev/urandom).
- Seuls les générateurs cryptographiquement sûrs (CSPRNG) conviennent en sécurité.

© ⓘ ⓘ Hugo Blanc, 2026

Ce document peut être distribué librement, selon les termes de la version 4.0 de la licence Creative Commons Attribution-ShareAlike: <http://creativecommons.org/licenses/by-sa/4.0/>.

Vous êtes libres de reproduire, distribuer et communiquer ce document au public et de modifier ce document, selon les conditions suivantes :

- Paternité. Vous devez citer le nom de l'auteur original.
- Partage des Conditions Initiales à l'Identique. Si vous modifiez, transformez ou adaptez cette création, vous n'avez le droit de distribuer la création qui en résulte que sous un contrat identique à celui-ci.
- À chaque réutilisation ou distribution, vous devez faire apparaître clairement aux autres les conditions contractuelles de mise à disposition de cette création.

Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits.