

1. Introduction

1.1. Communications sécurisées

- Confidentialité des données transmises : importance
 - Contexte personnel 
 - **Protection de la vie privée** : les données personnelles (médicales, bancaires, communications privées) doivent être gardées confidentielles pour protéger le droit des individus à la vie privée
 - **Protection contre le vol d'identité** : la confidentialité aide à empêcher l'accès non-autorisé au données personnelles, pour réduire les risques de vol d'identité et de fraude
 - **Sécurité personnelle** : elle peut être compromise si des informations sensibles (adresse personnelle, emploi du temps) tombent entre de mauvaises mains

1.1. Communications sécurisées

- Confidentialité des données transmises : importance
 - Contexte professionnel 
 - **Avantage compétitif** : stratégies d'entreprises et propriétés intellectuelles sont critiques pour les compagnies. Les dévoiler pourrait causer des pertes ou permettre leur exploitation par les concurrents
 - **Confiance des clients** : les entreprises qui traitent des données clients doivent maintenir leur confidentialité. Toute fuite serait dommageable pour la réputation et pourrait avoir des répercussions légales
 - **Obligation légale** : pour certaines industries
 - [https://fr.wikipedia.org/wiki/Règlement général sur la protection des données](https://fr.wikipedia.org/wiki/R%C3%A8glement_g%C3%A9n%C3%A9ral_sur_la_protection_des_donn%C3%A9es)

1.1. Communications sécurisées

- Confidentialité des données transmises : importance
 - Contexte gouvernemental 
 - **Sécurité nationale** : la confidentialité des opérations gouvernementales, stratégies de défense et informations classées est vitale pour la sécurité nationale
 - **Relations diplomatiques** : une communication confidentielle entre les gouvernements garantie des négociations et une diplomatie sans interférence
 - **Enquêtes policières et judiciaires** : garder certaines informations confidentielles facilite les enquêtes menées et protège l'identité des informateurs et témoins

1.1. Communications sécurisées

- Risques associés aux communications
 - Interception 
 - **Écoute clandestine** : des individus ou entités non-autorisés peuvent intercepter les communications (e-mails, appels téléphoniques, messages) pour récolter des informations sensibles
 - **Fuite de données** : des pirates peuvent infiltrer des systèmes ou réseaux pour voler des données, en accédant à des informations confidentielles enregistrées
- Falsification 
- **Altération de données** : des agents malveillants peuvent modifier l'information durant sa transmission, générant des information fausses ou fallacieuses
- **Intégrité des données** : elle peut être attaquée, afin d'affecter le processus de prise de décision

1.1. Communications sécurisées

- Risques associés aux communications
 - Imitation 
 - **Spoofing** : falsifier l'identité d'un expéditeur ou d'un destinataire, pour accéder de manière non-autorisée à des informations ou un système
 - **Hameçonnage (phishing)** : faux e-mails ou faux sites web piégeant les individus en les poussant à divulguer des informations sensibles, comme leur identifiants et mots de passes
 - Déni de service 
 - **Attaques DDoS** (attaque par déni de service distribuée) : inonder un système ou un réseau avec un trafic excessif pour rendre des services disponibles aux utilisateurs légitimes
 - **Coupure de service** : toute action interrompant la disponibilité normale d'un service, empêchant l'accès aux informations et systèmes critiques

1.1. Communications sécurisées

- Risques associés aux communications
 - Menaces internes 
 - Menaces de cybersécurité qui proviennent d'**utilisateurs autorisés**, c'est-à-dire d'employés, de fournisseurs ou de partenaires commerciaux, qui abusent, intentionnellement ou non, de leur accès légitime, ou dont les comptes sont détournés par des cybercriminels
 - Menaces physiques 
 - **Le vol ou la perte de matériel** (portables, clés USB) contenant des informations sensibles peut conduire à des accès non-autorisés
 - **Surveillance physique** : l'observation directe des canaux de communication physique, contournant ainsi les mesures de sécurité numérique

- **Définition**

- κρυπτός (*kryptós*), « secret » ; et γράφειν (*graphein*), « écrire »

Discipline de la cryptologie regroupant l'ensemble des techniques pour écrire l'information (communications et les données) de telle manière que seules les parties autorisées peuvent y accéder et la comprendre

La cryptographie se subdivise en chiffrement, hachage et encodage (voir ci-après)

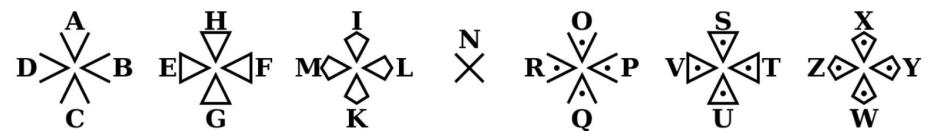
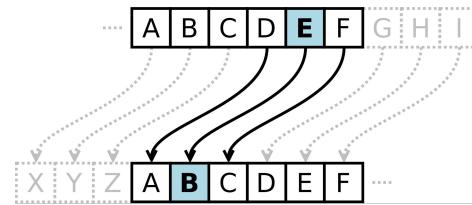
La **cryptologie** englobe également la cryptanalyse, qui est l'analyse de la cryptographie

- La cryptographie se distingue de la **stéganographie** qui fait passer inaperçu un message dans un autre message
(p. ex. message caché dans une image, avec OutGuess)

1.2. Bases de la cryptographie

- Rôles de la cryptographie, dont trois propriétés des données qu'elle vise à garantir
 - **Confidentialité** : encoder les données sorte que, même si elles sont interceptées, elles seront inintelligibles pour les utilisateurs non-autorisés
 - **Intégrité** : maintenir l'intégrité des données en détectant toute altération non-autorisée (voir fonctions de hachage et *checksums*)
 - **Authentification** : les signatures numériques et protocoles d'authentification utilisent des techniques cryptographiques pour garantir que les messages échangés proviennent de sources légitimes
 - **Non-réputation** : les signatures numériques permettent de s'assurer qu'un contrat ne peut être remis en cause par l'une des parties
 - **Transactions sécurisées** : e-commerce, banques en ligne
 - **Protection de la vie privée** : protéger les données et communications personnelles des accès non-autorisés

- Antiquité
 - **Chiffrement par substitution monoalphabétique**
 - Substituer dans un message chacune des lettres de l'alphabet par une autre
 - Chiffrement par décalage (César, ROT13), chiffre des Templiers



- **Chiffrement par transposition (ou permutation)**
 - Changer l'ordre des lettres (anagrammes)
 - Scytale



1.3. Évolution des techniques

- Renaissance
 - **Chiffrement par substitution polyalphabétique** : pour une lettre donnée, son chiffrement ne sera pas toujours le même selon l'endroit du texte
 - **Chiffre d'Alberti** (1404 - 1472)
 - Les substitutions sont définies par le positionnement relatif de deux disques concentriques
 - Ce positionnement peut changer selon l'endroit du texte
 - « dans mon message, j'écrirai un D majuscule et, à partir de ce point, k ne signifiera plus B mais D »



- Renaissance

- **Chiffrement par substitution polyalphabétique**

- **Chiffre de Trithémius** (1462 - 1516)

- On chiffre la première lettre du message clair avec la première ligne, la deuxième lettre avec la deuxième ligne, etc.
 - Cela revient à une suite de décalages de César
 - La première lettre n'est pas décalée, la deuxième est décalée d'un cran dans l'alphabet, la troisième de deux crans, etc.

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | |
| C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | |
| D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | |
| E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | |
| F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | |
| G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | |
| H | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F |
| I | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G |
| J | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H |
| K | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I |
| L | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J |
| M | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K |
| N | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L |
| O | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M |
| P | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
| Q | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| R | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
| S | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
| T | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
| U | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
| V | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |
| W | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U |
| X | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V |
| Y | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |
| Z | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X |

1.3. Évolution des techniques

- Renaissance

- Chiffrement par substitution polyalphabétique**

- Chiffre de Vigenère (1523 - 1596)**

- Introduit la notion de clé

| Lettre en clair | |
|------------------|---|
| Lettre de la clé | A B C D E F G H I J K L M N O P Q R S T U V W X Y Z |
| A | A B C D E F G H I J K L M N O P Q R S T U V W X Y Z |
| B | B C D E F G H I J K L M N O P Q R S T U V W X Y Z A |
| C | C D E F G H I J K L M N O P Q R S T U V W X Y Z A B |
| D | D E F G H I J K L M N O P Q R S T U V W X Y Z A B C |
| E | E F G H I J K L M N O P Q R S T U V W X Y Z A B C D |
| F | F G H I J K L M N O P Q R S T U V W X Y Z A B C D E |
| G | G H I J K L M N O P Q R S T U V W X Y Z A B C D E F |
| H | H I J K L M N O P Q R S T U V W X Y Z A B C D E F G |
| I | I J K L M N O P Q R S T U V W X Y Z A B C D E F G H |
| J | J K L M N O P Q R S T U V W X Y Z A B C D E F G H I |
| K | K L M N O P Q R S T U V W X Y Z A B C D E F G H I J |
| L | L M N O P Q R S T U V W X Y Z A B C D E F G H I J K |
| M | M N O P Q R S T U V W X Y Z A B C D E F G H I J K L |
| N | N O P Q R S T U V W X Y Z A B C D E F G H I J K L M |
| O | O P Q R S T U V W X Y Z A B C D E F G H I J K L M N |
| P | P Q R S T U V W X Y Z A B C D E F G H I J K L M N O |
| Q | Q R S T U V W X Y Z A B C D E F G H I J K L M N O P |
| R | R S T U V W X Y Z A B C D E F G H I J K L M N O P Q |
| S | S T U V W X Y Z A B C D E F G H I J K L M N O P Q R |
| T | T U V W X Y Z A B C D E F G H I J K L M N O P Q R S |
| U | U V W X Y Z A B C D E F G H I J K L M N O P Q R S T |
| V | V W X Y Z A B C D E F G H I J K L M N O P Q R S T U |
| W | W X Y Z A B C D E F G H I J K L M N O P Q R S T U V |
| X | X Y Z A B C D E F G H I J K L M N O P Q R S T U V W |
| Y | Y Z A B C D E F G H I J K L M N O P Q R S T U V W X |
| Z | Z A B C D E F G H I J K L M N O P Q R S T U V W X Y |

Texte en clair : j'adore écouter la radio toute la journée
Clé répétée : M USIQU EMUSIQU EM USIQU EMUSI QU EMUSIQU
^ ^ ^
| ||Colonne 0, ligne I : on obtient la lettre W.
| |Colonne D, ligne S : on obtient la lettre V.
| Colonne A, ligne U : on obtient la lettre U.
Colonne J, ligne M : on obtient la lettre V.

Texte chiffré : V'UVWHY IOIMBUL PM LSLYI XAOLM BU NAOJVUY
Clé répétée : M USIQU EMUSIQU EM USIQU EMUSI QU EMUSIQU
^ ^ ^
| ||Ligne I, on cherche W : on trouve la colonne 0.
| |Ligne S, on cherche V : on trouve la colonne D.
| Ligne U, on cherche U : on trouve la colonne A.
Ligne M, on cherche V : on trouve la colonne J.

Arithmétique modulaire

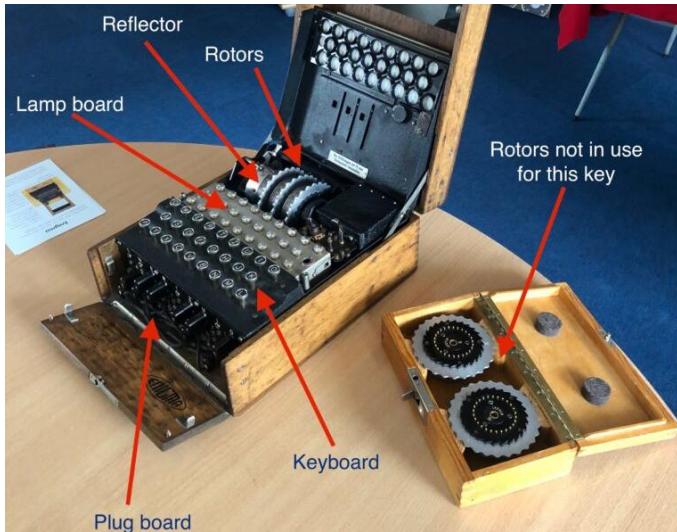
$$\text{Chiffré}[i] = (\text{Texte}[i] + \text{Clés}[i]) \bmod 26$$

$$\text{Texte}[i] = (\text{Chiffré}[i] - \text{Clés}[i]) \bmod 26$$

1.3. Évolution des techniques

- Seconde guerre mondiale
 - Enigma

- Famille de machines de chiffrement électromécaniques
- Version commerciale brevetée par Arthur Scherbius en 1918
- Plus tard, version militaire utilisée par l'armée allemande



Lorsqu'une touche du clavier est pressée :

- L'ampoule correspondant à la lettre chiffrée s'allume
- Le système de rotors tourne d'un cran
 - Presser la même lettre deux fois consécutives donnera deux lettres chiffrées différentes (chiffrement polyalphanumérique)

1.3. Évolution des techniques

- Seconde guerre mondiale
 - Enigma : design et combinaisons
 - 3 rotors crantés, sur 3 emplacements (droite, milieu, gauche) à choisir parmi 5 rotors différents
 - $5 \times 4 \times 3 = 60$
 - Chaque rotors possède 26 crans
 - $26 \times 26 \times 26 = 17\,576$
 - Tableau de connexions avec 10 câbles : permet de définir des permutations pour 10 paires de lettres (version militaire seulement)
 - $$\frac{26!}{6! \times 10! \times 2^{10}} = 150\,738\,274\,937\,250$$
 - Total = 158 962 555 217 826 360 000 configurations possibles

1.3. Évolution des techniques

- Seconde guerre mondiale
 - Enigma : **fonction de chiffrement involutive** (*self-reciprocal cipher*)
 - Les processus de chiffrement et déchiffrement sont les mêmes
 - On déchiffre en tapant le message chiffré sur une machine Enigma avec la même configuration que celle ayant servi au chiffrement
 - Propriété garantie par l'utilisation d'un **réflecteur** à la suite du dernier rotor
 - Facilite l'utilisation
 - Mais empêche de substituer une lettre à elle-même dans le texte chiffré
→ Vulnérabilité à une « attaque à texte clair connu »
 - Conception de la « bombe » par Alan Turing pour trouver la configuration d'Enigma utilisée, c.-à-d. trouver la clé
 - Note : les Alliés finir par capturer des exemplaires de ces machines...

1.3. Évolution des techniques

- Cryptologie moderne
 - **1940s - Avènement de l'informatique** : (classique → moderne) les ordinateurs révolutionnent la cryptographie, permettant des calculs plus rapides et des algorithmes plus complexes
 - **1970s - Développements mathématiques** : concepts comme la théorie des nombres, les structures algébriques et la théorie de la complexité
 - **1976 - Cryptographie asymétrique** : introduction généralement attribuée à Whitfield Diffie et à Martin Hellman
 - **1977 - Algorithme RSA** : développement par Rivest, Shamir et Adleman, aujourd'hui fondamental pour le chiffrement et les signatures numériques
 - **1980s - Fonctions de hachage cryptographiques** : leur émergence offre des moyens de vérifier l'authenticité et l'intégrité des données
 - **1990s - Cryptanalyse différentielle et linéaire** : développement de techniques capables de casser des algorithmes de chiffrement symétriques
 - **2001 - Advanced Encryption Standard (AES)** : adoption, en remplacement de Data Encryption Standard (DES), du fait de sa robustesse et de son efficience

- Informatique quantique et cryptographie post-quantique 

 - **Défis posés** : le développement des ordinateurs quantiques constitue une menace pour les méthodes de chiffrement actuelles, car ils pourraient résoudre, de manière efficiente, les problèmes mathématiques sur lesquels ces algorithmes reposent
 - RSA : décomposition en facteurs premiers
 - (EC)DH : problème du logarithme discret
 - **Solutions explorées** : développement d'algorithmes à résistance quantique basés sur des problèmes d'optimisation sur les réseaux euclidiens
 - Problème des vecteurs courts (SVP)
 - Problème de l'apprentissage avec erreur (LWE)

- Blockchain et crypto-monnaies 
 - **Défis posés** : résoudre les problèmes d'extensibilité (*scalability*), d'anonymat et de sécurité, sans compromettre la décentralisation
 - **Solutions explorées** : nouveaux mécanismes de consensus et techniques comme la ***zero-knowledge proof***
- Chiffrement homomorphe 
 - Algorithme permettant d'effectuer certaines opérations mathématiques sur des données chiffrées sans avoir à les déchiffrer d'abord
 - Permet de confier des calculs à un agent externe, sans que les données ni les résultats soient accessibles à cet agent
 - **Défis posés** : améliorer l'efficience
 - **Solutions explorées** : concernent principalement le *cloud computing*

- **Chiffrement** (*encryption*) : processus de conversion d'un message en clair, m (données intelligibles) en un message chiffré c (données inintelligibles), en utilisant un algorithme E et une clé k
 - $E(k_e, m) = c$
 - Déterministe ou probabiliste
- **Déchiffrement** (*decryption*) : processus réciproque du chiffrement, transformant c en m en utilisant une clé appropriée
 - $D(k_d, c) = m$
 - ! Toujours déterministe
- **(E, D)** : cryptosystème
- **Primitive cryptographique** : opération fondamentale cryptographique, de bas niveau et bien établi (*building blocks*), sur la base de laquelle est bâti tout système de sécurité informatique, algorithme ou protocole
 - Exemples : Chiffrement de flux, par bloc, fonction de hachage, MAC, PRNG, RSA, DH

- **Cryptanalyse** : ensemble des techniques d'analyse des systèmes cryptographiques, ayant pour but de trouver des vulnérabilités et d'être capable de déchiffrer sans connaître la clé. Le processus par lequel on tente de comprendre un message en particulier est appelé une **attaque**.
- **Adversaire** : entité cherchant à casser la sécurité d'un système cryptographique
 - Cryptanalyste : adversaire théorique (individu ou algorithme)
 - Attaquant : quelqu'un essayant activement d'exploiter des failles du système cryptographique
- **Alice et Bob** : pour les « personne A » et « personne B » qui cherchent à communiquer
 - D'autres noms sont utilisés, comme Eve (*eavesdropper*), Mallory (*malicious attacker*), Oscar (*opponent*) ou Trudy (*intruder*)



- **Chiffre (cipher) vs code** : le chiffrement d'un message s'opère au niveau des caractères de celui-ci, caractères individuels ou bien petits groupes de caractères. Un code, lui, transforme un message par remplacement des mots ou des phrases. Il opère donc au niveau de la signification du message.
 - Un message peut être encodé avant chiffrement pour compliquer la cryptanalyse
 - **Nécessité d'un livre-code** ou d'une table de correspondance
 - Exemples :
« Les sanglots longs des violons de l'automne... » = « Sabotez les voies ferroviaires ! »

| | |
|-------|----------------|
| 6706 | reichlich |
| 13850 | finanziell |
| 12224 | unterstützung |
| 6929 | und |
| 14991 | einverständnis |

Extrait du décodage du télégramme diplomatique Zimmermann (1917)

- **Cryptographie symétrique** (ou à clé secrète) : la même clé est utilisée pour le chiffrement et le déchiffrement
 - Problématique de la distribution des clés, qui doit être sécurisée
 - Chiffrement par flot (ou de flux, ou continu) : opère sur un flux continu de données (*i.e.* pas de découpage). Exemples : A5/1, RC4
 - Chiffrement par bloc : opère sur des données découpées en blocs de taille généralement fixe. Exemples : DES, AES
 - Code d'authentification de message (MAC) : code accompagnant des données dans le but d'assurer l'intégrité de ces dernières

- **Cryptographie asymétrique** (ou à clé publique) : système cryptographique utilisant des paires de clés, une publique pour le chiffrement, une privée pour le déchiffrement
 - Chiffrement asymétrique (RSA, DH, ...)
 - Schéma de signature numérique : mécanisme permettant d'authentifier l'auteur d'un document électronique et d'en garantir la non-répudiation
- **Cryptographie hybride** : combine
 - un chiffrement asymétrique (plus sécurisé) pour l'échange d'une clé symétrique,
 - et un chiffrement symétrique (plus rapide) pour les données transmises.
 - souvent couplées avec un mécanisme d'authentification
(p. ex. TLS, *Transport Layer Security*)



2. Chiffrements symétriques

- **Chiffre de Vernam** (1917) : XOR entre une clé secrète k et le message clair m pour produire le chiffré c

$$m \oplus k = c$$

Chiffrement

$$\begin{array}{r} 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ m \\ \oplus \\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ k \\ \hline =\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ c \end{array}$$

Déchiffrement

$$\begin{array}{r} 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ c \\ \oplus \\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ k \\ \hline =\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ m \end{array}$$

On peut également retrouver k à partir de m et c : $m \oplus c = k$

2.1. Masque jetable

- **Vulnérabilité** : utiliser deux fois la même clé k pour deux messages clairs m_1 et m_2 permet, à quelqu'un qui aurait intercepté m_2 , c_1 et c_2 , de déchiffrer m_1 sans la clé k

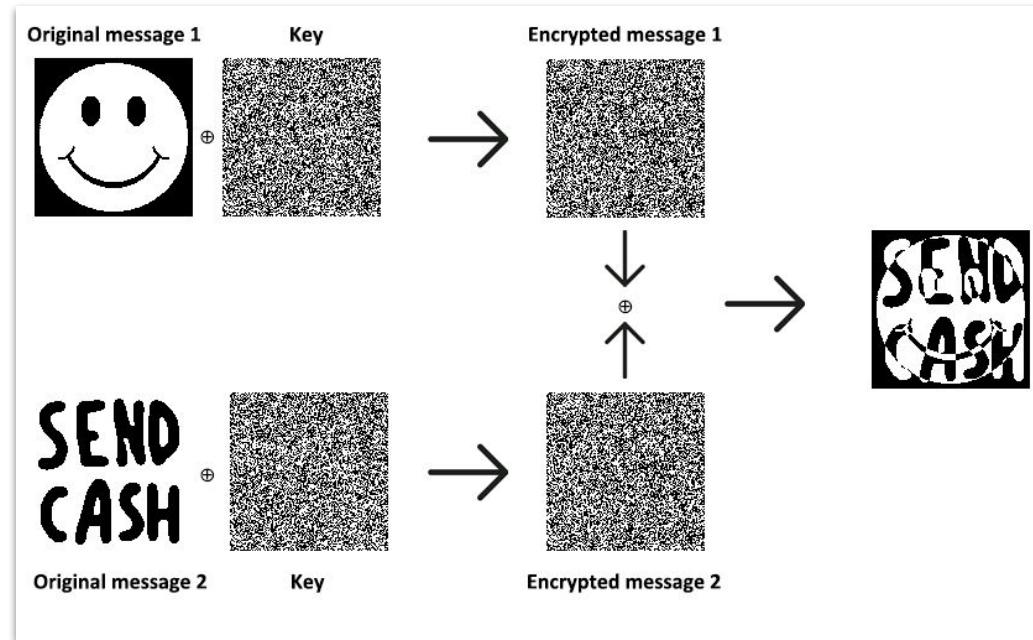
$$m_1 \oplus k = c_1$$

$$m_2 \oplus k = c_2$$

$$m_1 \oplus c_1 = m_2 \oplus c_2$$

$$m_1 = (m_2 \oplus c_2) \oplus c_1$$

Ci-contre : par commutativité, on a aussi $m_1 \oplus m_2 = c_1 \oplus c_2$
Cela crée une vulnérabilité en interceptant uniquement c_1 et c_2 (analyse fréquentielle, analyse par *crib*)



- La sécurité du chiffrement repose donc sur une **utilisation unique de la clé** (ou masque) → Masque jetable ou ***one-time pad (OTP)***
- La fonction **XOR** nécessite que la clé soit de la **même longueur que le message**
Très peu pratique (→ chiffréments de flux), surtout si la clé doit être gardée secrète

- Si la clé secrète k (à usage unique et de même longueur que m) **est parfaitement aléatoire**, alors OTP permet un « secret parfait » (*perfect secrecy*), concept introduit par Claude Shannon (1949)
OTP est le seul algorithme offrant une sécurité parfaite

Aucune information* sur le message clair m ou la clé k ne peut être extraite** du chiffré c , même en disposant d'une puissance de calcul illimitée
→ OTP résiste aux attaques par force brute avec un ordinateur quantique

- Clé parfaitement aléatoire : à chaque position de la séquence, les valeurs 0 et 1 sont équiprobables
→ Distribution uniforme (discrète) : $k \sim U(0^l, 1^l)$, où l est la longueur de la clé

*Information telle que définie par Claude Shannon

**La longueur de m est cependant connue et égale à celle de c

- Exemple pour $k \in K = \{0, 1\}^3$

$$\begin{array}{rclclclcl} \oplus & 111 & c & \oplus & 111 & c & \oplus & 111 & c \\ \oplus & 000 & k_1 & = & 001 & k_2 & = & 011 & k_3 \\ = & 111 & m_1 & = & 110 & m_2 & = & 100 & m_3 \\ & & & & & & & & \\ \oplus & 111 & c & \oplus & 111 & c & \oplus & 111 & c \\ \oplus & 110 & k_5 & = & 100 & k_6 & = & 101 & k_7 \\ = & 001 & m_5 & = & 011 & m_6 & = & 010 & m_7 \end{array}$$

$$\begin{array}{rclclclcl} \oplus & 111 & c & \oplus & 111 & c \\ \oplus & 000 & k_4 & = & 111 & k_4 \\ = & 000 & m_4 & = & 111 & m_4 \\ & & & & & \\ \oplus & 111 & c & \oplus & 111 & c \\ \oplus & 010 & k_8 & = & 010 & k_8 \\ = & 101 & m_8 & = & 101 & m_8 \end{array}$$

Avec une clé **k inconnue de l'adversaire** et générée de manière parfaitement aléatoire, le message chiffré c « 111 » pourrait résulter du chiffrement de tous les messages clairs possibles (m_1 à m_8) de façon équiprobable (1 chance sur 8).

Propriété : un **XOR** entre n'importe quelle séquence et une séquence aléatoire produit une séquence également aléatoire.

2.1. Masque jetable

$$\Pr[E(k, m_0) = c] = \Pr[k \oplus m_0 = c] \quad (1)$$

$$= \frac{|k \in \{0, 1\}^m : k \oplus m_0 = c|}{\{0, 1\}^m} \quad (2)$$

$$= \frac{1}{2^m} \quad (3)$$

$$\Pr[E(k, m_1) = c] = \Pr[k \oplus m_1 = c] \quad (4)$$

$$= \frac{|k \in \{0, 1\}^m : k \oplus m_1 = c|}{\{0, 1\}^m} \quad (5)$$

$$= \frac{1}{2^m} \quad (6)$$

donc $\Pr[E(k, m_0) = c] = \Pr[E(k, m_1) = c]$ $\forall m_0, m_1 \in M$, avec $|m_0| = |m_1|$
 $\forall c \in C$
avec $M = K = \{0, 1\}^m$

- **Convention d'écriture**
 - M : espace des messages en clair
 - $M = \{0, 1\}^l$
 - $|M| = 2^l$
 - C : espace des messages chiffrés
 - K : espace des clés
- La sécurité parfaite d'OTP requiert que $|K| \geq |M|$

- **Contenu en information** (ou auto-information)

- Défini par Claude Shannon, 1948

Soit une variable discrète X qui prend une valeur x

(p. ex. la valeur donnée à une position sur une séquence numérique)

Le contenu en information $I_X(x)$ de cet évènement est :

$$I_X(x) := -\log [p_X(x)] = \log \left(\frac{1}{p_X(x)} \right)$$

avec $p_X(x) \neq 0$ et $I_X(x) \in [0, +\infty]$

En utilisant un logarithme de base 2, l'unité de $I_X(x)$ est le shannon (Sh) ou bit ;
le hartley (Hart) pour la base 10 ; le *natural unit of information* (nat) pour la base e

- **Entropie de Shannon**

- Elle correspond au contenu en information moyen   sur toutes les positions de la s  quence. Elle s'  crit $H(X)$.

$$\begin{aligned} H(X) &= \sum_x -p_X(x) \log p_X(x) \\ &= \sum_x p_X(x) I_X(x) \\ &\stackrel{\text{def}}{=} E[I_X(X)], \end{aligned}$$

avec $H(X) \in [0, +\infty]$

- Pour une s  quence de longueur donn  e, **l'entropie de Shannon est maximum** si les diff  rentes valeurs possibles    chaque position sont equiprobables, c.-  -d. si la s  quence est parfaitement al  atoire (loi uniforme ; **maximum d'incertitude**)
Note : maximum mais pas ∞ , car s  quence de longueur finie

- **Entropie de Shannon**

- Exemple : entropie de la séquence de valeurs décimales « 31100427 »

$$p(0) = 2/8 = 0,25$$

$$p(1) = 2/8 = 0,25$$

$$p(2) = 1/8 = 0,125$$

$$p(3) = 1/8 = 0,125$$

$$p(4) = 1/8 = 0,125$$

$$p(5) = 0/8 = 0$$

$$p(6) = 0/8 = 0$$

$$p(7) = 1/8 = 0,125$$

$$p(8) = 0/8 = 0$$

$$p(9) = 0/8 = 0$$

$$\begin{aligned} H(X) &= \\ -0,25 \log_2(0,25) &+ \\ -0,25 \log_2(0,25) &+ \\ -0,125 \log_2(0,125) &+ \\ -0,125 \log_2(0,125) &+ \\ -0,125 \log_2(0,125) &+ \\ -0,125 \log_2(0,125) &+ \\ = 2,125 \text{ bits} \end{aligned}$$

• Entropie conditionnelle

- Pour deux séquences X et Y :

$$H(Y|X) = - \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)}$$

- $H(Y|X) \leq H(Y)$

$H(Y|X) = H(Y) \Leftrightarrow Y$ et X indépendantes

- Pour un message M et son chiffré C , on définit ainsi :

- Sécurité parfaite : $H(M|C) = H(M)$

- Connaître C n'apporte aucune information sur M (sauf sa longueur)
- La probabilité qu'un adversaire* casse le chiffrement est nulle

- Sécurité inconditionnelle : $H(M|C) \leq H(M)$

- La probabilité qu'un adversaire* casse le chiffrement est négligeable

* disposant d'une puissance de calcul illimitée

- OTP requiert une clé parfaitement aléatoire
 - Peut être obtenue grâce à un générateur de nombres aléatoires physique
True random number generator (TRNG)
Basé sur la variation aléatoire de propriétés physiques : p. ex. *timestamp*



Le mur de lampes à lave de Cloudflare

Sur les microprocesseurs de la famille x86 : instruction **RDSEED**

- **Limitation** : produisent un nombre limité de bits aléatoires par seconde

• Générateur de nombres pseudo-aléatoires

Pseudorandom number generator, PRNG

Algorithme qui génère une séquence de nombres qui apparaît aléatoire, mais qui est déterminée par une valeur initiale appelée « graine » (seed)

⚠ Sortie **déterministe** → **même graine = même sortie**

- Exemple : **générateur congruentiel linéaire**

$$X_{n+1} = (a \cdot X_n + c) \bmod m,$$

où X_n est le nombre pseudo-aléatoire courant,
et a , c et m sont des constantes, $m > 0$

(voir le théorème de Hull-Dobell pour les valeurs qui maximisent la période)

Sans nécessairement être aléatoire (c.-à-d. générée par un TRNG), la graine doit être choisie de sorte à ne pas pouvoir être prédite.

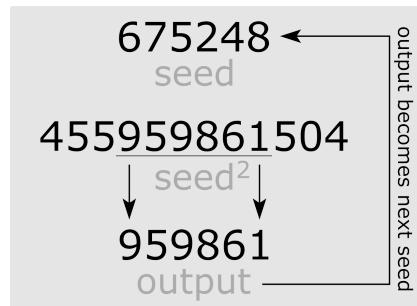
La valeur de X_0 est celle de la graine, à partir de laquelle la formule se répète de façon récurrente, pour générer une séquence de nombres.

P. ex., pour le **générateur d'UNIX**, $a = 1103515245$, $c = 12345$ et $m = 2^{31}$

Limitation : un mauvais choix des constantes peut générer une suite périodique

- **Générateur de nombres pseudo-aléatoires**

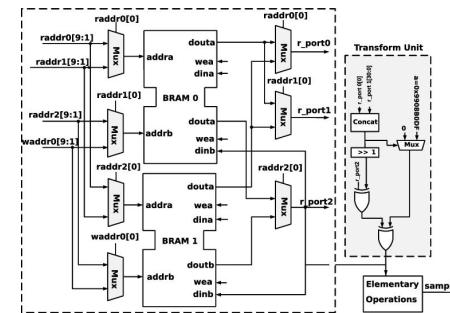
-  Une **périodicité** dans la clé pseudo-aléatoire compromet la sécurité, de la même manière qu'une **réutilisation de la même clef**
- Autres exemples :



Méthode du carré médian

Pour une graine de $2n$ chiffres, la période maximum est de 10^n (un peu plus, si nombre impair de chiffres)

John von Neumann, 1949



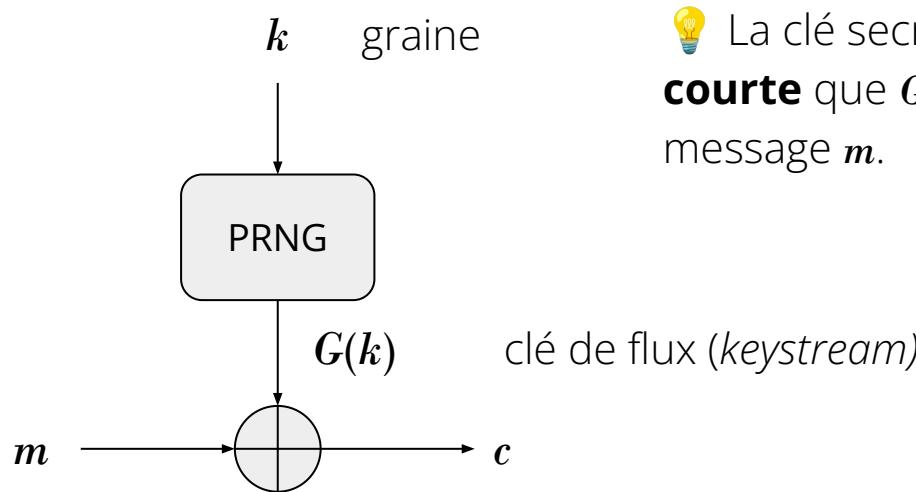
Mersenne Twister (MT19937)

Sa période maximum est de $2^{19937} - 1$. Mais **insuffisant pour la cryptographie**, car il existe des algorithmes qui permettent d'en prédire le comportement.



`import random`

- Architecture générale



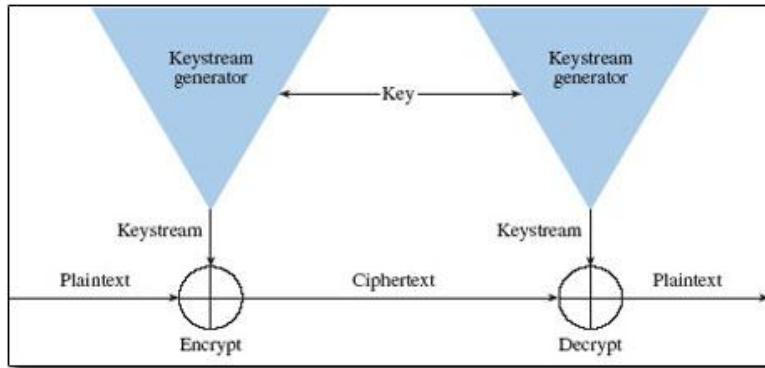
$$E(k, m) = m \oplus G(k)$$

$$D(k, c) = c \oplus G(k)$$

💡 La clé secrète échangée est k . Elle est **beaucoup plus courte** que $G(k)$ qui est de la même longueur que le message m .

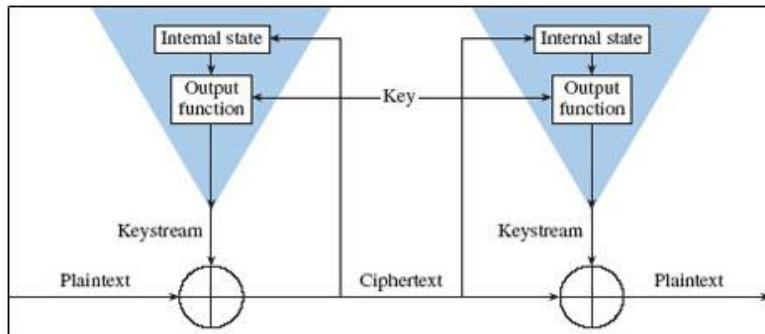
On peut donc considérer un chiffrement continu comme un **XOR** combiné à un PRNG. Ce dernier doit impérativement être **sécurisé pour la cryptographie**.

- Deux catégories



Synchrone

Calcule le flux de clé en fonction de la clé secrète k , mais indépendamment du texte clair m et du chiffré c .



Asynchrone (ou *self-synchronizing*)

Calcule chaque bit du flux de clé comme une fonction de k et des n précédents bits de c . Avantage : le récepteur se synchronise avec le générateur après réception de n bits de c , ce qui facilite la correction d'erreurs. Désavantage : 1 bit d'erreur à la transmission entraînera n bits d'erreurs pour le récepteur (propagation d'erreur).

- **Générateur de nombres pseudo-aléatoires sécurisé**

Cryptographically secure PRNG (CSPRNG)

- **RC4 (Rivest Cipher 4)**, 1987

- Utilisé depuis les premières versions des protocoles SSL/TLS, WEP, WPA
 - Génération d'un **flux d'octets** pseudo-aléatoires par modifications itératives (permutations) de l'état interne (vecteur de 256 octets), dépendantes de la clé secrète
 - ! Beaucoup de vulnérabilités, dont corrélations entre graine et flux

- **Salsa20** et **ChaCha**, 2008

- Protocoles TLS, IPsec, SSH et OpenVPN

- **Trivium**, A5/2 (2G), E0 (Bluetooth)

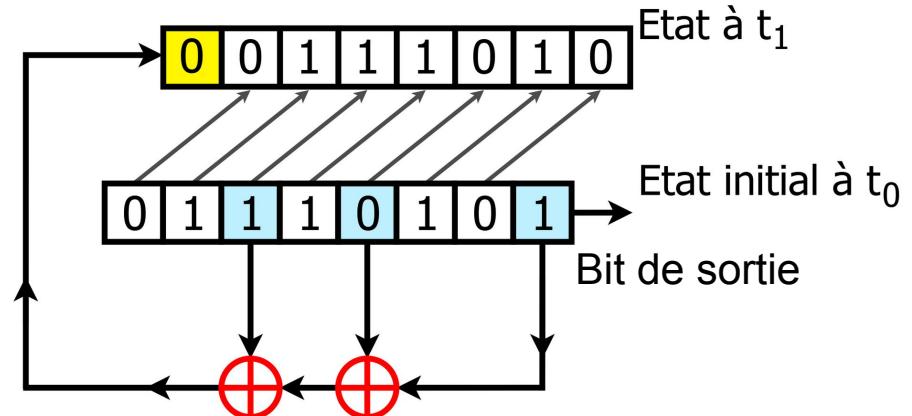
- Basés sur des **registres à décalage à rétroaction linéaire (LFSR)**...

- **Registre à décalage à rétroaction linéaire**

Linear feedback shift register (LFSR)

- Registre à décalage de type SIPO (*Serial In - Parallel Out*) dans lequel un ou plusieurs « étages » du registre subissent une transformation pour être réinjectés en entrée de celui-ci

- Il est dit de longueur r lorsqu'il est composé de r éléments, appelés « étages » ou « cellules », le contenu de l'ensemble de ces éléments à un instant t est **l'état du LFSR** à ce moment.



- À chaque top d'horloge, le contenu d'un étage est transféré au suivant et le premier est rempli (**bit de rétroaction**) par le résultat d'une **fonction linéaire** qui prend en compte **l'état d'un ou de plusieurs étages**.

- **Registre à décalage à rétroaction linéaire**

Linear feedback shift register (LFSR)

- Seulement utilisé comme **élément** de chiffrement de flux, car rétro-ingénierie à partir de la sortie
 - Combinaisons non-linéaires de LFSRs (p. ex. dans le chiffrement **Grain**)
- Convient peu à une implémentation logicielle (beaucoup de boucles)
→ Très efficace en **implémentation matérielle**
- *maximum length sequence* (MLS) : suite périodique qui explore toutes les valeurs pouvant être produites par un LFSR.
S'il comporte r bascules (*flip-flops*), $2^r - 1$ valeurs sont parcourues, l'état « tout à zéro » étant exclu, car il ne génère jamais de changement

- **Preuve de sécurité**

Démonstration qu'un schéma cryptographique respecte des propriétés de sécurités spécifiques, comme la confidentialité, l'intégrité et l'authenticité.

Il existe différents types de preuves cryptographiques, dont :

- Théorie de l'information
- Argument hybride 
- Preuve par jeu
- Réduction polynomiale

• Indistinguabilité calculatoire

- Deux distributions de probabilités sont calculatoirement indistinguables, s'il n'existe pas d'algorithme **efficace** qui puisse les discerner de manière **significative**.
 - A : algorithme probabiliste terminant en temps polynomial (proportionnel à la longueur de la séquence en entrée x)
 - Aussi appelé distinguiseur (*distinguisher*) ou test statistique
 - $A(x) = 1$: la sortie (booléen) de A vaut 1 (ou **Vrai**), s'il détecte que l'entrée x a été échantillonnée à partir d'**une distribution plutôt que l'autre**
 - Deux distributions D_n et E_n sont calculatoirement indistinguables si l'**avantage** $\delta(n)$ de l'adversaire est négligeable en fonction du « paramètre de sécurité » n :

$$\delta(n) = \left| \Pr_{x \leftarrow D_n} [A(x) = 1] - \Pr_{x \leftarrow E_n} [A(x) = 1] \right|$$

- Si la distribution est générée par un PRNG, n sera la longueur de la graine ; Dans RSA, n sera la valeur du module

• Indistinguabilité calculatoire

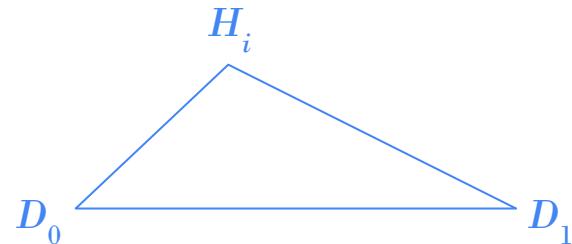
-  La question posée par le calcul de l'avantage est : est-ce que A se comporte différemment selon qu'on lui donne une entrée échantillonnée à partir de D_n ou bien de E_n ? Si oui, cette différence est-elle négligeable (notation : $D_n \approx E_n$), c.-à-d. est que $\delta(n) \rightarrow 0$ quand $n \rightarrow \infty$?
- **Fonction négligeable** : une fonction $f(n)$ est considérée comme négligeable si, \forall polynôme positif $\text{poly}(n)$, \exists une valeur suffisamment grande de n tq $f(n) < \frac{1}{\text{poly}(n)}$ c.-à-d. si $f(n) \rightarrow 0$ plus vite que l'inverse de n'importe quel polynôme
- Indistinguabilité calculatoire est donc une relaxation de l'indistinguabilité statistique prenant en compte le fait que la puissance de calcul des algorithmes est limitée (notions de complexité)

- **Argument hybride**

Pour montrer l'indistinguabilité calculatoire de deux distributions D_0 et D_1 , on définit une séquence de distributions hybrides tq $D_0 := H_0, H_1, \dots, H_n =: D_1$ qui permet de transformer graduellement la distribution D_0 en la distribution D_1 . L'adversaire \mathcal{A} ne pourra lire que les n premières hybrides (entier au plus polynomial).

Si $H_i \approx H_{i+1}, \forall 0 \leq i < n$, alors $D_0 \approx D_1$

[Note](#) : conséquence de l'inégalité triangulaire



Dans une preuve de sécurité d'un chiffrement par flot, D_0 serait la sortie du PRNG et D_1 une séquence binaire parfaitement aléatoire.

- **Argument hybride**

La manière de définir les hybrides dépendra du PRNG dont on souhaite prouver la sécurité cryptographique.

À titre d'exemple, pour un générateur et sa graine $G(k)$:

- $H_0 : G(k)$
- $H_1 : G(k) \parallel G(G(k))$
- $H_2 : G(k) \parallel G(G(k)) \parallel G(G(G(k)))$
- ...
- $H_n : G(k) \parallel G(G(k)) \parallel \dots \parallel G^{(n)}(k)$
- $H_{n+1} : U \parallel U \parallel \dots \parallel U$

avec U représentant une séquence parfaitement aléatoire (uniforme)

- **Exemple** : soit $G : \{0, 1\}^n \rightarrow \{0, 1\}^l$ un PRNG sécurisé (avec $l = f(n)$, fonction polynomiale) ; soit un hybride $G'(k) = G(k) \| G(k)$ et soit la distribution uniforme $U(0^{2l}, 1^{2l})$
Question : la sortie de $G'(k)$ est-elle calculatoirement indistinguabile de U ?

Réponse : On peut définir un distingueur A , qui détecte que la séquence $x \leftarrow G'(k)$ (et renvoie donc Vrai), en deux étapes :

1. Coupe x en 2 segments de même longueur tq $x_1 \| x_2 = x$
2. $A(x) = \text{Vrai}$ si $x_1 = x_2$

Calcul de l'avantage de A :

$$\delta(n) = |P_{x \leftarrow G'(k)}[A(x) = \text{Vrai}] - P_{x \leftarrow U}[A(x) = \text{Vrai}]|$$

Par la définition même de A , on a : $P_{x \leftarrow G'(k)}[A(x) = \text{Vrai}] = 1$

$$P_{x \leftarrow U}[A(x) = \text{Vrai}] = ?$$

- **Intuition** avec les plus petites valeurs possibles : 2 bits

Un générateur aléatoire peut générer $x \in \{00, 01, 10, 11\}$

Mais $G'(k)$ ne peut générer que $x \in \{00, 11\}$

$$\mathbf{P}_{x \leftarrow U} [A(x) = \text{Vrai}] = 2/4 = \frac{1}{2}$$

$$\delta(n) = | \mathbf{P}_{x \leftarrow G'(k)} [A(x) = \text{Vrai}] - \mathbf{P}_{x \leftarrow U} [A(x) = \text{Vrai}] | = 1 - \frac{1}{2} = \frac{1}{2}$$

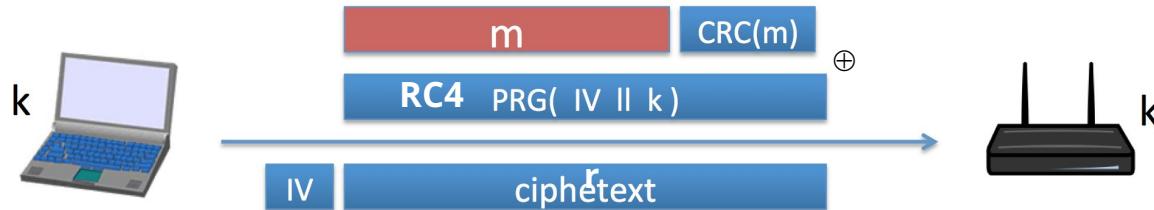
- Pour toute longueur l de x

$$\mathbf{P}_{x \leftarrow U} [A(x) = \text{Vrai}] = 2^l / 2^{2l} = 2^{-l}$$

$$\delta(n) = | 1 - 2^{-l} | = | 1 - 2^{-f(n)} | > \frac{1}{\text{poly}(n)} \quad \text{non-négligeable ; } G'(k) \text{ n'est pas sécurisé}$$

- **Limitations**

- **802.11b WEP:**



La même clé k était réutilisée ; seul le IV changeait à chaque message, mais était trop court (24 bits) → Utiliser WPA3 pour sécuriser les réseaux WiFi

- **Chiffrement de disque :**

- Des erreurs dans le chiffré peuvent empêcher le déchiffrement
 - Beaucoup de données : la périodicité de la clé de flux ou sa réutilisation cause l'apparition de motifs répétés dans le chiffré, détectables en cryptanalyse
 - ↳ Utiliser un **chiffrement par bloc**

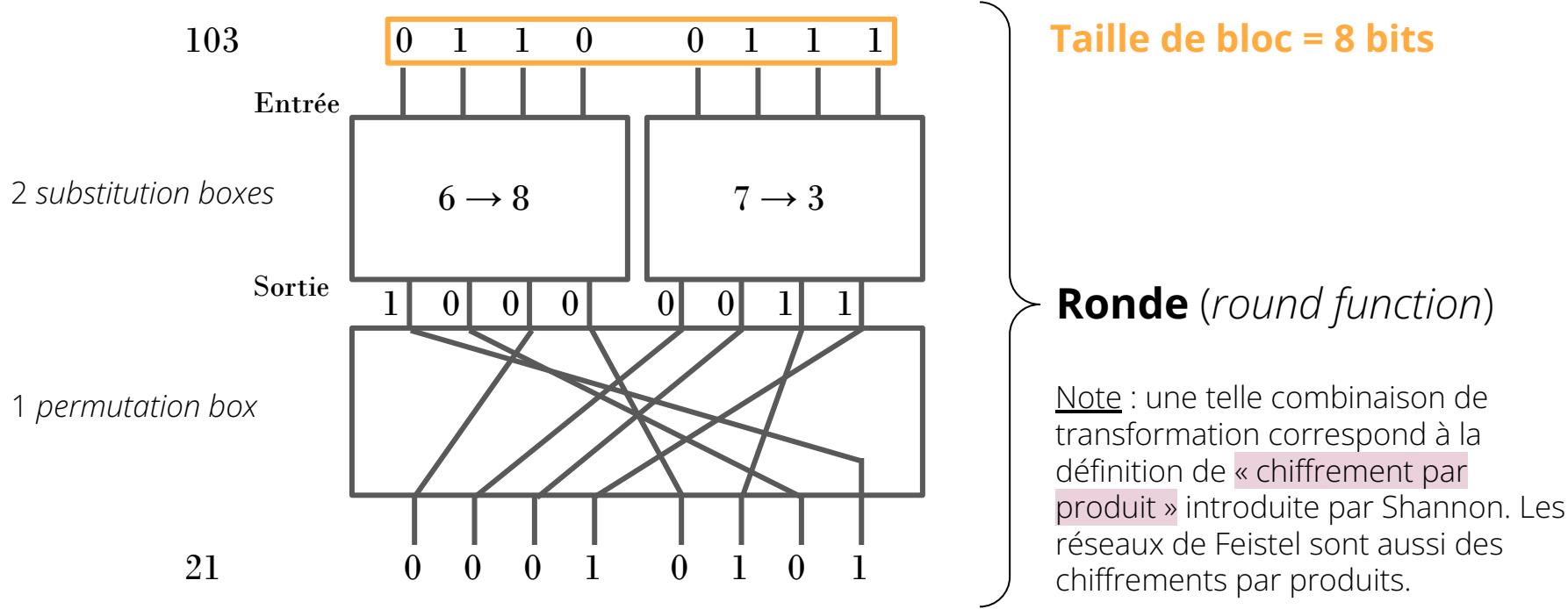
- **Chiffrement par bloc**

Un chiffrement par bloc découpe le message clair m en blocs d'une certaine taille (p. ex. 128 bits) et produit en sortie le message chiffré c , constitué d'autant de blocs chiffrés, chacun de cette même taille.

 Dans un chiffrement continu, le caractère $c[i]$ résulte du chiffrement du caractère $m[i]$. Cela équivaut à une **taille de bloc de 1**.

- **Réseau de substitution-permutation (SP)**

- Structure pour construire des chiffrements par bloc (Shannon, 1949)



- Réseau de substitution-permutation (SP)

- **S-Box (substitution box)**

Comme leur nom l'indique, elles sont utilisées pour effectuer des opérations de substitutions au sein de chaque bloc durant le processus de chiffrement.

Elles sont implémentées comme des tables de correspondances (*Lookup Table*, LUT) dans lesquelles sont définies les règles de substitutions.

Ces tables peuvent être fixes ou bien générées dynamiquement à partir de la clé.

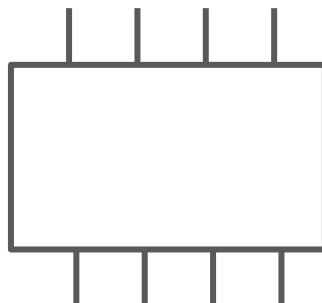


Illustration d'une S-Box : 4 bits en entrée et sortie.

La table de substitution comporte donc $2^4 = 16$ règles, pour des valeurs décimales de 0 à 15 : $0 \rightarrow 6 ; 1 \rightarrow 13 ; 2 \rightarrow 1 ; \dots$

La taille des S-Boxes AES est de 8 bits.

Dans DES, elle est de **6 bits en entrée, 4 bits en sortie**.

Note : les substitutions réciproques doivent être évitées.

- Réseau de substitution-permutation (SP)

- **S-Box (*substitution box*)**

Exemple d'une S-Box 3×3 **non-linéaire** :

| | | | | | | | | |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|
| Entrée | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| Sortie | 011 | 101 | 111 | 100 | 000 | 010 | 001 | 110 |

Table de correspondance souvent simplifiée sur 2 dimensions :

| | | | | |
|---|-----|-----|-----|-----|
| | 00 | 01 | 10 | 11 |
| 0 | 011 | 101 | 111 | 100 |
| 1 | 000 | 010 | 001 | 110 |

- Réseau de substitution-permutation (SP)

- S-Box (*substitution box*)

Exemple d'une S-Box 3×3 linéaire :

| | 00 | 01 | 10 | 11 |
|---|-----|-----|-----|-----|
| 0 | 000 | 011 | 111 | 100 |
| 1 | 110 | 101 | 001 | 010 |

Équations linéaires :

$$\begin{aligned}
 c_1 &= x_1 \oplus x_2 & = 1 \cdot x_1 \oplus 1 \cdot x_2 \oplus 0 \cdot x_3 \\
 c_2 &= x_1 \oplus x_2 \oplus x_3 & = 1 \cdot x_1 \oplus 1 \cdot x_2 \oplus 1 \cdot x_3 \\
 c_3 &= x_2 \oplus x_3 & = 0 \cdot x_1 \oplus 1 \cdot x_2 \oplus 1 \cdot x_3
 \end{aligned}$$

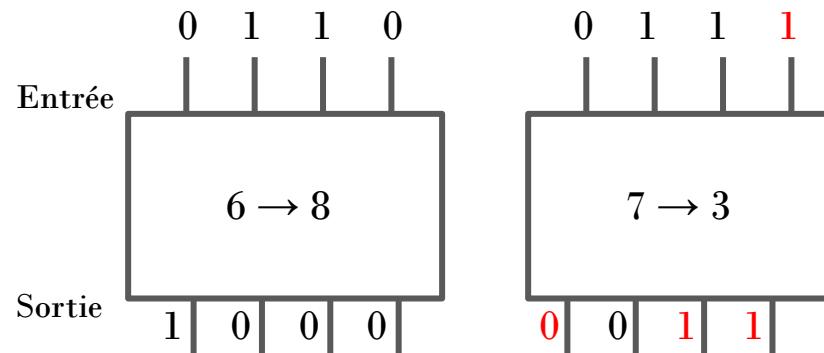
- Réseau de substitution-permutation (SP)

- **S-Box (substitution box)**

En pratique, les S-boxes sont des fonctions booléennes vectorielles **non-linéaires**.

Cette **non-linéarité** des S-boxes introduit ce que Shannon définit comme de la « **confusion** » dans la relation entre le chiffré et la clé secrète : pour un attaquant essayant de trouver la clé, le calcul réciproque à partir du chiffré est rendu difficile.

Une S-box est également source de « **diffusion** » : si l'on change un seul bit du texte clair (entrée), alors une grande partie du texte chiffré (sortie) change.



- **Réseau de substitution-permutation (SP)**

 Pour une bonne diffusion, il faudrait une grande S-box (p. ex. 128 bits). Mais cela serait beaucoup trop coûteux à implémenter. Ainsi, les S-box utilisées sont courtes et n'apportent quasiment pas de diffusion, que de la confusion.

- **P-Box (permutation box)**

Utilisées pour permuter (réarranger) les bits des données d'entrée, de manière systématique, selon un motif prédéfini. Mathématiquement, les opérations des P-boxes sont **linéaires**.

C'est la combinaison d'une P-Box **avec plusieurs S-Boxes courtes** qui introduit de la diffusion. Cette diffusion aide à ce que la structure statistique du texte clair (les *patterns*) ne soit pas préservée dans le chiffré.

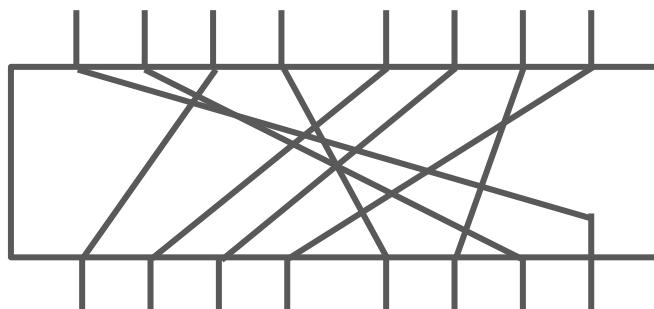
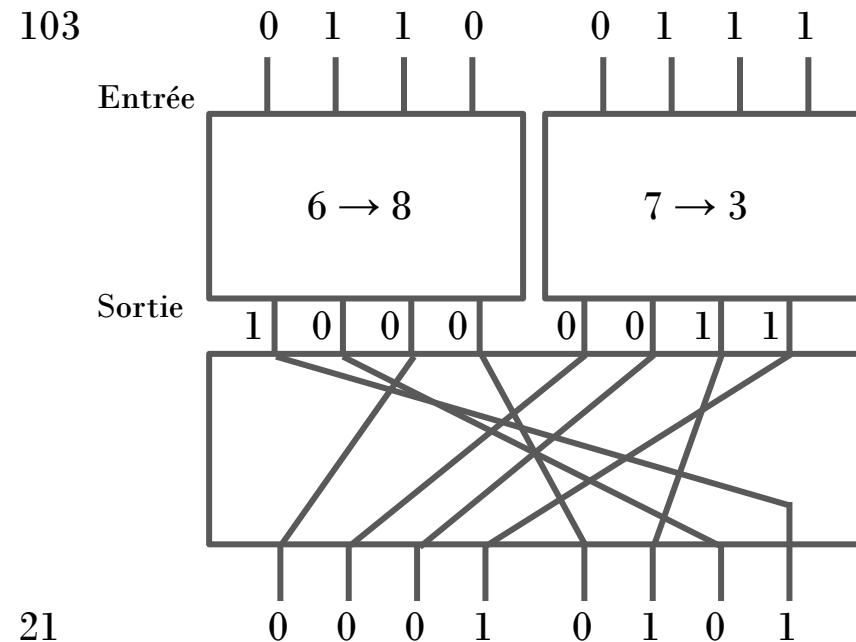
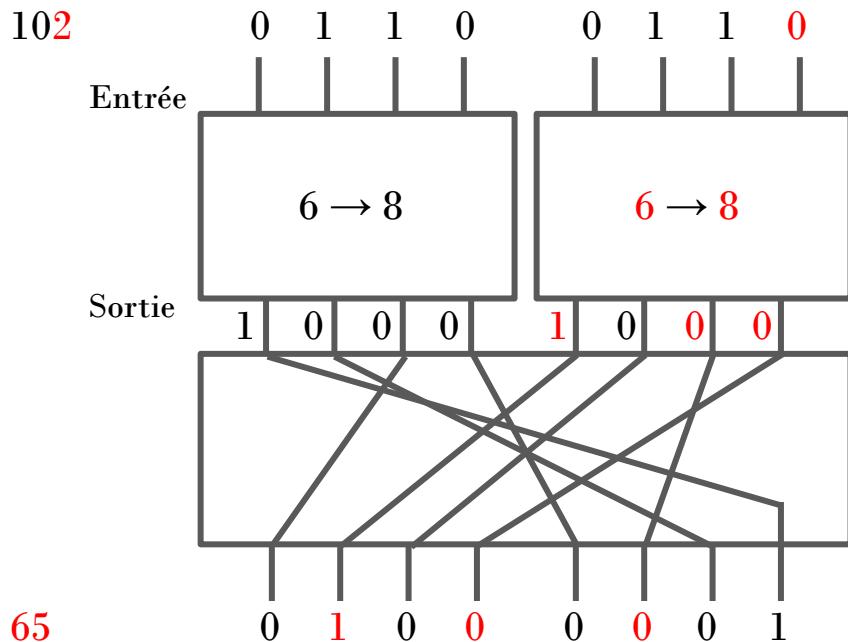


Illustration d'une P-Box droite : 8 bits I/O

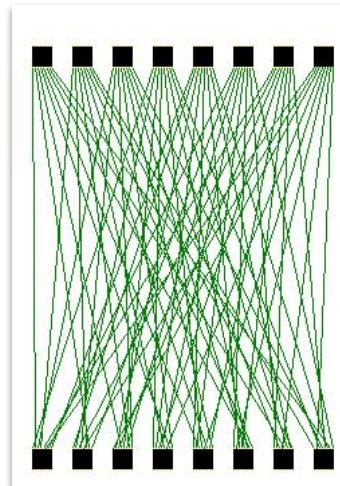
- **Diffusion**

- Exemple avec une seule ronde et sans ajout de clé



- **Remarque**

- La diffusion est également appelée « effet avalanche »
- Lorsque la diffusion est dite **complète** (propriété des fonctions booléennes)
 - On obtient un effet avalanche strict : la modification d'un seul bit en entrée affecte tous les bits en sortie avec une probabilité de 50%
 - Chaque bit en sortie est dépendant de tous les bits en entrée



- **Réseau de substitution-permutation (SP)**

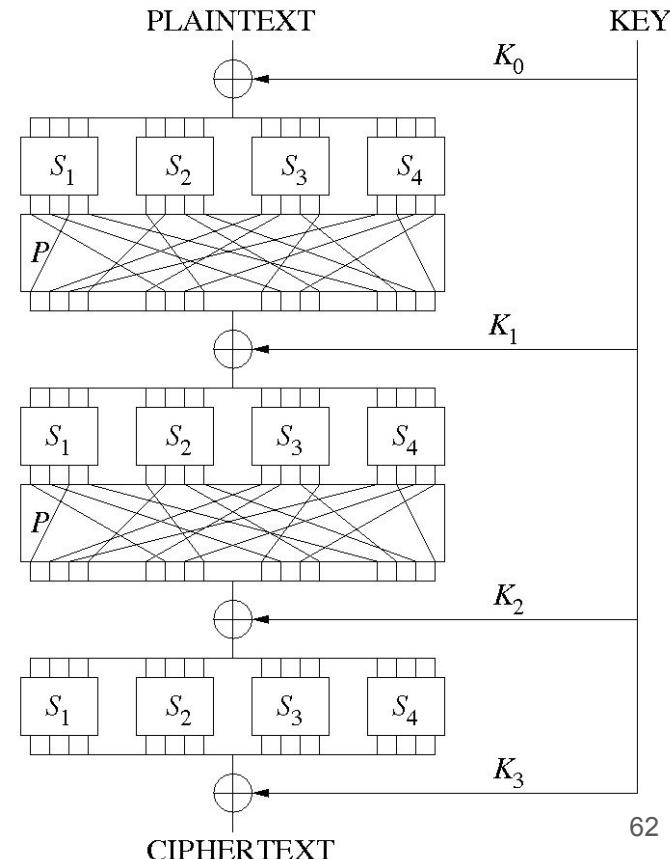
Plusieurs rondes doivent être effectués : leur nombre est un compromis en vitesse et sécurité

On introduit l'utilisation d'un clé secrète afin que, sans elle, **on ne puisse plus inverser la fonction** combinant S-Box et P-Box (« boîtes blanches »).

Préparation des clés (key schedule) : on crée des sous-clés à partir de la clé principale (« expansion » et division en clés de ronde).

Chaque ronde se termine donc par un **XOR** avec la sous-clé (*key mixing*). Ci-contre : 3 rondes.

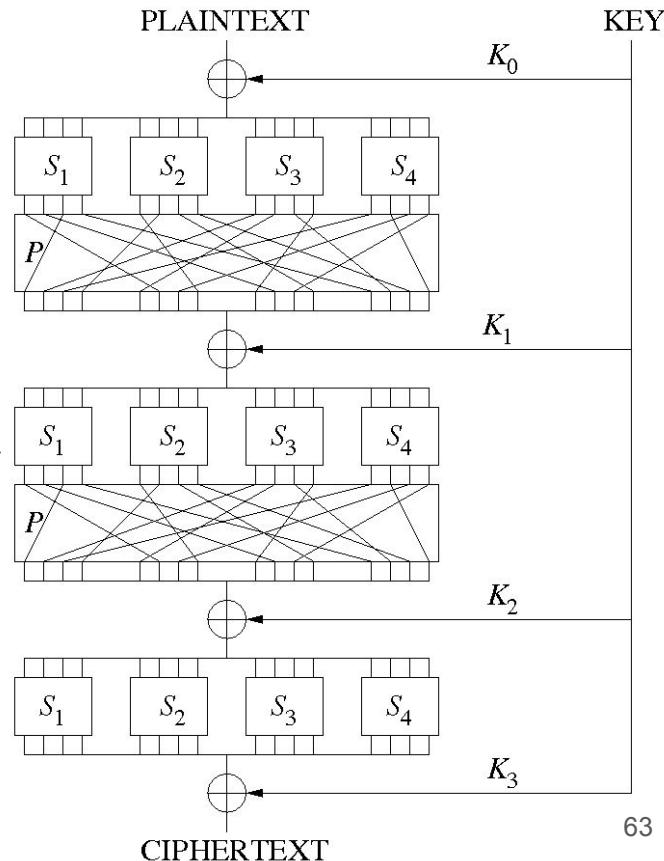
Déchiffrement : réciproques des S-boxes et P-boxes, avec utilisation des clés de rondes en ordre inversé.



- **Réseau de substitution-permutation (SP)**

Remarques :

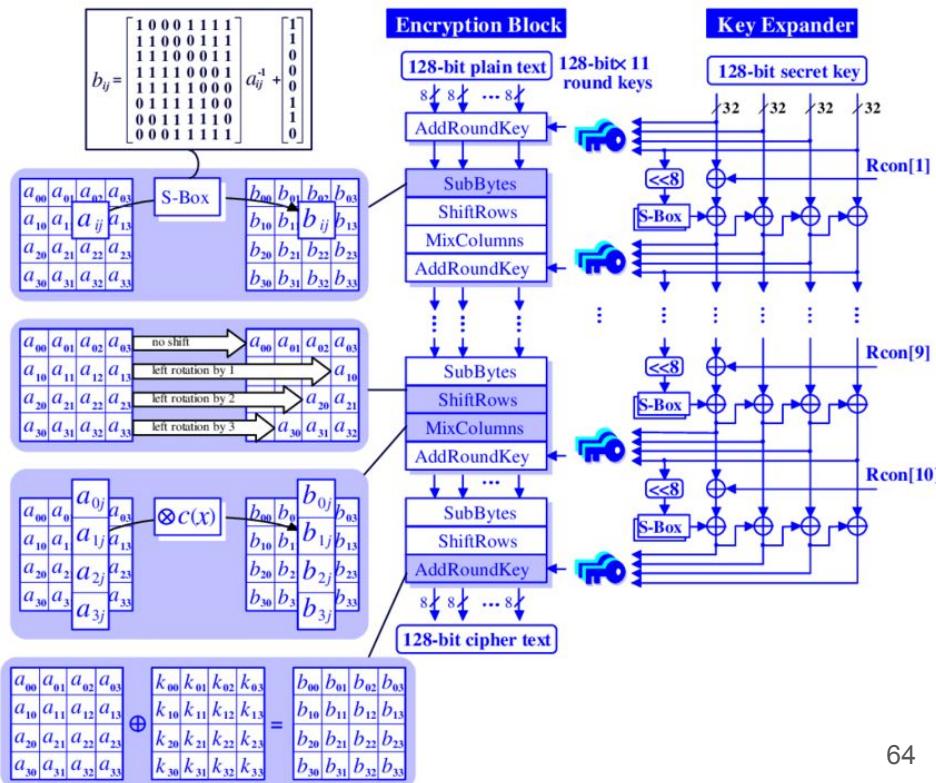
- La première ronde commence également par un **XOR** avec une sous-clé K_0
- Comme la dernière ronde produit le chiffré, elle n'a pas de P-Box : si celle-ci est publiée (c.-à-d. son schéma est rendu publique), alors un attaquant peut simplement « dé-permuter » les bits du chiffré. Elle n'a donc aucune valeur cryptographique. Le retrait de cette dernière P-box permet de diminuer les coûts de calculs et d'implémentation



Rijndael

Réseau SP de Rijmen et Daemen, 1998

- Bloc de données arrangé en grille
P. ex. $4 \times 4 = 16$ octets = 128 bits
- **Substitutions** (S-Box) :
Table de correspondance → rapide
 - ⚠ Pas de point fixe
 - Pas de point fixe opposé (p. ex. 1010 \Leftrightarrow 0101)
- **Permutations** :
 - Décalages des rangs
 - Mélange des colonnes
 - Multiplication par une matrice
→ Transformations linéaires
- **Addition de la clé de ronde**
- Dernière ronde : permutation inutile



- **Rijndael**

- Algorithme choisi par le NIST pour l'*Advanced Encryption Standard* (AES) pour remplacer (Triple) DES : devait être aussi sécurisé, mais beaucoup plus rapide
- L'AES fixe :
 - la longueur de bloc à 128 bits
 - la longueur des clés à 128, 192, ou 256 bits
 - et le nombre de rondes associé : 10, 12 et 14, respectivement
- Chaque étape peut être effectuée dans le **sens inverse** → **déchiffrement**
- <https://formaestudio.com/rijndaelinspector/archivos/Rijndael Animation v4 eng-html5.html>
- **Arithmétique modulaire**

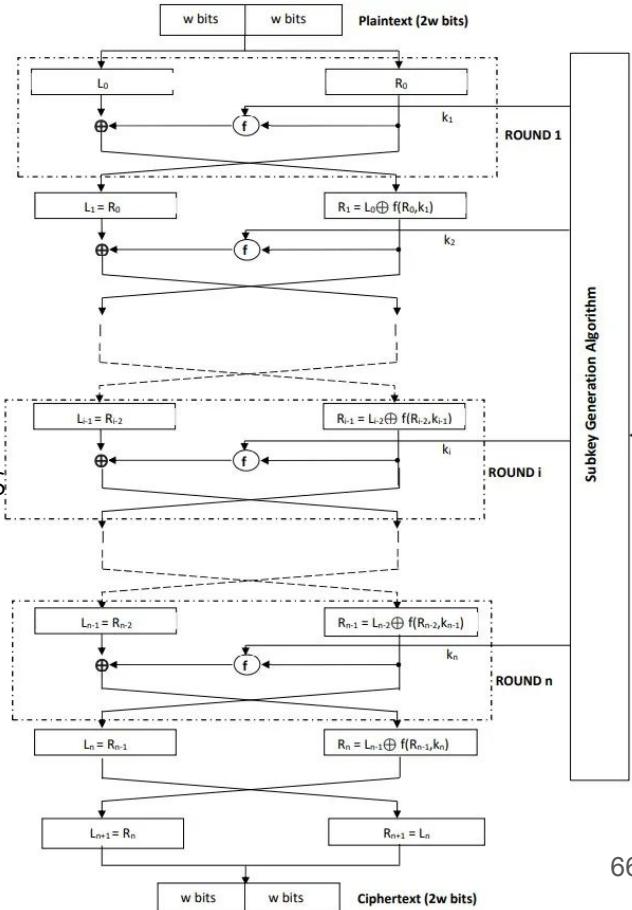
Toutes les opérations (addition, soustraction, multiplication, inversion) se font dans un « corps fini » (ou corps de Galois) de 2^8 éléments, soit 1 octet, les valeurs allant de 0000 0000 à 1111 1111. Les 256 éléments sont donc représentés par un polynôme irréductible de coefficients binaires $m(x)=x^8 + x^4 + x^3 + x + 1$ et les opérations se font modulo ce polynôme

2.6. Chiffrement par bloc

• Réseau de Feistel

Horst Feistel et Don Coppersmith, 1973

- Structure (comme réseaux SP) pour construire des chiffrements par bloc (p. ex. DES, Twofish)
- **Étapes du chiffrement :**
 - Le bloc en entrée est divisé en 2 parties (de tailles égales ou inégales)
 - Le chiffrement consiste en une série de rondes
 - La moitié droite du bloc est transformée par une fonction de ronde f , qui dépend d'une sous-clé dérivée de la clé principale
 - La sortie de f est ensuite XORée avec la moitié droite du bloc
 - **Après la dernière ronde** de (dé)chiffrement,
→ Permutation des deux moitiés

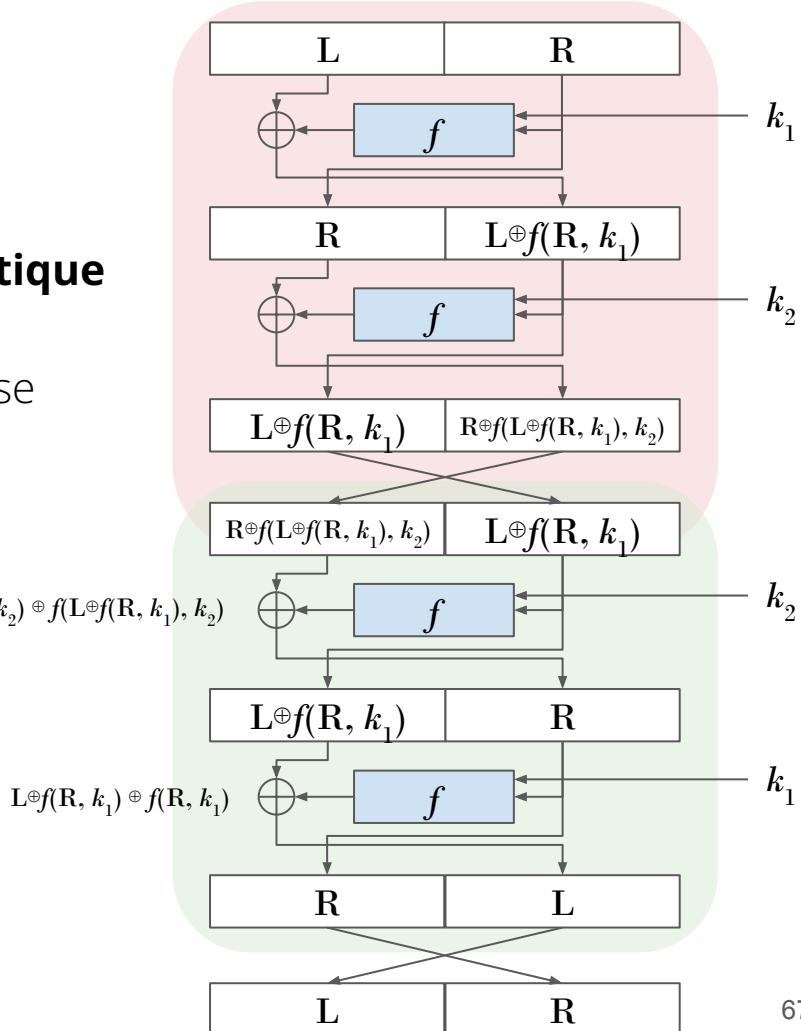


• Réseau de Feistel

- Le déchiffrement est essentiellement **identique au chiffrement**, la seule différence étant l'utilisation des sous-clés dans l'ordre inverse

Ci-contre : exemple avec 2 rondes

- Fonctionne
 - quelque soit f
 - pour tout nombre de rondes



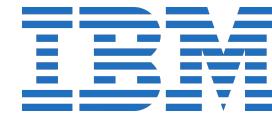
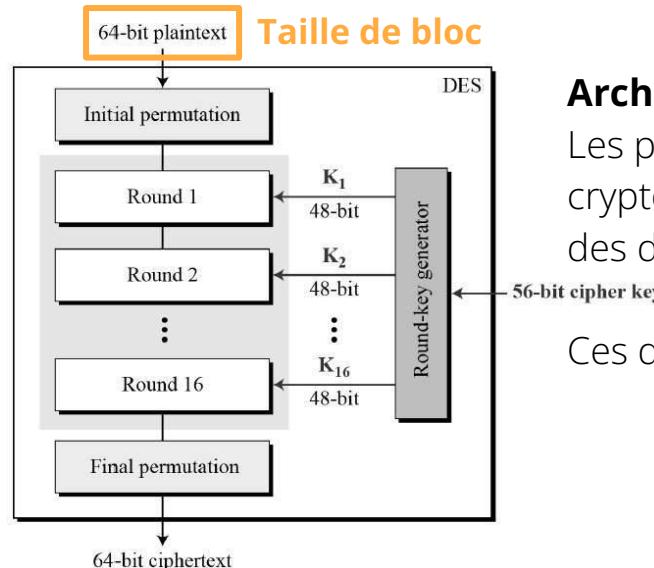
- Réseau de Feistel

- DES (*Data Encryption Standard*), 1975

Clé de 56 bits : peut être trouvée par force brute en 2^{55}

opérations **en moyenne** → Difficile à faire, jusque dans les années 90...

- Remplacé par Triple DES → Sécurisé, mais 3 fois plus lent !



Architecture générale du DES

Les permutations initiales et finales n'ont aucun intérêt cryptographique : elles ne sont là que pour le (dé)chargement des données sur le matériel 8 bits des années 70.

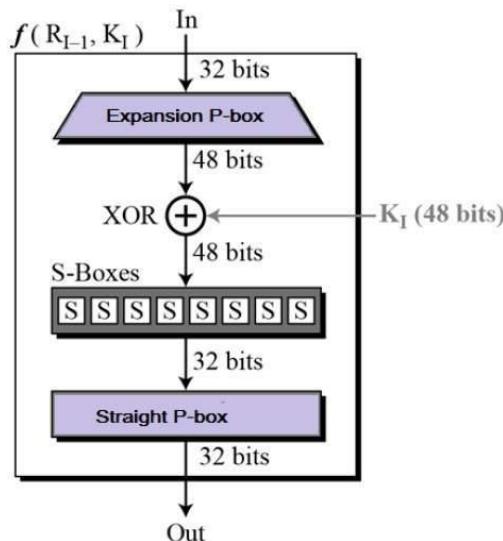
Ces deux fonctions sont mutuellement réciproques.

- Réseau de Feistel

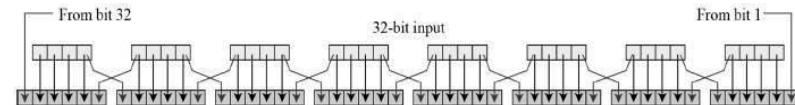
- DES (*Data Encryption Standard*), 1975

- Fonction de ronde f

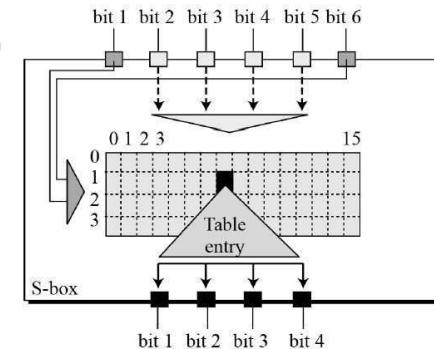
Combine une sous-clé de 48 bits avec les 32 bits de poids faibles (LSB) pour produire une sortie de 32 bits



Expansion P-Box (E-Box)



S-Box (6/4 bits)

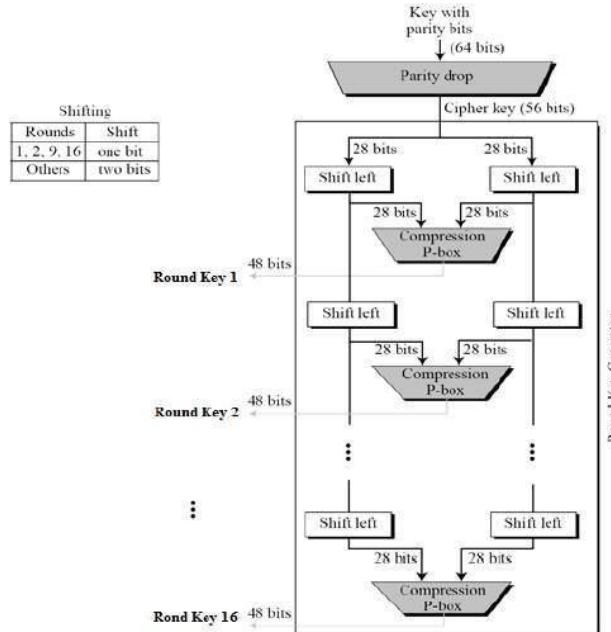


- Réseau de Feistel

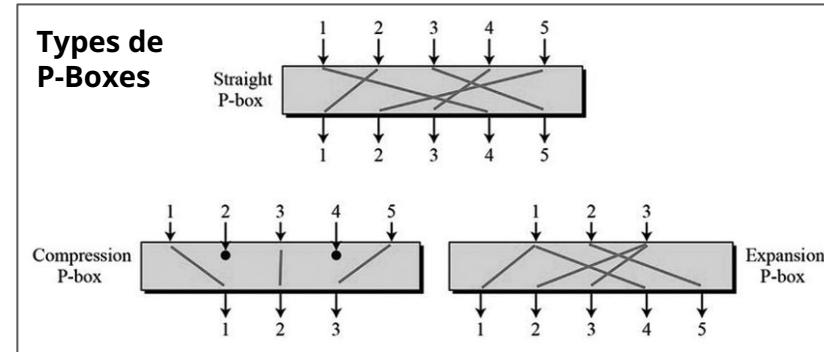
- DES (*Data Encryption Standard*), 1975

- **Key schedule** (préparation des clés de rondes)

Création de 16 clés de 48 bits, à partir de la clé principale de 56 bits



8 bits de parité, ajoutés aux 56 bits de données, pour assurer l'intégrité de la clé durant sa transmission (code correcteur)



- **Réseau de Feistel**

- DES-X (Rivest, 1984)

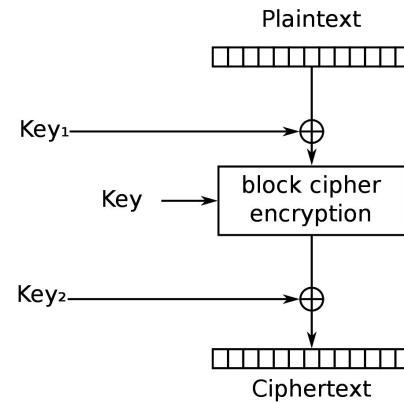
But : augmenter la complexité des attaques par force brute

 La force brute est, à ce jour, le seul type d'attaque connu contre DES

Utilisation d'une clé k de 184 bits, comprenant deux clés de 64 bits, k_1 et k_2 :

$$\text{DES-X}(m) = k_2 \oplus \text{DES}_k(m \oplus k_1)$$

Note : $184 = 56 + 2 \times 64$



Processus de *key whitening*

- **Mode de fonctionnement** (*mode of operation*)

Un mode d'opération décrit comment appliquer de manière répétée l'opération monobloc d'un chiffrement pour transformer en toute sécurité des quantités de données supérieures à un bloc

Afin d'utiliser les chiffrements par bloc dans diverses applications, **cinq** modes d'opération ont été définis par le NIST (SP 800-38A)

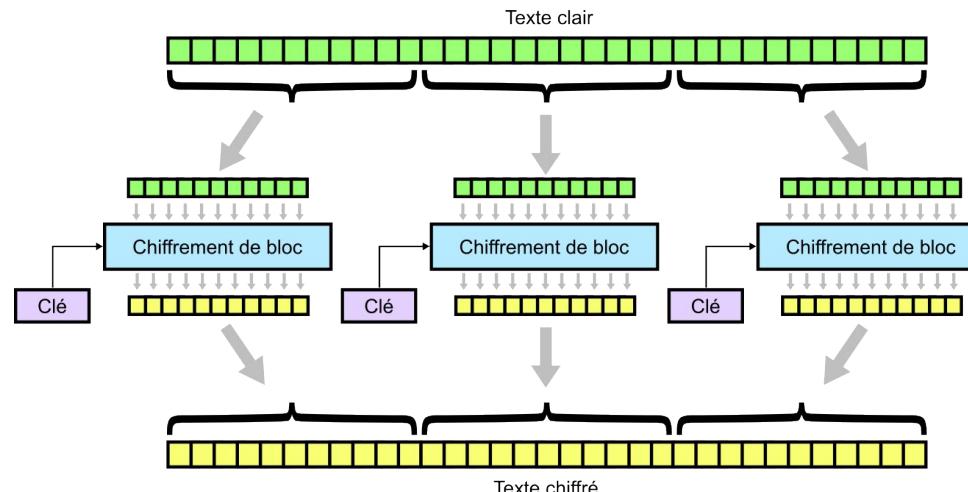
NIST SP 800-38A *Recommendation for Block Cipher Modes of Operation: Methods and Techniques* (déc. 2001)

- ECB
- CBC
- CTR
- CFB
- OFB

- o **ECB** (*Electronic Codebook Block*, dictionnaire de codes)

Approche la plus naïve

- Couper le message en blocs et les chiffrer un à la fois
- Si message trop court → *padding*, afin que sa longueur soit un multiple de la taille de bloc
- Déchiffrement : fonction réciproque



2.7. Modes de fonctionnement

- **ECB** (*Electronic Codebook Block*, dictionnaire de codes)

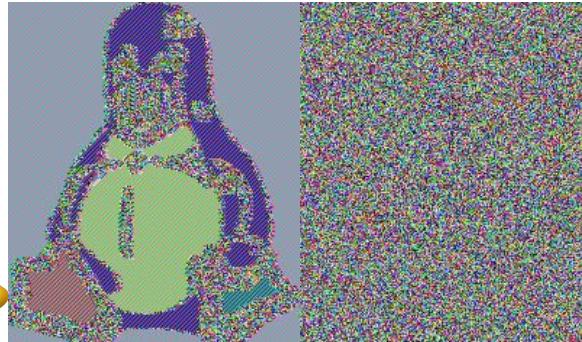
- Facile à implémenter
- Rapide et parallélisable
- Permet un déchiffrement direct d'une zone quelconque du texte chiffré (*random read access*)
- ⚠ Non-sécurisé : si deux blocs clairs sont identiques, les deux blocs chiffrés correspondants le seront aussi (même clé k)

Exemples : requête HTTP répétée à l'identique,  [zoom](#)

Original



Mode ECB

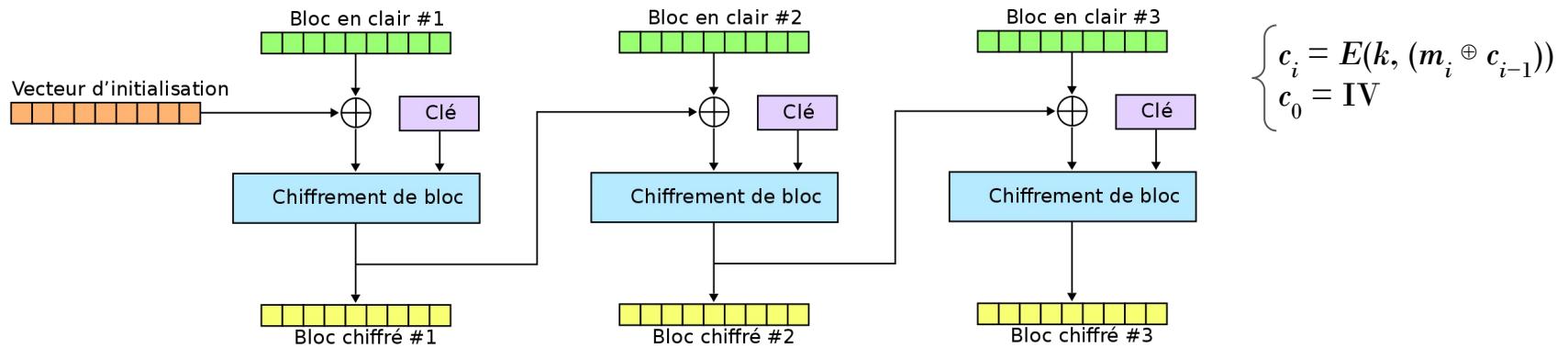


Mode sûr

Si des parties de l'image (données de pixels) sont identiques, leur chiffrements le sont aussi
→ Motifs (*patterns*) visibles

2.7. Modes de fonctionnement

- **CBC** (*Cipher Block Chaining*, enchaînement de blocs)



La sortie d'un bloc est reliée à l'entrée du bloc suivant : permet de masquer le i -ème bloc du texte clair par le chiffrement du bloc $i-1$ du texte chiffré

→ **2 blocs clairs identiques donnent 2 blocs chiffrés différents**

Premier bloc : entrée du **XOR**ée avec un bloc aléatoire « imaginaire », appelé **vecteur d'initialisation (IV)**

L'IV n'a pas besoin d'être secret, mais doit être aléatoire (cas du CBC) et **unique** : un nouvel IV pour chaque message (stockage dans une mémoire permanente)

→ Si deux blocs #1 sont les mêmes, leurs chiffrés seront différents

2.7. Modes de fonctionnement

- **CBC** (*Cipher Block Chaining*, enchaînement de blocs)

Le mode CBC est désormais peu utilisé



Séquentiel

↳ Perte du parallélisme d'ECB : on ne peut plus **chiffrer** le bloc $i+1$ avant le bloc i



Padding nécessaire, car **XOR** requiert un bloc clair complet

↳ Vulnérabilité à *padding attack* (sauf si *ciphertext stealing*, CTS)



Enchaînement de blocs et **XOR** → propagation d'erreur

■ Altération du bloc chiffré i → Problème avec les blocs clairs i et $i+1$

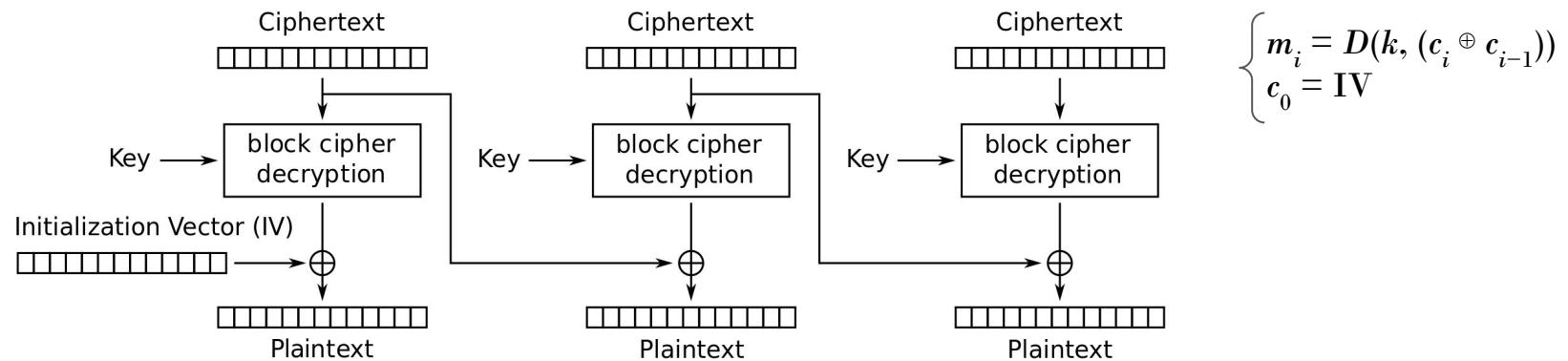
↳ Perte de 2 blocs !!

- Erreur de transmission réseau
- *bit flipping attack*

2.7. Modes de fonctionnement

- **CBC** (*Cipher Block Chaining*, enchaînement de blocs)

Le déchiffrement suit processus exactement inverse : le chiffré #1 est déchiffré, puis **XOR** avec **l'IV, qui est transmis avec le message**, pour obtenir le clair #1

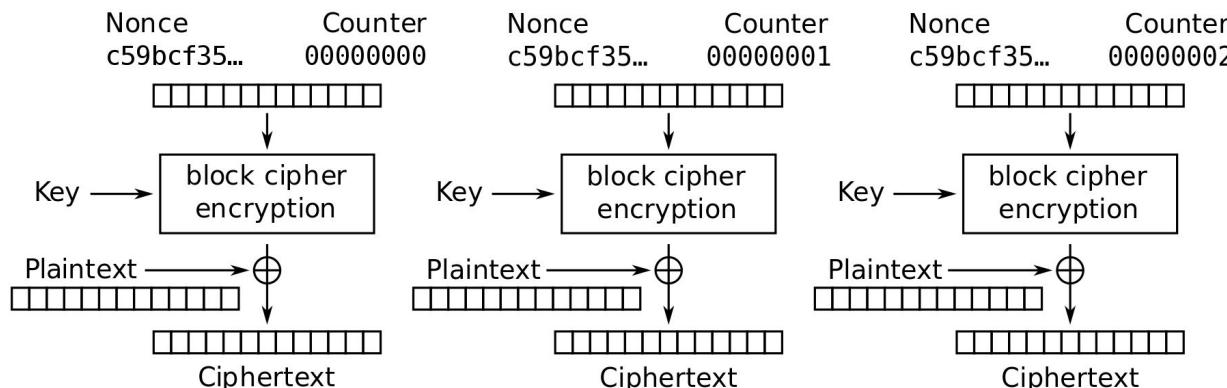


 **Le déchiffrement peut être parallélisé** : déchiffer avec un IV incorrect entraînera une corruption du premier bloc de texte clair, mais les blocs clairs suivants seront corrects

2.7. Modes de fonctionnement

- **CTR** (*CounTeR*), Diffie et Hellman, 1979

Nonce (number used once) : nombre arbitraire, unique à chaque communication ; chaque **paire clé-nonce** ne doit être utilisée qu'une seule fois. Il n'a pas besoin d'être secret, ni aléatoire
Il est concaténé à un **compteur** pour ensuite être chiffré par bloc à l'aide d'une clé



Terminologie :
le *nonce*, concaténé ou non, est parfois appelé vecteur d'initialisation

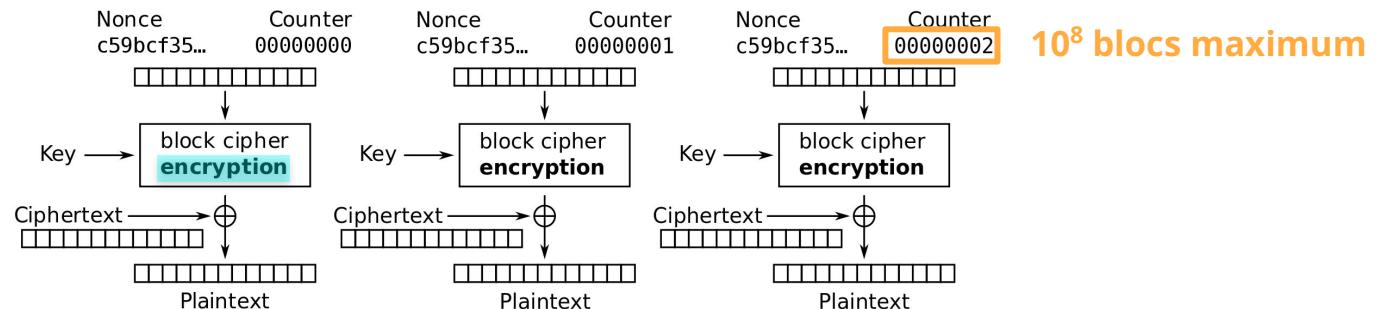
⚠ Ce n'est pas le message clair qui est chiffré mais le *nonce*||compteur.

Le bloc chiffré aléatoire produit en sortie est **XOR**é avec le bloc clair, comme une clé de flux.

→ Le mode CTR transforme un chiffrement par **bloc en chiffrement de flux**

- **CTR (CounteR)**

- ✓ Si même bloc clair → blocs chiffrés différents, car clés de flux différentes
- ✓ *Random read access*
- ✓ Parallélisation possible
- ✓ Plus besoin de *padding*, car transformation en chiffrement de flux
- ⚠ La taille du compteur limite la taille du message (*i.e.* le nombre de blocs) qu'on peut envoyer
- ⚠ **XOR** : inversion d'un bit dans le chiffré → inversion dans le clair
- ↳ Requiert un MAC (*Galois CTR mode, GCM* → permet confidentialité **et intégrité**)
- ✓ Le **déchiffrement** utilise le même processus que le chiffrement car on génère la même clé de flux
- ↳ Diminue les coûts d'implémentation (p. ex. on n'implémente pas la fonction de déchiffrement d'AES)

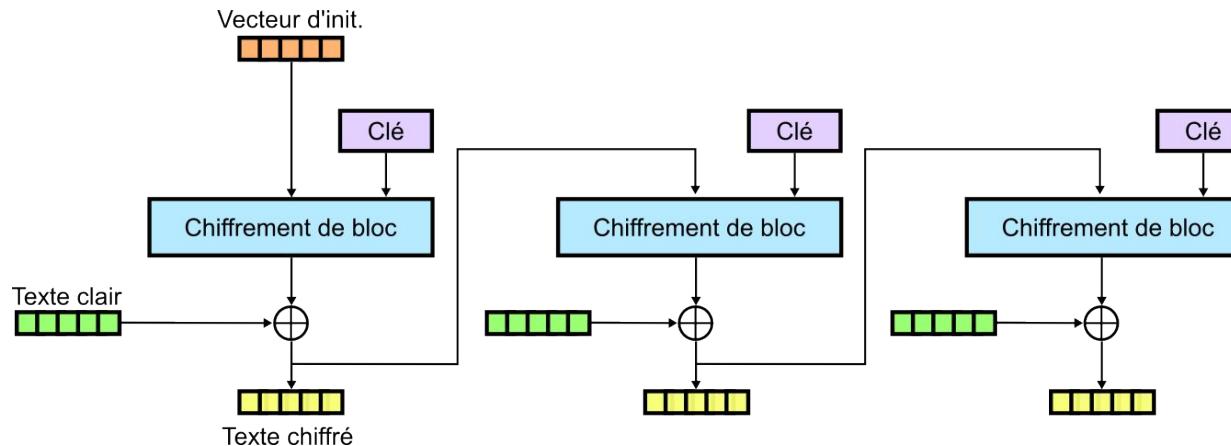


2.7. Modes de fonctionnement

- **CFB** (*Cipher Feedback Block*, chiffrement à rétroaction)

Comme CBC : la sortie d'un bloc est reliée à l'entrée du bloc suivant, ce qui permet de masquer le i -ème bloc du texte clair par le chiffrement du bloc $i-1$ du texte chiffré

Comme CTR : transformation en un chiffrement de flux (asynchrone) → pas besoin de *padding*

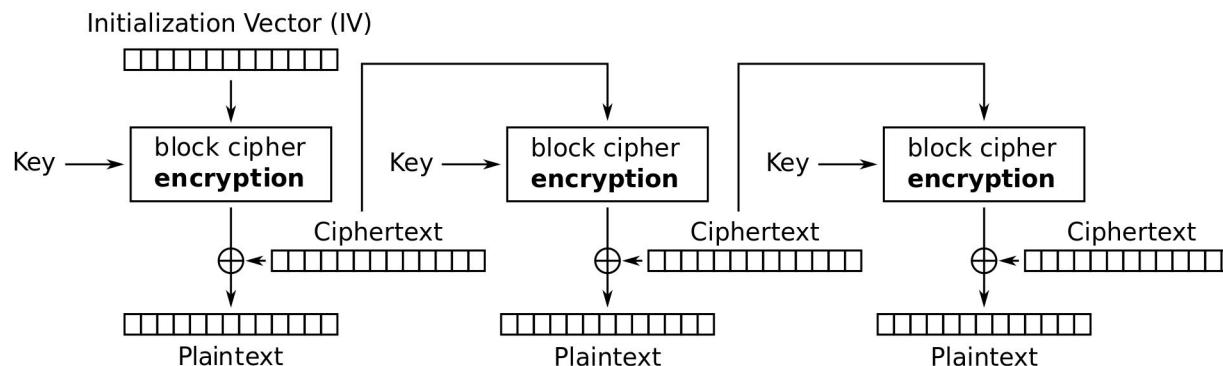


2.7. Modes de fonctionnement

- **CFB** (*Cipher Feedback Block*, chiffrement à rétroaction)

Comme CBC : seul le déchiffrement peut être parallélisé

Comme CTR : utilisation de la même fonction pour chiffrage et déchiffrement



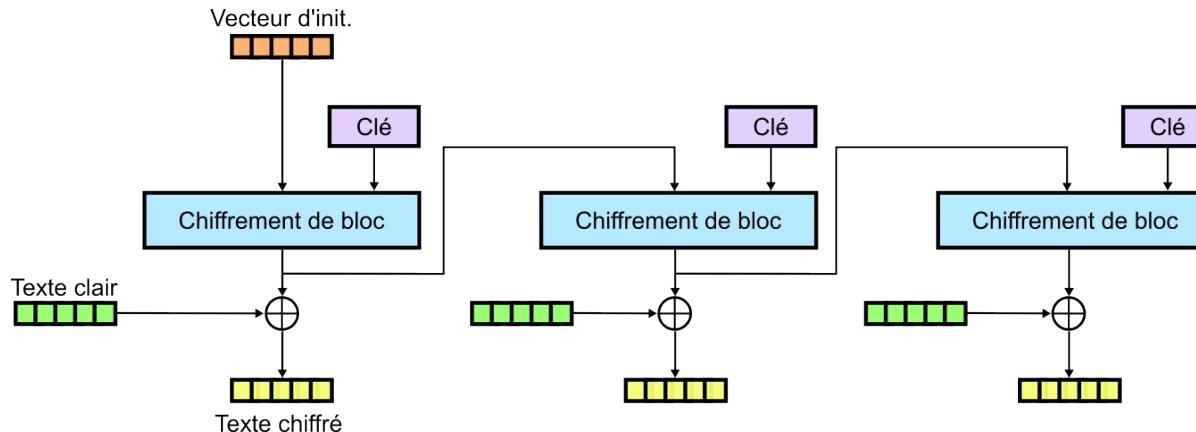
2.7. Modes de fonctionnement

- **OFB** (*Output Feedback*, chiffrement à rétroaction de sortie)

Le flux de clé est obtenu en chiffrant le précédent flux de clé

↳ Plus rapide que CFB, car permet un pré-calcul de toutes les clés de flux

↳ Plus fragile que CFB, car un attaquant n'a besoin de connaître que l'IV d'un message chiffré et le clair d'un autre message chiffré



2.7. Modes de fonctionnement

- **En résumé**

| | ECB | CBC | CTR | CFB | OFB |
|------------------------------|-----|-----|-----|-----|--------------------------|
| Chiffrement parallélisable | Oui | Non | Oui | Non | Précalcul des keystreams |
| Déchiffrement parallélisable | Oui | Oui | Oui | Oui | Précalcul des keystreams |
| <i>Random read access</i> | Oui | Oui | Oui | Oui | Non |

3. Sécurité des chiffrements

- **Modèles d'attaques cryptographiques**

- Ils représentent différents scénarios dans lesquels la sécurité du cryptosystème étudié est menacée par un adversaire
 - Adversaire « réaliste » : dispose des meilleurs algorithmes et capacités de calculs actuellement connus
 - À partir des données auxquelles l'adversaire a accès, différents types d'attaques sont possibles
 - Varient en force et en vraisemblance
 - La résistance du chiffrement caractérise son niveau de sécurité
-  **Preuve de sécurité par jeu** (*security game*)
p. ex. IND-CPA, IND-CCA1, ...

- **Attaque sur texte chiffré seul** (*ciphertext-only attack*, COA)

Dans ce scénario, le cryptanalyste a seulement accès à une collection de textes chiffrés, pas aux textes en clair

- Le plus vraisemblable en cryptanalyse
- Le plus faible, car manque d'information
- Chiffrements modernes très résistants aux COA
 - En général, ne peut réussir que si le cryptanalyste dispose d'informations sur le texte en clair : *langue utilisée*, formatage, etc.
- Cas particulier : **attaque par force brute**, qui recherche exhaustivement la clé de déchiffrement parmi toutes les possibilités
 - Tous les chiffrements y sont vulnérables, sauf OTP
 - En combien de temps (tentatives) ?
 - ↳ Sert de comparaison avec les autres attaques (*faster-than-brute-force attacks*)
 - *Attaque par dictionnaire* : s'appuie sur une liste de mots clairs vraisemblables

- **Attaque à texte clair connu** (*known-plaintext attack, KPA*)

Dans ce scénario, le cryptanalyste dispose d'un jeu de textes chiffrés et des textes clairs correspondants, pour une clé donnée. Ces paires aident à comprendre l'algorithme de chiffrement, voire à retrouver la clé.

- **Analyse par crib**

- Les *cribs* sont généralement des parties de texte en clair que l'on sait, **ou bien que l'on suspecte** (mots ou expressions récurrents), être présentes dans le chiffré
 - Exemple : « Cordialement » à la fin des mails
 - Les codes sources chiffrés sont vulnérables, car pleins de mots récurrents
- Cryptanalyse d'Enigma, avant que les Alliés ne capturent un exemplaire de la machine

Enigma : permutation sans point fixe

Une permutation f est dite sans point fixe, s'il n'existe aucun x tq $f(x) = x$

i.e. impossibilité de substituer une lettre à elle-même

→ Vulnérabilité à une analyse par *crib*

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|--|----------|---|----------|---|----------|---|----------|---|----------|----------|----------|----------|----------|---|----------|----------|----------|----------|---|----------|---|---|-----|---|----------|---|---|---|---|---|----------|---|---|---|----------|---|-----|--|--|--|--|--|--|--|--|
| 1 | <table border="1"><tr><td>H</td><td>Q</td><td>E</td><td>Y</td><td>S</td><td>A</td><td>W</td><td>Q</td><td>S</td><td>T</td><td>N</td><td>T</td><td>L</td><td>G</td><td>K</td><td>P</td><td>E</td><td>S</td><td>R</td><td>E</td><td>V</td><td>L</td></tr> <tr><td>S</td><td>E</td><td>C</td><td>R</td><td>E</td><td>T</td><td>M</td><td>E</td><td>S</td><td>S</td><td>S</td><td>A</td><td>G</td><td>E</td><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | S | E | C | R | E | T | M | E | S | S | S | A | G | E | ... | | | | | | | | |
| H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | | | | | | | | | | | | | | | | | | | | | | | | | |
| S | E | C | R | E | T | M | E | S | S | S | A | G | E | ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | <table border="1"><tr><td>H</td><td>Q</td><td>E</td><td>Y</td><td>S</td><td>A</td><td>W</td><td>Q</td><td>S</td><td>T</td><td>N</td><td>T</td><td>L</td><td>G</td><td>K</td><td>P</td><td>E</td><td>S</td><td>R</td><td>E</td><td>V</td><td>L</td></tr> <tr><td>...</td><td>S</td><td>E</td><td>C</td><td>R</td><td>E</td><td>T</td><td>M</td><td>E</td><td>S</td><td>S</td><td>A</td><td>G</td><td>E</td><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | |
| H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | <table border="1"><tr><td>H</td><td>Q</td><td>E</td><td>Y</td><td>S</td><td>A</td><td>W</td><td>Q</td><td>S</td><td>T</td><td>N</td><td>T</td><td>L</td><td>G</td><td>K</td><td>P</td><td>E</td><td>S</td><td>R</td><td>E</td><td>V</td><td>L</td></tr> <tr><td>...</td><td>S</td><td>E</td><td>C</td><td>R</td><td>E</td><td>T</td><td>M</td><td>E</td><td>S</td><td>S</td><td>A</td><td>G</td><td>E</td><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | |
| H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | <table border="1"><tr><td>H</td><td>Q</td><td>E</td><td>Y</td><td>S</td><td>A</td><td>W</td><td>Q</td><td>S</td><td>T</td><td>N</td><td>T</td><td>L</td><td>G</td><td>K</td><td>P</td><td>E</td><td>S</td><td>R</td><td>E</td><td>V</td><td>L</td></tr> <tr><td>...</td><td>S</td><td>E</td><td>C</td><td>R</td><td>E</td><td>T</td><td>M</td><td>E</td><td>S</td><td>S</td><td>A</td><td>G</td><td>E</td><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | |
| H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | <table border="1"><tr><td>H</td><td>Q</td><td>E</td><td>Y</td><td>S</td><td>A</td><td>W</td><td>Q</td><td>S</td><td>T</td><td>N</td><td>T</td><td>L</td><td>G</td><td>K</td><td>P</td><td>E</td><td>S</td><td>R</td><td>E</td><td>V</td><td>L</td></tr> <tr><td>...</td><td>S</td><td>E</td><td>C</td><td>R</td><td>E</td><td>T</td><td>M</td><td>E</td><td>S</td><td>S</td><td>A</td><td>G</td><td>E</td><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | |
| H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | <table border="1"><tr><td>H</td><td>Q</td><td>E</td><td>Y</td><td>S</td><td>A</td><td>W</td><td>Q</td><td>S</td><td>T</td><td>N</td><td>T</td><td>L</td><td>G</td><td>K</td><td>P</td><td>E</td><td>S</td><td>R</td><td>E</td><td>V</td><td>L</td></tr> <tr><td>...</td><td>S</td><td>E</td><td>C</td><td>R</td><td>E</td><td>T</td><td>M</td><td>E</td><td>S</td><td>S</td><td>A</td><td>G</td><td>E</td><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | |
| H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | <table border="1"><tr><td>H</td><td>Q</td><td>E</td><td>Y</td><td>S</td><td>A</td><td>W</td><td>Q</td><td>S</td><td>T</td><td>N</td><td>T</td><td>L</td><td>G</td><td>K</td><td>P</td><td>E</td><td>S</td><td>R</td><td>E</td><td>V</td><td>L</td></tr> <tr><td>...</td><td>S</td><td>E</td><td>C</td><td>R</td><td>E</td><td>T</td><td>M</td><td>E</td><td>S</td><td>S</td><td>A</td><td>G</td><td>E</td><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | |
| H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | <table border="1"><tr><td>H</td><td>Q</td><td>E</td><td>Y</td><td>S</td><td>A</td><td>W</td><td>Q</td><td>S</td><td>T</td><td>N</td><td>T</td><td>L</td><td>G</td><td>K</td><td>P</td><td>E</td><td>S</td><td>R</td><td>E</td><td>V</td><td>L</td></tr> <tr><td>...</td><td>S</td><td>E</td><td>C</td><td>R</td><td>E</td><td>T</td><td>M</td><td>E</td><td>S</td><td>S</td><td>A</td><td>G</td><td>E</td><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | |
| H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | <table border="1"><tr><td>H</td><td>Q</td><td>E</td><td>Y</td><td>S</td><td>A</td><td>W</td><td>Q</td><td>S</td><td>T</td><td>N</td><td>T</td><td>L</td><td>G</td><td>K</td><td>P</td><td>E</td><td>S</td><td>R</td><td>E</td><td>V</td><td>L</td></tr> <tr><td>...</td><td>S</td><td>E</td><td>C</td><td>R</td><td>E</td><td>T</td><td>M</td><td>E</td><td>S</td><td>S</td><td>A</td><td>G</td><td>E</td><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | |
| H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | <table border="1"><tr><td>H</td><td>Q</td><td>E</td><td>Y</td><td>S</td><td>A</td><td>W</td><td>Q</td><td>S</td><td>T</td><td>N</td><td>T</td><td>L</td><td>G</td><td>K</td><td>P</td><td>E</td><td>S</td><td>R</td><td>E</td><td>V</td><td>L</td></tr> <tr><td>...</td><td>S</td><td>E</td><td>C</td><td>R</td><td>E</td><td>T</td><td>M</td><td>E</td><td>S</td><td>S</td><td>A</td><td>G</td><td>E</td><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table> | H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | |
| H | Q | E | Y | S | A | W | Q | S | T | N | T | L | G | K | P | E | S | R | E | V | L | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | S | E | C | R | E | T | M | E | S | S | A | G | E | ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

- **Attaque à texte clair choisi** (*chosen-plaintext attack, CPA*)

Dans ce scénario, le cryptanalyste peut choisir arbitrairement un jeu de textes clairs et obtenir de l'**oracle de chiffrement** les chiffrés correspondants

- **Oracle** : entité informatique qui fournit des informations sur demande : chiffrement, déchiffrement, valeur de hachage, fuite d'information (cf. attaque par canal auxiliaire).

- Les détails de son fonctionnement ne sont pas censés être connus de l'adversaire, à l'exception de l'algorithme de chiffrement qu'il simule. La clé de chiffrement demeure secrète.

- Cela le définit comme une **boîte grise**

 **Preuve de sécurité par jeu** : l'oracle y sera représenté par le *challenger* → s'oppose à l'adversaire et interagit avec lui

- **Attaque à texte clair choisi** (*chosen-plaintext attack, CPA*)

Dans les schémas de **chiffrements asymétriques**, la clé utilisée pour chiffrer est distribuée publiquement, ce qui permet les tentatives de CPA

- Ils doivent donc être résistants aux CPA → **sécurité sémantique**
(sémantique *adj.* : qui concerne le sens, la signification)

- Chiffrements de flux résistants : Trivium, Salsa20, Grain-128, etc.

- **Attaque à texte clair choisi adaptative**

(*adaptive chosen-plaintext attack, CPA2*)

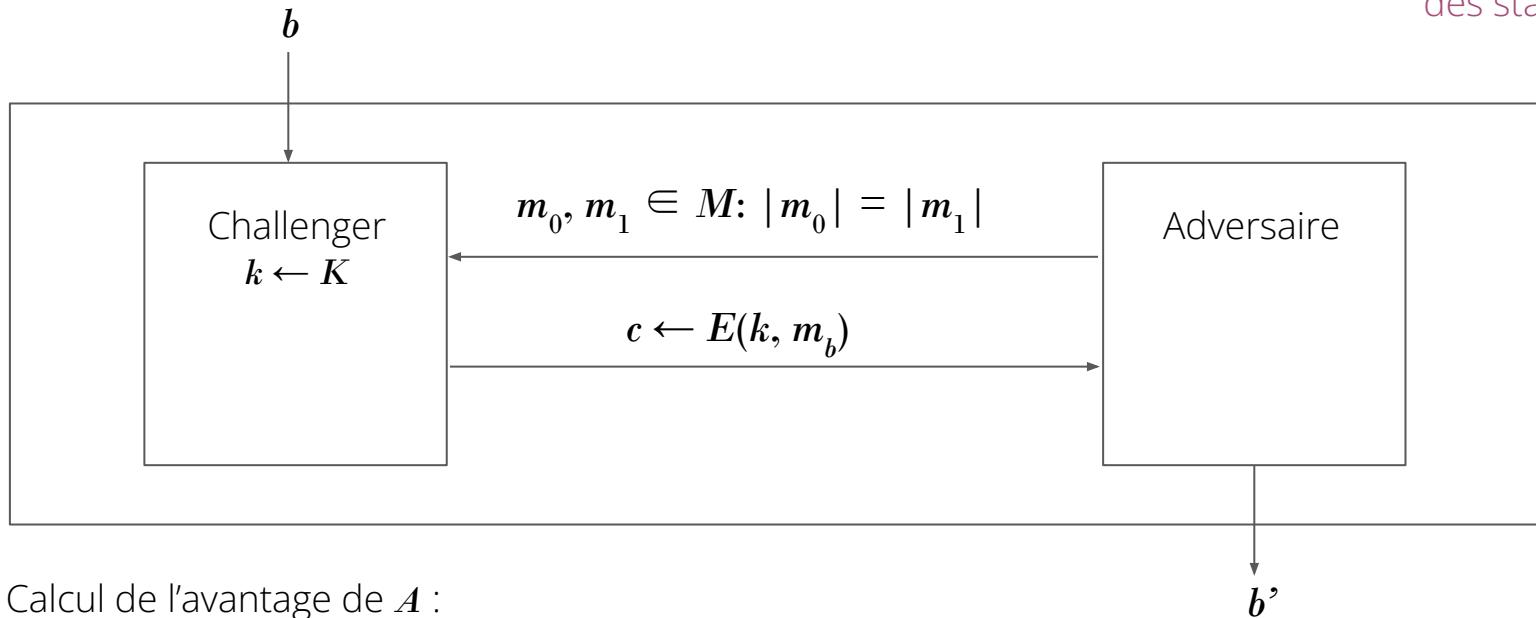
Ici, le cryptanalyste n'est pas obligé de choisir tous les textes clairs dès le début : il peut adapter son choix de texte clair en fonction des chiffrés renvoyés par l'oracle

3.3. CPA

Soit une expérience $\text{EXP}(b)$, avec $b \leftarrow U(0, 1)$ au cours de laquelle :

- l'adversaire A envoie deux messages m_0 et m_1 au *challenger*,
- le *challenger* renvoie le chiffrement c du message m_b à l'adversaire,
- ce dernier doit deviner qui de m_0 ou m_1 a été chiffré : prédiction $b' \in \{0,1\}$.

Répétitions de la procédure, pour calculer des statistiques



Calcul de l'avantage de A :

$$\delta(n) = |\Pr_{c \leftarrow E(k, m_0)}[A(c) = 1] - \Pr_{c \leftarrow E(k, m_1)}[A(c) = 1]|$$

Négligeable si $\delta(n) < \frac{1}{\text{poly}(n)}$ → « IND-CPA », E sémantiquement sécurisé

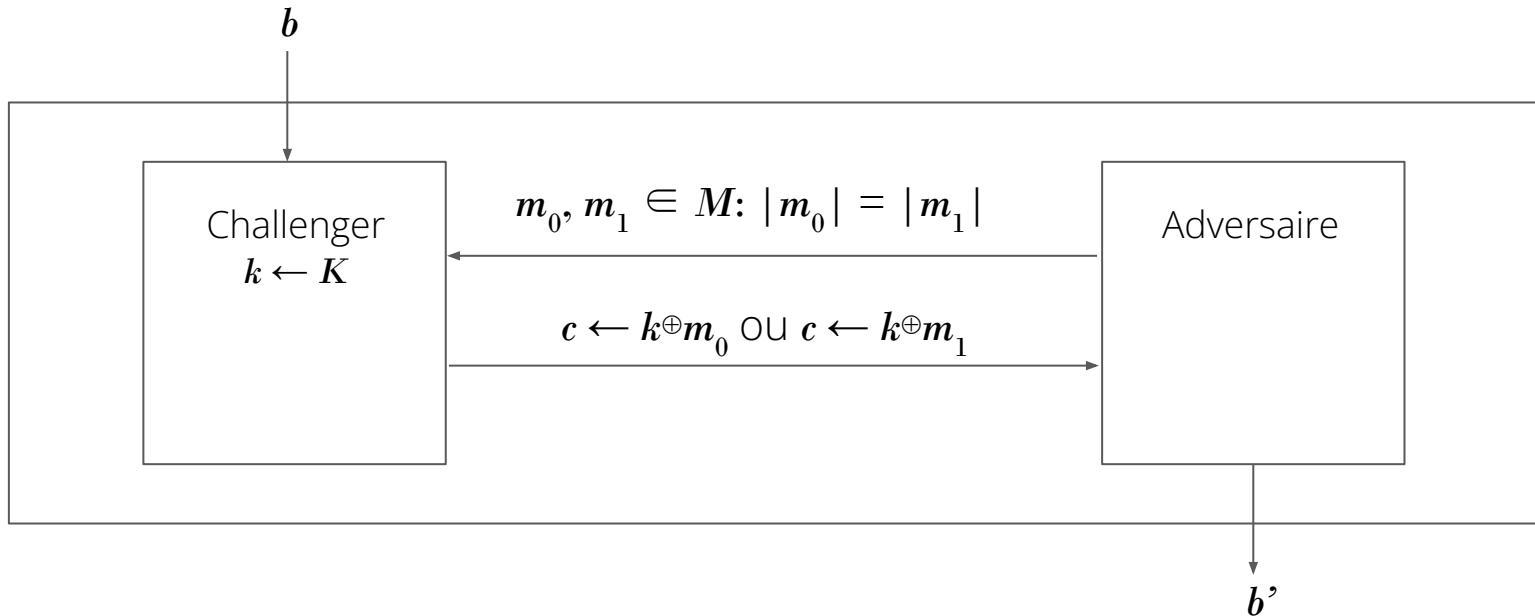
- **Attaque à texte clair choisi** (*chosen-plaintext attack*, CPA)

- **Sécurité sémantique (IND-CPA)** : pour un adversaire réaliste A (disposant des meilleurs algorithmes et capacités de calculs actuellement connus), deux messages chiffrés c_1 et c_2 , qui correspondent à des messages originaux différents m_1 et m_2 , sont calculatoirement indistinguables

Note : aussi appelée *ciphertext indistinguishability*

- Pour A , le chiffré c ne révèle rien sur la sémantique de m .
- $P_{c \leftarrow E(k, \square_o)}[A(c) = 1] = 1 - P_{c \leftarrow E(k, \square_i)}[A(c) = 1]$
 $\delta(n) = | 2 \cdot P_{c \leftarrow E(k, \square_o)}[A(c) = 1] - 1 |$

OTP est sémantiquement sécurisé



$$\delta(n) = |P[A(k \oplus m_0) = 1] - P[A(k \oplus m_1) = 1]| = |\frac{1}{2} - \frac{1}{2}| = 0, \text{ IND-CPA}$$

Rappel : un **XOR** entre une clef aléatoire et n'importe quel message produit une séquence également aléatoire

- **Attaque à texte chiffré choisi** (*chosen-ciphertext attack, CCA*)

Dans ce scénario, le cryptanalyste peut choisir arbitrairement un **nombre limité** de textes chiffrés et obtenir de l'oracle de déchiffrement les textes clairs correspondants

- **Attaque à texte chiffré choisi adaptative**

(*adaptive chosen-ciphertext attack, CCA2*)

Ici, le cryptanalyste n'est pas obligé de choisir tous les textes chiffrés dès le début : il peut adapter son choix de texte chiffré en fonction des textes clairs renvoyés par l'oracle

Sous ce modèle d'attaque, l'indistinguabilité est équivalente à la **non-malléabilité** ($\text{IND-CCA2} \Leftrightarrow \text{NM-CCA2}$)

- **Malléabilité**

Un cryptosystème est dit malléable s'il est possible de transformer un chiffré d'un message m en un chiffré pour un message $f(m)$ pour une fonction f connue sans connaître le message originel m ni obtenir d'information sur lui.

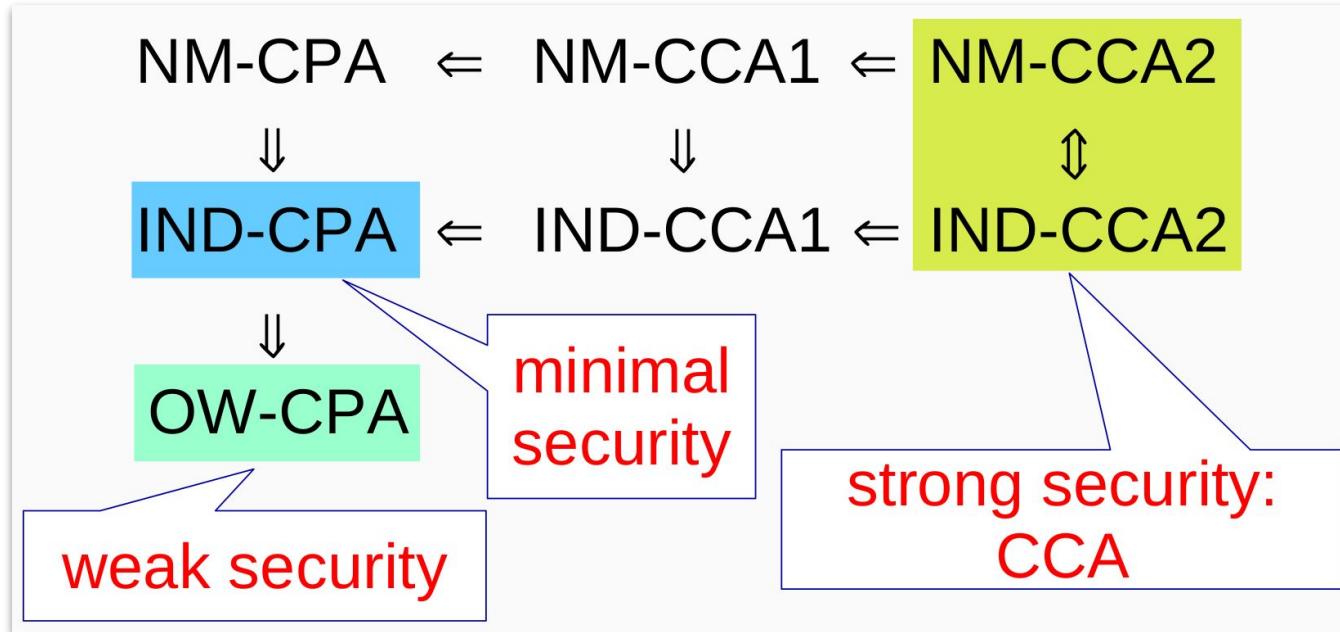
- Peut permettre à un attaquant de **modifier le contenu** des messages
- Permet d'exécuter des calculs sur des données chiffrés sans les connaître :

- ↳ **Chiffrement homomorphe**

- Chiffrement de flux

Avec un second message t , un adversaire peut construire un chiffrement de $m \oplus t$, tq $t \oplus E(m) = t \oplus m \oplus G(k) = E(t \oplus m)$

- **Solutions** : attacher au message un MAC, signature ou ZKP...



À voir dans le chapitre des fonctions de hachages et signatures numériques :
 OW-CPA, *One-Way Chosen-Plaintext Attack*
 EUF-NMA, *Existential Unforgeability under No-Message Attacks*

- **Principe de Kerckhoffs :**

- Axiome guidant la conception des cryptosystèmes (1883)
« La sécurité d'un cryptosystème ne doit reposer que sur le secret de la clé. »
- Autrement dit, on suppose que l'adversaire connaît tous les détails de l'algorithme, exceptée la clé de chiffrement
 - Expression connue sous le nom de la **maxime de Shannon** (1949)
- S'oppose à la **sécurité par l'obscurité**
 - Mesure insuffisante

4. Chiffrements asymétriques

- **Nombres premiers entre eux**

On dit que deux entiers a et b sont premiers entre eux (ou a est (co)premier avec b), si leur plus grand commun diviseur (pgcd) est égal à 1 :

$$\text{pgcd}(a, b) = 1$$

En d'autres termes, s'ils n'ont aucun diviseur autre que 1 et -1 en commun. De manière équivalente, ils sont premiers entre eux s'ils n'ont aucun facteur premier en commun.

- **Théorème de Bézout**

Pour deux entiers relatifs a et b , il existe deux entiers relatifs x et y tq :

$$ax + by = \text{pgcd}(a, b)$$

→ a et b sont copremiersssi l'équation $ax + by = 1$ admet des solutions

- **Indicatrice d'Euler**

Fonction arithmétique de la théorie des nombres, qui à tout entier naturel n non nul associe le nombre d'entiers compris entre 1 et n (inclus) et copremiers avec n .

Exemple pour $n = 8$:

- Les entiers positifs ≤ 8 sont $\{1, 2, 3, 4, 5, 6, 7, 8\}$
- Parmis eux, ceux qui sont copremiers avec 8 sont $\{1, 3, 7, 5\}$
- Donc $\varphi(8) = 4$

- Si n est premier, $\varphi(n) = n - 1$, car $\text{pgcd}(a, a) = a$
- Si n est le produit de deux nombres premiers distincts p et q : $\varphi(n) = (p - 1) \times (q - 1)$

Explications : il y a $p - 1$ multiples de p qui sont $\leq n$ et $q - 1$ multiples de q qui sont $\leq n$

Donc le nombre total d'entiers qui sont $\leq n$ et premiers avec n :

$$n - (p - 1) - (q - 1) = n - p - q + 1 = (p \times q) - (p + q) + 1 = (p - 1) \times (q - 1)$$

- **Algorithme d'Euclide**

Permet de déterminer si deux nombres entiers sont premiers entre eux

1. Soit deux entiers a et b , avec $a \geq b$
2. On divise a par b et on garde le reste r
3. On remplace a par b et b par r
4. On répète le processus, jusqu'à ce que b soit égal à 0

Le **pgcd** des deux entiers initiaux est la valeur non-nulle de b quand le processus se termine

Exemple

$$\text{pgcd}(252, 105) = ?$$

$$252 \% 105 = 42$$

$$105 \% 42 = 21$$

$$42 \% 21 = 0$$

$$\rightarrow \text{pgcd}(252, 105) = 21$$

Rappel : l'opérateur modulo % donne le reste d'une division **euclidienne**

- **Algorithme d'Euclide étendu**

Permet de calculer le $\text{pgcd}(a, b)$ — comme l'algorithme d'Euclide — ainsi que les entiers x et y du théorème de Bézout tq :

$$ax + by = \text{pgcd}(a, b)$$

Pseudo-code :

```
function extended_gcd(a, b):
    if b == 0:
        return a, 1, 0
    else:
        gcd, x, y = extended_gcd(b, a%b)
        return gcd, y, x - (a//b) * y
```

Algorithme d'Euclide

Pourrait être sur une seule colonne avec l'opérateur %

Exemple : $a = 240, b = 46$

| index i | quotient q_{i-1} | Remainder r_i | s_i | t_i |
|-----------|--------------------|--------------------------|------------------------|----------------------------|
| 0 | | 240 | | 0 |
| 1 | | 46 | | 1 |
| 2 | $240 \div 46 = 5$ | $240 - 5 \times 46 = 10$ | $1 - 5 \times 0 = 1$ | $0 - 5 \times 1 = -5$ |
| 3 | $46 \div 10 = 4$ | $46 - 4 \times 10 = 6$ | $0 - 4 \times 1 = -4$ | $1 - 4 \times -5 = 21$ |
| 4 | $10 \div 6 = 1$ | $10 - 1 \times 6 = 4$ | $1 - 1 \times -4 = 5$ | $-5 - 1 \times 21 = -26$ |
| 5 | $6 \div 4 = 1$ | $6 - 1 \times 4 = 2$ | $-4 - 1 \times 5 = -9$ | $21 - 1 \times -26 = 47$ |
| 6 | $4 \div 2 = 2$ | $4 - 2 \times 2 = 0$ | $5 - 2 \times -9 = 23$ | $-26 - 2 \times 47 = -120$ |

$$240 \times -9 + 46 \times 47 = 2$$

• Arithmétique modulaire

Ensemble de méthodes permettant la résolution de problèmes sur les nombres entiers.

Ces méthodes dérivent de l'étude du reste obtenu par une division euclidienne.

Exemple : sur un cadran d'horloge, 21h00 se lit également « 09h00 du soir ».

9 n'est pas égal à 21, mais sur un cadran de 12 entiers, on dit que 9 et 21 sont « congrus modulo 12 » et cela s'écrit : $9 \equiv 21 \pmod{12}$

Cette relation d'équivalence s'applique également aux entiers relatifs : $-1 \equiv 11 \pmod{12}$

Propriété de la congruence sur les entiers a et $b \in \mathbb{Z}$:

$a \equiv b \pmod{n}$, avec $n \in \mathbb{N}^*$

→ n divise $a - b$

→ $a = b + kn$, avec $k \in \mathbb{Z}$

→ les divisions euclidiennes de a et b par n ont le même reste

Remarques : il n'y a pas de congruence modulo 0, mais la relation d'égalité ;

le module correspond à la base de numération;

un **XOR** est une addition modulo 2 sans retenue

- **Classe d'équivalence**

Ensemble des entiers qui sont congrus modulo n

La classe d'équivalence d'un entier a modulo n , notée $[a]_n$, est l'ensemble des entiers qui sont congrus à a modulo n . Elle est définie formellement comme :

$$[a]_n = \{x \in \mathbb{Z} : x \equiv a \pmod{n}\}$$

Exemple : la classe d'équivalence de **9 mod 12** inclus tous les entiers qui laissent **un reste de 9** quand on les divise par 12, comme

$$[9]_{12} = \{-15, -3, 9, 21, 33, \dots\}$$

- **Notations**

L'ensemble des entiers modulo n est noté \mathbb{Z}_n .

Il peut être défini formellement comme :

$$\mathbb{Z}_n = \{0, 1, 2, \dots, n - 1\}$$

C'est l'ensemble de **tous les restes** possibles quand on divise des entiers par n .

Une notation alternative est $\mathbb{Z}/n\mathbb{Z}$.

Elle représente **l'ensemble des classes d'équivalence** modulo n formée en divisant

- l'ensemble \mathbb{Z} de tous les entiers
- par l'ensemble $n\mathbb{Z}$ des entiers multiple de n

- **Inverse modulaire**

u est l'inverse modulaire de a si $au \equiv 1 \pmod{n}$

On peut alors écrire $u \equiv a^{-1} \pmod{n}$

Condition

u existe ssi $\text{pgcd}(a, n) = 1$

Unicité

$$11 \times 2 = 22 \equiv 1 \pmod{7}$$

$$11 \times 9 = 99 \equiv 1 \pmod{7}$$

Mais 2 est l'unique inverse modulaire de 11 modulo 7, car $9 \equiv 2 \pmod{7}$

- **Calcul de l'inverse modulaire**

Si a et b sont copremiers, alors $\text{pgdc}(a, b) = 1$

Il existe un inverse modulaire

- de a modulo b
- de b modulo a

Par le théorème de Bézout, on a : $ax + by = \text{pgdc}(a, b) = 1$

$ax = 1 + (-yb)$ x est l'inverse modulaire de a modulo b

$by = 1 + (-xa)$ y est l'inverse modulaire de b modulo a

Rappel : $au \equiv 1 \pmod{b} \rightarrow au = 1 + kb$

Ainsi, l'algorithme d'Euclide étendu permettra de calculer l'inverse modulaire

- **Division modulaire**

$$a \div b = a \times (1/b) \equiv a \times c \bmod n$$

avec c inverse modulaire de b , tq $b \times c \equiv 1 \bmod n$

Exemples :

$$8 \div 3 \bmod 5 ?$$

$$3 \times 2 = 6 \equiv 1 \bmod 5$$

$$8 \div 3 \bmod 5 = 8 \times 2 = 16 \equiv 1 \bmod 5$$

$$11 \div 4 \bmod 5 ?$$

$$4 \times 4 = 16 \equiv 1 \bmod 5$$

$$11 \div 4 \bmod 5 = 11 \times 4 = 44 \equiv 4 \bmod 5$$

- **⚠ L'opérateur modulo, noté %, n'existe qu'en informatique !**

En mathématique, **mod** désigne le module de la relation de congruence.

$$a \equiv b \text{ mod } n$$
 

$$a \equiv b$$
 

$$a \% b$$
 

$$a \text{ mod } b$$
 

$$a = b \% n$$
 

$$\text{On a : } a \equiv b \text{ mod } n \Leftrightarrow b \equiv a \text{ mod } n$$

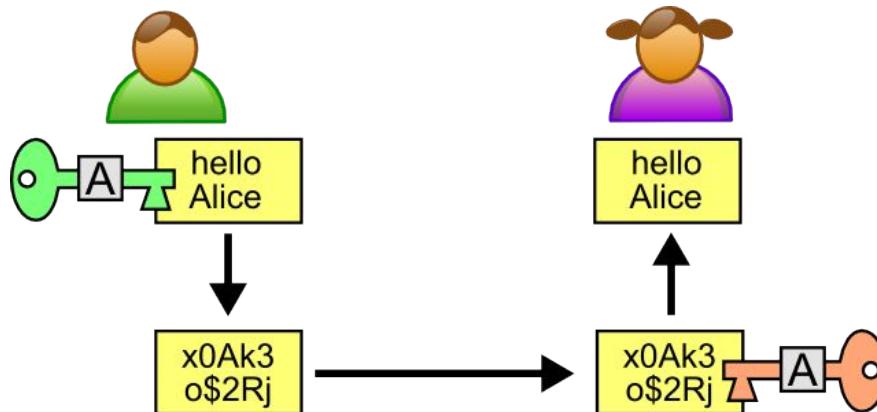
Cela serait faux si **mod** pouvait désigner un opérateur.

L'utilisation de parenthèses est donc facultative

- ### Motivation

En cryptographie symétrique, la même clé est utilisée pour le chiffrement et le déchiffrement (→ symétrie). Cette clé doit donc être **transmise tout en étant maintenue secrète** (→ aussi appelée « cryptographie à clé secrète »), ce qui peut être très contraignant...

La cryptographie asymétrique fonctionne avec **deux clés** : une clé publique, partagée avec tous les expéditeurs pour le chiffrement des messages et une clé privée que seul le destinataire possède pour déchiffrer les messages qui lui sont adressés.



Bob utilise la **clé publique d'Alice** pour chiffrer les messages qu'il lui envoie.

Alice utilise sa **clé privée** pour les déchiffrer.

On parle aussi de « cryptographie à clé publique ».

- **Théorème d'Euler-Fermat**

Pour tout module n et pour tout entier a copremier de n :

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

- **Chiffrement RSA** (Rivest, Shamir, Adleman, 1977)

Soit n le produit de deux nombres premiers distincts p et $q \rightarrow \varphi(n) = (p - 1) \times (q - 1)$

Soit l'exposant e tq $\text{pgcd}(e, \varphi(n)) = 1$

Soit le chiffré C du message clair M , produit avec la clé publique (e, n) :

$$C \equiv M^e \pmod{n}$$

Soit une clé privée (d, n) définie comme l'**inverse modulaire de e mod $\varphi(n)$** :

$$ed \equiv 1 \pmod{\varphi(n)} = 1 \pmod{(p - 1) \times (q - 1)}$$

On a donc le déchiffrement :

$$C^d = (M^e)^d = M^{ed} = M^{1 + k\varphi(n)} = M \times M^{\varphi(n)k} = M \times 1^k \equiv M \pmod{n}$$

- **Preuve de sécurité**

Un attaquant qui intercepterait C aurait besoin de générer la clé secrète d pour le déchiffrer. Il dispose également des valeurs publiques (e, n) . Il lui manque $\varphi(n)$.

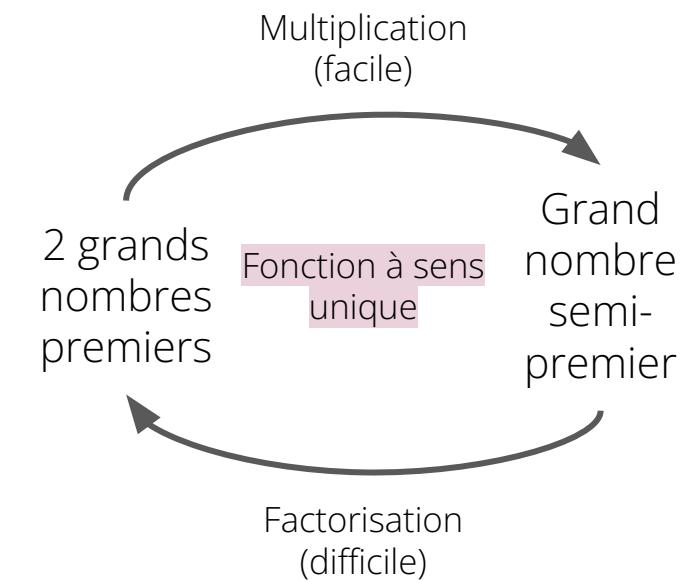
La **sécurité de RSA** repose sur le fait que calculer $\varphi(n)$ seulement à partir de n nécessite de [décomposer \$n\$ en produit de facteurs premiers](#), ce qui est un problème difficile.

- **Remarque**

À la place de $\varphi(n)$, il est possible d'utiliser l'indicatrice de Carmichael $\lambda(n)$. Cela permet de générer des clés privées plus courtes et d'accélérer les calculs, mais peut créer des vulnérabilités à prendre en considération...

- **Fonction à trappe**

Fonction qu'il est facile d'évaluer en chaque point de son domaine, mais qu'il est difficile d'inverser (fonction à sens unique) à moins de disposer d'une information particulière, appelée « trappe ».



Dans RSA, la trappe est la connaissance des facteurs premiers d'un grand nombre composite.

- **Choix des valeurs**

Mathématiquement, e pourrait être n'importe quel entier relatif, tq $\text{pgcd}(e, \varphi(n)) = 1$

Pour la vitesse des calculs, on choisit $1 < e < \varphi(n)$

Une valeur souvent utilisée pour e est 65537 qui vaut 1000000000000001 en base 2.
Cela facilite les calculs.

Le module n est choisi comme le produit de **deux grands nombres premiers**.

Par exemple, la taille de la clé n peut être de 2048 bits.

- **Déchiffrement accéléré** 

Il est possible de réduire la complexité du calcul de l'exposant de déchiffrement d , inverse modulaire de l'exposant public e , par rapport à l'algorithme d'Euclide étendu.

Pour cela, il faut décomposer le calcul en un système de congruences :

$$d_1 \equiv e^{-1} \pmod{(p-1)}$$

$$d_2 \equiv e^{-1} \pmod{(q-1)}$$

La solution est alors donnée par le **théorème des restes chinois** :

$$d = \sum e^{-1} N_i d_i \pmod{N}$$

$$\text{avec } N = (p-1) \times (q-1) = \varphi(n)$$

- **Théorème des restes chinois**

Dans un système à k équations (p. ex. $k = 3$), permet de trouver x tq :

$$x \equiv b_1 \pmod{n_1}$$

$$x \equiv b_2 \pmod{n_2}$$

$$x \equiv b_3 \pmod{n_3}$$

avec $\text{pgcd}(n_1, n_2) = \text{pgcd}(n_1, n_3) = \text{pgcd}(n_2, n_3) = 1$

On définit :

$$N = n_1 n_2 n_3$$

$$N_i = N/n_i$$

$$x_i \equiv N_i^{-1} \pmod{n_i}$$

| b_i | N_i | x_i | $b_i N_i x_i$ |
|-------|-----------------|-------|---------------|
| b_1 | $N_1 = n_2 n_3$ | x_1 | $b_1 N_1 x_1$ |
| b_2 | $N_2 = n_1 n_3$ | x_2 | $b_2 N_2 x_2$ |
| b_3 | $N_3 = n_1 n_2$ | x_3 | $b_3 N_3 x_3$ |

Solution : $x = \sum b_i N_i x_i \pmod{N}$

- De même que la décomposition en facteurs premiers, il existe d'autres problèmes mathématiques dont la difficulté garantit la sécurité de certaines méthodes cryptographiques (fonctions à sens unique)
- **Problème du logarithme discret** (*DLOG problem*)

Le logarithme réel donne l'exposant x dans l'exponentiation de a par x :

$$a^x = b$$

Le logarithme discret peut être vu comme son analogue en arithmétique modulaire. Il donne l'exposant x dans l'exponentiation de g par x (modulo n) :

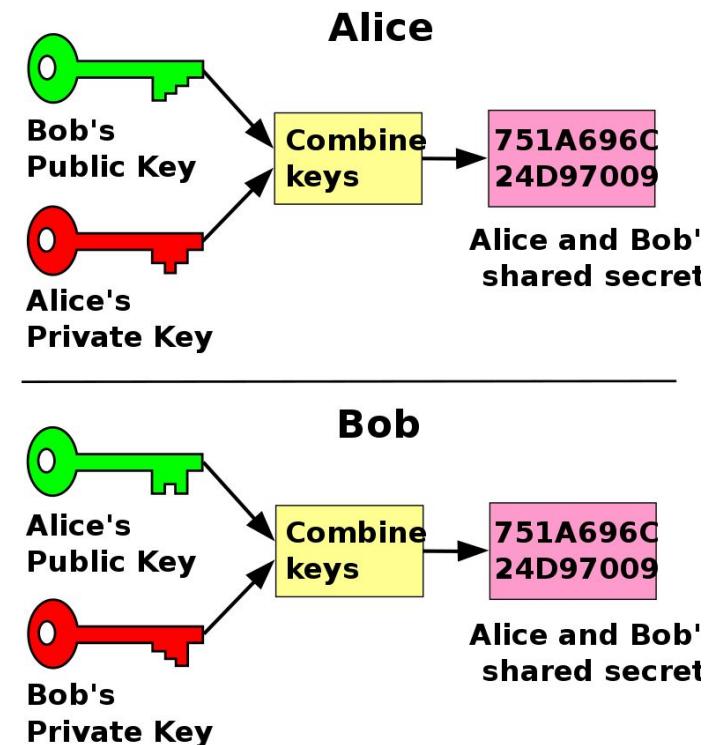
$$a^x \equiv b \pmod{n}$$

Cette exponentiation est facile à calculer, mais le calcul réciproque (logarithme discret) peut être très difficile pour certaines valeurs de a et de n . **Cette difficulté à calculer x garantit la sécurité du protocole d'échange de clé Diffie-Hellman (DH, 1976).**

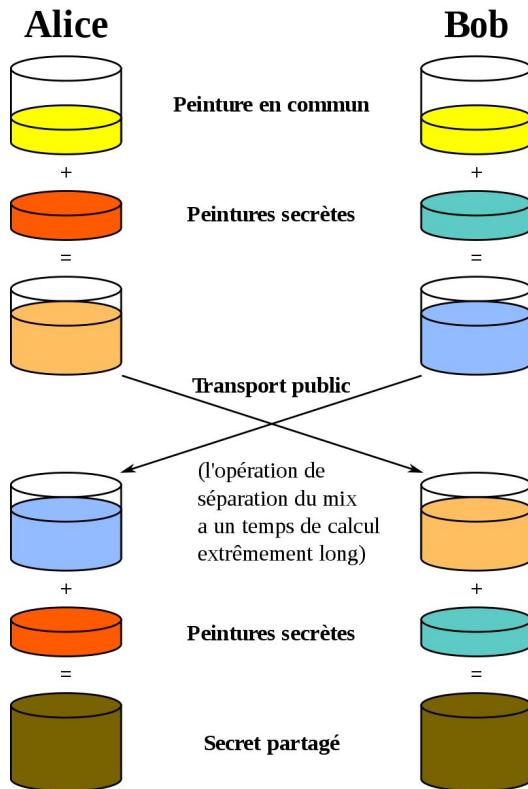
- **Principe général**

Il s'agit d'**établir une clé secrète** commune, que seuls Alice et Bob connaissent

Elle peut être utilisée ensuite pour un **chiffrement symétrique** des données à transmettre



- Calculs**



Variables publiques : le module p et le « générateur » g (échangées au début du *handshake* ou bien définies dans les standards)

Variables privées :

- Alice choisit aléatoirement (PRNG) $a \in [1, p-1]$
- Bob choisit aléatoirement (PRNG) $b \in [1, p-1]$

Étapes :

1. Alice calcule $g^a \pmod{p}$ et le transmet à Bob
2. Bob calcule $g^b \pmod{p}$ et le transmet à Alice
3. Alice calcule $(g^b)^a \pmod{p} = g^{ab} \pmod{p}$
4. Bob calcule $(g^a)^b \pmod{p} = g^{ab} \pmod{p}$

Même si Eve (*eavesdropper*) intercepte $g^a \pmod{p}$ ou $g^b \pmod{p}$, elle ne pourra pas extraire les couleurs privées a et b (même en connaissant g et p), et ne pourra donc pas calculer $g^{ab} \pmod{p}$, la couleur partagée.

- **Groupe**

Ensemble (de nombres ou autres objets) muni d'une loi de composition ou « opération binaire », notée $*$ et qui satisfait certaines propriétés.

Un groupe $(G, *)$ est défini par une loi de composition :

- **interne** : \forall éléments a et $b \in G$, $(a * b) \in G$
- **associative** : \forall éléments a , b et $c \in G$, $(a * b) * c = a * (b * c)$
- admettant un **élément neutre** $e \in G$ tq $e * a = a * e = a$, $\forall a \in G$
- avec, $\forall a \in G$, un **élément symétrique** $b \in G$ tq $a * b = b * a = e$, élément neutre

En cryptographie, on utilisera des groupes d'entiers relatifs modulo n , notés $(\mathbb{Z}_n, *)$, munis d'une opération d'addition ou de multiplication.

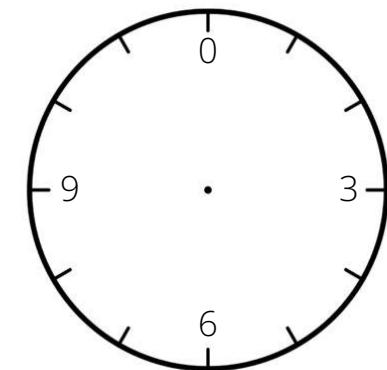
- **Exemple de groupe**

L'ensemble des entiers relatifs modulo 12 muni de l'addition $(\mathbb{Z}_{12}, +)$ est un groupe avec :

- 0 comme élément neutre
- l'opposé comme élément symétrique

Note : 0 et l'opposé sont les éléments neutre et symétrique, respectivement, pour tout groupe additif.

C'est un **groupe fini** : il est constitué d'un nombre fini d'éléments.



Ce nombre, noté $|(\mathbb{Z}_{12}, +)| = 12$, est appelé **ordre** du groupe.

C'est la **cardinalité** de l'ensemble qui forme le groupe.

- L'**ordre d'un élément** a du groupe est le plus petit entier $k > 0$ tq l'opération de groupe répétée k fois à partir de a donne l'élément neutre. Il est noté $|a|$.

Groupe additif : $a \cdot k \equiv 0 \pmod{n}$

Groupe multiplicatif : $a^k \equiv 1 \pmod{n}$

Exemple : l'élément 4 du groupe $(\mathbb{Z}_{12}, +)$ est d'ordre 3.

$$4 \times 1 \% 12 = 4$$

$$4 \times 7 \% 12 = 4$$

$$4 \times 2 \% 12 = 8$$

$$4 \times 8 \% 12 = 8$$

Note : la multiplication est l'opération d'addition répétée

$$4 \times 3 \% 12 = 0$$

$$4 \times 9 \% 12 = 0$$

$$4 \times 4 \% 12 = 4$$

$$4 \times 10 \% 12 = 4$$

$$4 \times 5 \% 12 = 8$$

$$4 \times 11 \% 12 = 8$$

$$4 \times 6 \% 12 = 0$$

$$4 \times 12 \% 12 = 0$$

L'ordre d'un élément est l'ordre du **sous-groupe** qu'il génère.

Il est parfois appelé **période**.

- **Autre exemple de groupe**

L'ensemble des entiers relatifs modulo **12** muni de la multiplication $(\mathbb{Z}_{12}, \times)$ est un groupe avec :

- 1 comme élément neutre
- **l'inverse comme élément symétrique**

Il comporte 4 éléments (« ordre 4 ») : ceux **ayant un inverse modulaire** et qui doivent donc être copremiers avec 12 ; $|(\mathbb{Z}_{12}, \times)| = \varphi(12) = 4 : \{1, 5, 7, 11\}$

Note : pour tout groupe multiplicatif, $|(\mathbb{Z}_n, \times)| = \varphi(n)$



À retenir...

Pour $(\mathbb{Z}_n, +)$, l'ordre est toujours **n**. Mais pour (\mathbb{Z}_n, \times) , l'ordre est égal à $\varphi(n) < n$, car il ne comporte que les éléments ayant un inverse modulo **n**.

L'ordre est le plus proche de **n**, quand **n** est premier, avec $|(\mathbb{Z}_n, \times)| = \varphi(n) = n - 1$

- **Générateur**

Un générateur est un élément qui, par répétitions de l'opération du groupe, peut générer tous les autres éléments du groupe.

Il est également appelé **élément primitif** du groupe.

L'ordre d'un générateur est égal à l'ordre du groupe.

Exemple : 5 est un générateur du groupe $(\mathbb{Z}_{12}, +)$.

$$5 \times 1 \% 12 = 5$$

$$5 \times 7 \% 12 = 11$$

$$5 \times 2 \% 12 = 10$$

$$5 \times 8 \% 12 = 4$$

$$5 \times 3 \% 12 = 3$$

$$5 \times 9 \% 12 = 9$$

$$5 \times 4 \% 12 = 8$$

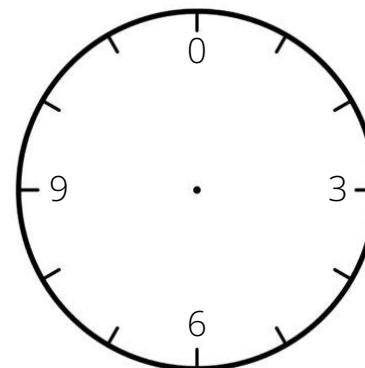
$$5 \times 10 \% 12 = 2$$

$$5 \times 5 \% 12 = 1$$

$$5 \times 11 \% 12 = 7$$

$$5 \times 6 \% 12 = 6$$

$$5 \times 12 \% 12 = 0$$



Groupe avec générateur(s) \Leftrightarrow **groupe cyclique**

- Le logarithme discret est aussi défini pour un groupe additif (même s'il devrait alors s'appeler « division discrète ») ; il donne x dans
$$a \times x \equiv b \pmod{n}$$
- Par analogie avec le logarithme réel, il faut que $x \exists$ et soit unique $\forall b$
Pour cela, **a doit être générateur** du groupe des entiers modulo n

Exemples pour $(\mathbb{Z}_n, +)$ et (\mathbb{Z}_n, \times) :

$x \not\exists$ pour

$$4 \times x \equiv 10 \pmod{12}$$

$$4^x \equiv 7 \pmod{12}$$

x n'est pas unique pour

$$4 \times x \equiv 8 \pmod{12}$$

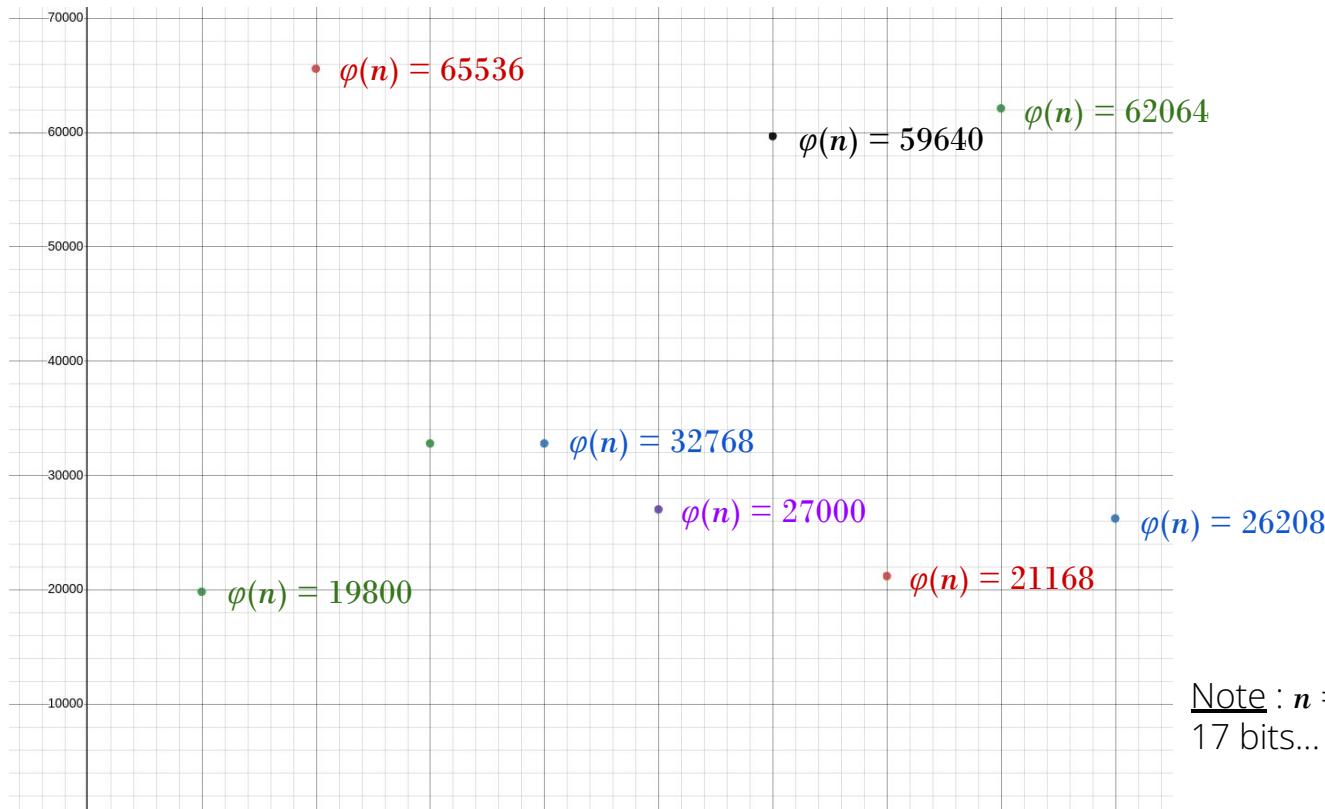
$$2^x \equiv 8 \pmod{12}$$

- La complexité du DLOG est bien moins grande pour un groupe additif que multiplicatif
On calcule donc : $a^x \equiv b \pmod{n}$, avec a générateur du groupe $(\mathbb{Z}/n\mathbb{Z})^\times$
Note : a est aussi appelé « racine primitive modulo n »
Par exemple, on utilise souvent $a = 2$ ou $a = 3$
 **La taille du générateur n'a pas d'effet sur la difficulté du DLOG** (cf. *random self-reducibility*)
- On choisit un **très grand nombre** pour le module n , entre 2048 et 4096 bits,
afin que l'ordre de $(\mathbb{Z}/n\mathbb{Z})^\times$ soit très grand pour qu'aucun adversaire ne puisse calculer le
DLOG par la force brute, ni par crible algébrique (cf. *function field sieve* et Logjam)
Ce très grand nombre n **doit être premier** :
 - Garantit que le groupe est cyclique (présence d'au moins un générateur)
[Corollaire du Théorème de Lagrange](#)
 - $|(\mathbb{Z}/n\mathbb{Z})^\times| = \phi(n) = n - 1$

Note : Pour se prémunir de certaines attaques, on utilise des nombres premiers p de Sophie Germain (*safe prime*) tq $p = 2q + 1$, avec q premier

4.6. Choix des paramètres

- **Illustration** : valeurs de $\varphi(n)$ pour n décroissant de 65538 à 65530



Note : $n = 65537$ fait seulement
17 bits...

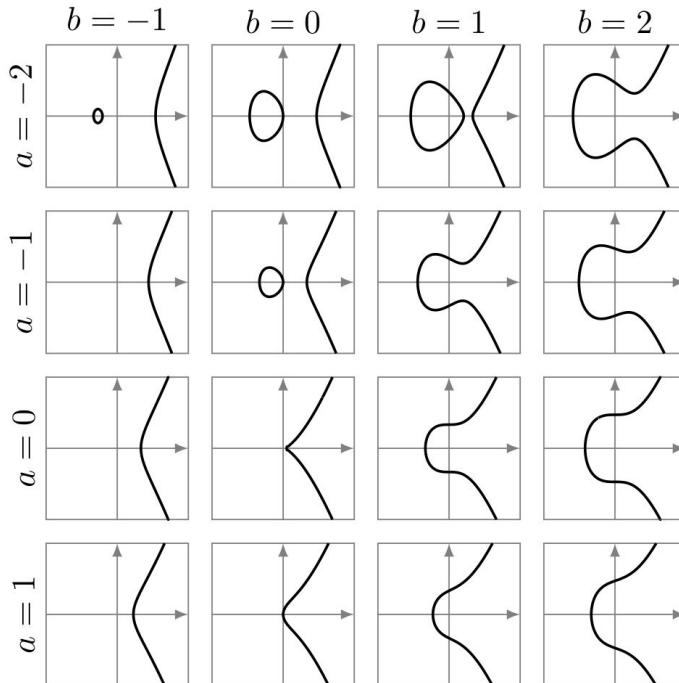
- Utiliser un module premier p fait de $(\mathbb{Z}/p\mathbb{Z})^\times$ le groupe multiplicatif d'un « corps fini » (*finite field* ou *Galois field*)
- **Finite Field DH (FFDH) vs Elliptic Curve DH (ECDH)**
Alternativement, il est possible d'utiliser une structure de groupe définie par des points sur une **courbe elliptique**
 - L'opération de ce groupe est l'addition, mais le calcul du DLOG y est plus complexe que sur $(\mathbb{Z}/p\mathbb{Z})^\times$
 - Cela permet d'utiliser des clés plus courtes, 256 ou 384 bits, ce qui accélère les calculs (p. ex. protocole TLS)

⚠ FFDH manipule directement des éléments dans un corps fini, alors que ECDH opère dans un groupe défini par des points sur une courbe elliptique, qui est elle-même définie sur un corps fini. Le calcul des points sur la courbe elliptique se fait donc modulo n , ordre du corps fini.

• Courbe elliptique

Courbe algébrique définie par une équation de la **forme de Weierstrass** :

$$y^2 = x^3 + ax + b$$

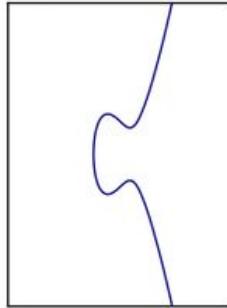


L'ellipse n'est pas une courbe elliptique !
 Le nom des courbes elliptiques vient historiquement de leur association avec les intégrales elliptiques, elles-mêmes appelées ainsi car elles servent en particulier à calculer la longueur d'arcs d'ellipses.

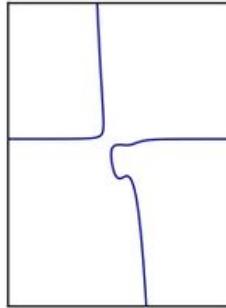
4.7. Courbes elliptiques

- Il existe d'autres formes de courbes elliptiques, chacune ayant ses avantages...

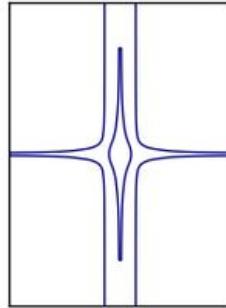
Short Weierstrass
 $y^2 = x^3 + ax + b$



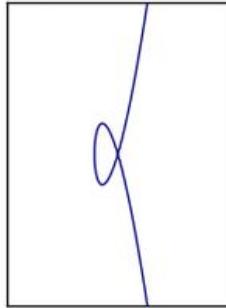
Koblitz
 $y^2 + xy = x^3 + ax^2 + 1$



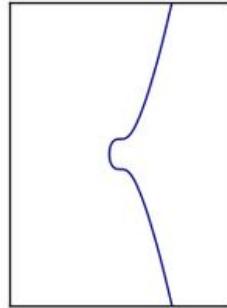
Edwards
 $y^2 + x^2 = 1 + dx^2y^2$



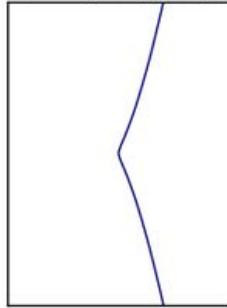
Montgomery
 $by^2 = x^3 + ax^2 + x$



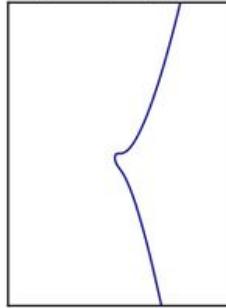
Barreto-Naherig
 $y^2 = x^3 + b$



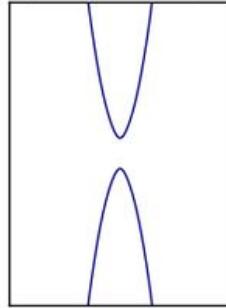
Jacobian
 $y^2 = x(x + 1)(x + \lambda)$



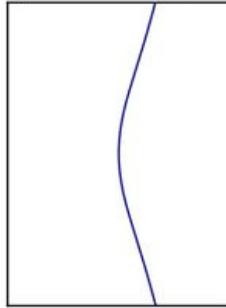
Hessian
 $by^2 + axy + by = x^3$



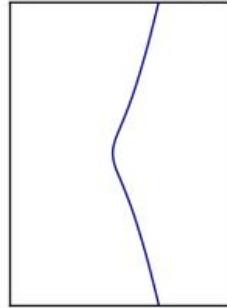
Jacobi quartic
 $y^2 = cy^4 + 2ax^2 + 1$



Doche-Icart-Kohel 2
 $y^2 = x^3 + ax^2 + 16ax$

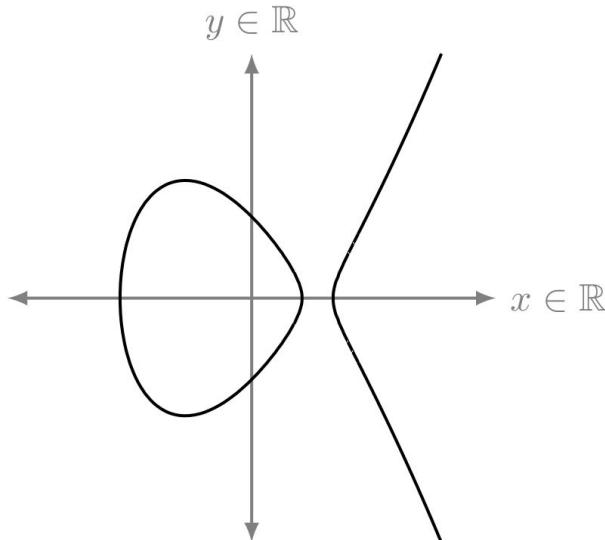


Doche-Icart-Kohel 3
 $y^2 = x^3 + 3a(x + 1)^2$

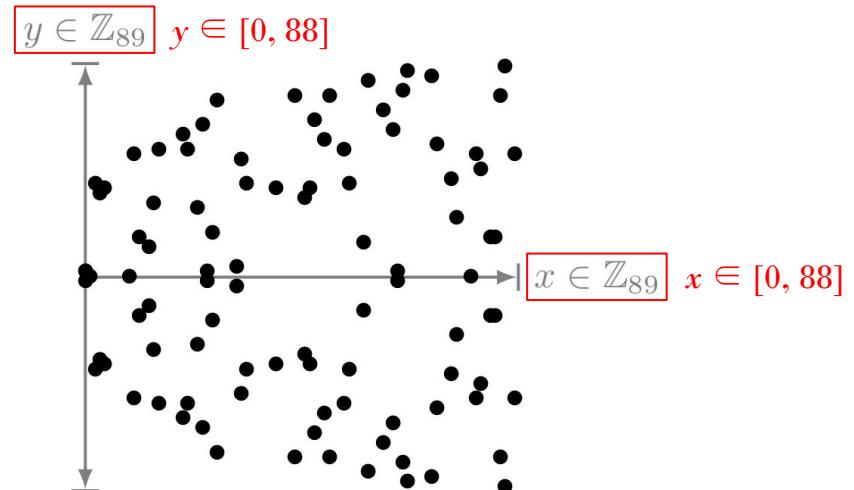


4.7. Courbes elliptiques

- Définition d'une courbe elliptique sur un corps fini **vs** sur les réels :



$$y^2 = x^3 - 2x + 1 \text{ over } \mathbb{R}$$



$$y^2 = x^3 - 2x + 1 \text{ over } \mathbb{Z}_{89}$$

$$y^2 \equiv x^3 + ax + b \pmod{n}$$

- **Deux caractéristiques notables** d'une courbe elliptique :

- 1.** symétrique par rapport à l'axe des abscisses
- 2.** intersection avec une droite en trois points maximum
équation polynomiale de degré 3

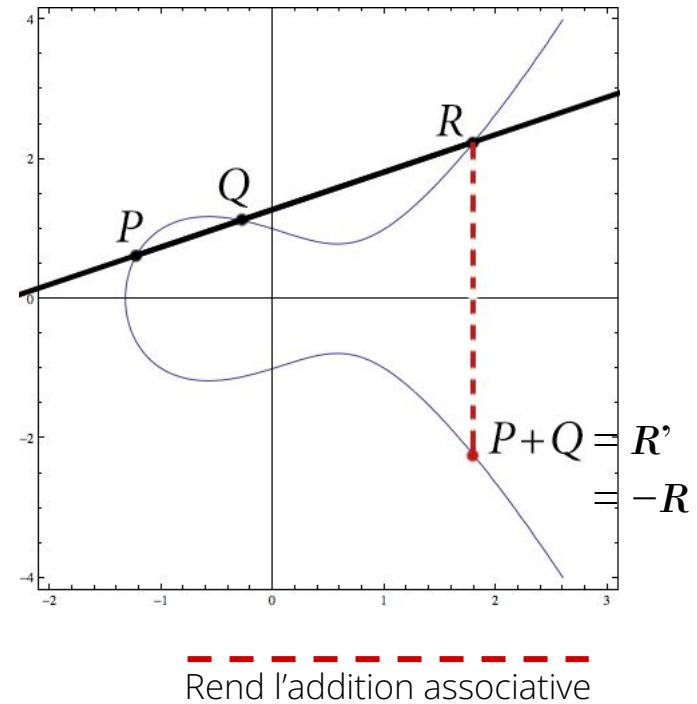
- **Addition de deux points distincts**

Calcul de la pente λ entre les deux points P et Q :

$$\lambda = \frac{y_q - y_p}{x_q - x_p}$$

$$x_{r'} = \lambda^2 - x_p - x_q$$

$$y_{r'} = \lambda(x_p - x_{r'}) - y_p$$



- **Addition d'un point P avec lui-même**

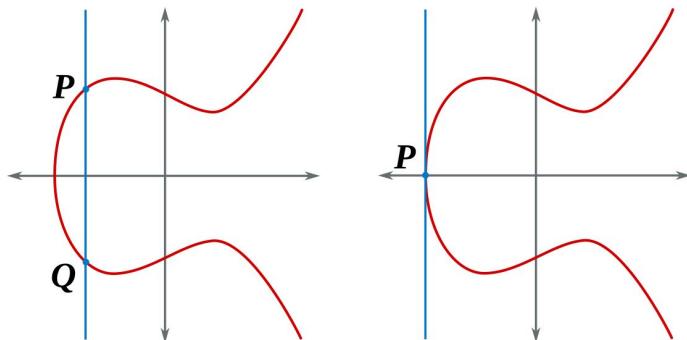
Pente λ de la tangente en P : dérivée de la courbe en P

$$\lambda = \frac{3x_p^2 + a}{2y_p}$$

- **Cas particuliers pour l'addition**

Si P et Q sont opposés, on définit $P + Q = P + (-P) = O$

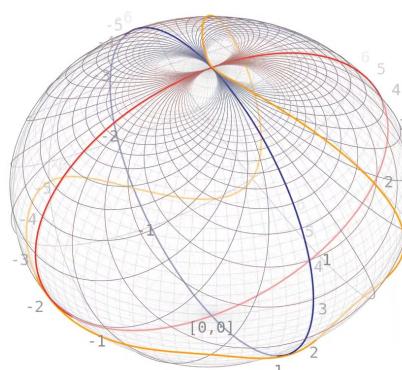
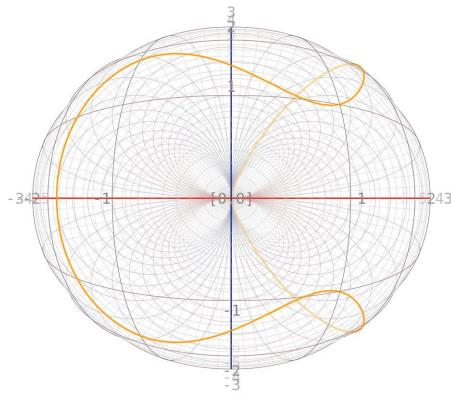
Si P est un point d'inflection (la tangente est verticale), on définit $P + P = O$



Le « point (imaginaire) à l'infini » O est l'élément neutre pour l'opération d'addition du groupe formés par les points de la courbe elliptique.

Sur le plan cartésien, ce n'est pas un point de la courbe : il représente la limite de la courbe à l'infini.

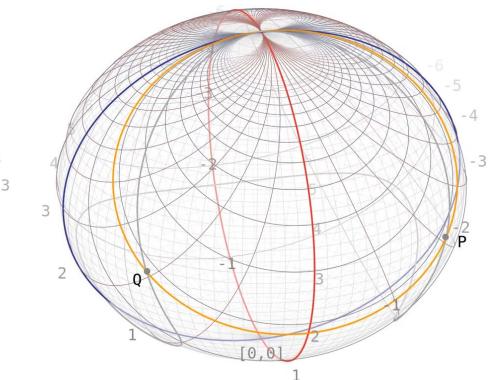
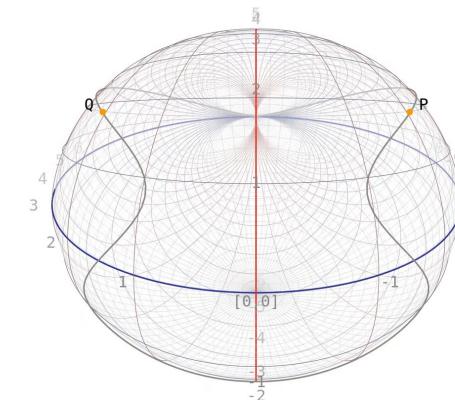
- Projection du plan de la courbe elliptique sur une sphère



L'origine du système de coordonnées est au pôle Sud. En allant vers l'infini dans n'importe quelle direction, on se rapproche infiniment du pôle Nord.

Sur le plan projectif (surface de la sphère), on voit que la courbe elliptique passe par le **point à l'infini**, au pôle nord.

Si on **additionne deux points opposés** P et Q , la ligne infinie passant par ces deux points forme un cercle qui coupe la courbe au point à l'infini.



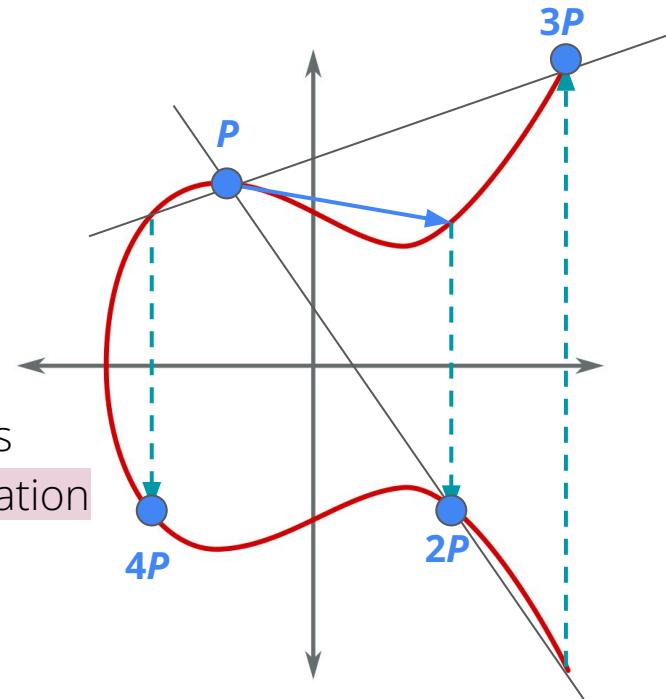
- **Génération des points sur une courbe elliptique**

À partir d'un point générateur G , les éléments du groupe sont calculés par additions répétées de G avec lui-même (multiplication par un scalaire modulo n)

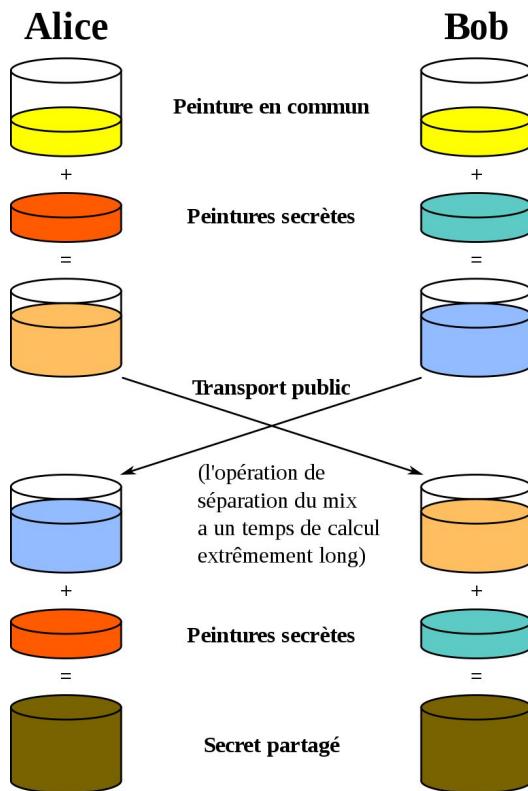
Le calcul du DLOG donne donc x dans
 $G \times x \equiv h \pmod{n}$

- **L'ordre de la courbe elliptique** est le nombre de points sur la courbe. Il est égal au nombre de solutions de l'équation de la courbe dans le corps fini sur lequel elle est définie.

Il doit être très grand afin de rendre difficile pour l'adversaire le calcul du DLOG



- **Calculs**



Variables publiques :

- une courbe elliptique spécifique
- un générateur G

Variables privées (grand nombres) :

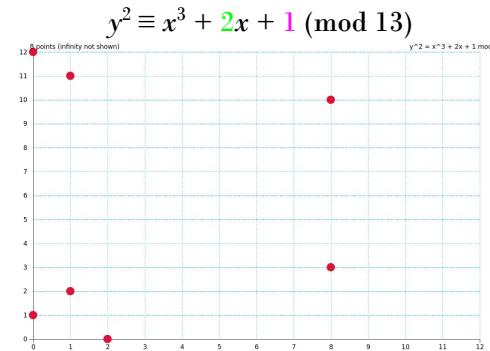
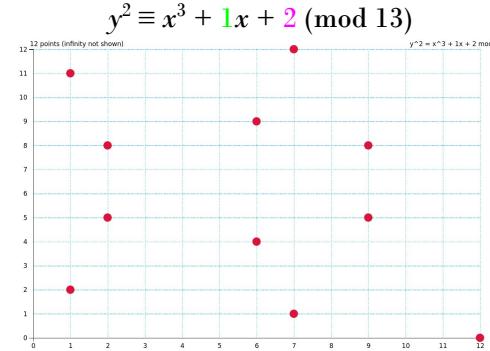
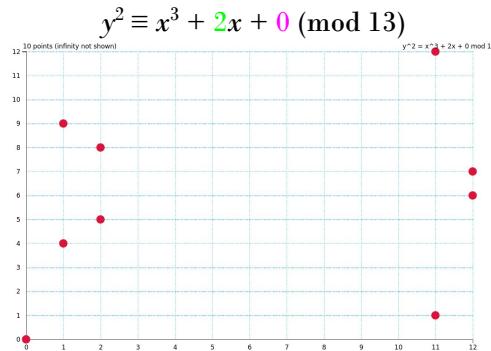
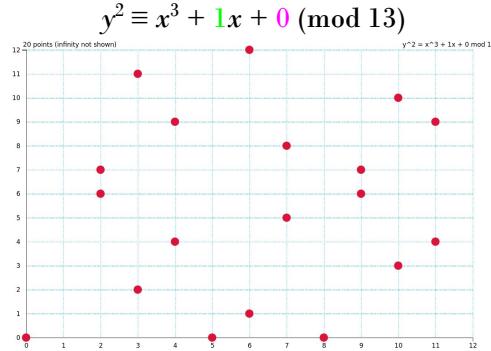
- Alice choisit aléatoirement (PRNG) $a \in [1, p-1]$
- Bob choisit aléatoirement (PRNG) $b \in [1, p-1]$

Étapes :

1. Alice calcule aG et le transmet à Bob
2. Bob calcule bG et le transmet à Alice
3. Alice calcule abG
4. Bob calcule abG

4.8. Choix des paramètres

- **Le nombre de points sur la courbe doit être le plus grand possible**
Ce nombre dépend **de la courbe choisie...**



- **Le nombre de points sur la courbe doit être le plus grand possible**

... dont **le module n** , ordre du corps fini sur lequel est définie la courbe

⚠ Ne pas confondre avec l'ordre de la courbe

- **n doit être un grand nombre**, car la limite supérieure du nombre de points sur la courbe est égale à n^2
- **n doit être premier**, afin que le nombre de point soit le plus proche de cette limite supérieure (cf. endomorphisme de Frobenius)

- **L'ordre de la courbe doit être premier**, afin que tous les points soient générateurs, excepté **0**

Corollaire du Théorème de Lagrange

- **Exemple** : courbe secp256k1, utilisée dans Bitcoin pour les signatures numériques...

$$y^2 \equiv x^3 + 7 \pmod{p}$$

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

Pour que p soit premier

Le corps fini doit être assez grand pour des clés privées de 256 bits



$$p =$$

115792089237316195423570985008687907853269984665640564039457584007908834671663

Ordre =

115792089237316195423570985008687907852837564279074904382605163141518161494337

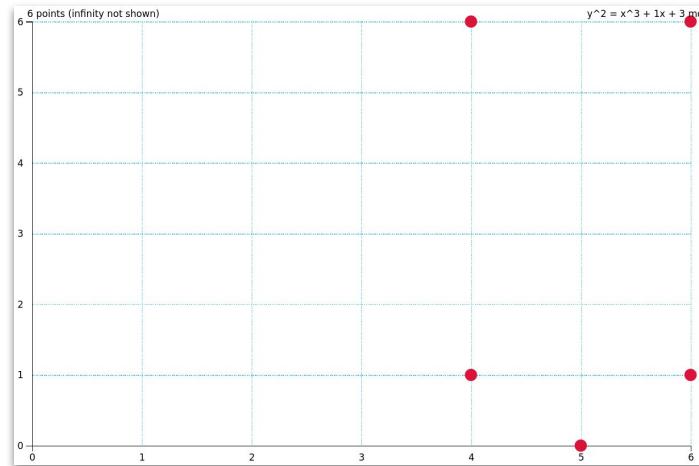
- Compter les points sur une courbe elliptique : approche naïve (exhaustive)**

Exemple : $y^2 \equiv x^3 + x + 3 \pmod{7}$

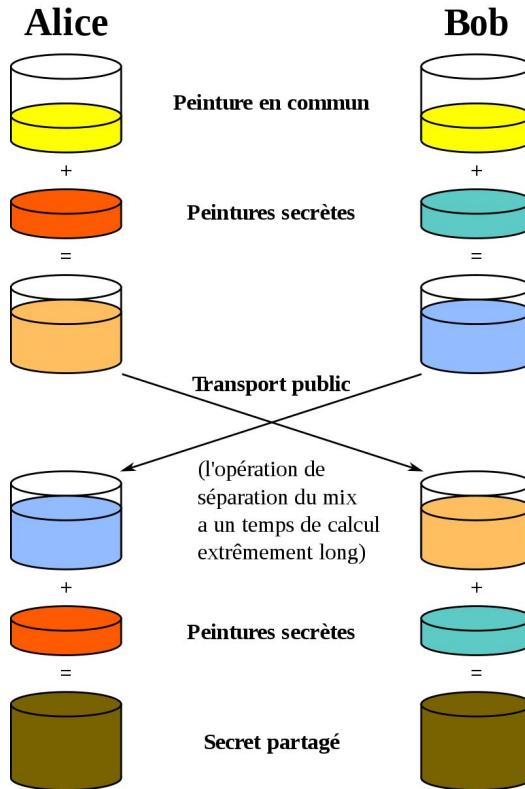
| x | $x^3 + x + 3$ | y | y^2 |
|-----|---------------|-----|-------|
| 0 | 3 | 0 | 0 |
| 1 | 5 | 1 | 1 |
| 2 | 6 | 2 | 4 |
| 3 | 5 | 3 | 2 |
| 4 | 1 | 4 | 2 |
| 5 | 0 | 5 | 4 |
| 6 | 1 | 6 | 1 |

Les valeurs (x, y) , solutions de l'équation, sont les coordonnées des points de la courbe.

Les **six points** faisant partie de cette courbe sont :
(4,1), (4,6), (5,0), (6,1), (6,6) et O , point imaginaire à l'infini.



4.9. Chiffrement ElGamal (1985)



Variables publiques : le module p et le « générateur » g (échangées au début du *handshake* ou bien définies dans les standards)

Variables privées :

- Alice choisit aléatoirement (PRNG) $a \in [1, p-1]$
- Bob choisit aléatoirement (PRNG) $b \in [1, p-1]$

Étapes :

1. Alice calcule $g^a \pmod{p}$ et le transmet à Bob
2. Bob calcule $g^b \pmod{p}$ et le transmet à Alice
3. Alice calcule $(g^b)^a \pmod{p} = g^{ab} \pmod{p}$
4. Bob calcule $(g^a)^b \pmod{p} = g^{ab} \pmod{p}$

$g^{ab} \pmod{p}$, est une clé secrète k qui sert à chiffrer un message m tq :

$$c = m \cdot k \pmod{p}$$

Celui qui reçoit c peut le déchiffrer grâce à k^{-1} , inverse modulaire de k :

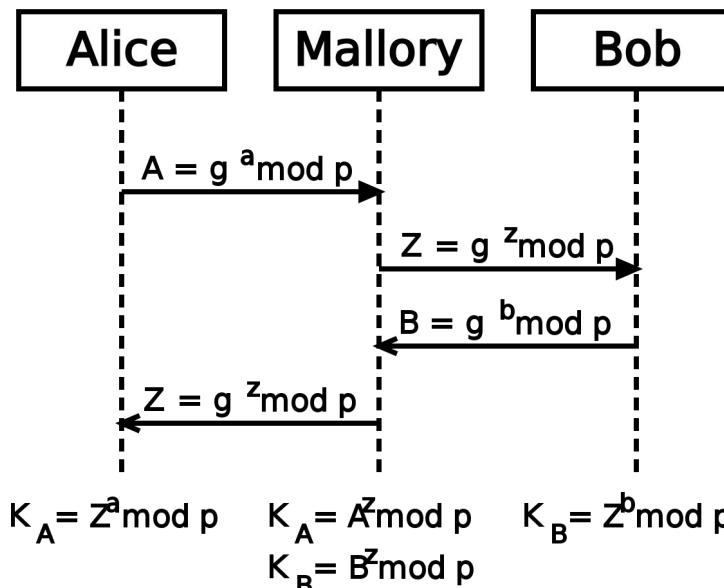
$$m = c \cdot k^{-1} \pmod{p}$$

ElGamal fonctionne aussi sur les courbes elliptiques, avec abg

- **Attaque de l'homme du milieu (*man-in-the-middle attack*)**

L'échange de clé (EC)DH est vulnérable à ce type d'attaque.

C'est pourquoi il est utilisé avec RSA pour **authentifier** l'origine des messages reçus...



● Méthode de factorisation de Fermat

Basée sur la représentation de tout entier naturel impair N comme la différence entre deux carrés :

$$N = a^2 - b^2 \text{ qui se factorise : } N = (a + b)(a - b)$$

FactorFermat(N) :

```
a ← ⌈√N⌉ // Partie entière supérieure : ⌈2,4⌉ = 3 et ⌈-2,4⌉ = -2
```

```
b2 ← axa - N
```

```
tant que b2 n'est pas un carré:
```

```
    a ← a + 1
```

```
    b2 ← axa - N
```

```
fin tant que
```

```
retourner A - √b2 // ou A + √b2
```

 Dans RSA, le module public est $n = pq$, avec p et q deux très grands nombres premiers

Si p et q **sont proches**, on obtient une solution après seulement quelques itérations !

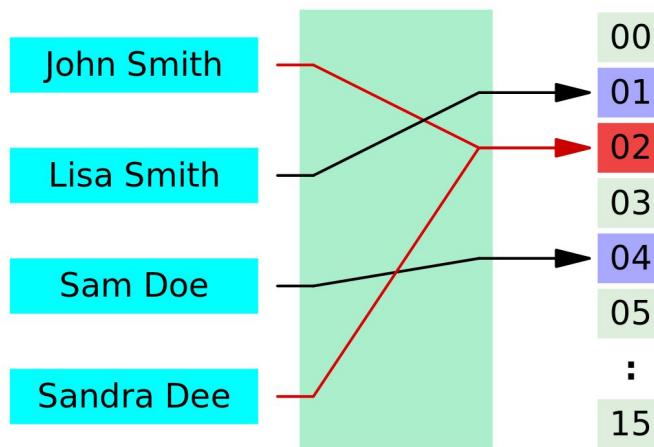
Böck, H. (2023). Fermat factorization in the wild. *Cryptology ePrint Archive*

5. Intégrité des données

● Définition

Une fonction de hachage est une fonction pouvant être utilisée pour associer des données de tailles arbitraires (entrée) à des valeurs de taille fixe (sortie).

Cette sortie est appelée image, ou « valeur de hachage », ou encore « empreinte »



Propriétés de base

- Déterministe
- Rapide à calculer
- Résistante aux **collisions** : minimiser la duplication des valeurs de sortie

⚠ Une fonction de hachage n'est **pas un chiffrement** : on ne peut pas retrouver l'entrée à partir de la sortie.
C'est une **fonction à sens unique**.

- **Fonctions de hachage simples**

Méthode par division (*modulo hashing*)

À partir d'une entrée m de longueur arbitraire, on calcule une valeur de hachage $m \% N$, qui sera donc toujours de longueur $N - 1$.

Exemples :

$$5454138768116315486435136 \% 19 = 13$$

$$789966531159753462179663222114447526991735569456454317 \% 19 = 11$$

$$465631201684153255378956238136873187676435154 \% 19 = 13$$

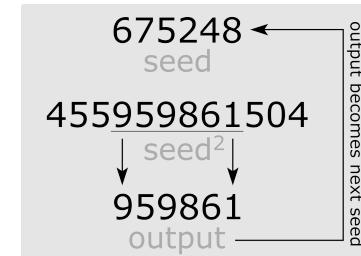
$$465631201684153255378956278136873187676435154 \% 19 = 15$$

Pour minimiser les collisions, il faudrait donc choisir un module N très grand et premier.

Méthode du carré médian (*mid-square hashing*)

Cette méthode, inventée par John Von Neumann en 1949, peut servir de PRNG et de fonction de hachage ; exemple : $h(3121) = 31212 = 9740641$

Note : on peut utiliser les PRNG comme fonction de hachage, l'entrée étant la graine. ⚠ La période dans les valeurs générées sera source de collisions.

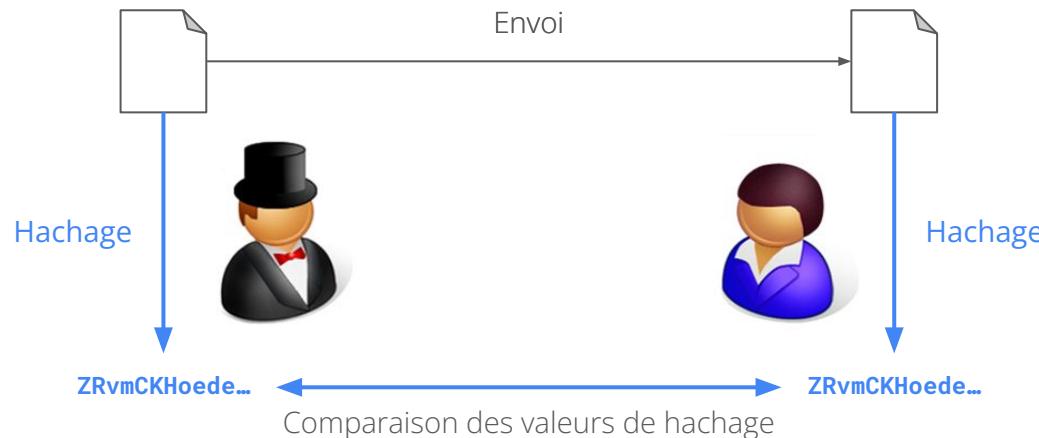


- **Intégrité**

Déetecter toute altération non-autorisée des données est une des trois propriétés des données que la cryptographie vise à garantir.

Les fonctions de hachage permettent une vérification rapide de cette intégrité, sans avoir à comparer la totalité des données.

La valeur de hachage étant une **séquence courte, elle est moins sujette aux erreurs**.



- **Méthode par pliage (*folding hash code*)**

Dans ce type de fonction de hachage, **l'entrée est divisée en blocs de même taille**, qui sont combinés entre eux (typiquement, par une opération **XOR** ou **ADD**) pour produire la valeur de hachage en sortie.

Exemple (avec addition de valeurs décimales) :

$$h(123456789) = 123 + 456 + 789 = 1368$$

Pour une sortie de longueur 3, on peut soit

- enlever le 1 ou le 8,
- ou bien calculer $1368 \% 997 = 371$, avec 997 le plus grand nombre premier à trois chiffres

Une variante (*boundary folding*) consiste à inverser alternativement un bloc sur deux, avant de les combiner :

$$h(123456789) = 123 + 654 + 789 = 1566$$

$$h(123456789) = 321 + 456 + 978 = 1764$$

- **Contrôle de redondance longitudinal (*longitudinal redundancy check, LRC*)**

Forme de **code correcteur**, qui est une technique destinée à corriger les erreurs de transmission d'un message.

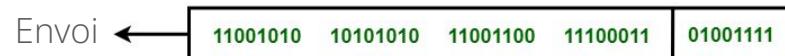
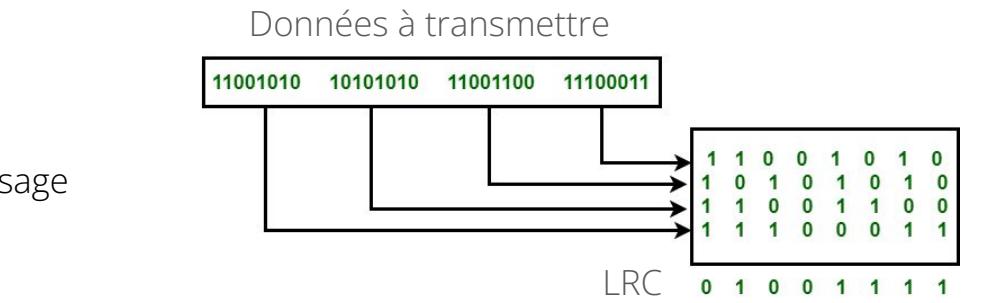
$$C_i = b_{i1} \oplus b_{i2} \oplus \dots \oplus b_{im}$$

avec

C_i : le i -ème bit du hachage, tq $1 \leq i \leq n$

m : nombre de **blocs de n bits** dans le message

b_{ij} : le i -ème bit dans le j -ème bloc



Bits de parité/LRC
Note : les bits d'**imparité** seraient 10110000

La longueur du LRC est celle des blocs.

Il est calculé pour tout le message (*message coding*).

- **Contrôle de redondance longitudinal (*longitudinal redundancy check, LRC*)**

LRC n'est généralement utilisé que pour **corriger des erreurs sur 1 seul bit**.

Message original, découpé en blocs de 3 bits : 000 000 000

Une seule erreur (tous les cas possibles) :

| | | | | | | | | | |
|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| 000 | 100 | 010 | 001 | 000 | 000 | 000 | 000 | 000 | 000 |
| 000 | 000 | 000 | 000 | 100 | 010 | 001 | 000 | 000 | 000 |
| 000 | 000 | 000 | 000 | 000 | 000 | 000 | 100 | 010 | 001 |
| 000 | 100 | 010 | 001 | 100 | 010 | 001 | 100 | 010 | 001 |

Plusieurs erreurs :

| | | | | | | | | | |
|------------|------------|------------|------------|------------|------------|------------|------------|------------|-----|
| 000 | 100 | 010 | 001 | 000 | 000 | 000 | 000 | 000 | 000 |
| 000 | 000 | 000 | 000 | 100 | 010 | 001 | 000 | 000 | 000 |
| 000 | 001 | 001 | 001 | 001 | 001 | 001 | 101 | 011 | 011 |
| 000 | 101 | 011 | 000 | 101 | 011 | 000 | 101 | 011 | |

Erreurs non-détectées, si **deux erreurs à la même position**, chacune sur un bloc différent.

- **Contrôle de redondance longitudinal (*longitudinal redundancy check, LRC*)**

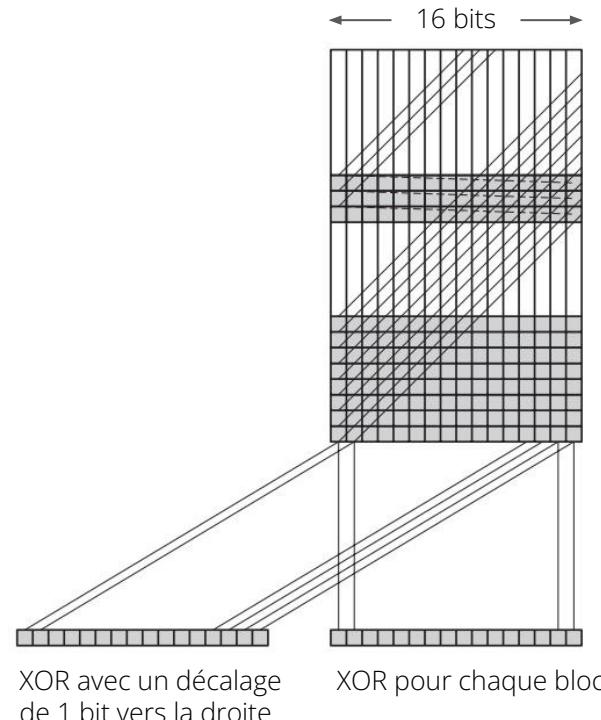
Amélioration possible

Pour des données aléatoires (uniformes), la probabilité qu'une erreur sur 1 bit ne change pas la valeur de hachage est de 2^{-n} .

Cette probabilité est plus élevée (donc le LRC est moins efficace) pour des données formatées.

Par exemple, dans la plupart des fichiers textes, le MSB de chaque octet est toujours égal à 0 (codage ASCII). Ainsi, si on utilise un LRC de 128 bits (soit 16 caractères ASCII), alors son efficacité n'est pas de 2^{-128} , mais de 2^{-112} .

Une façon de régler ce problème est de faire un **décalage circulaire** (à gauche ou à droite) de 1 bit sur la valeur de hachage avant le **XOR** de chaque bloc (figure ci-contre).



- **Contrôle de redondance vertical (*vertical redundancy check*, VRC)**

Le VRC est un code correcteur calculé pour chaque bloc du message.

C'est le bit de parité dans le codage ASCII (*character error coding*) :

| Carac | Hex | Binaire |
|-------|------|-----------|
| P | 0x50 | 0b1010000 |
| O | 0x4f | 0b1001111 |
| S | 0x53 | 0b1010011 |
| T | 0x54 | 0b1010100 |
| I | 0x49 | 0b1001001 |
| C | 0x43 | 0b1000011 |

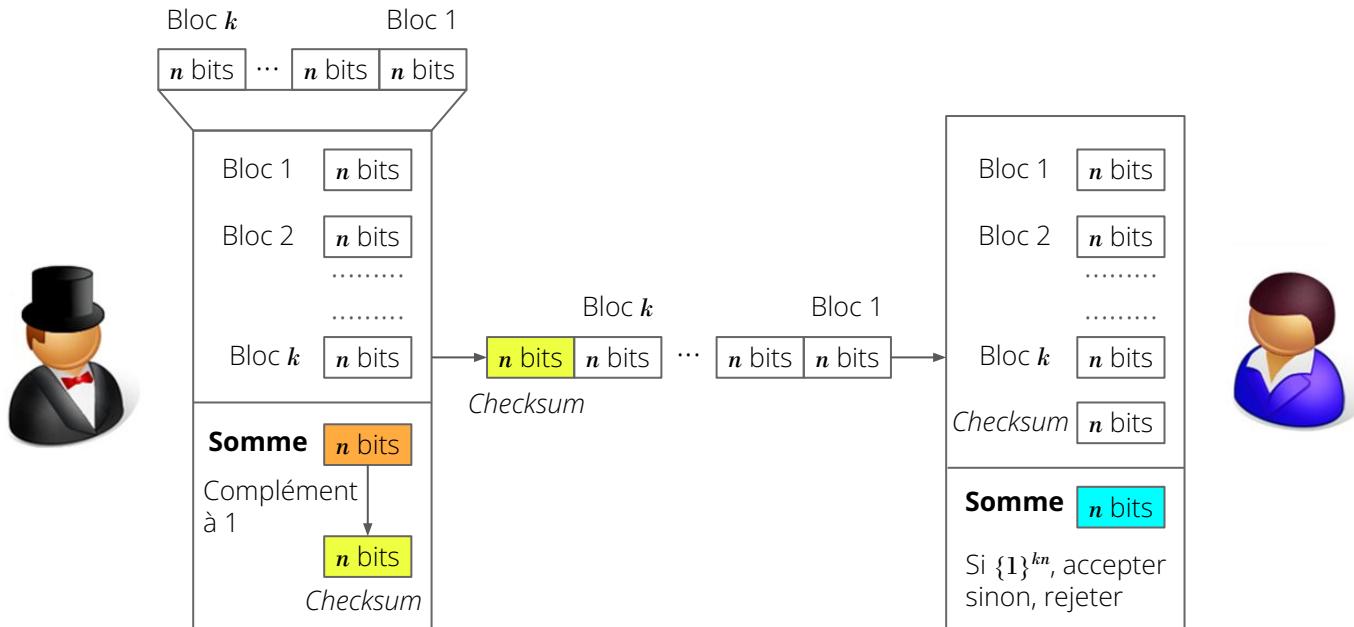
| Bit | Binaire | LRC |
|------------|--------------------|----------|
| | P O S T I C | |
| b 0 | 0 1 1 0 1 1 | 0 |
| b 1 | 0 1 1 0 0 1 | 1 |
| b 2 | 0 1 0 1 0 0 | 0 |
| b 3 | 0 1 0 0 1 0 | 0 |
| b 4 | 1 0 1 1 0 0 | 1 |
| b 5 | 0 0 0 0 0 0 | 0 |
| b 6 | 1 1 1 1 1 1 | 0 |
| VRC | 1 0 1 0 0 0 | 1 |

| | | | | | | |
|----------|----------|----------|----------|----------|----------|-----------------|
| P | 0 | S | T | I | C | |
| 00001011 | 11110010 | 11001011 | 00101010 | 10010010 | 11000010 | 01001001 |

● Somme de contrôle (checksum)

La combinaison entre les blocs se fait par **addition** binaire.

Le calcul du « complément à 1 » consiste à remplacer les 0 par des 1 et les 1 par des 0.



- **Somme de contrôle (*checksum*)**

Notes :

- Après chaque addition, la valeur obtenue peut nécessiter plus de ***n*** bits.

On coupe alors les **bits « surnuméraires »** pour les additionner avec les autres (*wrap around*) :

$$\begin{array}{r} 1011 \\ + 0110 \\ \hline 10001 \\ + 1 \\ \hline 0010 \end{array}$$

-  Le terme *checksum* est souvent utilisé de manière plus générale pour désigner toute méthode destinée à vérifier l'intégrité des données.

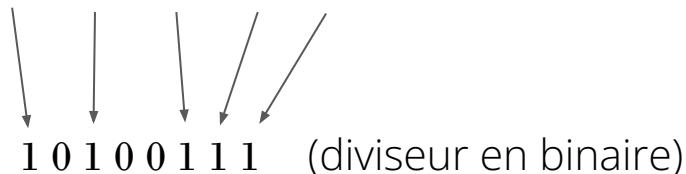
- **Contrôle de redondance cyclique (cyclic redundancy check, CRC)**

Deux étapes pour l'expéditeur :

1. Ajout de $L-1$ zéros au message original, avec L longueur d'un polynôme (le polynôme est prédéfini dans les standards)
2. Division binaire par le polynôme : **le reste est le CRC** (de longueur $L-1$ bits)

Le calcul du CRC s'applique sur des données formant un seul bloc, c.-à-d. **pas de pliage**.

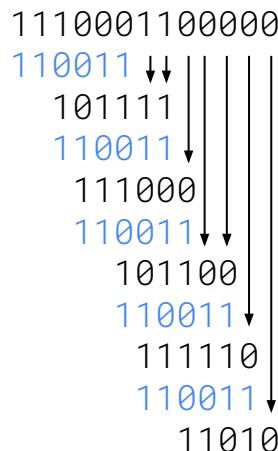
Exemple de polynôme : $x^7 + x^5 + x^2 + x + 1$



- **Contrôle de redondance cyclique (*cyclic redundancy check, CRC*)**

Division binaire par un polynôme

Exemple : 1110001100000 divisé par 110011



Étapes :

1. Alignement du diviseur sur le 1 le plus à gauche
2. XOR ; les 0 à gauches ne sont pas écrits
3. Ajouts de bits à droite pour égaler la longueur du diviseur

Le quotient n'a pas d'importance.

Dans le contexte du CRC, les données originales seraient 11100011, avec un *zero-padding* de 00000.

Le reste est ici de 11010. Le CRC sera donc 11010.

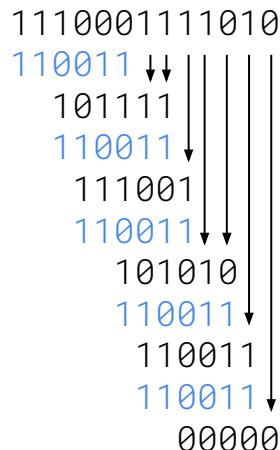
Le message transmis sera 1110001111010.

Note : c'est l'ajout des 0 qui donne son caractère **cyclique** à la division

- **Contrôle de redondance cyclique (*cyclic redundancy check, CRC*)**

Le destinataire divise le message reçu 1110001111010 par le diviseur 110011

↳ Si le reste est nul, le message a été transmis sans erreur



- **Contrôle de redondance cyclique (*cyclic redundancy check, CRC*)**

CRC est très largement utilisé pour la détection d'erreurs

- Protocoles réseaux : ethernet, Wi-Fi, TCP/IP, PPP...
- Stockage et systèmes de fichiers : RAID, NTFS, ZIP...
- Interfaces de communication : USB, SATA, PCIe...
- Etc.

Mais cette fonction de hachage est **facilement « réversible »** : pour une sortie donnée, il est facile de trouver une entrée possible.

Pour protéger l'intégrité des données contre les erreurs **et les altérations intentionnelles**, il est nécessaire d'utiliser des **fonctions de hachage cryptographiques**.

Celles-ci se caractérisent par les propriétés décrites ci-après.

- **Propriété n°1 : résistance aux collisions**

Pour les fonctions de hachage en général, le nombre de collisions possibles doit être minimisé.

Mais dans le cas des fonctions de hachage cryptographique, il doit être très difficile de trouver deux messages différents m_1 et m_2 tq

$$h(m_1) = h(m_2)$$

c.-à-d. ayant la même valeur de hachage.

Une attaque par force brute nécessiterait $2^{n/2}$ opérations.

- **Propriété n°2 : résistance à la préimage**

Pour une valeur de hachage donnée (ou image), il devrait être très difficile de construire un message ayant cette valeur.

Une attaque par [force brute nécessiterait \$2^n\$ opérations](#).

Exemple

Les mots de passe utilisateurs ne sont pas directement stockés sur un serveur : **seules leurs valeurs de hachage le sont**. Si un attaquant obtient une telle image, il devra trouver la préimage correspondante (c.-à-d. le mot de passe).

- **Propriété n°3 : résistance à la seconde préimage**

Pour une valeur de hachage donnée (ou image) et son message correspondant, il devrait être très difficile de construire un second message ayant cette valeur.

Une attaque par [force brute nécessiterait \$2^n\$ opérations](#).

Comme pour la résistance aux collisions, il devrait être très difficile de trouver des messages différents m_1 et m_2 tq $h(m_1) = h(m_2)$

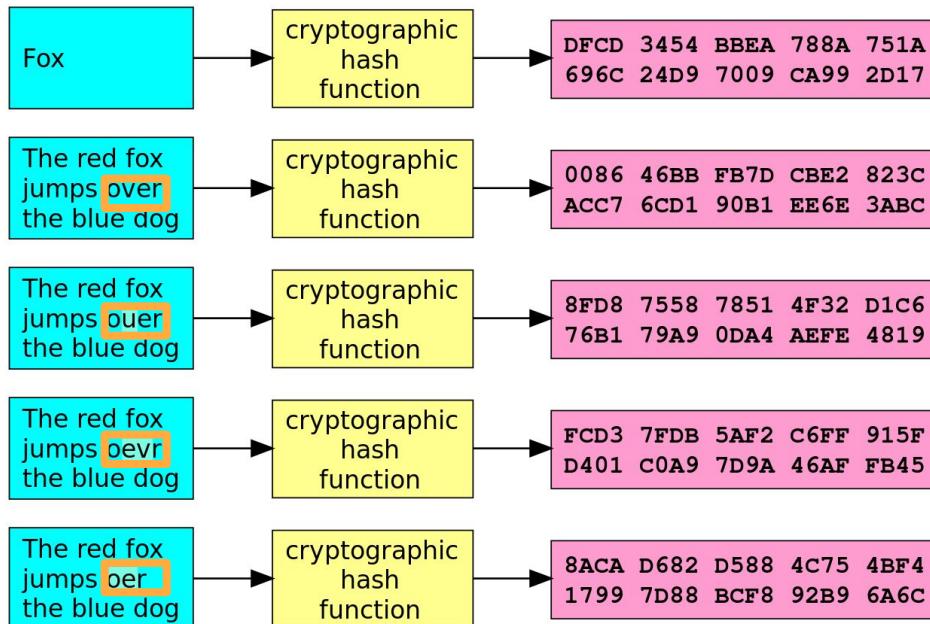
Mais ici, l'attaquant ne peut choisir que le message m_2 .

Exemple

Sans cette résistance, si un attaquant obtient un fichier et son image, il pourrait l'altérer sans que cela change la valeur de hachage.

- **Propriété n°4 : effet avalanche complet (ou uniformité)**

La sortie de la fonction doit être uniformément distribuée : **chaque bit en sortie dépend de tous les bits en entrée**. Ainsi, modifier un tant soit peu un message modifie considérablement la valeur de hachage.



Note : un effet avalanche est aussi désirable pour les fonctions de hachage non-cryptographiques, mais pour des raisons de performance

- **Propriété n°5 : rapidité de calcul**

La valeur de hachage d'un message doit se calculer « facilement ».

 Si le calcul est trop rapide, la fonction sera vulnérable aux attaques par force brute.

- **Propriété n°6 : déterminisme**

Comme pour toute fonction de hachage en général, une même valeur en entrée devra toujours produire la même valeur en sortie.

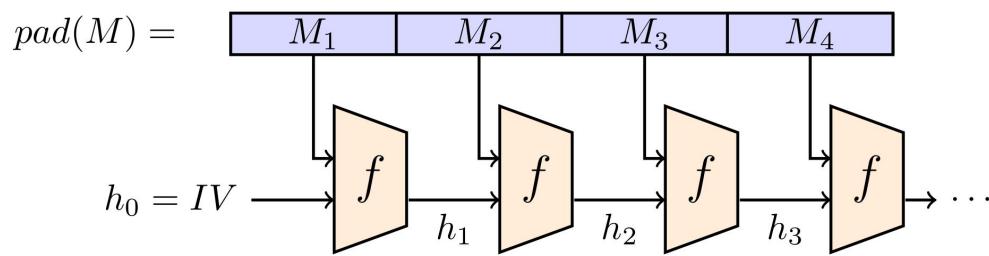
- **Construction de Merkle-Damgård** (1979)

Le message à hacher M est divisé en blocs de taille fixe et est rempli (*padding* du dernier bloc).

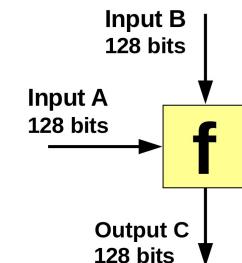
Les blocs sont ensuite envoyés itérativement dans une **fonction de compression à sens unique** f .

Le résultat de chaque compression est ensuite transmis au bloc suivant.

Ce résultat h_i est l'**état interne** ou « variable de chaînage ».



« Compression » :
en sortie, f produit
une séquence de
taille strictement
inférieure à celles
des **deux entrées**.



Propriété

si la fonction de compression est résistante aux collisions } alors la fonction de hachage sera aussi résistante
si un schéma de remplissage approprié est utilisé } aux collisions

- **Schéma de remplissage (*padding*)**

Remplissage avec des 0 (mentionné par Merkle)

↳ Engendre des **collisions** entre les messages se terminant par des 0

Exemple : avec des blocs de 4 bits, les messages

1111 1111 1, 1111 1111 10, 1111 1111 100 et 1111 1111 1000
produiront tous la même valeur de hachage

C'est pourquoi, **le premier bit de *padding* est un 1**. Ainsi, ces mêmes messages

1111 1111 1**00**, 1111 1111 1**010**, 1111 1111 1**001** et 1111 1111 1**000**

produiront toutes des valeurs de hachage différentes, car leurs blocs de *padding* sont différents

Renforcement de Merkle-Damgård

Pour encore plus de sécurité, on ajoute la longueur du message sur un nombre fixe de bits, à la fin :

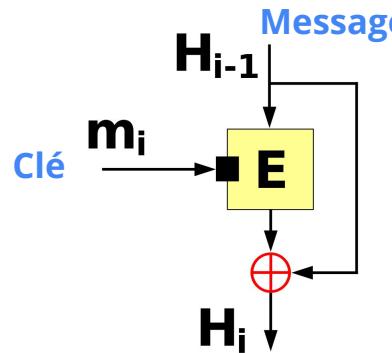
$$pad(M) = M \parallel 1 \parallel 0 \dots 0 \parallel \text{encodage de la longueur}$$

↳ ! Remplissage même si la longueur du message est un multiple de la taille de bloc, afin de distinguer le message original et le remplissage lui-même

- Fonction de compression et chiffrement par bloc**

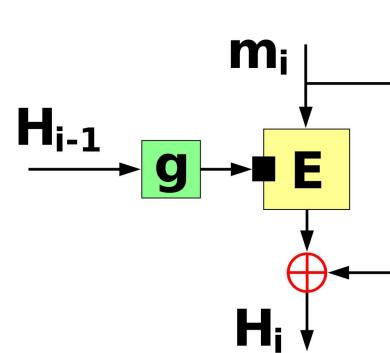
Il est possible de construire une fonction de hachage cryptographique en utilisant un algorithme de chiffrement par bloc **E** (p. ex. AES) comme fonction de compression.

Plusieurs schémas sont possibles :



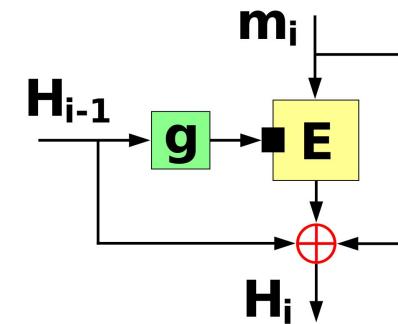
Davies-Meyer

$$H_i = E_{m_i}(H_{i-1}) \oplus H_{i-1}$$



Matyas-Meyer-Oseas

$$H_i = E_{g(H_{i-1})}(m_i) \oplus m_i$$



Miyaguchi-Preneel

$$H_i = E_{g(H_{i-1})}(m_i) \oplus H_{i-1} \oplus m_i$$

L'algorithme de chiffrement **E** peut avoir des contraintes concernant la clé, comme sa taille ou son format → « fonction de conversion » **g**

- **Fonction de compression et chiffrement par bloc**

👍 **Intérêt** : si un chiffrement par bloc est déjà déployé

- On peut prouver la sécurité de la fonction de hachage à partir de l'analyse du chiffrement par bloc
- En réutilisant des éléments du chiffrement par bloc, on gagne en espace et en temps d'implémentation

👎 **Inconvénient** : la fonction de hachage sera plus lente qu'une **fonction dédiée au hachage**, comme MD5, SHA-1, etc.

5.4 Construction de Merkle-Damgård

- **MD2** (*Message-Digest algorithm 2*)

Ron Rivest, 1989

Description dans la **RFC 1319**

Blocs et hachage de 128 bits

S-Box basée sur les décimales de π

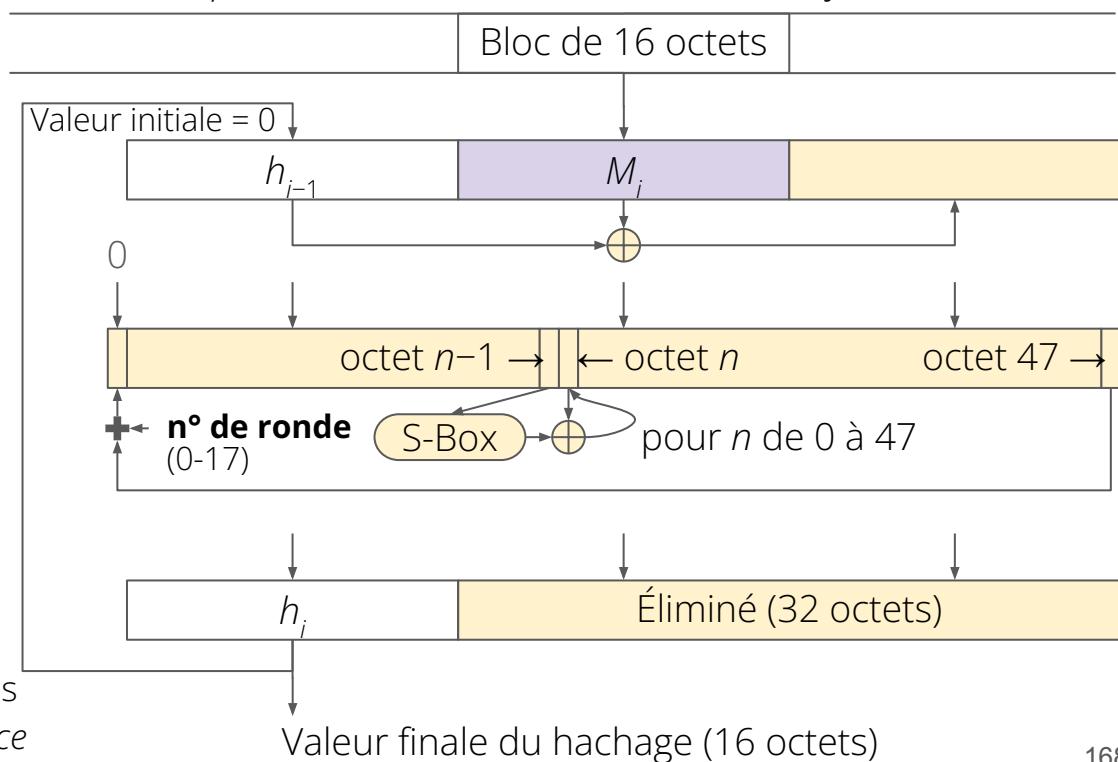
nothing-up-my-sleeve number :

Nombre dont la définition lui évite d'être soupçonné d'avoir des propriétés cachées (cf. *backdoor*)

🚫 **MD2 n'est plus considéré sûr**

Complexité : attaque en 2^{104} opérations contre 2^{128} pour une attaque *brute force*

pad(M) avec checksum de 16 octets ajouté



5.4 Construction de Merkle-Damgård

- **MD4** (*Message-Digest algorithm 4*)

Ron Rivest, 1990

Description dans la **RFC 1320**

Blocs de 512 bits et hachage de 128 bits

Attaque de préimage en $2^{99,7}$ opérations

- **MD5** (*Message-Digest algorithm 5*)

Ron Rivest, 1992

Description dans la **RFC 1321**

Blocs de 512 bits et hachage de 128 bits

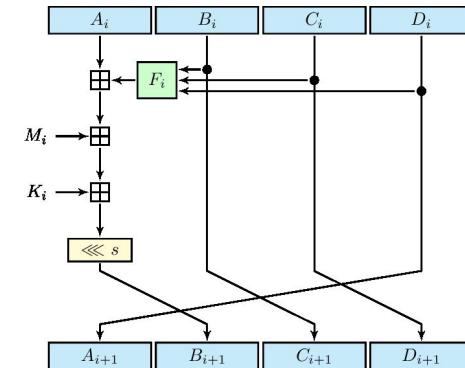
Attaque de collisions en $2^{24,1}$ opérations

🟡 Mais parfois encore utilisé en sécurité...

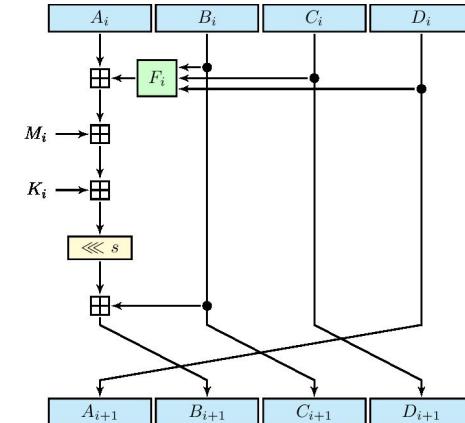
✓ Utilisation toujours valable pour vérifier l'intégrité non-cryptographique

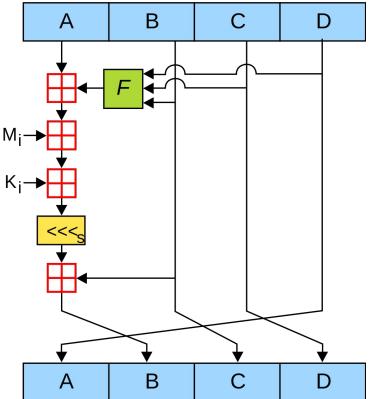
```
$ md5sum fichier.txt > hash.md5
```

MD4



MD5





$i \in [0 ; 63]$, c.-à-d. 64 tours

4 « rondes » différentes F (# fonction de compression)

↳ Chacune utilisée 16 fois : (16×4) tours

Le bloc M de 512 bits entre dans la fonction de compression par sous-blocs (« mots ») M_i , de 32 bits

↳ 16 mots différents, chacun utilisé 4 fois : (4×16) tours

Le nombre de bits s du décalage à gauche ne prend que 16 valeurs différentes, chacune utilisée 4 fois

Les 64 constantes K_i sont toutes différentes

⌘ mod 32 : perte d'information, **sens inverse impossible**

```
import math

def left_rotate(x, n): # Décalage de n bits vers la gauche
    return ((x << n) | (x >> (32 - n))) & 0xFFFFFFFF # Valeur sur 32 bits

def md5(message):
    # 64 constantes : nombre de bits pour la rotation vers la gauche
    s = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]
    # 64 constantes de 32 bits : sinus d'entiers
    K = [math.floor(abs(math.sin(i + 1)) * 2**32) & 0xFFFFFFFF for i in range(64)]

    message = bytearray(message, 'utf-8')
    original_length_bits = (8 * len(message)) & 0xffffffffffff # Valeur sur 64 bits

    # Schéma de remplissage (renforcement de Merkle-Damgård)
    message.append(0x80) # Ajout du "1"
    while len(message) % 64 != 56:
        message.append(0x00) # Ajout des "0"
    message.extend(original_length_bits.to_bytes(8, byteorder='little')) # Ajout de la longueur

    # Initialisation de l'état interne : 128 bits, divisé en 4 mots de 32 bits (A, B, C et D)
    a0 = 0x67452301
    b0 = 0xefcdab89
    c0 = 0x98badcfe
    d0 = 0x10325476
```

Découpage en blocs de 512 bits

```
for chunk_start in range(0, len(message), 64):
    chunk = message[chunk_start:chunk_start + 64]
```

Le bloc de 512 bits est divisé en 16 mots de 32 bits

```
M = [int.from_bytes(chunk[i:i+4], byteorder='little') for i in range(0, 64, 4)]
```

A = a0

B = b0

C = c0

D = d0

Boucle principale : 64 itérations (4 fonctions non-linéaires x 16 itérations)

```
for i in range(64):
```

```
    if 0 <= i <= 15: # Fonction 1 : (B  $\wedge$  C)  $\vee$  ( $\neg$ B  $\wedge$  D)
        F = (B & C) | ((~B) & D)
        g = i
```

```
    elif 16 <= i <= 31: # Fonction 2 : (B  $\wedge$  D)  $\vee$  (C  $\wedge$   $\neg$ D)
        F = (D & B) | ((~D) & C)
        g = (5 * i + 1) % 16
```

```
    elif 32 <= i <= 47: # Fonction 3 : B  $\oplus$  C  $\oplus$  D
        F = B ^ C ^ D
        g = (3 * i + 5) % 16
    else: # Fonction 4 : C  $\oplus$  (B  $\vee$   $\neg$ D)
        F = C ^ (B | (~D))
        g = (7 * i) % 16
```

Additions modulo 32, décalage vers la gauche,

((F \oplus A \oplus M[i] \oplus K[i]) [$<<$ s]) \oplus B, et permutations des mots A, B, C et D

```
F = (F + A + K[i] + M[g]) & 0xFFFFFFFF
```

A = D

D = C

C = B

```
B = (B + left_rotate(F, s[i])) & 0xFFFFFFFF
```

Mise à jour de l'état interne pour le bloc suivant

a0 = (a0 + A) & 0xFFFFFFFF

b0 = (b0 + B) & 0xFFFFFFFF

c0 = (c0 + C) & 0xFFFFFFFF

d0 = (d0 + D) & 0xFFFFFFFF

Dernier état interne : concaténation des 4 mots de 32 bits

```
digest = a0.to_bytes(4, byteorder='little') + \
        b0.to_bytes(4, byteorder='little') + \
        c0.to_bytes(4, byteorder='little') + \
        d0.to_bytes(4, byteorder='little')
```

```
return digest.hex()
```

5.4 Construction de Merkle-Damgård

- **SHA-1** (*Secure Hash Algorithm 1*)

National Security Agency, 1995

Description dans la **RFC 3174**

Blocs de 512 bits et [hachage de 160 bits](#)

80 tours

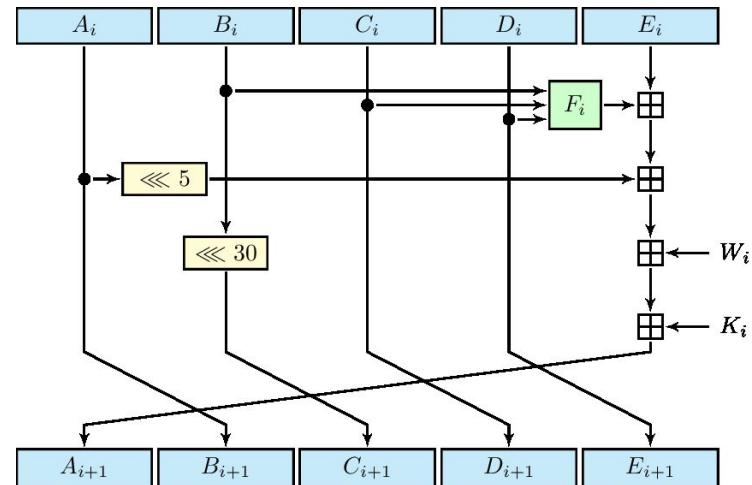
2011 : utilisation obsolète dans les **signatures numériques**

2017 : les navigateurs Web cessent de prendre en charge les certificats signés avec SHA-1

2020 : Microsoft cesse de prendre en charge les signatures SHA-1 pour Windows Update
Le NIST déclare SHA-1 obsolète, avec une utilisation progressivement supprimée d'ici le 31/12/2030

Utilisation encore valable pour :

- les **HMAC**
- vérifier l'intégrité non-cryptographique (« sommes de contrôle »)
`$ sha1sum fichier.txt > hash.sha1`



5.4 Construction de Merkle-Damgård

- **SHA-2** (*Secure Hash Algorithm 2*)

National Security Agency, 2002

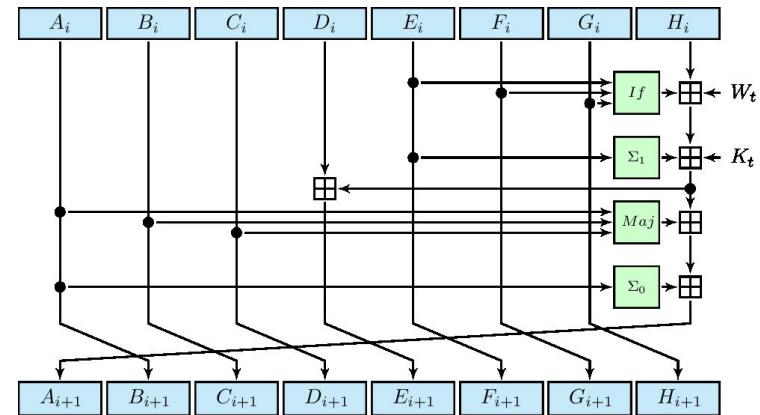
Description dans le **FIPS 180-2**

Federal Information Processing Standard

| | Sortie | État interne | Bloc | Tours |
|-------------|--------|--------------|------|-------|
| SHA-224 | 224 | 256 | 512 | 64 |
| SHA-256 | 256 | (8 × 32) | | |
| SHA-384 | 384 | 512 | 1024 | 80 |
| SHA-512 | 512 | (8 × 64) | | |
| SHA-512/224 | 224 | | | |
| SHA-512/256 | 256 | | | |

Valeurs en bits

💡 Construction de Merkle-Damgård basé sur le chiffrement par bloc **SHACAL-2** configuré selon le mode Davies-Meyer



$$\text{Ch}(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$$

$$\text{Ma}(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$

$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

Constantes pour SHA-256

5.4 Construction de Merkle-Damgård

- **RIPMD** (*RIPE Message Digest*)

Dobbertin, Bosselaers, Preneel (1992)

Taille de bloc : 512 bits

Taille de l'empreinte : 128, 160, 256 ou 320 bits

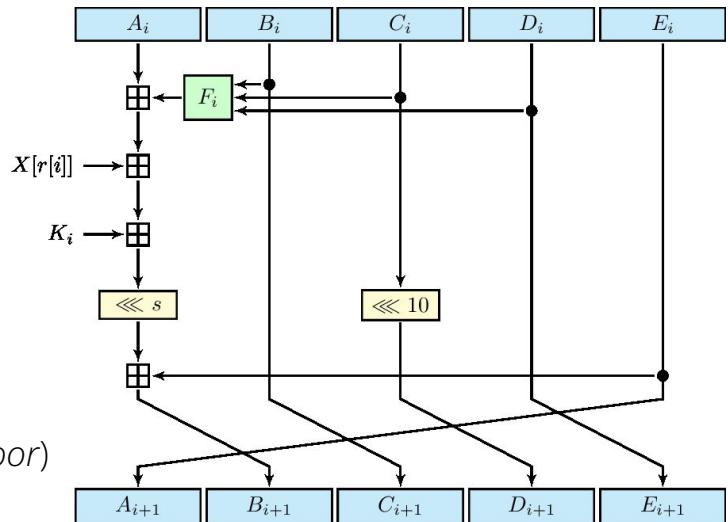
Nombre de rondes : 80

RIPMD-160 (1996)

Seule fonction de hachage 160 bits sans collision trouvée

Issue de la recherche académique (plutôt que militaire, cf. *backdoor*)

Utilisée dans Bitcoin



RIPE : RACE Integrity Primitives Evaluation

RACE : Research and Development in Advanced Communications Technologies in Europe

5.5 Construction de l'éponge

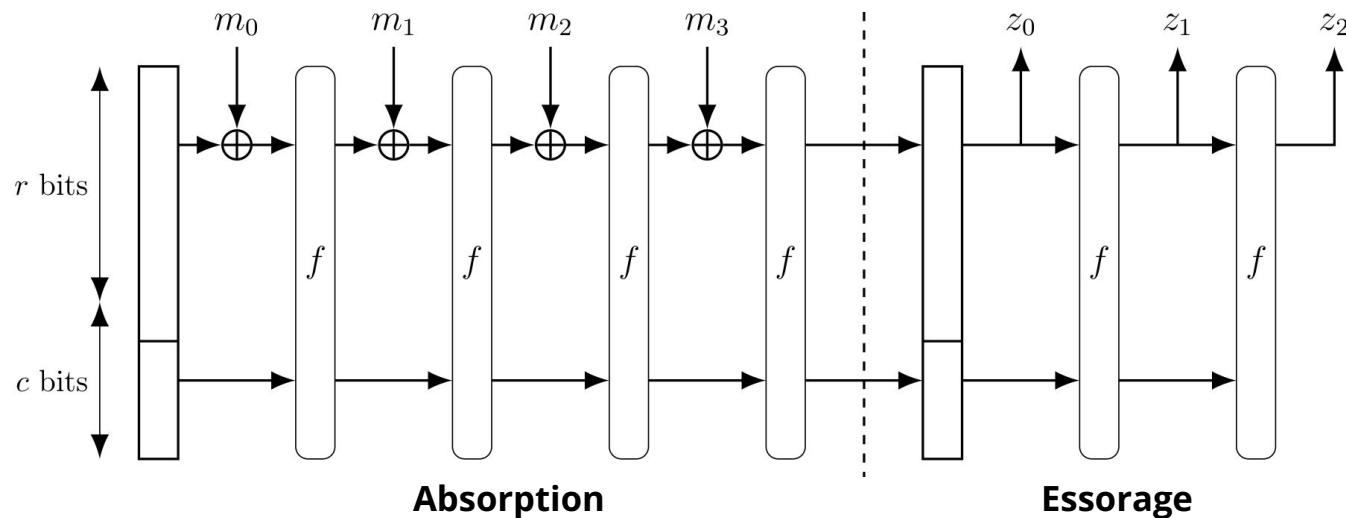
- **SHA-3** (*Secure Hash Algorithm 3*)

Bertoni, Daemen, Peeters, van Assche (2015)

État interne S de 1600 bits, composé de deux paramètres : un débit (*rate*) r et d'une capacité c

Plus r est grand, plus la vitesse des calculs est grande ; plus c est grand, plus la sécurité est grande

Fonction de transformation (« permutation ») f : **algorithme de Keccak**



- **SHA-3** (*Secure Hash Algorithm 3*)

État interne S : tableau 3D de $5 \times 5 \times w$ bits,
 w étant la taille des mots (une puissance de 2, p. ex. 64 bits)

Indices

i : rang

j : colonne

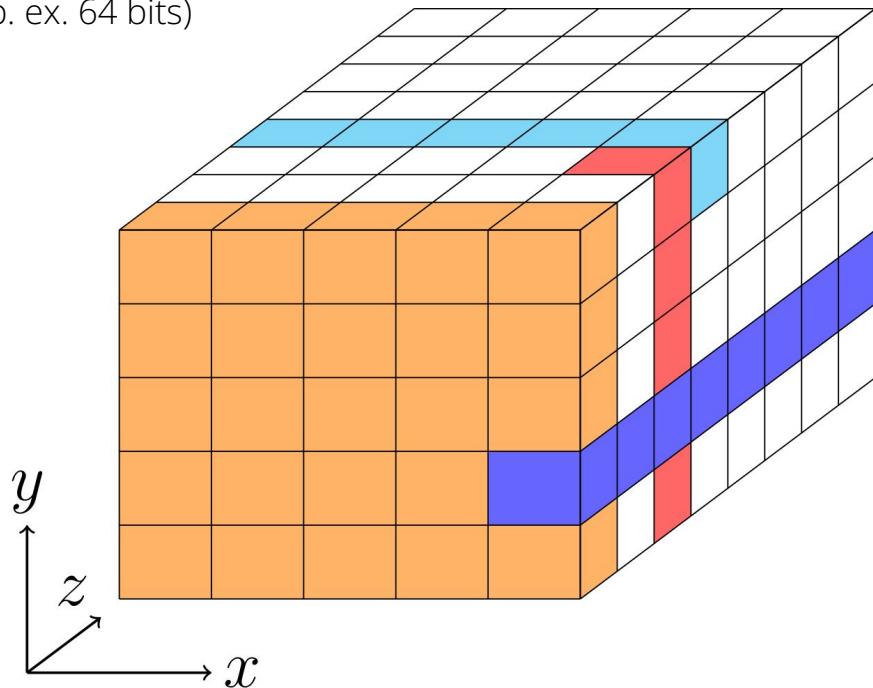
k : ligne

Note : tranche

Arithmétique modulaire

i et j : mod 5

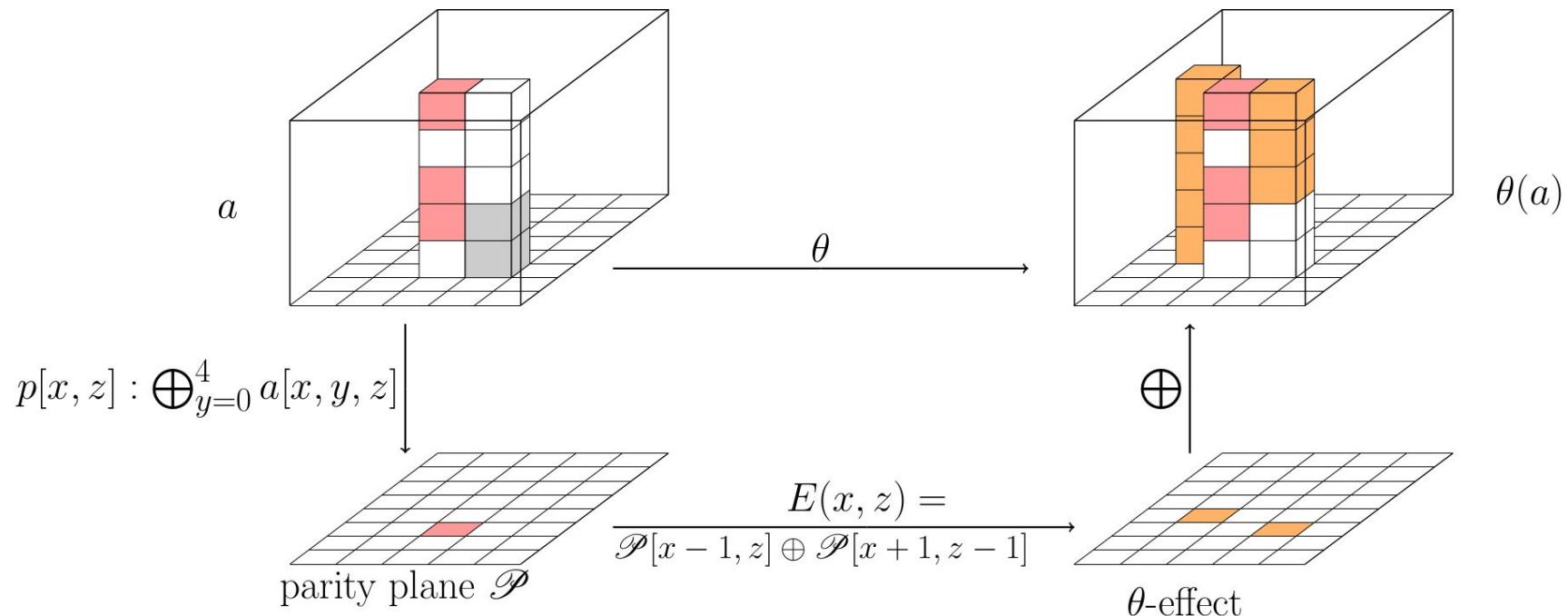
k : mod w



5.5 Construction de l'éponge

- **SHA-3** (*Secure Hash Algorithm 3*)

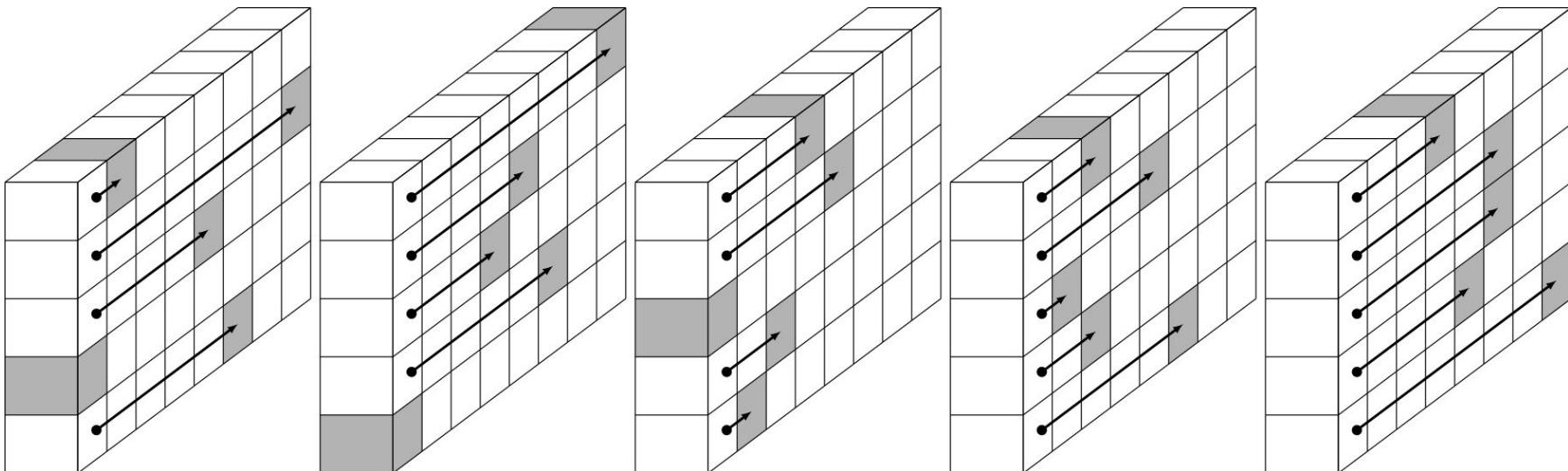
Étape θ (theta) : calcule la parité de chacune des $5 \times w$ colonnes de 5 bits, puis XOR avec les colonnes voisines selon un motif régulier



5.5 Construction de l'éponge

- **SHA-3** (*Secure Hash Algorithm 3*)

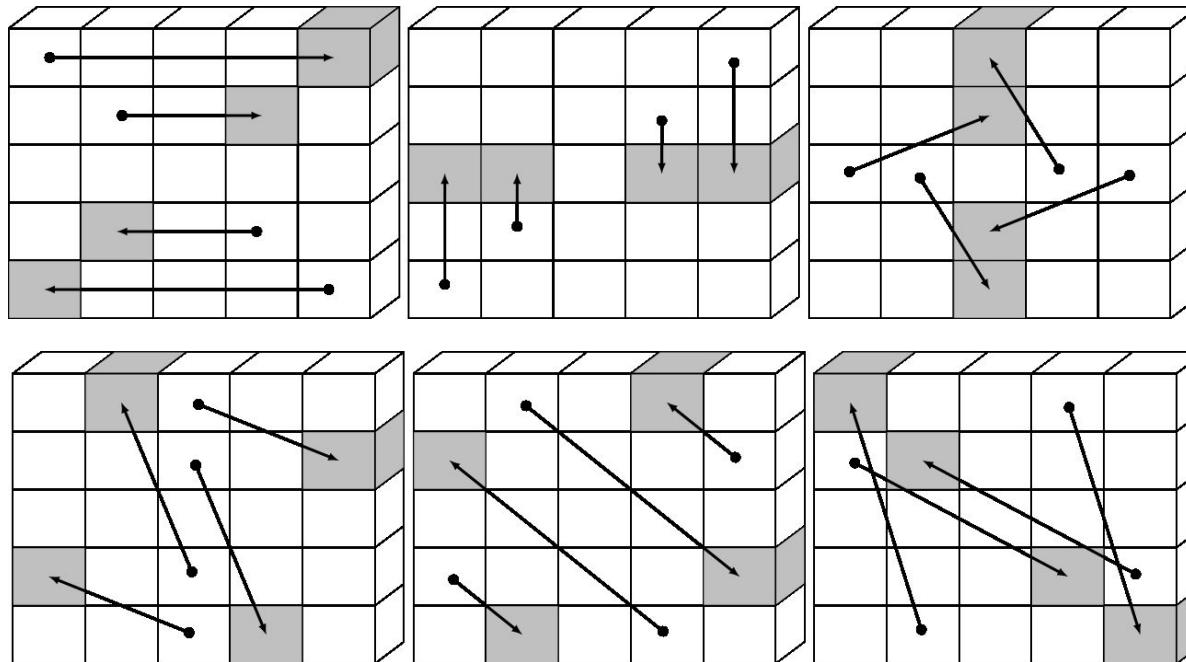
Étape ρ (rho) : rotation des 25 mots, chacun d'un nombre triangulaire différent



5.5 Construction de l'éponge

- **SHA-3** (*Secure Hash Algorithm 3*)

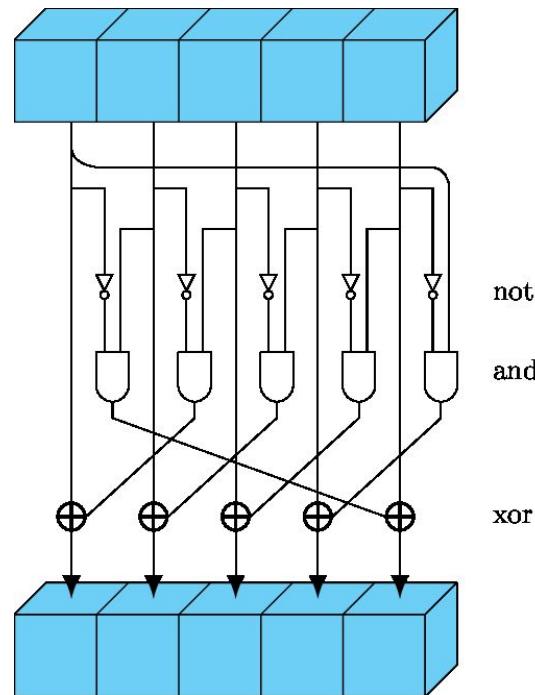
Étape π (pi) : permutation des 25 mots, selon le motif $a[3i+2j][i] \leftarrow a[i][j]$



5.5 Construction de l'éponge

- **SHA-3** (*Secure Hash Algorithm 3*)
Étape χ (chi) : combinaison au sein de chaque rang
 $a[i][j][k] \leftarrow a[i][j][k] \oplus (\neg a[i][j+1][k] \& a[i][j+2][k])$

Note : seule opération non-linéaire de SHA-3

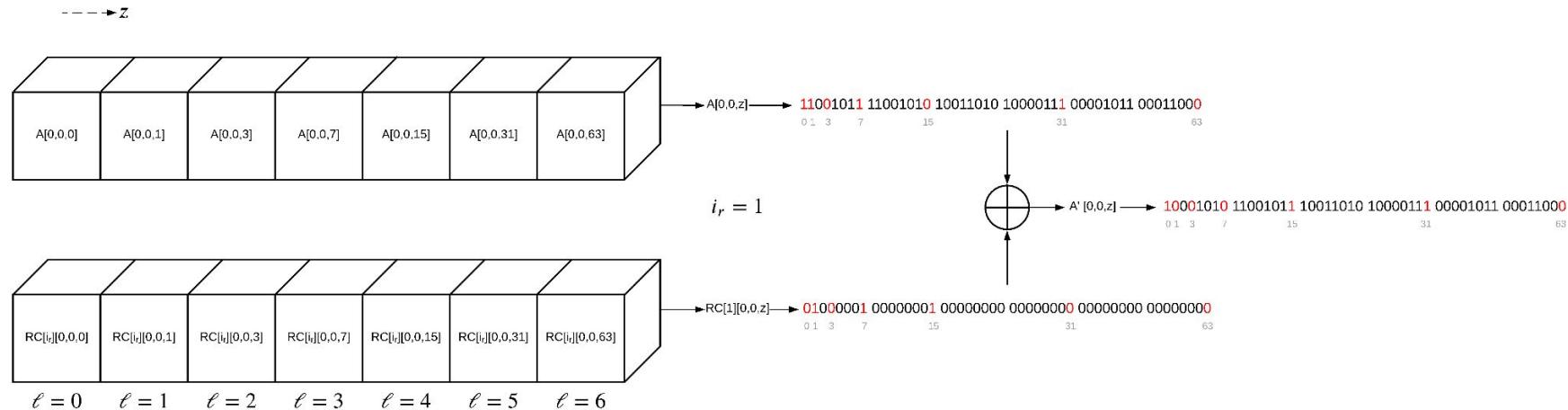


5.5 Construction de l'éponge

- **SHA-3 (Secure Hash Algorithm 3)**

Étape 1 (iota) : XOR d'une constante de ronde (RC) avec un mot de l'état (ligne)

Casse la symétrie préservée par les autres étapes



5.6 Hachage et chiffrement de flux

● ChaCha20

Daniel J. Bernstein, 2008

Chiffrement de flux qui fonctionne comme un hachage

Entrée : [clé de 128 bits ou 256 bits](#) en entrée ; [sortie de 512 bits](#)

↳ Tourne en mode CTR pour obtenir une clé de flux

Chiffrement ARX : opérations ADD, <<< (rotation), XOR (**4 cellules ARX**)

- ↳ Léger : ARX prennent toujours le même temps, quelle que soit la clef
- ↳ Implémentation software presque aussi rapide que AES sur hardware
- ↳  ChaCha20 est **recommandé dans TLS 1.3** comme alternative à AES et défini comme norme de chiffrement de flux par Google (RFC 7539)

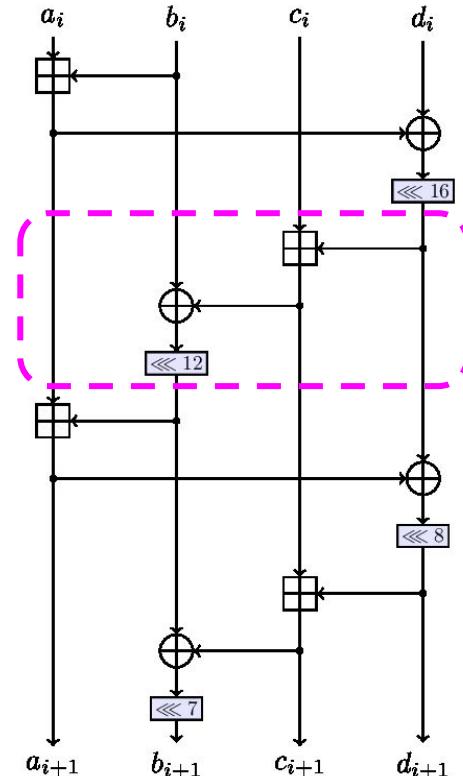
| C | C | C | C |
|-----|-----|-----|-----|
| k | k | k | k |
| k | k | k | k |
| # | # | N | N |

État interne : $4 \times 4 \times 32$ bits

Initialisation : clé k , constante C , **compteur** # et **nonce** N (cf. mode CTR)

C est définie par "expa", "nd 3", "2-by", "te k".

Note : parfois les tailles de C et N sont 32 et 96 bits.



« Quart de ronde (QR) »

5.6 Hachage et chiffrement de flux

● ChaCha20

Quarts de ronde « impaire » :

QR(0, 4, 8, 12)

QR(1, 5, 9, 13)

QR(2, 6, 10, 14)

QR(3, 7, 11, 15)

a b c d

Quarts ronde « paire » :

QR(0, 5, 10, 15)

QR(1, 6, 11, 12)

QR(2, 7, 8, 13)

QR(3, 4, 9, 14)

2 rondes successives
× 10 (d'où « Chacha20 »)

| | | | |
|------------------|------------------|------------------|------------------|
| θ _{1a} | 1 _{2a} | 2 _{3a} | 3 _{4a} |
| 4 _{1b} | 5 _{2b} | 6 _{3b} | 7 _{4b} |
| 8 _{1c} | 9 _{2c} | 10 _{3c} | 11 _{4c} |
| 12 _{1d} | 13 _{2d} | 14 _{3d} | 15 _{4d} |

| | | | |
|------------------|------------------|------------------|------------------|
| θ _{5a} | 1 _{6a} | 2 _{7a} | 3 _{8a} |
| 4 _{8b} | 5 _{5b} | 6 _{6b} | 7 _{7b} |
| 8 _{7c} | 9 _{8c} | 10 _{5c} | 11 _{6c} |
| 12 _{6d} | 13 _{7d} | 14 _{8d} | 15 _{5d} |

Permet une meilleure
diffusion que Salsa20
(chiffrement prédecesseur,
par le même auteur)

6. Authentification

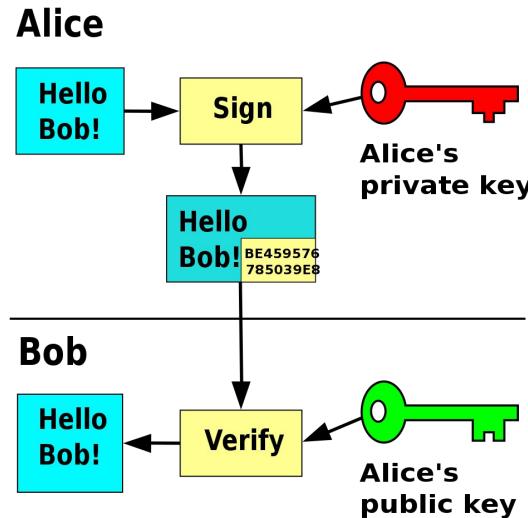
- **Principe général**

La signature numérique suit un modèle de **cryptographie asymétrique**.

Elle implique donc une paire de clés, publique et privée.

La signature du message à authentifier se fait par **chiffrement avec la clé privée** de l'expéditeur.

↳ Le message n'est déchiffrable qu'avec sa clé publique → authentification de l'expéditeur



Ci-contre : envoi d'un message authentifié, mais sans confidentialité

- Signature numérique et chiffrement**

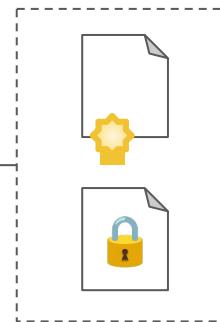
Calcul de l'empreinte $H(m)$ → Intégrité

Signature par chiffrement avec la clé privée → Authentification

} Certificat



1. Chiffrement : $E(b_{\text{pub}}, m) = c_1$
2. Hachage : $H(m)$
3. Certificat : $E(a_{\text{priv}}, H(m)) = c_2$
4. Données signées : $c_1 \parallel c_2$



4. Déchiffrement : $D(b_{\text{priv}}, c_1) = m$
5. Déchiffrement : $D(a_{\text{pub}}, c_2) = H(m)$
6. Hachage et comparaison : $H(m) \text{ vs } H(m)$

Le cryptosystème (E, D) utilisé peut être n'importe quel chiffrement asymétrique, comme RSA ou ElGamal. **CEPENDANT...**

- **Confidentialité persistante (*perfect forward secrecy*)**

Propriété qui garantit que la découverte par un adversaire de la clé privée d'un correspondant (secret à long terme) ne compromet pas la **confidentialité des communications passées**.

Une solution est d'utiliser une **clé de session** pour le chiffrement.

- ↳ Clé **symétrique**, à usage unique, utilisée pour tous les messages dans une session de communication
- ↳ Générée au moyen d'un protocole d'échange de clé « éphémère »

Pour cet échange de clé, RSA pourrait être utilisé (transmission d'une graine pour PRNG, chiffrée par RSA)

- ↳ Mais est déjà utilisé pour les signatures numériques
- ↳ Utiliser un algorithme différent pour l'échange de clés : **DHE**, *Diffie-Hellman Ephemeral*

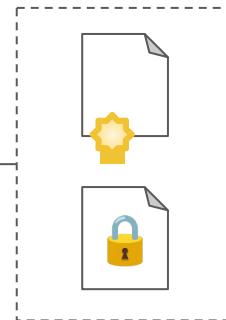
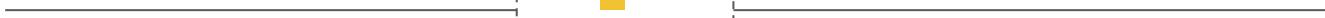
P. ex., dans **TLS**, ce type de combinaison offre un bon compromis entre vitesse et sécurité :
ECDHE-RSA-AES256-GCM-SHA384

- **Signature numérique et chiffrement**

Calcul de l'empreinte $H(m)$ → Intégrité

Signature par chiffrement avec la clé privée → Authentification

} Certificat



1. Chiffrement : $E(k, m) = c_1$

2. Hachage : $H(m)$

3. Certificat : $E(a_{priv}, H(m)) = c_2$

4. Données signées : $c_1 \parallel c_2$

4. Déchiffrement : $D(k, c_1) = m$

5. Déchiffrement : $D(a_{pub}, c_2) = H(m)$

6. Hachage et comparaison : $H(m) \text{ vs } H(m)$

Le cryptosystème (E, D) utilisé peut être n'importe quel chiffrement asymétrique, comme RSA ou ElGamal. **CEPENDANT...**

- **Certificat de clé publique**

But : permettre à **Alice** de s'assurer que la clé publique qu'elle utilise pour

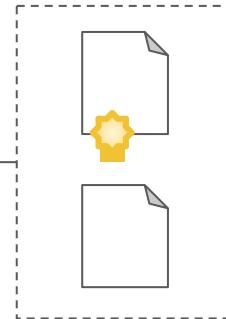
- le chiffrement des messages qu'elle envoie à Bob et
- la vérification de ses signatures

proviennent bien de Bob lui-même et non d'un « homme du milieu » (*man in the middle*, MITM).

Bob envoie ses informations et sa clé publique à une **autorité de certification (CA)**, un tiers de confiance qui vérifie l'identité de Bob (personne physique, société, etc.). Elle crée alors un certificat signé avec sa clé privée (c_{priv}).

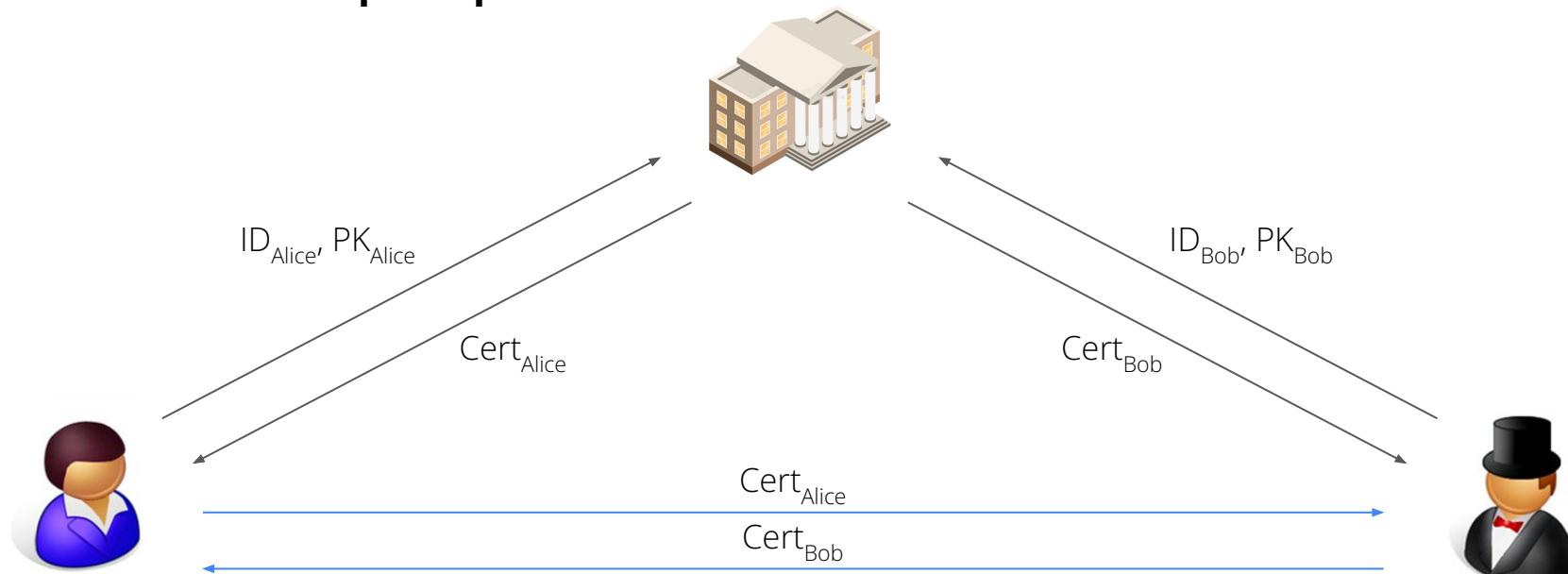


1. ID Bob, ID CA et b_{pub} : m
2. Hachage : $H(m)$
3. Certificat : $E(c_{priv}, H(m)) = c_1$
4. Données signées : $m \parallel c_1$



4. Déchiffrement : $D(c_{pub}, c_1) = H(m)$
5. Hachage et comparaison : $H(m)$ **vs** $H(m)$

- Certificat de clé publique


$$Cert_{Alice} = \langle ID_{Alice}, SN, Expiration, PK_{Alice}, \text{Sign}_{CA}(ID_{Alice}, SN, Expiration, PK_{Alice}) \rangle$$

SN : numéro de série du certificat, pour validation/renouvellement/révocation

- **Certificat de clé publique**

Les clés publiques des autorités de certification sont largement distribuées : pré-installées dans les systèmes d'exploitation et navigateurs web.

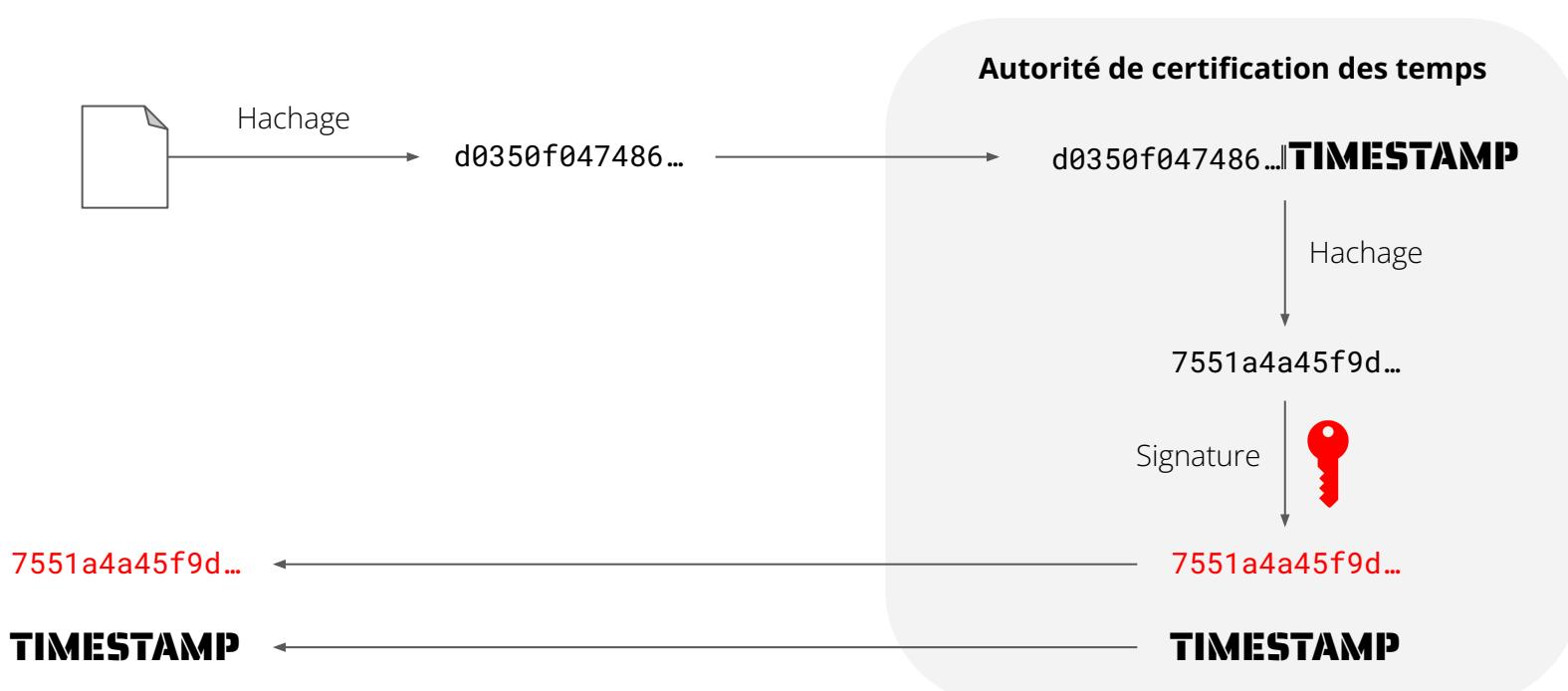


Les sites web sécurisés avec le protocole HTTPS (*HyperText Transfer Protocol Secure*) présentent leur certificat au navigateur au moment du *handshake*.

Deux normes pour les certificats : X.509 et OpenPGP

- **Horodatage certifié**

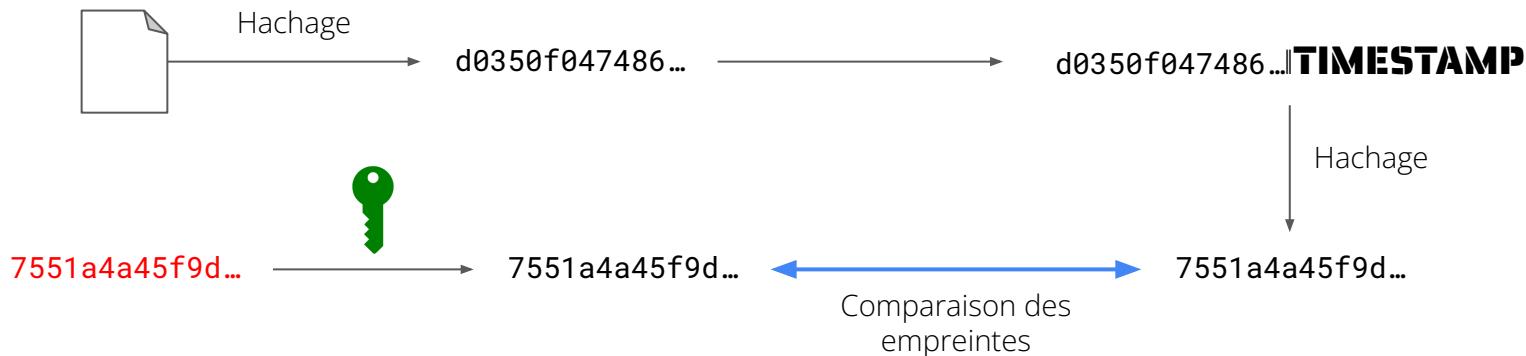
Système permettant de garder la preuve de l'existence d'un document et son contenu à une date donnée.



- **Horodatage certifié**

Après réception des données, du *timestamp* et du **hachage signé**

↳ **Vérification** : calcul de l'empreinte et utilisation de la **clé publique** du **tiers de confiance**



- **DSA (Digital Signature Algorithm)**

Proposé par le NIST en 1991

Faisait partie de la spécification DSS (*Digital Signature Standard*) adoptée en 1993 avant d'être retiré en 2023

Variant du schéma de signature ElGamal

Basé sur le problème du **logarithme discret**

↳ Extension **ECDSA** implémentée avec des courbes elliptiques

↳ Extension **EdDSA** : *Edwards-curve Digital Signature Algorithm*

Ed25519 : signature avec SHA-512 la courbe elliptique 25519

- **DSA**

Variables publiques : deux modules premiers p et q , ainsi qu'un générateur g

Clé privée d'Alice : Alice choisit aléatoirement (PRNG) $x \in [1, q-1]$

Clé publique d'Alice : Alice calcule $y = g^x \bmod p$

Signature du message M : Alice choisit aléatoirement un $nonce k \in [2, q-1]$, puis calcule deux valeurs :

1. $r = (g^k \bmod p) \bmod q$
2. $s = k^{-1}(H(M) + xr) \bmod q$

Utilisation de la **clé privée d'Alice** et calcul du **hachage** (p. ex. SHA-256)

Envoi du certificat, attaché au message chiffré : $(r, s) \| C$, avec $C = E_k(M)$

Vérification de la signature (r, s) : à partir de $M = D_k(C)$ et de (p, q, g, y) , Bob calcule quatre valeurs :

1. $w = s^{-1} \bmod q$;
2. $u_1 = H(M) \times w \bmod q$ Calcul du **hachage**
3. $u_2 = rw \bmod q$;
4. $v = (g^{u1} \times y^{u2} \bmod p) \bmod q$ Utilisation de la **clé publique d'Alice**
5. **Test** : $v = r$? Si vrai → signature validée

Fonctionne car : $v = g^{hw} \times y^{rw} = g^{hw} \times g^{xrw} = g^{hw + xrw} = g^{h/s + xr/s} = g^{(h + xr)/(k-1)(h + xr))} = g^k = r$,

avec $h = H(M)$

6.1 Signature numérique

- **En résumé**

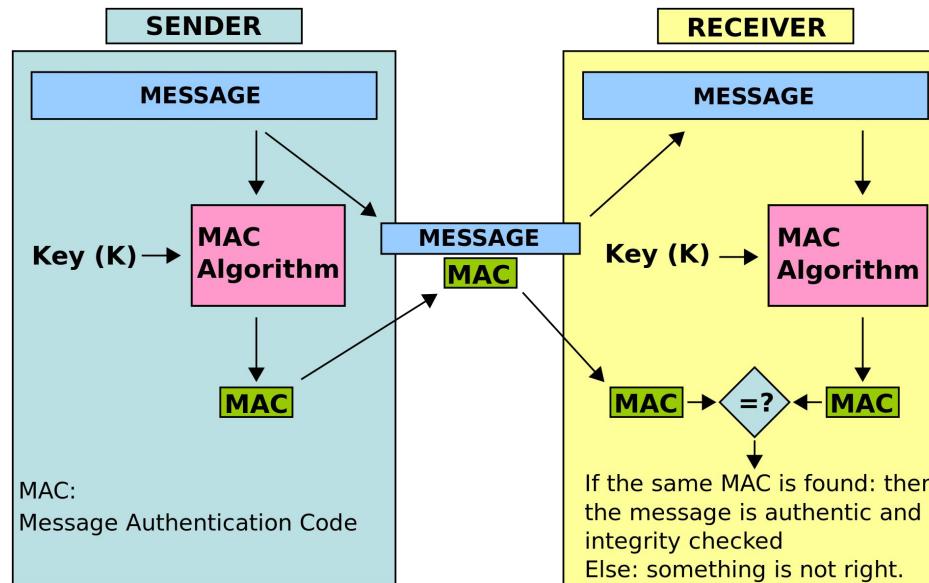
| | Chiffrement | Echange de clé | Signature numérique |
|-----|-------------|----------------|---------------------|
| RSA | ✓ | ✓ | ✓ |
| DH | | ✓ | |
| DSA | | | ✓ |

- **Principe général**

Le MAC (*message authentication code*) suit un modèle de **cryptographie symétrique**.

Il implique donc une clé secrète **K** partagée entre l'expéditeur et le destinataire.

⚠ Mais un MAC n'est pas un chiffrement.



- **HMAC (code d'authentification de message basé sur le hachage)**

Calcul basé sur une fonction de hachage cryptographique \mathbf{H} :

$$\text{HMAC}(K, m) = \mathbf{H} \left((K' \oplus opad) \parallel \mathbf{H} \left((K' \oplus ipad) \parallel m \right) \right)$$

$$K' = \begin{cases} \mathbf{H}(K) & \text{si } K \text{ est plus grande que la taille de bloc} \\ K & \text{autrement} \end{cases}$$

avec

K' , une clé de la taille du bloc, dérivée de la clé secrète \mathbf{K} par hachage et/ou remplissage avec des 0 ;

opad (*outer padding*), une constante faite par répétition de la valeur hexadécimale **0x5c**

ipad (*inner padding*), une constante faite par répétition de la valeur hexadécimale **0x36**

L'utilisation de ces constantes permet un mélange de la clé secrète avec les parties externes et internes du message.

La sécurité d'un HMAC dépend de la fonction \mathbf{H} , mais également de la clé secrète \mathbf{K} .

C'est pourquoi les HMACs sont plus résistants aux collisions que leurs fonctions \mathbf{H} .

Ainsi, on utilise encore SHA-1 dans des HMACs sécurisés.

- **CBC-MAC (*cipher block chaining message authentication code*)**

Pour calculer le CBC-MAC d'un message m , on calcule le chiffrement de m en mode CBC, avec un **IV** constitués de zéros, et on garde le dernier bloc chiffré.

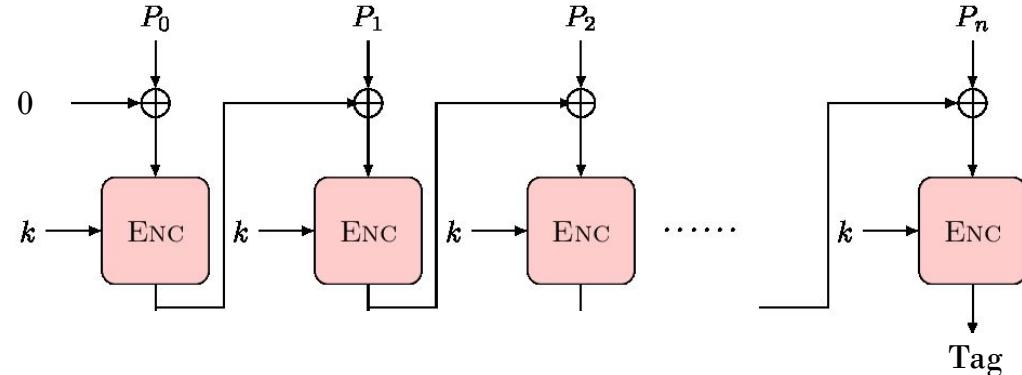
Si le chiffrement par bloc utilisé est sécurisé, alors le CBC-MAC l'est aussi pour les messages de tailles fixes.

Mais pour les messages de tailles variables, si un adversaire connaît les CBC-MAC de deux messages (m, t) et (m', t') , alors il peut forger un troisième message m'' , dont le CBC-MAC sera également t' :

$$m'' = m \parallel [(m'_1 \oplus t) \parallel m'_2 \parallel \dots \parallel m'_n]$$

Solutions possibles

1. Ajouter la longueur du message dans le premier bloc (mais nécessite de la connaître dès le début)
2. Chiffrer le dernier bloc (*encrypt last bloc*) : $\text{CBC-MAC-ELB}(m, (k_1, k_2)) = E(k_2, \text{CBC-MAC}(k_1, m))$

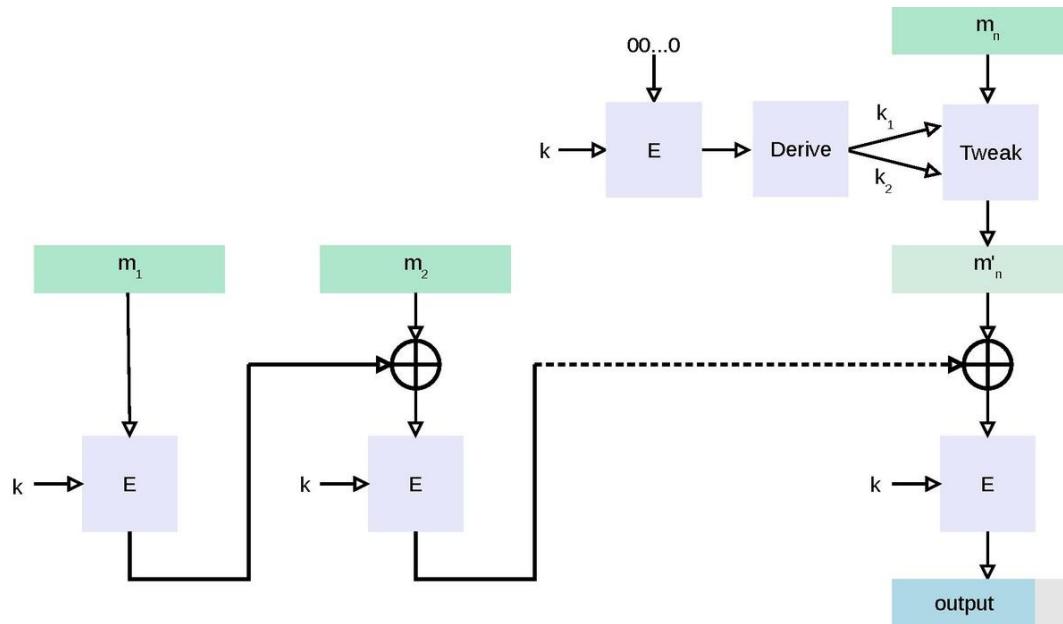


6.2 Code d'authentification de message

- **OMAC (one-key MAC)**

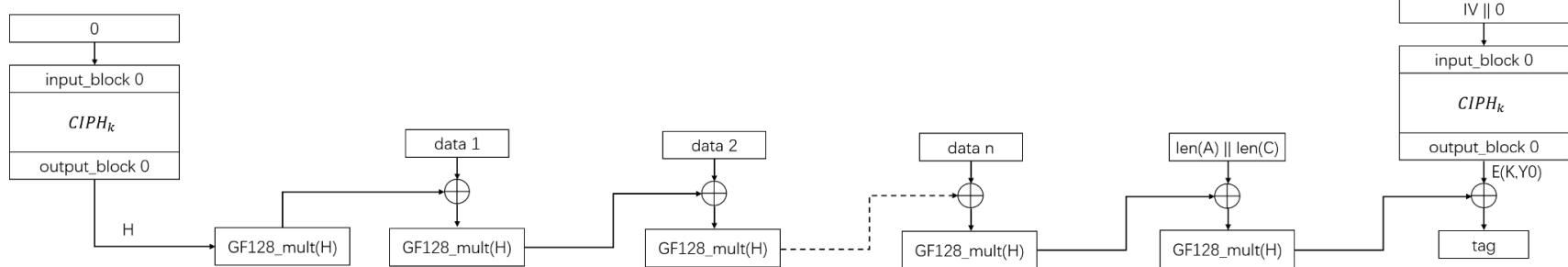
Aussi appelé CMAC

Résout les défauts de sécurité de CBC-MAC, mais nécessite 3 clés



6.2 Code d'authentification de message

- **GMAC, Galois Message Authentication Code**



4 entrées authentifiées :

- un vecteur d'initialisation \mathbf{IV} ,
- données utilisateur chiffrées, \mathbf{C} ,
- leur longueur $\mathbf{len(C)}$ et
- la longueur des données associées non-chiffrées $\mathbf{len(A)}$; ci-dessus = $\mathbf{len(H)}$

H : produit par le chiffrement d'un bloc de zéros

$\mathbf{GF128_mult(H)}$: multiplication par H , effectuée dans le **corps fini** $F_2[X] / \langle X^{128} + X^7 + X^2 + X + 1 \rangle$
↳ Nécessite donc de convertir le bloc et H , chacun en un polynôme sur 128 bits

- **Non-répudiation**

Situation dans laquelle la signature d'un contrat ne peut être remise en question

L'utilisation d'un **MAC n'assure pas la non-répudiation**, car le destinataire partage la clé secrète avec l'expéditeur. Ce dernier peut donc légitimement nier avoir signé les données, arguant du fait qu'il n'est pas le seul à pouvoir générer le MAC.



Seules les signatures numériques permettent la non-répudiation.

L'implication d'une autorité de certification évite qu'un expéditeur puisse répudier sa propre signature.

6.3 Chiffrement authentifié

● Galois/Counter Mode (GCM)

Mode de fonctionnement CTR, avec authentification

AEAD : *Authenticated encryption with associated data*

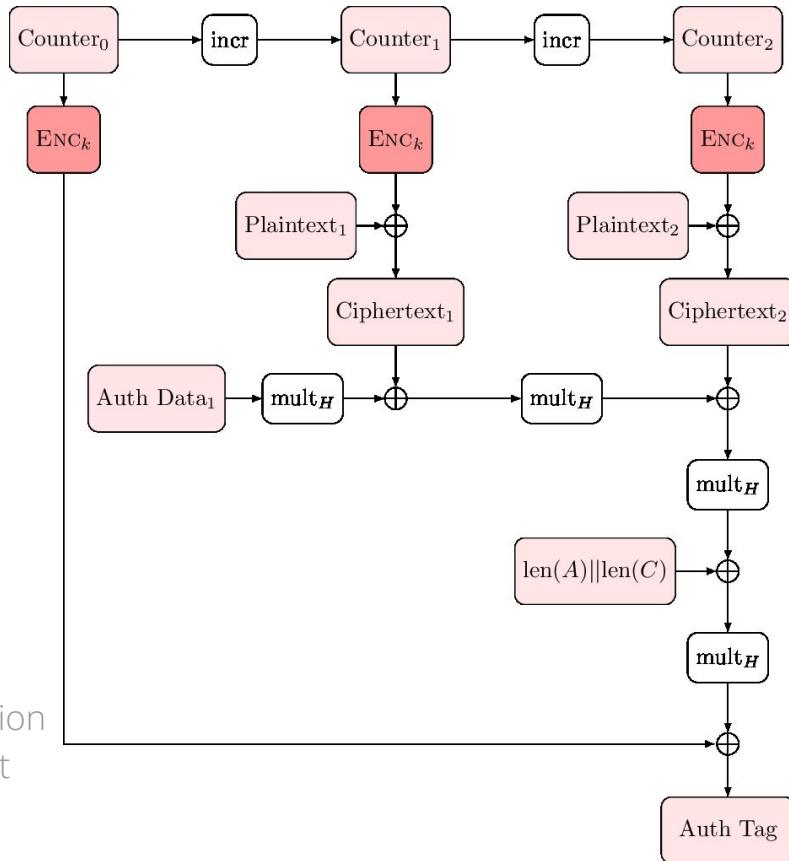
Permet simultanément :

- Authentification par **GMAC**
- **Chiffrement** par bloc (typiquement, AES, 128 bits)
- Ajout de **données non-chiffrées mais authentifiées** (**Auth Data₁**, ci-contre)

Utilité : dans un paquet réseau, l'en-tête et la charge utile (données utilisateur) ont toutes les deux besoin d'intégrité et d'authentification. Mais seule la *payload* doit être chiffrée, le *header* doit être visible pour le routage.

Note : le caractère simultané des chiffrement et authentification fait que $\text{len}(A) \parallel \text{len}(C)$ doit être fournie avant le (dé)chiffrement

Auth Tag : 128 bits (parfois tronqués à 96 bits)

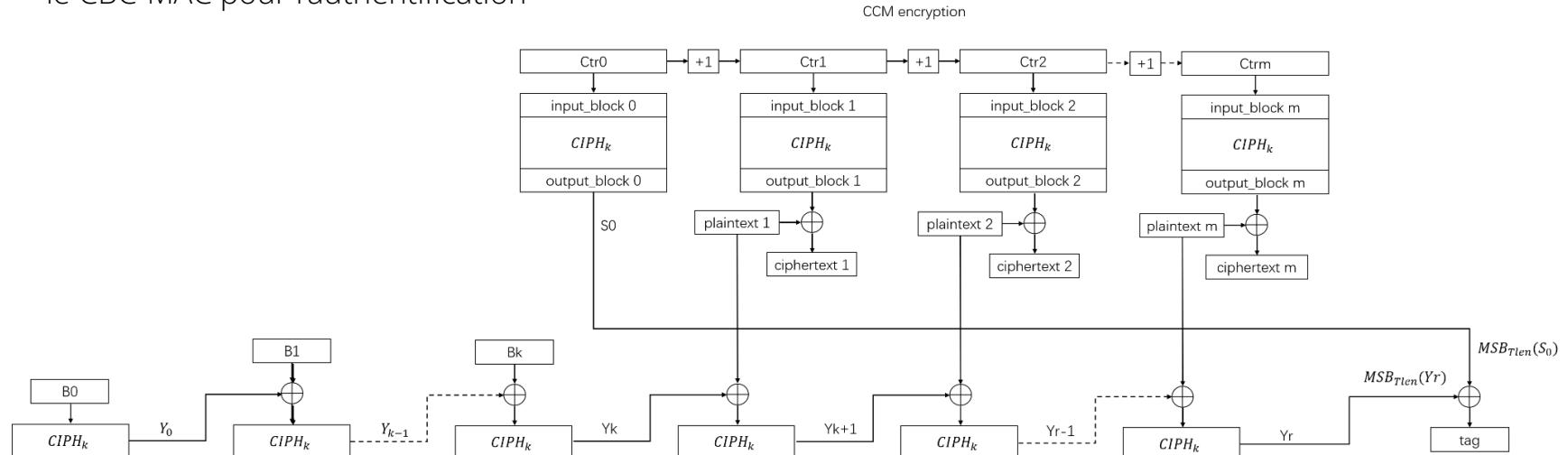


- **CCM (counter with CBC-MAC)**

Alternative au GCM largement répandue (p. ex. AES-CCM dans WPA2).

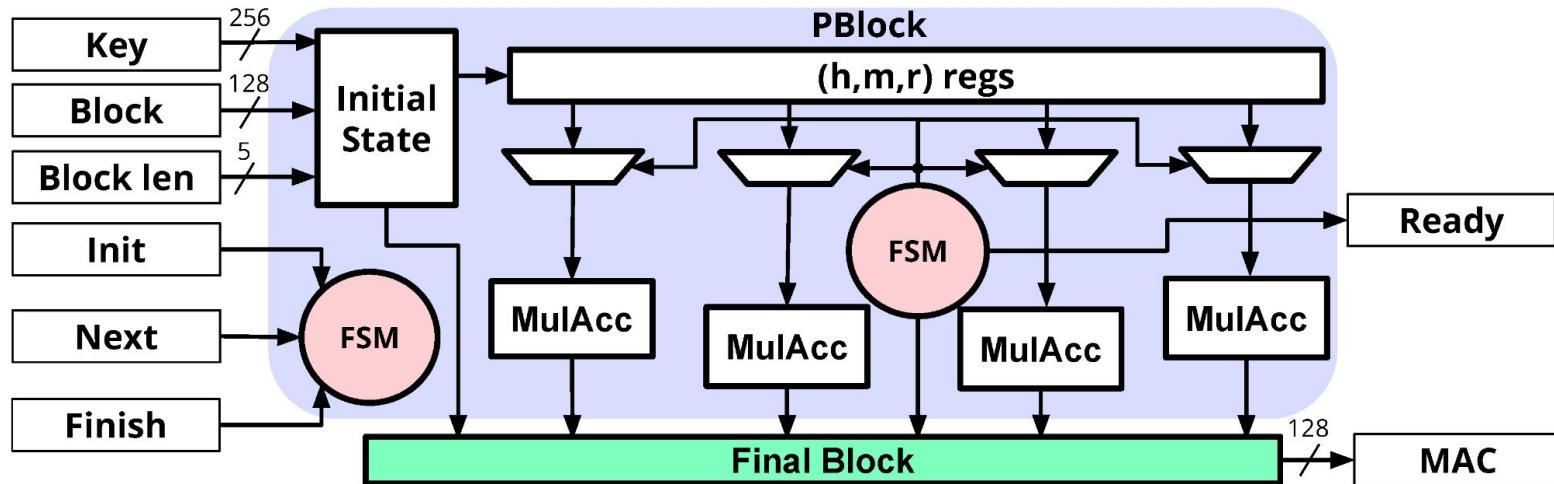
Comme son nom l'indique, CCM combine :

- le mode CTR pour le chiffrement et
- le CBC-MAC pour l'authentification



- **Poly1305** (Daniel J. Bernstein, 2005)

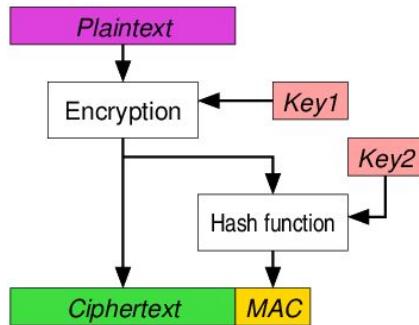
Hachage universel : sélection aléatoire d'une fonction de hachage dans une famille de fonctions de hachages ayant certaines propriétés mathématiques → Permet de minimiser la probabilité de collision



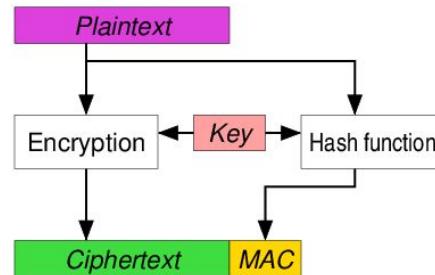
ChaCha20-Poly1305 : AEAD alternatif à AES-GCM dans TLS

Rappel : ChaCha20 est un chiffrement de flux

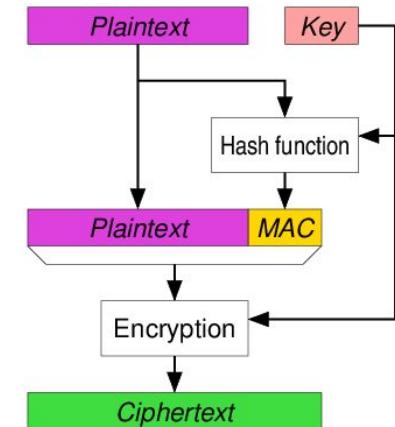
- Differentes approches**



Encrypt-then-MAC (EtM)
 GCM
 ChaCha20-Poly1305
 EAX



Encrypt-and-MAC (E&M)
 CMAC



MAC-then-Encrypt (MtE)
 CCM
 GCM-SIV

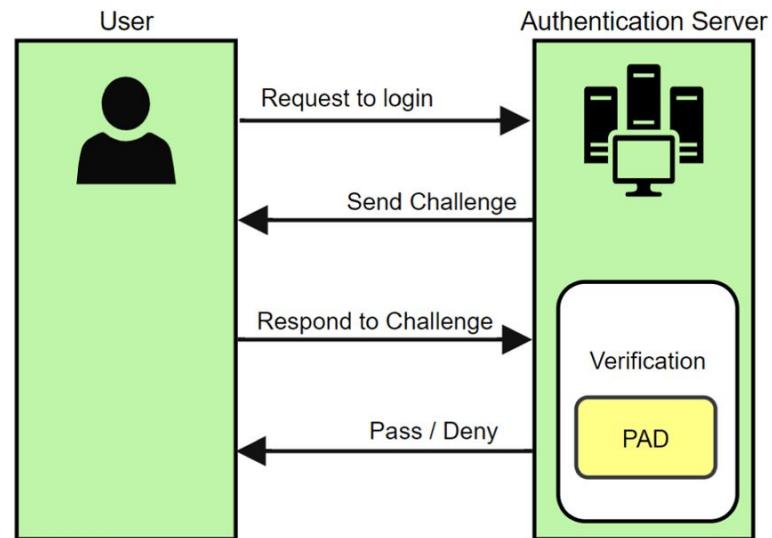
- **Principe général**

Famille de protocoles dans lesquels une partie présente une question (le défi) et une autre partie doit fournir une réponse valide (la réponse) pour être authentifiée.

Authentification par mot de passe :

- le défi est de demander le mot de passe,
- la réponse valide est le mot de passe correct.

La vérification du mot de passe se fait par comparaison avec une valeur stockée sur le serveur.



- **Stockage des mots de passe**

Il existe plusieurs façons de stocker les mots de passe sur le serveur.

Les voici présentées par ordre du niveau de sécurité qu'elles assurent.

 Les méthodes d'**authentification sans mot de passe** offrent une sécurité encore plus élevée.

1. **Stockage en clair** : aucune sécurité en cas d'intrusion sur le serveur et vol des données
2. **Chiffrement** : mieux, mais aucune sécurité en cas de vol de la clé secrète ; aucune protection contre les menaces internes
3. **Hachage** : le système vérifie que le **hachage du mdp** entré correspond au hachage stocké dans la base.
Si un intrus s'empare d'une valeur de hachage, il ne pourra pas retrouver le mdp correspondant (résistance à la préimage), ni générer un mdp ayant la même valeur de hachage (résistance à la seconde pré-image). Mais problème de sécurité
 - si la fonction de hachage n'est pas résistante ;
 - si le mdp est faible : vulnérabilité aux attaques par dictionnaire et **tables arc-en-ciel**

- **Stockage des mots de passe**

4. **Salage** : lors de la création d'un compte utilisateur, une courte valeur aléatoire est générée : le sel (*salt*)
 - ↳ Elle est concaténée au mdp avant hachage et, également, stockée dans la base
 - Rainbow icon: Protège contre les tables arc-en-ciel ; pour deux utilisateurs avec le même mdp, les hachages stockés seront différents
 - Red warning icon: Ne protège pas contre les attaques par dictionnaires, car le sel est stocké dans la base
5. **Fonction de hachage dédiée** : fonction conçue pour être « lente », coûteuse en calculs, et utilisée spécifiquement pour le hachage des mots de passe ; le salage est inclus

Exemples :

bcrypt (Provos et Mazières, 1999), dérivée du chiffrement par bloc Blowfish

scrypt (Percival, 2009)

Argon2 (Biryukov, Dinu et Khovalovich, 2015)

Vitesse d'une attaque : de l'ordre du millier de tentatives par seconde

⚠ Ne protège que les mdp forts

- **Jeton d'accès (access token)**

Après avoir authentifié l'utilisateur, le serveur génère un jeton unique, grâce à une **clé privée**

- ↳ Le **client reçoit ce jeton** et l'utilisera pour ses futures requêtes d'authentification
- ↳ Le serveur n'aura qu'à **valider la signature**

Intérêts

 Le serveur ne stocke pas les jetons, contrairement aux données d'une **session**

Note : l'identifiant de session est stocké côté client dans un *cookie*, qui peut être modifié par le client et n'est donc pas fiable

 Une définition des droits d'accès aux fonctionnalités par l'utilisateur est associée au jeton

 Pour éviter les attaques par force brute, le jeton expirera après une certaine date

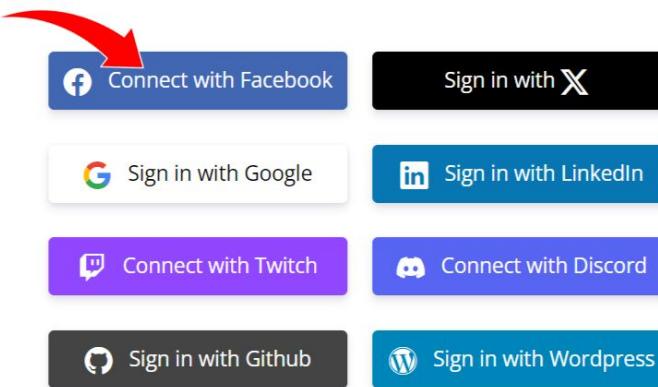
Risque

 Un jeton est similaire à un mot de passe qui serait temporaire et fourni par le serveur

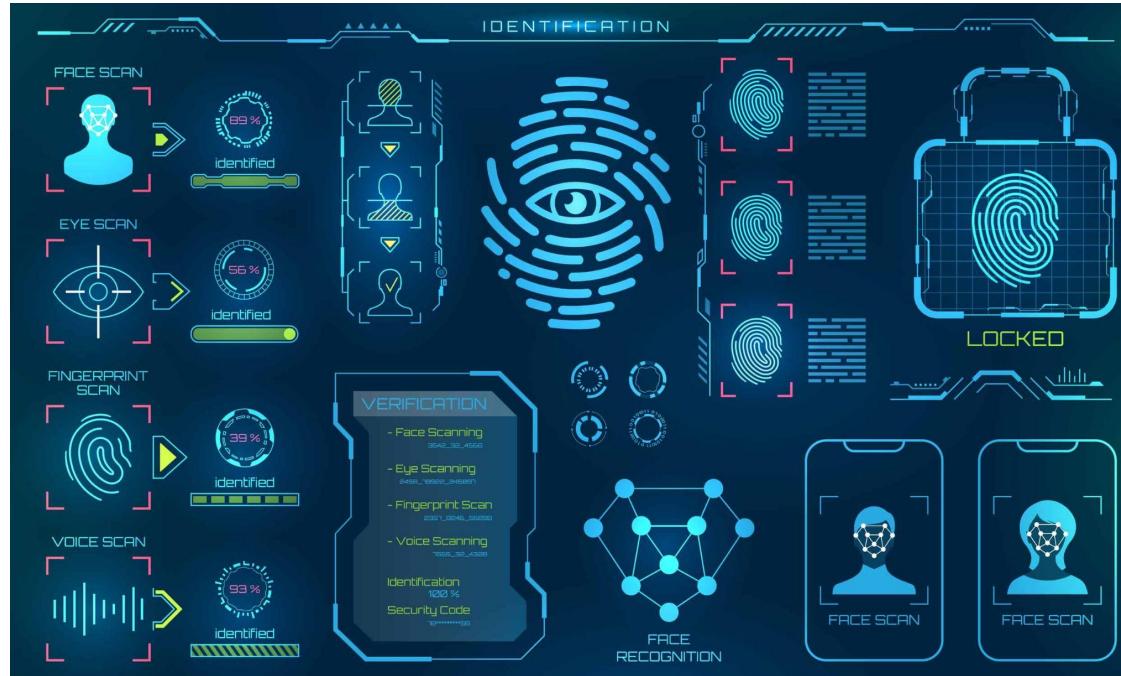
- ↳ Il peut donc être volé à un utilisateur imprudent

- **OAuth (*Open Authorization*)**

Protocole de délégation d'autorisation (basé sur les jetons)



- **Authentification biométrique**



- **Définition**

Si le canal de communication n'est pas sûr (risque d'interception), une solution implique que les parties communicantes doivent se convaincre l'une et l'autre qu'elles connaissent le secret partagé (mot de passe) sans que ce secret ne soit transmis en clair.

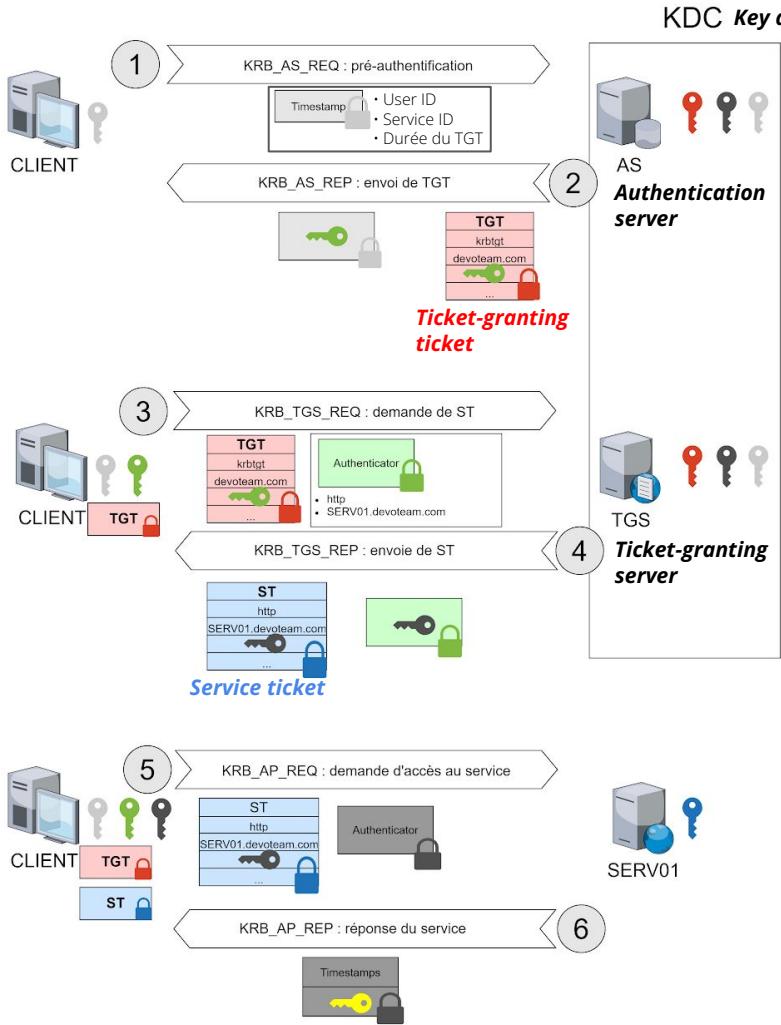
Kerberos (MIT, 1988)

Protocole dans lequel la permission de communiquer entre deux machines est donnée par un serveur tiers de confiance ([authentification centralisée](#)), qui génère des clés de session temporaires.

Pour communiquer avec ce serveur, les machines ont toutes des **clés à long terme** (« permanentes ») basées sur le **hachage du mot de passe** du compte.



Dans la mythologie grecque, Cerbère (en grec ancien Κέρερος) est le chien à trois têtes gardant l'entrée des Enfers.



- Les chiffrements sont **symétriques**
- Les mdp ne sont jamais envoyés sur le réseau
- Les clés ne sont transmises que de manière chiffrée
- **Authentification mutuelle**
 - Le client s'authentifie en
 - transmettant un ticket chiffré avec la clé secrète du serveur (principe semblable à celui d'un jeton)
 - envoyant ses informations chiffrée avec sa clé secrète
 - Le serveur s'authentifie en répondant avec un message chiffré par la clé secrète du client
- KRB_AS_REQ : l'étape de pré-authentication n'est pas utilisée dans les systèmes *Legacy* ou ceux mal configurés
- KRB_AP_REQ : le service ne répond que si la requête contenant le flag **mutual-required**. La **sous-clé** sera utilisée par le client et le serveur pour communiquer de manière sécurisée.

- **Attaque par rejeu (*replay attack*)**

Forme d'attaque réseau dans laquelle un adversaire intercepte et retransmet malicieusement des données qui ont précédemment été échangées entre deux parties.

C'est l'une des attaques que peut mener un *man-in-the-middle*.

Pour se protéger des attaques par rejeu, les **tickets de Kerberos** contiennent :

- un *nonce*,
- un horodatage et
- une durée de validité.

7. Protocole SSH (*secure shell*)

- **SSH (*secure shell*)**

Historique : les protocoles d'accès distant via une interface en ligne de commande, comme **Telnet** (*teletype network*) ou **rsh** (*remote shell*), permettaient de communiquer sur un réseau TCP, avec une **transmission en clair des paquets** de données.

Mais avec Internet, le risque d'interception des paquets (*sniffing*) a nécessité de chiffrer la connexion entre les machines : Tatu Ylönen (Helsinki University of Technology) développe **SSH-1** en 1995

Version 2

SSH-2 est proposé comme norme par le groupe « secsh » en 2006 ; incompatible avec **SSH-1**
↳ Nouveautés : échange de clé DH, intégrité avec HMAC, chiffrement par AES, etc.

Utilisation

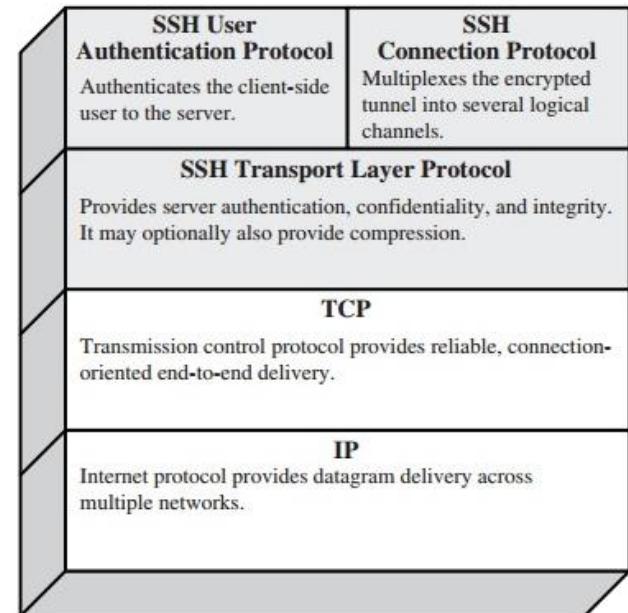
En plus d'un accès à distance en ligne de commande, un déport de l'affichage graphique (fenêtres X11) est possible, avec l'option **ssh -X**.

SSH permet également de sécuriser la création de tunnels, les transferts de fichiers (**scp**) ou la modification de fichiers (**sftp**).

● Pile de protocoles

SSH est organisé en **trois protocoles** s'appuyant sur TCP (typiquement, sinon WebSocket...)

- Couche de transport : authentification du serveur, confidentialité et intégrité, compression
- Authentification utilisateur : authentifier le client au serveur
- Connexion : multiplexage du tunnel chiffré en plusieurs canaux logiques



- **Clés hôtes (*host keys*)**

Paire **clé publique/clé privée du serveur**, pour son authentification auprès du client.

Cette authentification nécessite que le client possède déjà la **clé publique** du serveur K_s .

Un serveur peut avoir plusieurs clés hôtes, utilisant différents algorithmes de chiffrements asymétriques. Plusieurs utilisateurs peuvent partager la même clé hôte publique.

Confiance dans les clés hôtes publiques : deux modèles

1. Le client gère lui-même ses clés, via une base de données locale, qui associe chaque nom d'utilisateur à une clé publique

↳ Lourd à maintenir

2. Le client ne connaît que la clé publique d'une autorité de certification, afin de vérifier la validité des clés hôtes

↳ Nécessite que les clés hôtes publiques aient été certifiées

7.2 Protocole de la couche de transport

● Format des paquets

Taille maximum d'un paquet = 65535 octets

La **compression** (avec **zlib** ou **xz**) et l'authentification par MAC sont optionnelles

Numéro du paquet dans la séquence (sur 32 bits)

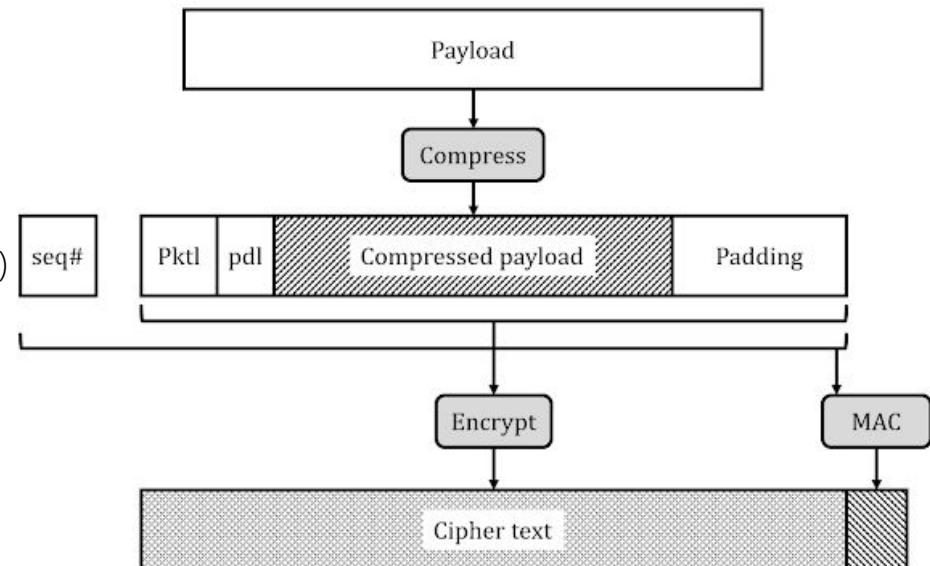
- ↳ Démarre à 0, puis incrémentation
- ↳ Inclus dans le calcul du MAC

Longueur du paquet (sur 4 octets)

⚠ N'est pas chiffrée : lecture par routeur TCP

Longueur du remplissage (sur 1 octet)

Remplissage : octets aléatoires (max. 4)
Si chiffrement de flux : obtenir un multiple de 1 octet



💡 La compression se fait toujours **avant** le chiffrement

29 mars 2024

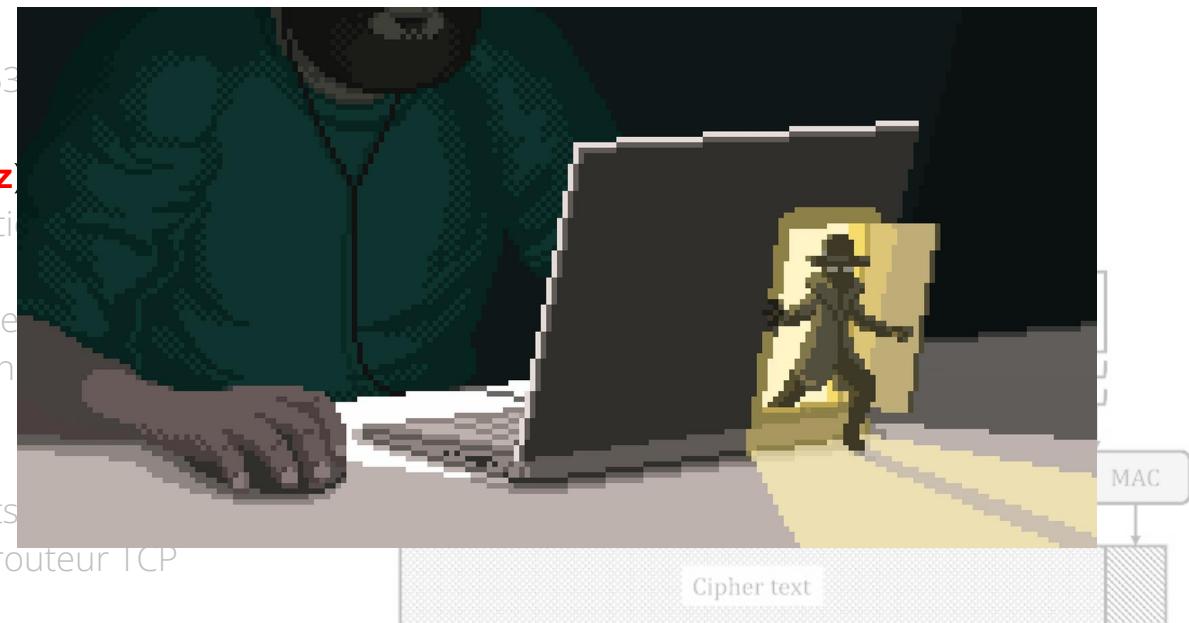
- Format des paquets

Taille maximum d'un paquet = 65535 octets

La compression (avec zlib ou xz)

| Rating | CVSS Score |
|----------|------------|
| None | 0.0 |
| Low | 0.1-3.9 |
| Medium | 4.0-6.9 |
| High | 7.0-8.9 |
| Critical | 9.0-10.0 |

Si chiffrement de flux : obtenir un multiple de 1 octet



💡 La compression se fait toujours **avant** le chiffrement

CVE-2024-3094

- « Poignée de main » (*handshake*)

Au préalable, le client établit une connexion TCP avec le serveur (protocole TCP, pas celui de la couche de transport)

Dans la première étape du *handshake*, le client envoie un paquet avec une chaîne de caractères d'identification (de version) V de la forme :

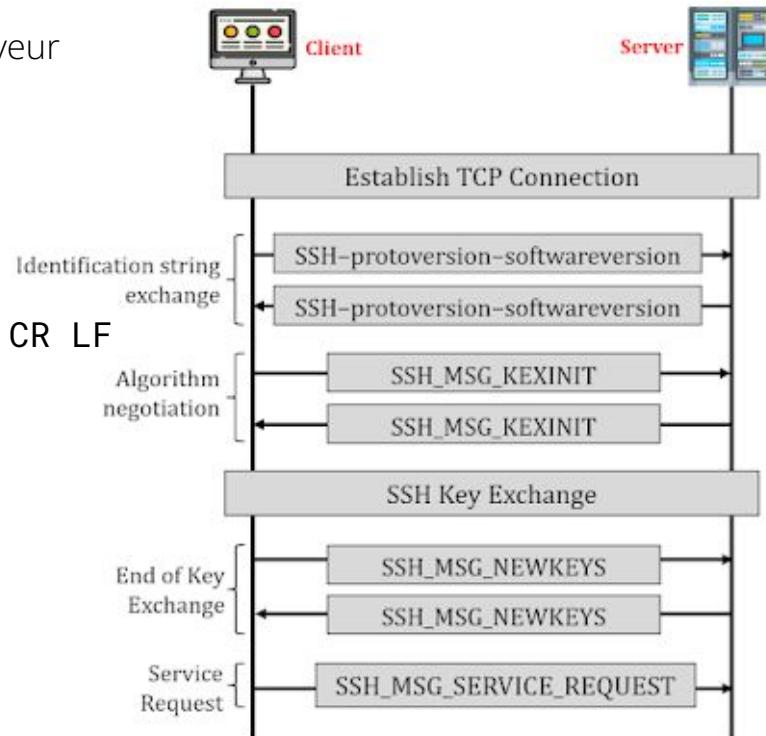
SSH-protoversion-softwareversion SP comments CR LF

SP : space ; CR : carriage return ; LF : line feed

Exemple :

SSH-2.0-billsSSH_3.6.3q3<CR><LF>

Le serveur répond avec sa propre chaîne d'identification.



7.2 Protocole de la couche de transport

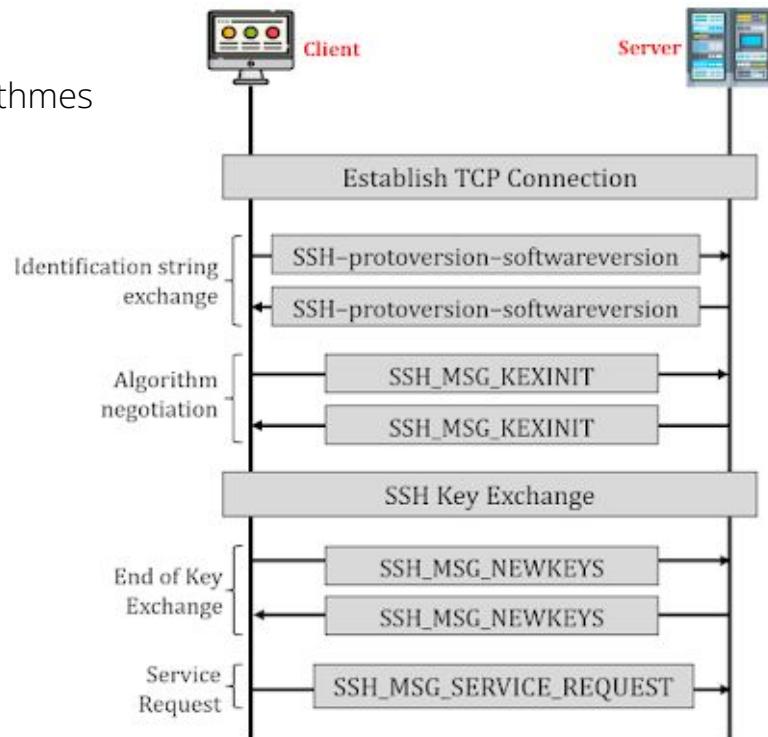
- « Poignée de main » (*handshake*)

Ensuite, le client et le serveur s'envoient mutuellement un message **I SSH_MSG_KEXINIT** contenant des listes d'algorithmes supportés, classés par ordre de préférences de l'expéditeur

4 catégories d'algorithmes :

- échange de clé
- chiffrement
- MAC
- compression

L'algorithme choisi est le premier sur la liste du client qui est aussi supporté par le serveur



- « Poignée de main » (*handshake*)

L'étape suivante est l'échange de clé Diffie-Hellman.

Une valeur de hachage H est calculée pour cet échange :

$$H = h(V_c \| V_s \| I_c \| I_s \| K_s \| e \| f \| K), \quad \text{p. ex. avec } h \in \text{SHA-2}$$

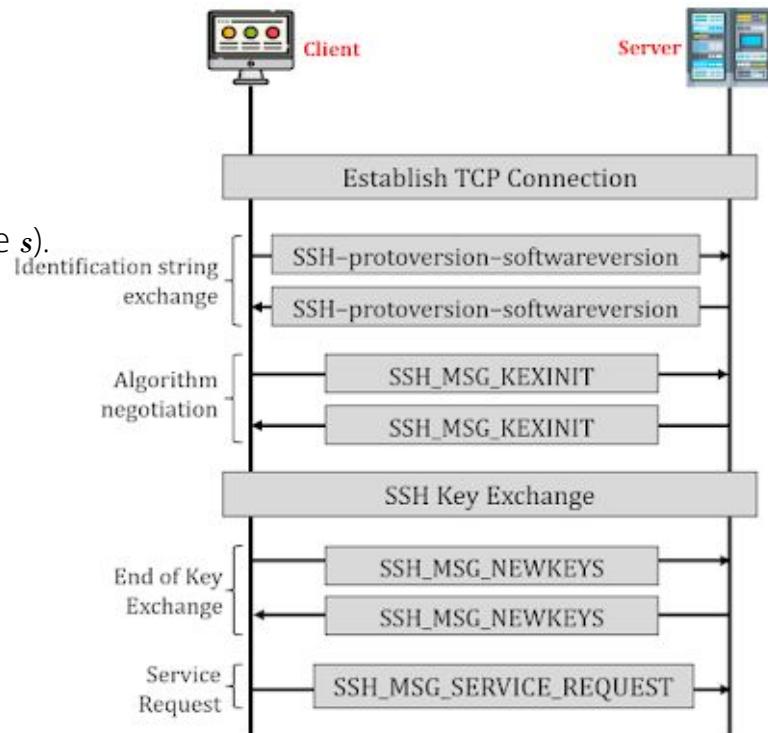
⚠ Le **serveur chiffre** H avec sa clé d'hôte privée (signature s).

Il envoie donc $(K_s \| f \| s)$, afin que le client puisse valider f (en déchiffrant s avec K_s).

La fin de l'échange de clé est signalée par un échange de paquets **SSH_MSG_NEWKEYS**.

Dans la dernière étape, le client envoie un paquet **SSH_MSG_SERVICE_REQUEST**, pour demander une authentification utilisateur ou bien une connexion.

🛡 **À partir de là**, toutes les données utilisateur (*payload*) des paquets échangés sont protégées par chiffrement et MAC.



- ### Génération des clés

Les clés **symétriques** utilisées pour le chiffrement et le calcul du MAC, ainsi que les éventuels vecteurs d'initialisation, sont générés à partir de la clé secrète partagée \mathbf{K} , la valeur de hachage \mathbf{H} de l'échange de clé et l'identifiant de session. Ce dernier est égal à \mathbf{H} , à moins qu'il n'y ait eu un autre échange de clé, consécutif à l'échange de clé initial.

Les calculs de la **fonction de dérivation de clés** (*key derivation function, KDF*) sont les suivants :

A, B, C, D, E et F sont des *nonces*

$$\text{IV}_{C \rightarrow S} = h(K \| H \| A \| \text{id_session})$$

$$\text{IV}_{S \rightarrow C} = h(K \| H \| B \| \text{id_session})$$

Chiffrement :

$$k_{C \rightarrow S} = h(K \| H \| C \| \text{id_session})$$

$$k_{S \rightarrow C} = h(K \| H \| D \| \text{id_session})$$

Intégrité (MAC) :

$$k'_{C \rightarrow S} = h(K \| H \| E \| \text{id_session})$$

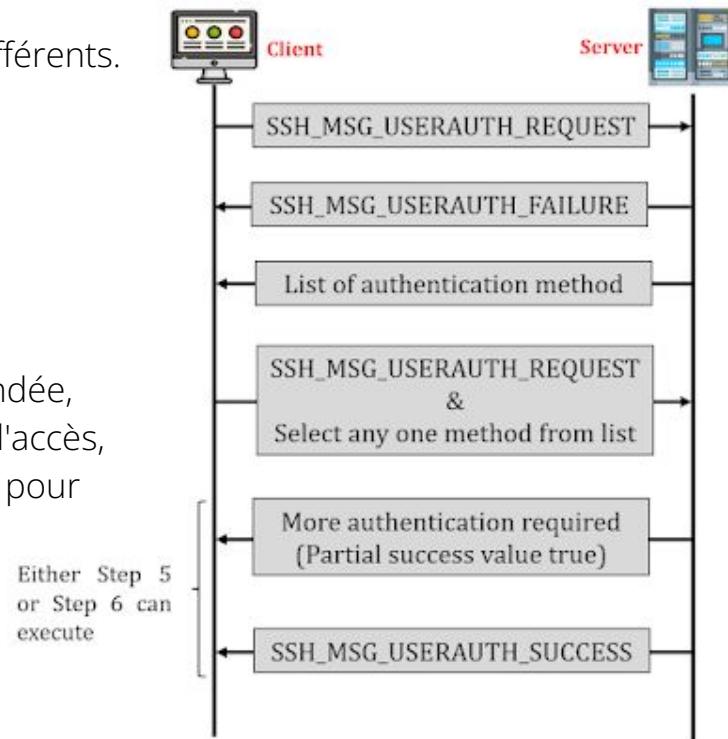
$$k'_{S \rightarrow C} = h(K \| H \| F \| \text{id_session})$$

● Messages échangés

Le client et le serveur échangent **trois types de messages** différents.

Par défaut, le premier octet du message envoyé par le client a une valeur décimale de **50**. Il sera alors interprété comme **SSH_MSG_USERAUTH_REQUEST**. Son format est :

```
byte SSH_MSG_USERAUTH_REQUEST (50)
string user name, l'identité associée à l'autorisation demandée,
string service name, le service auquel le client demande l'accès,
string method name, la méthode d'authentification utilisée pour
cette requête.
```



● Messages échangés

Si le serveur :

- rejette la requête d'authentification ou bien
- demande une ou plusieurs méthodes d'authentification supplémentaires, il envoie un message au format :

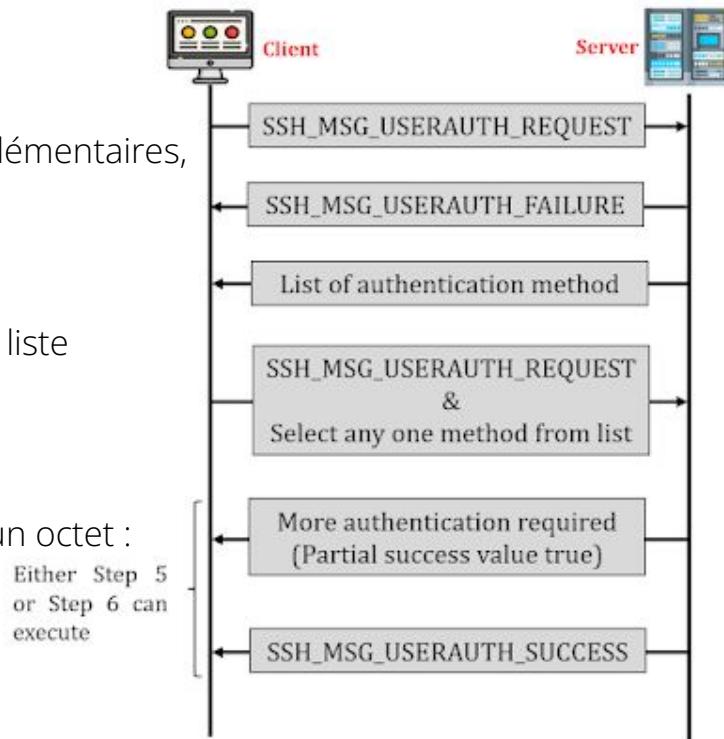
`byte SSH_MSG_USERAUTH_FAILURE (51)`

`name-list authentications that can continue`, une liste des méthodes permettant de poursuivre le dialogue

`Boolean partial success`

Si le serveur accepte l'authentification, il envoie un message d'un octet :

`SSH_MSG_USERAUTH_SUCCESS (52)`



- **Méthodes d'authentification**

Le serveur peut demander une ou plusieurs des méthodes d'authentification.

Les principales sont les suivantes.

Par clé publique : les détails varient selon l'algorithme de cryptographie asymétrique demandé.

Dans tous les cas, le client envoie un message au serveur contenant sa clé publique, avec un message signé (par sa clé privée). À la réception, le serveur vérifie si la clé publique est autorisée (fichier `authorized_keys`). Si tel est le cas, il vérifie ensuite que la signature est correcte.

Par mot de passe : le client envoie un message contenant son mdp en clair, qui est protégé par le chiffrement du protocole de la couche de transport.

Basée sur l'hôte : le serveur vérifie l'identité du client hôte plutôt que celle de l'utilisateur lui-même (adresses IP, noms de domaines, clé privée de l'hôte...). Cette méthode est moins sécurisée que celles authentifiant les utilisateurs.

```
guillaume@computer:~$ ssh -v gpostic@ssh2.ibisc.univ-evry.fr
```

OpenSSH_7.6p1 Ubuntu-4ubuntu0.7, OpenSSL 1.0.2n 7 Dec 2017
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: /etc/ssh/ssh_config line 19: Applying options for *

La librairie OpenSSL fournit des implémentations des différents algorithmes cryptographiques

debug1: Connecting to ssh2.ibisc.univ-evry.fr [195.221.162.152] **port 22**. Connexion TCP/IP au serveur SSH, avec le port par défaut
debug1: Connection established.

debug1: **identity file** /home/guillaume/.ssh/id_rsa type 0 **Recherche de la clé privée du client**

debug1: key_load_public: No such file or directory
debug1: identity file /home/guillaume/.ssh/id_rsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/guillaume/.ssh/id_dsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/guillaume/.ssh/id_dsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/guillaume/.ssh/id_ecdsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/guillaume/.ssh/id_ecdsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/guillaume/.ssh/id_ed25519 type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/guillaume/.ssh/id_ed25519-cert type -1 **SSH essaye de charger un certificat**

debug1: Local version string **SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.7**

debug1: Remote protocol version 2.0, remote software version OpenSSH_9.2p1 Debian-2+deb12u2

debug1: match: **OpenSSH_9.2p1 Debian-2+deb12u2** pat OpenSSH* compat 0x04000000 **Le client cherche une correspondance avec la version du serveur, sur la base d'un motif (pattern)**

debug1: **SSH2_MSG_KEXINIT** sent

debug1: SSH2_MSG_KEXINIT received

debug1: kex: algorithm: curve25519-sha256 **Courbe elliptique X25519**

debug1: kex: host key algorithm: ecdsa-sha2-nistp256 **Courbe elliptique NIST P-256**

debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none **AEAD**

debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none

debug1: expecting SSH2_MSG_KEX_ECDH_REPLY **Attente d'une réponse du serveur**

debug1: Server host key: ecdsa-sha2-nistp256 SHA256:PBw1NOEr6/SRK9KCzzeFXnWwrrzGhMqHzjF1gs5VR4 **Empreinte de la clé publique présentée par le serveur**
debug1: Host 'ssh2.ibisc.univ-evry.fr' is known and matches the ECDSA host key.

debug1: Found key in /home/guillaume/.ssh/known_hosts:4

debug1: **rekey after 134217728 blocks** **Les clés cryptographiques sont renégociées de façon périodique**

debug1: SSH2_MSG_NEWKEYS sent

debug1: expecting **SSH2_MSG_NEWKEYS**

debug1: SSH2_MSG_NEWKEYS received

```
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_EXT_INFO received Le client a reçu du serveur des informations étendues (extended) durant l'établissement de la connexion
debug1: kex_input_ext_info:
server-sig-algs=<ssh-ed25519, sk-ssh-ed25519@openssh.com, ecdsa-sha2-nistp256, ecdsa-sha2-nistp384, ecdsa-sha2-nistp521, sk-ecdsa-sha2-nistp256@openssh.com, webauthn-sk-ecdsa-sha2-nistp256@openssh.com, ssh-dss, ssh-rsa, rsa-sha2-256, rsa-sha2-512>
debug1: kex_input_ext_info: pubkey-hostbound@openssh.com (unrecognised) Extension de sécurité non-supportée
debug1: SSH2_MSG_SERVICE_ACCEPT received Le serveur a accepté la requête du client
debug1: Authentications that can continue: pubkey
debug1: Next authentication method: pubkey
debug1: Offering public key: RSA SHA256:P3LXX3EZ3BLtArfBwdD0W9rPftFGRWsw7gP2Gght5wg /home/guillaume/.ssh/id_rsa
debug1: Server accepts key: pkalg rsa-sha2-512 blen 279 La clé publique fait 279 octets
debug1: Authentication succeeded (pubkey).
Authenticated to ssh2.ibisc.univ-evry.fr ([195.221.162.152]:22).
debug1: channel 0: new [client-session] Canal SSH 0 : connexion logique de la session principale
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session. Interactive, par opposition à une session où le client est contrôlé par un script
debug1: pledge: network Les appels systèmes du client seront restreint au réseau
debug1: client_input_global_request: rtype hostkeys-00@openssh.com want_reply 0
debug1: Remote: /home/gpostic/.ssh/authorized_keys:1: key options: agent-forwarding port-forwarding pty user-rc
x11-forwarding
debug1: Remote: /home/gpostic/.ssh/authorized_keys:1: key options: agent-forwarding port-forwarding pty user-rc
x11-forwarding
debug1: Sending environment.
debug1: Sending env LC_MEASUREMENT = fr_FR.UTF-8 Le client envoie des variables d'environnement au serveur
debug1: Sending env LC_PAPER = fr_FR.UTF-8
debug1: Sending env LC_MONETARY = fr_FR.UTF-8
debug1: Sending env LANG = en_US.UTF-8
debug1: Sending env LC_NAME = fr_FR.UTF-8
debug1: Sending env LC_ADDRESS = fr_FR.UTF-8
debug1: Sending env LC_NUMERIC = fr_FR.UTF-8
debug1: Sending env LC_TELEPHONE = fr_FR.UTF-8
debug1: Sending env LC_IDENTIFICATION = fr_FR.UTF-8
debug1: Sending env LC_TIME = fr_FR.UTF-8
Linux ssh-bridge 6.5.13-1-pve #1 SMP PREEMPT_DYNAMIC PMX 6.5.13-1 (2024-02-05T13:50Z) x86_64
```

Liste des algorithmes de signature supportés, envoyées par le serveur

Le client présente l'empreinte de sa clé publique RSA

