# Cryptographie Bachelor 2

Simon Abelard
(Inspiré par un cours de L. Rouquette)

Printemps 2026

# Organisation du cours

## Contenu

- Cours, TD, TP
- Algorithmes cryptographiques
- Cryptanalyse (un peu)
- Protocoles et PKI (authenticité)

## Évaluation:

- Exam final le mardi 3 mars de 9h30 à 11h30

## Matériel autorisé:

- Pas de calculatrice ni de cours/TD
- Une feuille A4 que vous remplissez comme vous voulez
- Cette feuille est personnelle et porte votre nom

# Contenu du cours

- Fonctions de hachage
- Chiffrement symétrique
- Chiffrement asymétrique
- Générateurs d'aléa
- Certains protocoles (SSH)
- Infrastructure à clé publiques (PKI)
- Post-quantique ?

# Plan du cours

- Semaine 1
  - CM : Introduction
  - TD : Hachage
  - TP : Chiffrements par substitution
- Semaine 2
  - CM : Cryptographie symétrique
  - TD : Architecture 1
  - TP : Substitution
- Semaine 3
  - CM : Cryptographie asymétrique
  - TD : Logarithme discret
  - TP : Schéma d'El Gamal
- Semaine 4
  - CM : Protocoles et PKI
  - TD : Architecture 2
  - TP : examen !

# Problématique

## Système d'information

- Une multitude d'utilisateurs
- Qui ont des machines en réseau
- Qui échangent des informations numériques
- Via des canaux **publics**

- Donnée $\neq$ information
- Donnée $=$ représentation d'une information
- Une donnée porte plusieurs informations
- Protéger une donnée $\neq$ protéger l'information
  (Si A envoie à B un mail chiffré c'est déjà une information en soi)

# Les besoins de la sécurité

- Disponibilité
- Confidentialité
- Intégrité
- Authenticité
- Non-répudiation

Quiz: quels sont les besoins ciblés par les attaques suivantes ?

- Écouter aux portes
- Imiter une signature
- Voler un badge d'accès
- Attaque DDoS
- Brouillage GPS
- Usurper une identité

# Les besoins de la sécurité

## Disponibilité

Dans le domaine de l'ingénierie de fiabilité, la disponibilité d'un équipement ou d'un système est une mesure de performance obtenue en divisant la durée pendant laquelle ledit équipement ou système est opérationnel par la durée totale pendant laquelle on aurait souhaité qu'il le soit.

## Intégrité

L'intégrité des données est l'assurance que les données de l'entreprise sont exactes, complètes et cohérentes tout au long de leur cycle de vie.

## Confidentialité

La confidentialité est le fait de s'assurer que l'information n'est accessible qu'à ceux dont l'accès est autorisé.

# Les besoins de la sécurité II

## Non-répudiation

Aucun utilisateur ne doit pouvoir contester les opérations qu'il a réalisées dans le cadre de ses actions autorisées et aucun tiers ne doit pouvoir s'attribuer les actions d'un autre utilisateur.

## Authenticité

L'authentification est le processus visant à confirmer qu'un commettant est bien légitime pour accéder au système.
Parmi les méthodes usuelles on utilise par exemple un secret que seul le commettant connaît (code, mot de passe), un appareil que seul le commettant possède (smartphone personnel, carte à puce, dongle) ou encore une opération que seul le commettant sait faire.

# Comment faire ?

La sécurité de l'information repose sur :

- Une culture de sécurité
- Des règles organisationnelles (GRC)
- L'anticipation des menaces (Counter-threat intelligence)
- La détection et la réponse à incidents (SoC, SIEM, SOAR, CSIRT etc...)
- L'utilisation d'outils adaptés, dont la **cryptographie**

**La cryptographie est importante mais ce n'est qu'un maillon de la chaîne!**

# La cryptologie

Au sens large, c'est l'étude de la protection de l'information numérique contre des actions malveillantes. (Par opposition aux codes correcteurs qui protègent d'une altération accidentelle)

La cryptologie se divise en de multiples branches :

**La cryptographie** (défensive)

- Cryptographie symétrique
- Cryptographie asymétrique
- Protocoles (ZK, TLS, etc.)
- Function de hachage

**La cryptanalyse** (offensive)

- Cryptanalyse algorithmique
- Canaux auxiliaires
- Bug Exploit
- Social Engineering

# Pourquoi faire de la cryptographie ?

Le maillon faible est souvent l'utilisateur, presque jamais la cryptographie, MAIS

- Ne pas faire de cryptographie c'est risquer que la cryptographie devienne le maillon faible !
- Une faille cryptographique est moins probable mais plus globale donc potentiellement désastreuse !

**Exemples:** panne du Playstation network pendant quelques jours (mauvaise implémentation de la cryptographie), Attaque LOGJAM qui cible **8% d'internet** (cryptographie sous-dimensionnée), Cassage d'ENIGMA et des variantes japonaises pendant la IIe Guerre Mondiale (mauvais usage et sous-dimensionnement)

# Brève histoire de la cryptographie

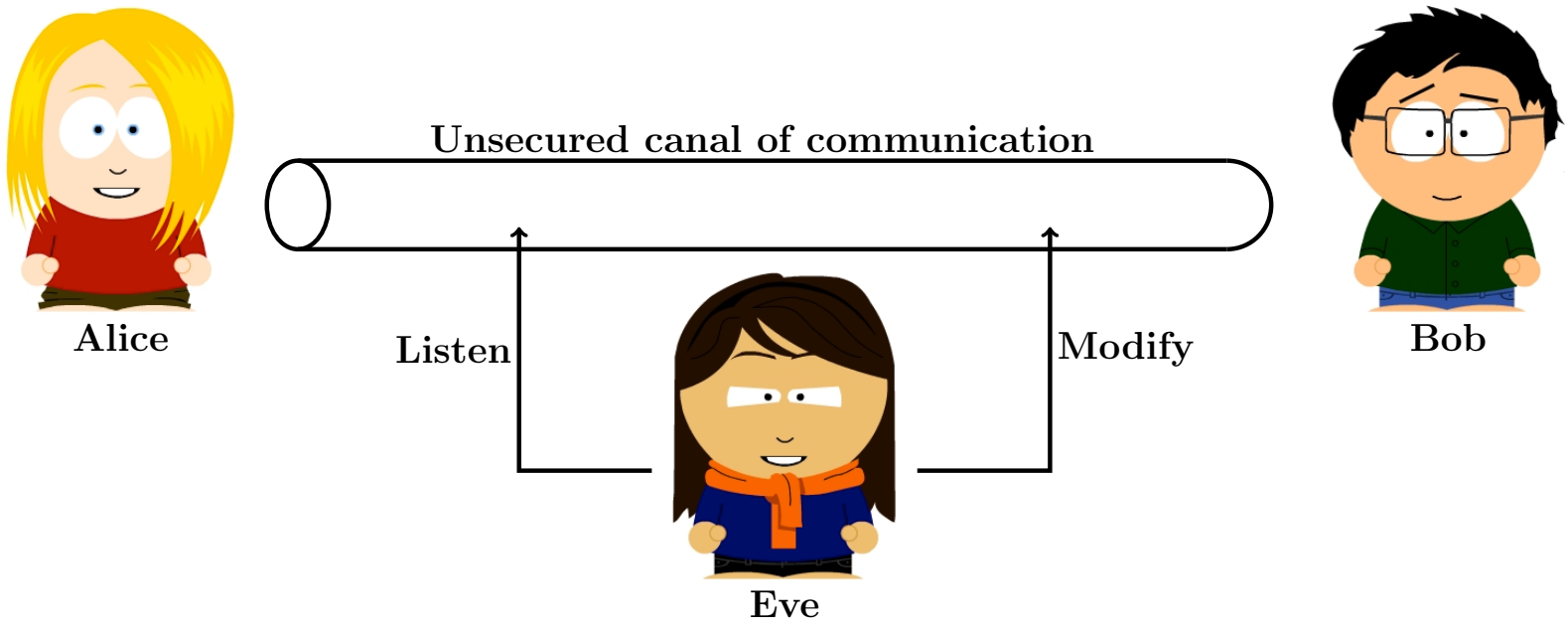
La cryptographie est presque aussi vieille que l'écriture, mais la cryptographie dite moderne n'a même pas 60 ans !

- Antiquité : stéganographie (scytale, esclave rasé, etc.)
- De la Rome Antique jusqu'à 1960 : chiffrement par substitutions et permutations. D'abord à la main puis avec des appareils mécaniques (rotors)
- Au milieu des années 50 : étude systématique
- Dans les années 1970 : standard DES (crypto symétrique)
- En 1976, invention de la cryptographie asymétrique
- Rupture moderne (2015) : la cryptographie post-quantique
- Nouvelle technologie : la cryptographie quantique

# Modèle de communications

Préalable à toute analyse de sécurité rigoureuse : la modélisation.
Il s'agit de préciser :

- Quelles sont les propriétés du canal de communication ?
- Quelles sont les capacités de l'attaquant ? (passif vs actif)
- Quelles informations sont en possession des protagonistes ?




;

# Différentes menaces

**Attaquant passif :** espionne/stocke les communications

**Attaquant actif :** va tenter de

- Usurper une identité
- Altérer des données
- Rejouer des messages/commandes
- Bloquer ou retarder la transmission de messages/commandes
- Détruire des messages

**Un message n'est pas toujours un simple email ! Il peut s'agir de données vitales ou de commandes destinées à des systèmes complexes mettant en jeu des vies humaines !**

# La cryptographie : un empilement de couches

- Des textes réglementaires (articles de lois, certifications) vous disant ce que vous devez protéger contre quel type d'attaque. Exemple : ISO 27001, RGPD, etc.
- Des mécanismes cryptographiques : protocoles, algorithmes, modes, paramètres. Exemple : standards NIST, RGS ANSSI.
- Des implémentations (logiciel). Exemple : OQS, OpenSSL
- Du matériel. Exemple : HSM, chiffreur IP, etc.

Exemple de raisonnement pour concevoir une architecture :

- Quel besoin ?
- Quelle primitive ?
- Quelle sécurité ?
- Paramètres ?

# La cryptographie : un empilement de couches

- Des textes réglementaires (articles de lois, certifications) vous disant ce que vous devez protéger contre quel type d'attaque. Exemple : ISO 27001, RGPD, etc.
- Des mécanismes cryptographiques : protocoles, algorithmes, modes, paramètres. Exemple : standards NIST, RGS ANSSI.
- Des implémentations (logiciel). Exemple : OQS, OpenSSL
- Du matériel. Exemple : HSM, chiffreur IP, etc.

Exemple de raisonnement pour concevoir une architecture :

- Quel besoin ?
- Quelle primitive ?
- Quelle sécurité ?
- Paramètres ?

- Confidentialité
- Le standard AES
- 128 bits (civil)
- Aucun

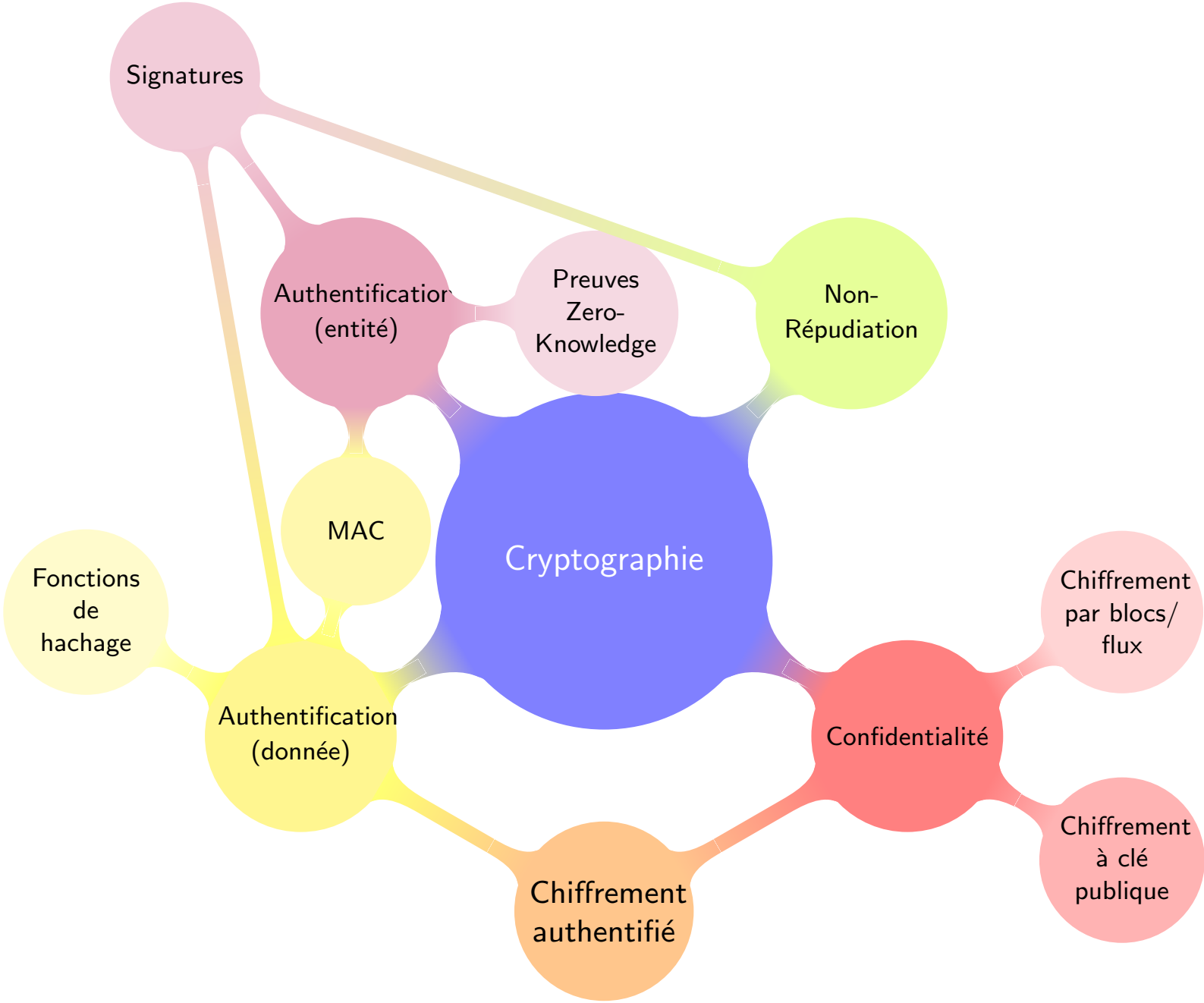
# La cryptographie : un empilement de couches

- Des textes réglementaires (articles de lois, certifications) vous disant ce que vous devez protéger contre quel type d'attaque. Exemple : ISO 27001, RGPD, etc.
- Des mécanismes cryptographiques : protocoles, algorithmes, modes, paramètres. Exemple : standards NIST, RGS ANSSI.
- Des implémentations (logiciel). Exemple : OQS, OpenSSL
- Du matériel. Exemple : HSM, chiffreur IP, etc.

Exemple de raisonnement pour concevoir une architecture :

| | | |
|---|---|---|
| Quel besoin ? | Confidentialité | Authenticité |
| Quelle primitive ? | Le standard AES | ECDSA |
| Quelle sécurité ? | 128 bits (civil) | 128 bits (civil) |
| Paramètres ? | Aucun | Courbe FRP256v1 |

# Les mécanismes fondamentaux de la cryptographie



# Fonctions de hachage

## Fonction de hachage

Une fonction de hachage est un algorithme (que l'on souhaite efficace) qui calcul une valeur **de taille fixe** appelée **empreinte ou haché** à partir d'un message de taille quelconque.

$$H : \{0,1\}^* \to \{0,1\}^n, H(m) = h.$$

**Exemples:** troncation, table d'indices, etc.

Outil très utile en cryptographie mais rarement employé seul !

**Cas d'usage :** une source fiable me dit que le haché du fichier $\mathrm{toto.pdf}$ vaut $h$. Je récupère $\mathrm{toto.pdf}$ sur un serveur public. Si $H(\mathrm{toto.pdf}) \neq h$, alors je sais que quelqu'un a truqué le fichier !

**Question :** Et si un attaquant arrive à fabriquer un fichier $\mathrm{tutu.pdf}$ tel que $H(\mathrm{tutu.pdf}) = h$ ?

# Fonction de hachage cryptographique

## Fonction de hachage cryptographique

Une fonction de hachage telle que :

- **Premier antécédent :** à partir de $y$, il est "impossible" de trouver $x$ tel que $H(x) = y$
- **Deuxième antécédent :** à partir de $x$ et $H(x)$ il est "impossible" de trouver $x' \neq x$ tel que $H(x') = H(x)$.
- **Collision :** il est "impossible" de trouver $x$ et $x'$ distincts tel que $H(x) = H(x')$.

**Exemples:** MD5 (128 bits), SHA-1 (160 bits), SHA-2 & SHA-3 offrent plusieurs choix de taille (224, 256, 384 et 512 bits).

# Sécurité calculatoire

Que veut dire "impossible"? Si le haché fait 256 bits, je calcule $H(x)$ pour $2^{256}+1$ valeurs de $x$ et j'ai trouvé une collision.

## Paradoxe des anniversaires

S'il y a $2^{\ell}$ hachés possibles, j'ai plus d'une chance sur deux d'avoir une collision dans un ensemble de

$$\sqrt{\frac{2^{\ell}\pi}{2}} = O(2^{\frac{\ell}{2}})$$

empreintes aléatoires.

Pour MD5 cela fait environ $2^{64}$ hachés, et $2^{80}$ pour SHA-1.

En dessous de $2^{80}$ opérations, une attaque est possible en pratique, au-dessus de $2^{128}$ opérations on est à l'abri.

Un ordinateur cadencé à 1GHz mettrait 15 milliards d'années à faire $2^{90}$ opérations.

# Sécurité calculatoire : un peu de hauteur

Tous les mécanismes cryptographique vus en cours sont cassables : Objets de une taille bornée $\Rightarrow$ essayer toutes les possibilités.

## Sécurité pratique ou calculatoire

Un mécanisme est sûr si un attaquant possédant les spécifications de l'algorithme, autant de données que possibles à sa disposition (ex: des messages chiffrés) et **une grande puissance de calcul** ne peut pas le casser en un temps humainement raisonnable.

**Exemples :** s'il faut couvrir la planète de superordinateurs ou s'il faut faire un calcul qui dure des milliards d'années pour casser un schéma, il est considéré comme sûr.

**Remarque :** la sûreté est relative (MD5 et RSA-512 en 1990) et pas toujours simple à évaluer. Rassurez-vous, ce sont les agences et les normes qui font ce travail !

# Chiffrement symétrique et générateur d'aléa

# Cryptographie symétrique

**Le principe :** Alice et Bob partagent la même clé utilisée pour chiffrer et déchiffrer les messages.

**Avantages :** très efficaces même pour de grands messages.

**Inconvénient :** comment Alice et Bob partagent la clé ?
Ils ne peuvent pas l'envoyer sur un canal public.

**Idée :** des morceaux du message (lettres, bits, groupes de lettres) subissent des substitutions et des permutations.

## Exemple : le chiffrement de César

La clé secrète $s$ est un nombre entre 1 et 25.
Chiffrer = décaler les lettres de $s$ positions dans l'alphabet.
Déchiffrer = décaler de $s$ positions dans l'autre sens.
Pour $s = 25$, AVE CESAR devient ZUD BDRZQ.
Pour $s = 1$, AVE CESAR devient BWF DFTBS.

# Le chiffrement de Vigenère

**Faille de César :** deux lettres identiques ont le même chiffré. En français, la lettre E est la plus fréquente. Ainsi, la lettre la plus fréquente du chiffré de César est le chiffré de $E$. On déduit la clé.

**Contre-mesure :** ne pas décaler identiquement toutes les lettres. La clé devient $s = s_1 s_2 \cdots s_k$ et on décale la première lettre de $s_1$ positions, la deuxième de $s_2$ et ainsi de suite.

**Cryptanalyse :** une fois la longueur de la clé connue, l'analyse fréquentielle permet de casser le schéma. Comment calculer la longueur de la clé ? Utiliser les répétitions.

**Faille de Vigenère :** deux lettres identiques espacées de $n$ fois la taille de la clé ont le même chiffré.

# Chiffrement de Vernam ou masque jetable (1917)

**Chiffrement de Vigenère :** addition modulo 26.
Version binaire: XOR ou addition dans $\mathbb{F}_2$ ($1 \oplus 1 = 0$).

## Chiffrement de Vernam

Le chiffré du message $m$ avec la clé $K$ est $E_K(m) = m \oplus K$. Ce chiffrement est prouvé parfait à condition que :

- $K$ soit purement aléatoire
- $K$ soit de la taille de $m$
- $K$ ne soit utilisée qu'une seule fois

Ce chiffrement n'est évidemment pas pratique car :

- Qu'est-ce que l'aléa pur ?
- $K$ est beaucoup trop longue.

Cependant les schémas symétriques essaient de s'en rapprocher en utilisant diverses techniques (pseudo-aléa, key schedule, etc.)

# Générateur d'aléa physique (TRNG)

Appareil qui utilise un phénomène physique **bien modélisable** et connu pour son caractère aléatoire : la désintégration des radionucléides, le bruit thermique, la mesure de certains états quantiques, etc.

Par extension, certains générateurs utilisent des phénomènes incontrôlables mais moins bien modélisés physiquement (dev/random utilise du bruit issu de divers périphériques).

Un TRNG peut donc être assez coûteux et son débit de génération d'aléa de qualité peut être assez faible.

**La qualité d'un TRNG est soumise à des certifications** (par exemple la norme AIS20/31).

# Les générateurs de pseudo-aléa

- On les appelle DRNG/PRNG (Deterministic/Pseudorandom Number Generators).
- Algorithmes mathématiques efficaces pour générer des suites de nombres qui semblent aléatoire (au sens statistique) à partir d'une valeur initiale (graine ou seed).
- Pour une même valeur de la graine, le comportement est toujours le même (reproductibilité)
- Pour une même valeur de la graine, le comportement est toujours le même (reproductibilité).
- Les PRNG ont un débit bien plus rapides mais peuvent souffrir de (légers) biais statistiques.

# Les générateurs de pseudo-aléa cryptographique

Appelés lisseurs (cryptographiques), et en anglais CSPRNG (Cryptographically Secure Pseudorandom Number Generators). Ce sont des PRNG qui sont tellement proches de l'aléa qu'ils peuvent être utilisés en cryptographie.

- Formellement cela signifie qu'un attaquant ayant observé de nombreuses valeurs retournées par le PRNG dans le passé ne disposera d'aucun avantage pour prédire la sortie du PRNG par rapport à quelqu'un qui prédit au hasard.
- Ces CSPRNG sont soumis à des normes strictes et sont testés et certifiés
- Ils sont testés en continu pour détecter une chute d'entropie (ex : défaut matériel ou logiciel).

**La combinaison gagnante : utiliser un CSPRNG en l'alimentant périodiquement avec de l'entropie (des graines) issues d'un TRNG.**

# Création d'un canal confidentiel

Alice veut échanger avec Bob.

- Alice utilise un générateur d'aléa pour créer une clé secrète $K$
- Avec un canal authentifié et confidentiel, elle envoie cette clé à Bob
- Tout message transitant entre Alice et Bob est ensuite chiffré avec un algorithme symétrique en utilisant la clé $K$
- Le canal authentifié et confidentiel est physique (Alice et Bob se voient), repose sur la cryptographie asymétrique ou sur une session précédente (si pas de compromission).
- Quel algorithme symétrique choisir à part Vernam?

# Chiffrement par flux

**Idée:** simuler un chiffrement de Vernam avec une clé pseudo-aléatoire issue de la clé $K$.

Soit $G$ un générateur pseudo-aléatoire **déterministe** tel que $G(s, n)$ soit une chaîne de $n$ bits produite en utilisant la graine $s$.

## Chiffrement par flux

Fonction de chiffrement $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$, telle que

$$E_K(M) = E(K, M) = M \oplus G(K, m).$$

- Partant d'une clé $K$ courte (ex. 128 bits),
- On produit une clé aussi longue que le message,
- On XOR avec le message.

**Question:** pourquoi $G$ doit-il être déterministe ?
**Remarque:** $E_K(E_K(M) = M$ ($E$ permet de déchiffrer !)

# Chiffrement par blocs

**Idée:** plutôt que d'agrandir $K$, couper le message en blocs de taille inférieure à $K$.

## Chiffrement par flux

Fonction de chiffrement $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$.
Fonction de déchiffrement $D : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ telle que $D_K(E_K(M)) = M$.

- Pour chiffrer $M$ on le découpe en bloc
- Le chiffré est la concaténation des chiffrés de chaque bloc
- On déchiffre bloc par bloc

# Un bon chiffrement par bloc

**Un bon algorithme vérifie les propriétés de Shannon (1949).**

## Confusion

La relation entre la clé de chiffrement et le texte chiffré doit être aussi complexe que possible.

## Difusion

La diffusion est une propriété où la redondance statistique dans un texte en clair est dissipée dans les statistiques du texte chiffré. En d'autres termes, un biais en entrée ne doit pas se retrouver en sortie et les statistiques de la sortie doivent donner le moins possible d'informations sur l'entrée. ( Wikipédia)

**Réflexion :** est-ce que les chiffrements de César et de Vigenère possèdent ces propriétés ?

# Chiffrement par bloc:

**Idée :** puisque chaque chiffrement ajoute un peu de désordre, nos algorithmes vont itérer plusieurs **tours** de chiffrement successifs.

## Chiffrement itératif

$$E_K(M) = E^r_{K_r}\left(E^r_{K_{r-1}} \cdots \left(E^r_{K_2}\left(E^r_{K_1}(M)\right)\right)\right)$$

## Exemple, les SPN (Substitution Permutation Networks)

Chaque tour de chiffrement est composée de :

- Couche de substitution et de $\oplus K_i$ (génère la confusion)
- Couche de permutation (génère la diffusion)

La répétition de plusieurs tours au sein d'un SPN permet de générer suffisamment de confusion et de diffusion.

# Chiffrement par bloc: AES

Il s'agit d'un chiffrement symétrique par blocs standardisé par le NIST en 2001 et qui a succédé au DES.

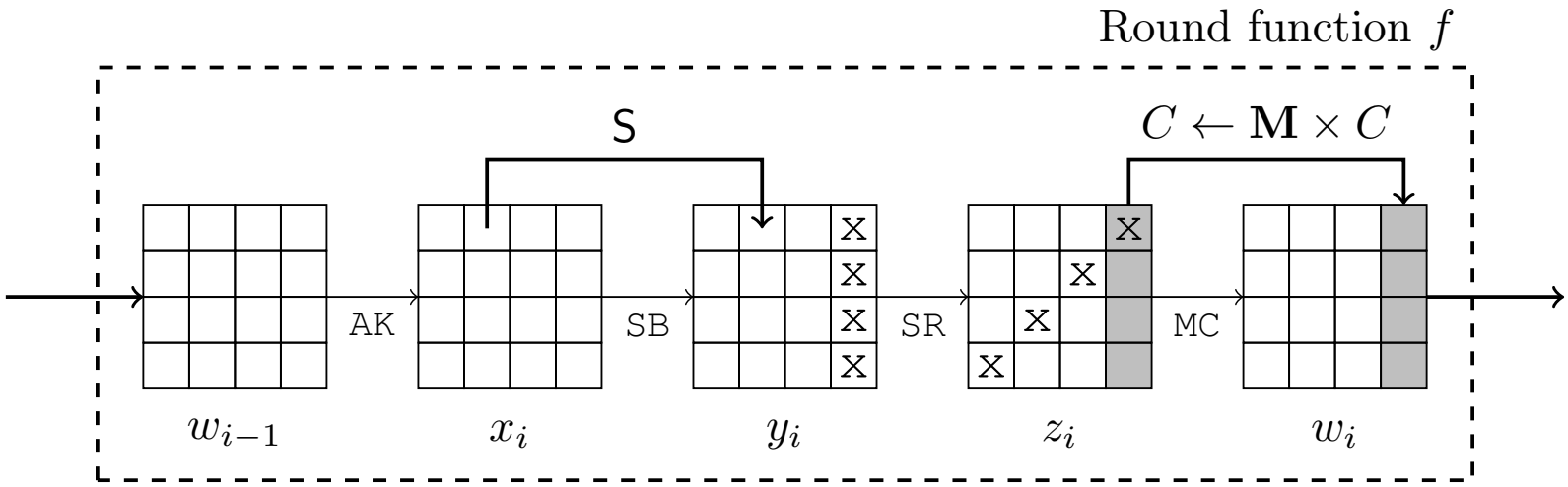
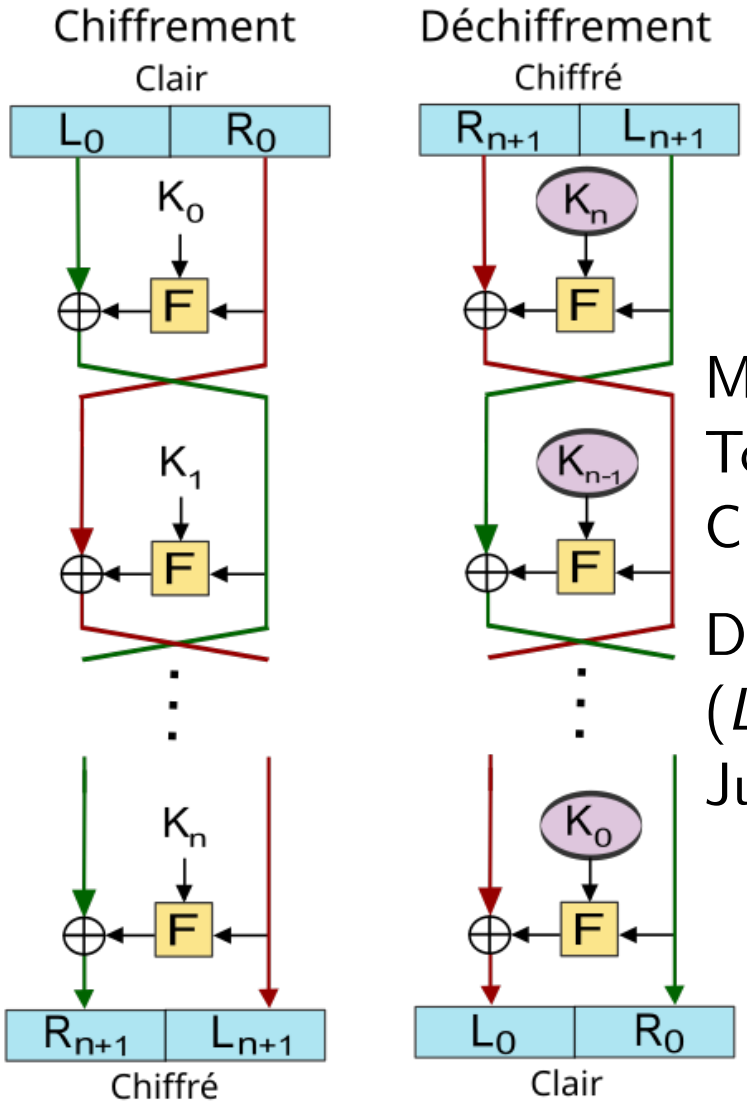



**Schéma de la fonction tour de l'AES.**

# Chiffrement par bloc: réseaux de Feistel




Message coupé en deux $M = (L_0, R_0)$

Tour $(L_i, R_i) = (f_{K_i}(R_{i-1}) \oplus L_{i-1}, L_{i-1})$

Chiffré $C = (L_r, R_r)$

Déchiffrement = calculer les

$(L_i, R_i) = (f_{K_i}(L_{i+1}) \oplus R_{i+1}, R_{i+1})$

Jusqu'à $(L_0, R_0) = M$.

source: Wikipedia

# Les modes d'opérations

**Question :** comment construire un chiffrement par flux ?

**Idée :** couper en blocs, chiffrer chaque bloc et concaténer !

**Réflexion :** quid de la diffusion et de la confusion?

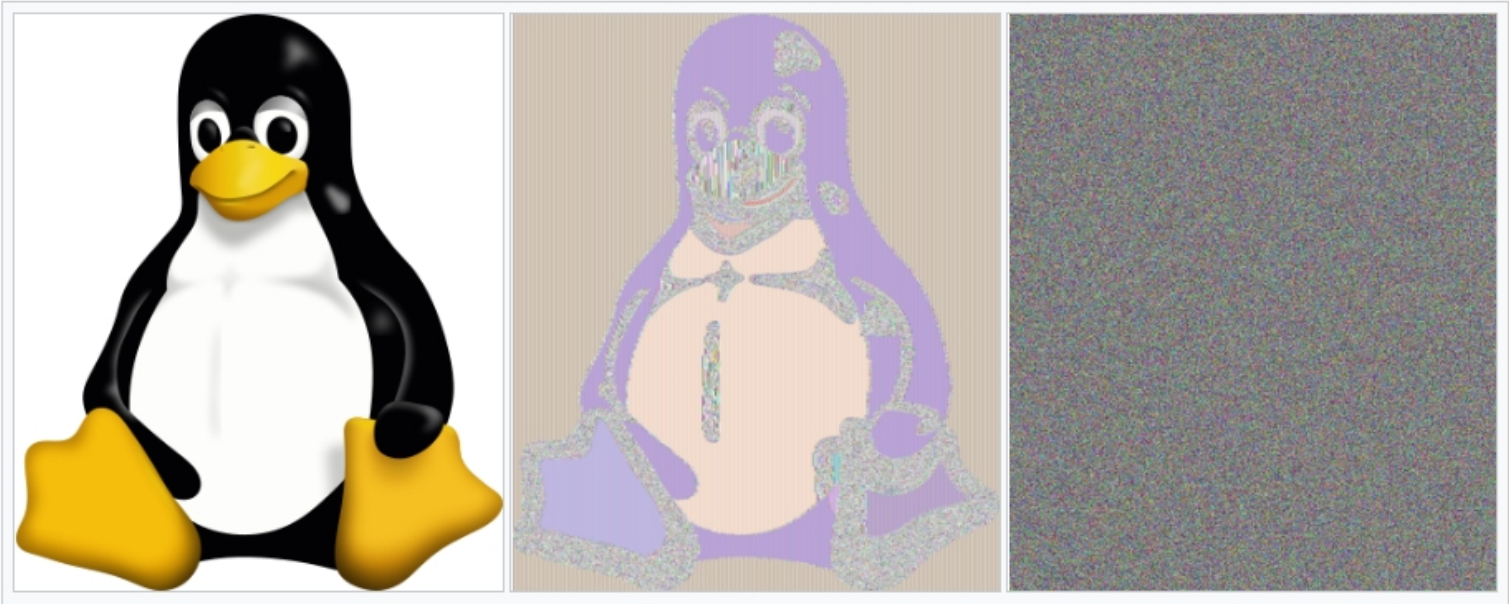
# Les modes d'opérations

**Question :** comment construire un chiffrement par flux ?

**Idée :** couper en blocs, chiffrer chaque bloc et concaténer !

**Réflexion :** quid de la diffusion et de la confusion?

**La réponse par l'exemple :** (source Wikipedia)

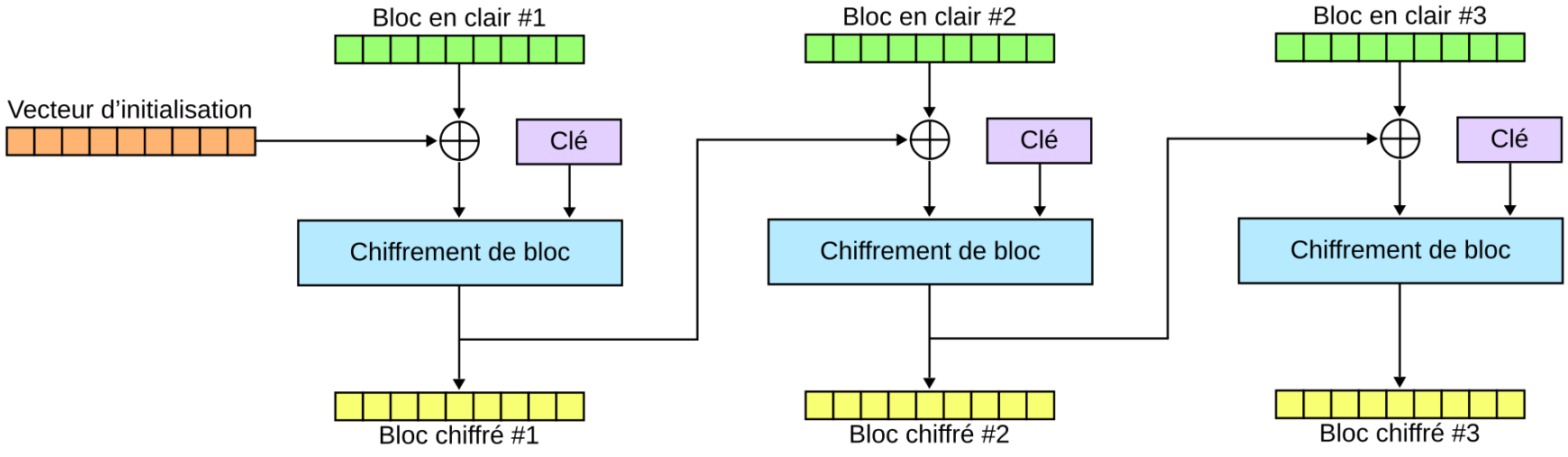


**Remarque :** cette idée est le mode ECB, mode déconseillé mais satisfaisant sur des données non-structurées.

# Modes d'opérations

**Solution :** lier le traitement d'un bloc aux autres pour obtenir une bonne diffusion.

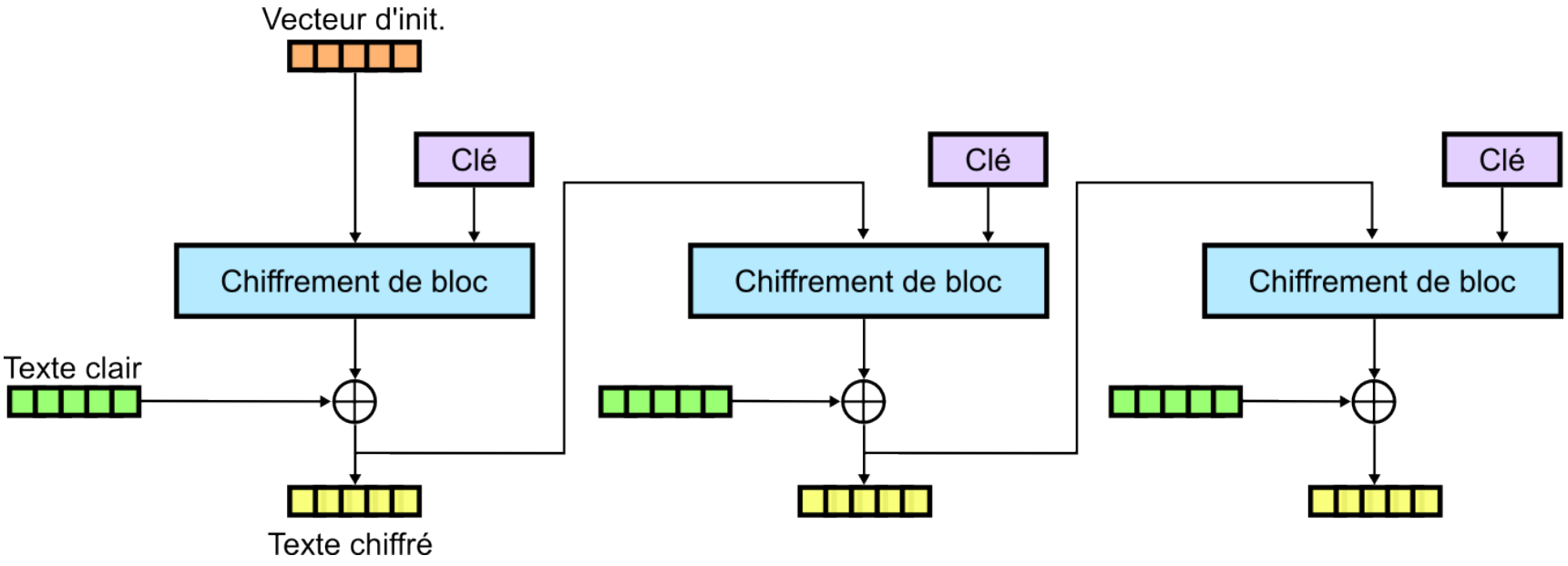
**Mode CBC (Cipher Block Chaining) :**




(source Wikipedia)

Le chiffrement n'est pas parallélisable mais le déchiffrement l'est.

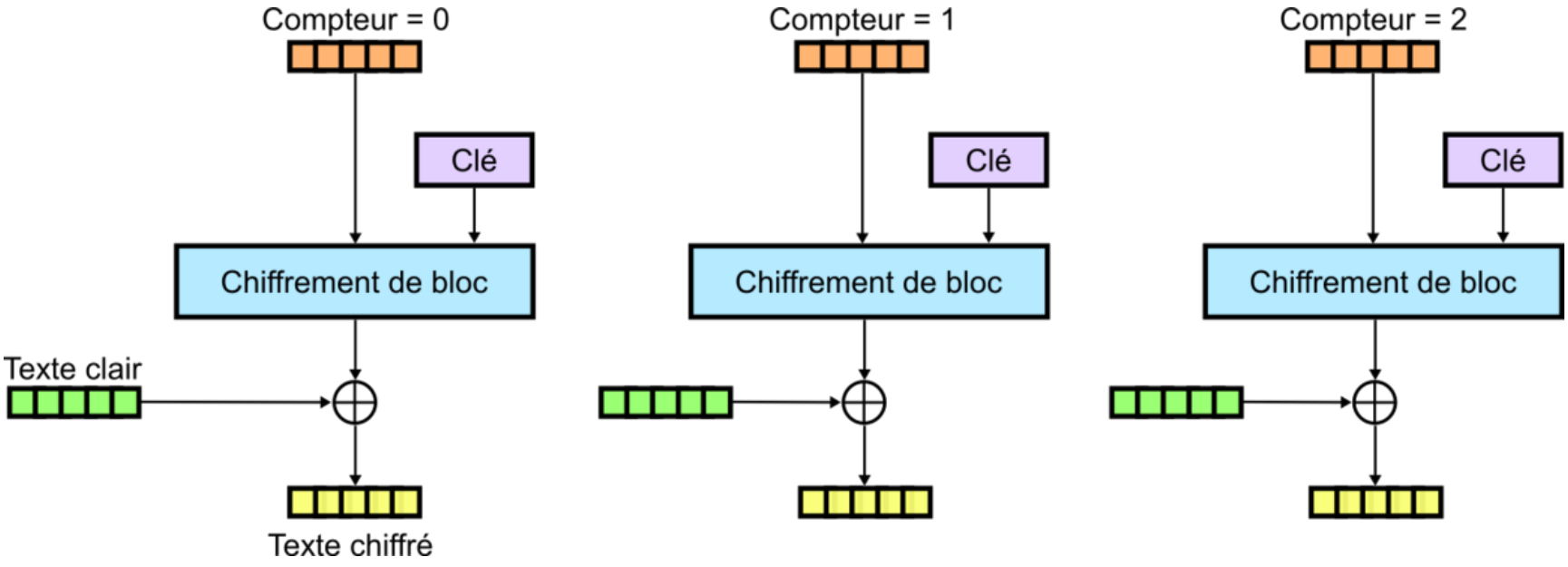
# Mode CFB (Cipher Feedback)




(source Wikipedia)

Le chiffrement n'est pas parallélisable mais le déchiffrement l'est.
Contrairement à CBC, la longueur du message peut ne pas être un multiple de la taille du bloc.

# Mode compteur (CTR)




(source Wikipedia)

Le mode CTR est parallélisable en chiffrement et en déchiffrement.
**Attention :** de manière générale les vecteurs d'initialisation (IV) ne sont pas secrets mais imposent des précautions (ex : ne pas réutiliser le même IV avec la même clé)

# Documents normatifs

- Standard NIST FIPS 197 pour AES
- Modes d'opérations SP800-38A du NIST (pour ceux du cours)
- Modes supplémentaires : NIST SP800-38B à SP800-38G
- Hachage : NIST FIPS 180-4 (SHA2) et FIPS 202 (SHA3)

## Contenu de ces standards

Ces documents décrivent précisément les algorithmes et la manière de les implémenter de façon sécurisée et interopérable (par exemple dans quel ordre lire les octets/bits), quels paramètres choisir pour quel niveau de sécurité, etc.

Le NIST fournit par ailleurs des séquences entrée/sortie permettant de tester ses implémentations (NIST CAVP).

# Cryptographie asymétrique

# Les limites de la cryptographie symétrique

- D'après vous, quelle est la plus grande faiblesse du chiffrement symétrique ?

- Une entreprise a 100 terminaux (téléphones, ordinateurs) et chacun doit pouvoir communiquer de manière sécurisée avec n'importe quel autre. Combien de clés symétriques faut-il ?

# Les limites de la cryptographie symétrique

- D'après vous, quelle est la plus grande faiblesse du chiffrement symétrique ?
  Il faut qu'Alice et Bob possèdent la même clé secrète, ils ne peuvent donc pas la diffuser sur le canal.
- Une entreprise a 100 terminaux (téléphones, ordinateurs) et chacun doit pouvoir communiquer de manière sécurisée avec n'importe quel autre. Combien de clés symétriques faut-il ?

# Les limites de la cryptographie symétrique

- D'après vous, quelle est la plus grande faiblesse du chiffrement symétrique ?
  Il faut qu'Alice et Bob possèdent la même clé secrète, ils ne peuvent donc pas la diffuser sur le canal.
- Une entreprise a 100 terminaux (téléphones, ordinateurs) et chacun doit pouvoir communiquer de manière sécurisée avec n'importe quel autre. Combien de clés symétriques faut-il ?
  Une clé par paire de terminaux, soit $\frac{100 \times 99}{2} = 4950$ clés !

# Les limites de la cryptographie symétrique

- D'après vous, quelle est la plus grande faiblesse du chiffrement symétrique ?
  Il faut qu'Alice et Bob possèdent la même clé secrète, ils ne peuvent donc pas la diffuser sur le canal.
- Une entreprise a 100 terminaux (téléphones, ordinateurs) et chacun doit pouvoir communiquer de manière sécurisée avec n'importe quel autre. Combien de clés symétriques faut-il ?
  Une clé par paire de terminaux, soit $\frac{100 \times 99}{2} = 4950$ clés !

Historiquement, ces deux problèmes ont été plus ou moins gérés de la façon suivante :

- Peu d'utilisateurs (diplomates, chefs d'états, généraux)
- Dissémination physique (ambassadeurs, agents du chiffre)
- Registres de clés / livre-codes (cf télégramme Zimmermann)

# Cryptographie moderne

- Aujourd'hui avec internet et les cartes bleues il serait impensable de ne faire que de la cryptographie symétriques.
- Pour certains systèmes rares et critiques, la cryptographie symétrique seule reste envisageable, les clés symétriques sont soit pré-placées dans des modules sécurisés, soit transportées via des injecteurs de clés (ex: EKMS 308)

Module sécurisé = HSM (hardware security module) contient une mémoire et un cœur cryptographique censé garantir la sécurité du stockage et de l'éxécution des opérations.

C'est une sécurité logique et physique (résistance aux sondes, à l'ouverture du boîtier, etc.) garantie par des normes du type Critères Communs (ISO 15408) à 7 niveaux (EAL 1 à 7)

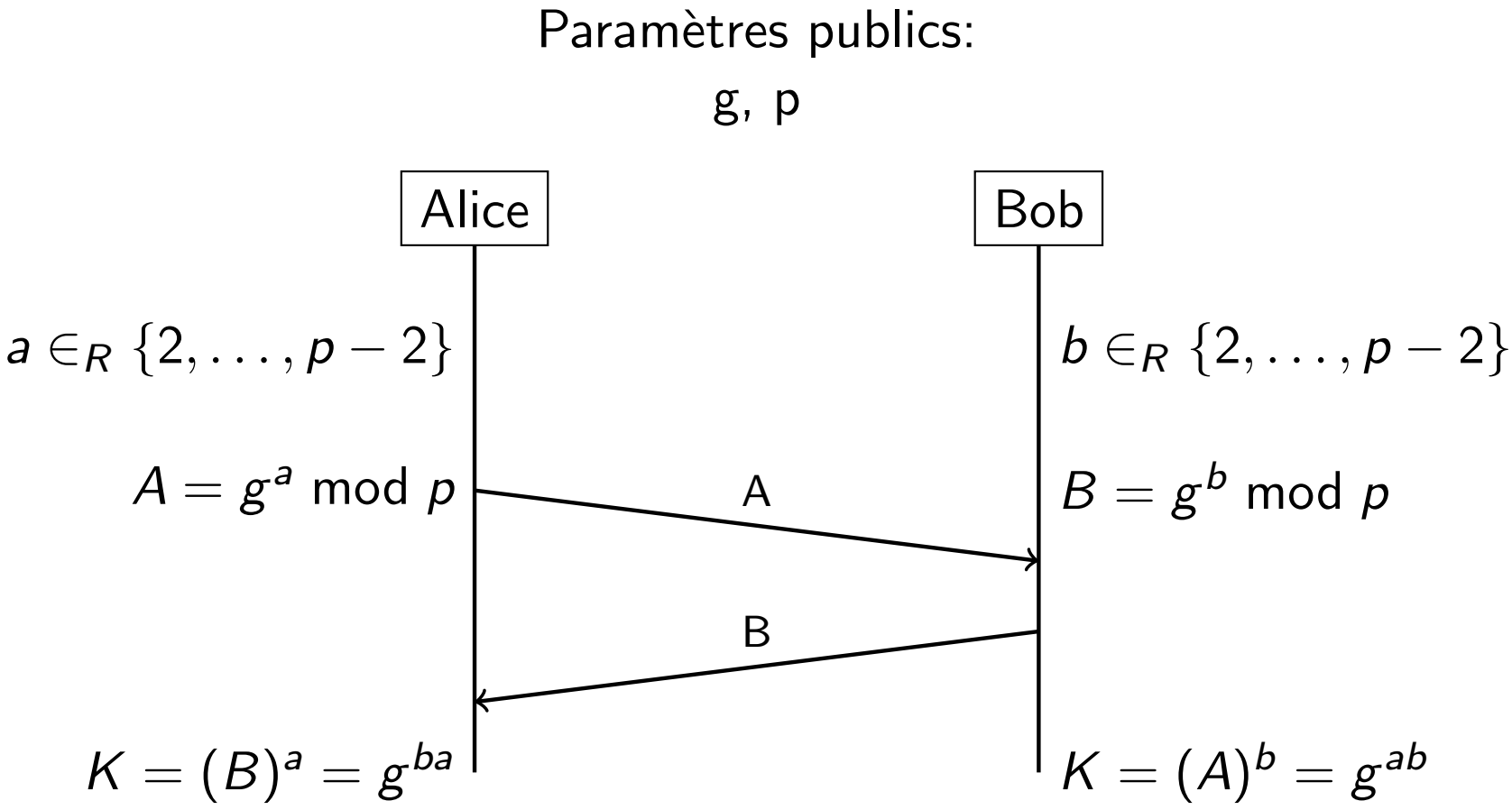
# Exemples de produits cryptographiques


Injecteur de clés DTD-II
(Thales Germany)


HSM Proteccio (Atos)


HSM Luna (Thales)

# Le protocole de Diffie-Hellman (1976)



# La cryptographie asymétrique

Chaque interlocuteur possède une paire de clé (privée, publique) et on dispose de protocoles permettant :

- à Alice de chiffrer $m$ avec la clé publique de Bob pour que Bob retrouve $m$ via sa clé secrète (KEM= Key Encapsulation Mechanism)
- à Alice et Bob de négocier une clé secrète comme dans Diffie-Hellman (KEX = Key EXchange)
- à Alice de signer un message avec sa clé secrète, Bob pouvant authentifier le message avec la clé publique d'Alice.

Pour que ça fonctionne il faut :

- qu'il soit facile de chiffrer ou de vérifier l'authenticité avec la clé publique (efficacité)
- mais qu'il soit impossible de signer ou de déchiffrer sans la clé secrète ! (sécurité)

# Fonctions à sens unique

## Fonction à sens unique

Une fonction calculable en temps polynomial $f : \{0,1\}^* \to \{0,1\}^*$ est dite **à sens unique** si pour tout algorithme polunomial probabiliste $A$, il existe une fonction négligeable $\epsilon : \mathbb{N} \to [0,1]$ telle que pour tout $n$ on ait:

$$\mathbb{P}_{x \in \{0,1\}^n, y = f(x)}[A(y) = x' \text{ tq } f(x') = y] < \epsilon(n)$$

(Source : Wikipedia)

En d'autres termes, il n'existe pas d'algorithme polynomial permettant de calculer des antécédents de $f$ de manière fiable.

# Exemples de fonctions à sens unique (ou presque)

- Etant donné $N = p \times q$, calculer $p$ et $q$ (produit vs factorisation)
- Etant donnés $g$ et $g^x$, calculer $x$ (exponentiation vs logarithme discret)
- Etant donné des équations quadratiques, trouver une solution de ces équations (évaluation vs résolution)

## Exemple pratique

En une seconde, un ordinateur portable des années 2015 peut

- Calculer $N = p \times q$ avec $p$ et $q$ des entiers à 30 millions de chiffres
- Factoriser un entier $N$ (retrouver $p$ et $q$) ayant 42 chiffres

Aujourd'hui **personne** n'a jamais factorisé un nombre à plus de 260 chiffres.

# Factorisation record (2020)

RSA-250 $= p \times q =$
21403246502407449612644230728393335630086147151447550177977
54920881418023447140136643345519095804679610992851872470914
58768739626192155736304745477052080511905649310668769159001
97594056934574522305893259766974716817380693648946998715784
94975937497937,
$p =$
64135289477071580278790190170577389084825014742943447208116
85963202453234463023862359875266834770873766192558569463979
8853367,
$q =$
33372027594978156556226010605355114227940760344767554666784
52098702384172921003708025744867329688187756571898625803693
2062711.

**Ce calcul durerait plus de 2830 années sur un seul cœur.**

# Un peu de cryptanalyse...

Logarithme discret : $G$ un groupe de taille $q$, $g$ un générateur. Connaissant $g$ et $g^x$ on veut calculer $x$. Comment faire ?

- Essayer tous les $x \rightsquigarrow$ complexité $O(q)$ opérations
- Calculer $g^y$ pour plein de $y$ pris au hasard $\rightsquigarrow$ $O(\sqrt{q})$ (question : pourquoi ?)

Pour certains groupes $G$ (courbes elliptiques), on ne sait pas faire mieux que $O(\sqrt{\#G})$ en termes de complexité asymptotique.

**Question :** pourquoi les courbes elliptiques standard ont des noms comme SECp256 ou Brainpool384 ?

**Attention :** on ne sait pas faire mieux à l'heure actuelle, ça ne prouve pas que c'est impossible !

# RSA (Rivest, Shamir, Adleman)

Alice veut envoyer un message chiffré à Bob.

## Génération de clé

Bob choisit deux nombres premiers $p$ et $q$ et calcule $n = p \times q$.
Bob calcule $\phi(n) = (p-1)(q-1)$.
Bob choisit un $e < \phi(n)$ premier à $\phi(n)$ (i.e. $\mathrm{pgcd}(e, \phi(n)) = 1$)
Bob calcule $d$ l'inverse de $e$ modulo $\phi(n)$, i.e. $de = 1 \bmod \phi(n)$.
La clé publique de Bob est $(n, e)$ et sa clé secrète $d$.

**Remarque:** le choix de $p$ et $q$ doit venir d'un aléa cryptographique car ils sont secrets. En revanche $e$ est public donc son choix importe peu.

# RSA (Rivest, Shamir, Adleman)

## Chiffrement

Alice cherche dans l'annuaire la clé publique de Bob $(n, e)$.
Elle calcule le chiffré $c = m^e \bmod n$ qu'elle envoie à Bob.

## Déchiffrement

Bob reçoit $c$ et utilise sa clé secrète pour calculer $c^d \bmod n$.
Comme $c^d \bmod n = m^{ed} \bmod n = m \bmod n$, Bob obtient bien le message $m$.

# Pourquoi le chiffrement est-il correct ?

**Petit théorème de Fermat:**
Si $p$ premier ne divise pas $m$, alors $m^{p-1} = 1 \bmod p$.

Puisque $ed = 1 + k(p-1)(q-1)$, on a $m^{ed} = m \times m^{k(p-1)(q-1)}$.
En appliquant le théorème à $p$ puis à $q$ on obtient

$$\begin{cases} m^{ed} = m \bmod p \\ m^{ed} = m \bmod q \end{cases}$$

Comme $n = pq$ et $p \neq q$, d'après le théorème des restes chinois on a $m^{ed} = m \bmod n$.

Le chiffrement est donc correct : si Alice envoie le chiffré de $m$ à Bob, alors Bob obtiendra bien $m$ en déchiffrant.

Il est efficace car tout repose sur des opérations modulaires rapides.

# Le chiffrement est-il sûr ?

Autrement dit, peut-on trouver $m$ à partir de $m^e$ ?

Actuellement on connaît seulement trois moyens :

- Les attaques par force brute (infaisables pour $p$ et $q$ assez grands)
- La factorisation de $n$ : si je calcule $p$ et $q$ alors j'en déduis $\phi(n)$ puis $d$
- Des algorithmes quantiques comme celui de Shor (mais il faut avoir un ordinateur quantique)

**Remarque :** $\phi(n)$ doit rester secret car $e$ est public donc il est facile de calculer $d = e^{-1} \bmod \phi(n)$.
Comment ? Via l'algorithme d'Euclide on calcule $(u, v)$ tq $ue + v\phi(n) = 1$, $d = u \bmod \phi(n)$.

# Factorisation

Plusieurs méthodes utilisées conjointement en pratique selon les cas :

- Force brute : tester $n$ mod $p$ pour tout $p$ premier $\leq \sqrt{n}$, complexité $O(\sqrt{n})$
- Algorithme de Fermat : efficace si $p - q$ est petit
- Algorithme ECM (Elliptic Curve Method) : efficace pour trouver des petits facteurs ($p$ petit)
- Crible algébrique (NFS en anglais) : méthode dont la complexité est sous-exponentielle $O\left(e^{c \log(n)^{1/3} \log\log(n)^{2/3}}\right)$

**Remarques :**

- $e^{c \log n} = n^c$ est exponentiel, d'où le nom de sous-exponentiel
- Un entier $n$ se représente avec $\log n$ bits, donc $n$ est bien exponentiel (en $\log n$) et pas polynomial !

# Choisir des paramètres sûrs

**Choisir une taille suffisante** en fonction des attaques connues.
Exemples :

- Le meilleur algorithme pour calculer des logarithmes discrets sur une courbe elliptique est en $O(\sqrt{n}) \rightsquigarrow n \simeq 2^{256}$
- Sur $\mathbb{F}_p$, on sait faire mieux car dispose d'un algorithme sous-exponentiel en $p \rightsquigarrow p \simeq 2^{2048}$

**Cela ne suffit pas toujours**, l'instance doit être robuste.

- Pour le logarithme discret, le groupe doit avoir un cardinal premier "ou presque" (Pohlig-Hellman)
- Certaines courbes elliptiques sont vulnérables
- Dans certains groupes le logarithme discret est facile : par exemple dans $(\mathbb{Z}/n\mathbb{Z}, +)$, DLP = relation de Bézout

# Documents normatifs

- NIST SP-800 56, Key Exchange using Discrete Logarithm
- BSI TR-02102-1, Recommendations and Key length
- BSI TR-03111, Elliptic Curve Cryptography
- NIST FIPS 203 & 204, Post-quantum KEM

## Contenu de ces standards

Ces documents décrivent précisément les algorithmes et la manière de les implémenter de façon sécurisée et interopérable (par exemple dans quel ordre lire les octets/bits), quels paramètres choisir pour quel niveau de sécurité, etc.

Le NIST fournit par ailleurs des séquences entrée/sortie permettant de tester ses implémentations (NIST CAVP).

# Authentification

# Pourquoi authentifier ?

**Attaque Man-in-the-middle :**

- Eve se fait passer pour Bob auprès d'Alice
- Eve se fait passer pour Alice auprès de Bob
- Eve négocie des clés secrètes $K_A$ et $K_B$ auprès d'Alice et Bob
- Alice chiffre son message avec $K_A$
- Eve le déchiffre et le renvoie à Bob chiffré avec $K_B$
- Et réciproquement de Bob vers Alice

**Conséquence :** Eve intercepte tous les messages et ni Alice ni Bob ne se rendent compte de rien.

**Contre-mesure :** il faut un mécanisme permettant à Alice d'être la seule à pouvoir revendiquer son identité. C'est l'authentification.

# Intégrité : authentification symétrique

## Message authentication code (MAC)

Un code d'authentification de message (MAC) est un algorithme qui calcule une valeur de taille fixe appelée MAC (ou tag) à partir d'un message de taille quelconque et d'une clé secrète.

- Le MAC/tag est public et ne révèle rien sur le message
- La clé secrète doit être partagée entre les deux parties (et elle est secrète bien sûr)
- Idée : un MAC utilise souvent des fonctions de hachage pour hacher une expression qui combine le message et la clé.

**Limites :** il faut la clé secrète pour vérifier l'intégrité.
De plus, si Alice, Bob et Charlie ont la même clé et qu'Alice reçoit un message avec un tag correct, elle ne peut pas être sûre que c'est Bob et non Charlie l'auteur du message.

# Signature : authentification asymétrique

## Signature numérique

Un algorithme de signature calcule une valeur de taille fixe appelée signature à partir d'un message de taille quelconque et de la clé privée du signataire. Pour vérifier la validité de la signature, le destinataire utilise la clé publique du signataire.

- Alice signe un message avec sa clé secrète
- Bob reçoit un message signé par Alice, il récupère la clé publique d'Alice dans l'annuaire
- Bob peut vérifier que le message est bien authentique (il vient d'Alice et personne ne l'a modifié par la suite)

**Question :** et si je mets ma clé publique dans l'annuaire en prétendant m'appeler Alice ?

# Créer un canal sécurisé

Pour exploiter les avantages du symétrique et de l'asymétrique :

- Alice et Bob font un échange de clés authentifié (asymétrique)
  - Echange de clés : à la fin du protocole, Alice et Bob obtiennent une clé secrète commune
  - Authentifié : à la fin du protocole Alice et Bob sont certains de se parler l'un à l'autre
- Alice et Bob utilisent la clé secrète en question dans des algorithmes symétriques pour avoir des échanges chiffrés (par exemple par AES) et authentifiés (par un MAC)

Prochain chapitre :

- Un exemple de protocole d'échange de clés authentifié, TLS
- La difficulté de garantir l'authenticité : le concept de PKI

# Le protocole TLS

# Le protocole SSL/TLS

De 1995 à nos jours : protocole SSL (Secure Sockets Layer) puis TLS (Transport Layer Security), utilisé dans HTTPS.

- Trois versions de SSL 1.0, 2.0 et 3.0 (RFC 6101)
- Quatre versions de TLS 1.0 (RFC 2246) puis 1.1, 1.2 et 1.3 (RFC 4346)

Fin 2024, 99.9% des serveurs HTTPS sont compatibles avec TLS 1.2 et 66.9% avec TLS 1.3

# La sécurité dans TLS

Un protocole tel que TLS est utilisé au niveau mondial par de nombreux terminaux différents. Il doit donc être flexible. Il permet par exemple :

- Authentification :
  - asymétrique (RSA, DSS, ECDSA)
  - symétrique (clé ou mot de passe)
  - pas d'authentification
- Échange de clés :
  - RSA ou Diffie Hellman (DH, ECDH) avec clé statique
  - clé secrète ou mot de passe
  - Diffie-Hellman avec clés éphémères

**Question :** pourquoi préférer Diffie-Hellman avec des clés éphémères selon vous ?

# Les primitives cryptographiques dans TLS

Liste non-exhaustive des options proposées par TLS :

- Chiffrement :
  - par blocs : AES, DES, 3DES
  - par blocs authentifiés : AES-CCM, AES-GCM
  - par flot : RC4
  - mode sans chiffrement
- Intégrité :
  - HMAC-SHA1, HMAC-SHA256
  - AEAD (intégrité dans le chiffrement)

Le choix d'une combinaison de toutes ces primitives est appelée une cipher suite, les deux parties (client et serveur) doivent évidemment se mettre d'accord sur la même cipher suite.

# Comment établir une connexion SSL/TLS ?

Un client veut établir une connexion sécurisée avec un serveur (exemples : banque en ligne, ameli, ...)

Il faut :

- Se mettre d'accord sur une cipher suite
- Authentifier le serveur (est-ce vraiment ma banque ?)
- Échanger des clés secrètes pour sécuriser les messages

C'est ce qu'on appelle un **handshake**.

# Première partie : Client/Server Hello

**Étape 1 :** Le client envoie $\mathrm{ClientHello}$ : la plus haute version du protocole qu'il supporte, les cipher suite et les algorithmes de compression qu'il supporte.

**Étape 2 :** Le serveur répond avec

- $\mathrm{ServerHello}$ version du protocole, cipher suite et compression choisis par le serveur
- $\mathrm{Certificate}$ la clé publique du serveur pour assurer l'authenticité (vide si pas d'authentification)
- $\mathrm{ServerKeyExchange}$ une clé publique du serveur pour échanger des clés via Diffie-Hellman (vide si pas de DH)
- $\mathrm{ServerHelloDone}$

# Seconde partie :

**Étape 3 :** le client envoie

- ClientKeyExchange soit un secret symétrique protégé par la clé publique du serveur, soit une clé publique Diffie-Hellman
- ChangeCipherSpec marque la fin du handshake côté client, et donc le passage à la cryptographie symétrique (si handshake réussi)
- Finished message chiffré et authentifié contenant un MAC des messages précédents

**Étape 4 :** Si le Finished du client est valide, le serveur envoie

- ChangeCipherSpec marque la fin du handshake côté serveur, et donc le passage à la cryptographie symétrique
- Finished message chiffré et authentifié contenant un MAC des messages précédents

# Remarques :

- Si le $\mathrm{Finished}$ du serveur est valide (vérifié par le client) alors la connexion est établie
- Ce message a pour but d'associer le contexte (notamment la Cipher suite) à la session : si quelqu'un essaie de se faire passer pour le serveur ou de modifier a posteriori la ciphersuite, le $\mathrm{Finished}$ qui en résulte est invalide.
- Dans le handshake, le serveur envoie sa propre clé publique. Est-ce raisonnable ?
- Comment faire pour authentifier la clé publique du serveur ?

# Authentifier le serveur

- Option 1 : la clé publique est signée par le serveur lui-même
  Cela ne règle pas le problème...
- Option 2 : la clé publique est signé par un tiers (CA : certificate authority)
  Comment vérifier cette signature ?
- Option 3: le serveur envoie sa clé signée par la CA ainsi que la clé publique de la CA.

**Question :** quid de l'authenticité de la clé publique de la CA ?

Pas de miracle : à un moment il faut faire confiance à quelqu'un.
En réalité on progresse un peu : il y a beaucoup moins de CA que de serveurs donc on donne sa confiance à un plus petit nombre de personnes.

# Les infrastructures à clés publiques (PKI)

Deux possibilités : décentralisée (Web of trust) ou hiérarchique via des chaînes de CA.

**Pour le cas hiérarchique, différents niveaux :**

- CA racines : certifient les clés publiques d'autres CA
- CA intermédiaires : certifient les clés publiques des serveurs (plus rarement d'autres CA)

Un client va donc vérifier le certificat du serveur, puis de la CA qui l'a signé, puis de CA' qui a signé le certificat de CA, et ainsi de suite jusqu'à la vérification de la signature émanant de la CA racine. Si on fait confiance à la CA racine, on peut faire confiance au serveur.

# En pratique

Les navigateurs ont une liste de CA racines de confiance (un peu moins d'une centaine)... mais encore faut-il faire confiance aux navigateurs !

Les CA sont auditées et on peut révoquer les certificats qui sont compromis via des mécanismes comme CRL (Certificate Revocation List) ou OCSP (Online Certificate Status Protocol).

In fine, on fait confiance à l'infrastructure, ce n'est finalement pas si différent de ce qui se passe pour nos documents d'identité (passeport, permis, carte d'identité).