Chapitre 2 : Chiffrements Symétriques
Stream Ciphers & Block Ciphers

Cours de Cryptographie

12 janvier 2026

Table des matières

1 Introduction

1.1 Motivations

Le chapitre précédent a montré que la sécurité parfaite nécessite des clés aussi longues que les messages ($|| \geq ||$). Cette contrainte est rédhibitoire en pratique.

Objectif de ce chapitre : Construire des systèmes de chiffrement avec :

— Clés **courtes** et **réutilisables**

— Sécurité basée sur la **complexité computationnelle**

— Efficacité pratique (chiffrement rapide)

Prix à payer : Abandon de la sécurité parfaite au profit de la **sécurité computationnelle**.

1.2 Plan du chapitre

1. **Sécurité computationnelle** : Définitions, modèle de l'adversaire
2. **Pseudorandom Generators (PRG)** : Étendre une courte clé en un long pad
3. **Stream Ciphers** : Construction à partir de PRG
4. **Block Ciphers** : Permutations pseudoaléatoires (PRP)
5. **Modes opératoires** : ECB, CBC, CTR, OFB
6. **Sécurité CPA** (Chosen Plaintext Attack)
7. **AES** : Standard moderne de chiffrement par blocs

2 Sécurité computationnelle

2.1 Principe

Au lieu d'exiger qu'un adversaire ne puisse **jamais** casser le système (sécurité parfaite), on exige qu'il ne puisse pas le casser en **temps raisonnable**.

Définition

Sécurité Computationnelle
Un système est **computationnellement sécurisé** si tout adversaire efficace (temps polynomial) a une probabilité négligeable de succès.
Formalisation :

— Adversaire limité à 2^{80} opérations (minimum)

— Probabilité de casser $< 2^{-80}$ (négligeable)

— Paramètre de sécurité λ : clé de longueur λ bits

2.2 Hypothèses cryptographiques

La sécurité repose sur des **hypothèses** (non prouvées) :

— Certains problèmes sont difficiles (ex : factorisation, log discret)

— $\mathcal{P} \neq \mathcal{NP}$ (conjecture centrale)

— Existence de fonctions à sens unique

— Existence de générateurs pseudoaléatoires

3 Pseudorandom Generators (PRG)

3.1 Définition informelle

Un PRG étend une courte graine (seed) aléatoire en une longue séquence indistinguable de l'aléatoire.

Définition

Pseudorandom Generator (PRG)
Un PRG est une fonction déterministe $G : \{0,1\}^\lambda \to \{0,1\}^n$ avec $n \gg \lambda$ telle que :

1. **Expansion** : $n > \lambda$ (typiquement $n = \lambda \cdot c$ pour c constant)
2. **Efficacité** : G est calculable en temps polynomial
3. **Pseudoaléatoire** : $G(s)$ (avec $s \xleftarrow{\$} \{0,1\}^\lambda$) est indistinguable d'une chaîne uniforme $r \xleftarrow{\$} \{0,1\}^n$ pour tout adversaire efficace

3.2 Jeu d'indistinguabilité PRG

Théorème

Jeu PRG-IND
Challenger choisit $b \xleftarrow{\$} \{0,1\}$
if $b = 0$ **then**
$s \xleftarrow{\$} \{0,1\}^\lambda$, envoie $y = G(s)$ à
else
$r \xleftarrow{\$} \{0,1\}^n$, envoie $y = r$ à
end if
retourne $b' \in \{0,1\}$
gagne si $b' = b$

Avantage de l'adversaire :

$$\mathsf{Adv}_G^{\mathrm{PRG}}() = \left|\Pr[\text{ gagne}] - \frac{1}{2}\right|$$

G est un PRG sécurisé si $\mathsf{Adv}_G^{\mathrm{PRG}}() \leq \epsilon$ négligeable pour tout efficace.

3.3 Exemples de PRG

3.3.1 Linear Congruential Generators (LCG) - INSÉCURISÉS !

Construction :

$$x_{n+1} = (a \cdot x_n + c) \bmod m$$

Avertissement

DANGEREUX : Ne JAMAIS utiliser pour la cryptographie !
LCG est prédictible : connaissant quelques sorties successives, on peut retrouver tous les paramètres et prédire les sorties futures. Attaque connue depuis les années 1980.
Usage acceptable : Simulations Monte-Carlo, jeux vidéo (non-crypto)

3.3.2 Blum-Blum-Shub (BBS)

Construction :

— Choisir deux grands premiers $p, q \equiv 3 \pmod 4$

— $n = pq$

— Seed : $x_0 \in \mathbb{Z}_n^*$

— Itération : $x_{i+1} = x_i^2 \bmod n$

— Output : bit de poids faible de x_i

Sécurité : Prouvé sécurisé sous l'hypothèse de difficulté de la factorisation (problème de résidus quadratiques).

Limitation : Très lent (une exponentiation modulaire par bit généré).

3.3.3 ChaCha20 (Recommandé)

Construction : Stream cipher basé sur la fonction de permutation ChaCha (variante de Salsa20).

Propriétés :

— **Très rapide** : ∼3-4 cycles/byte sur processeurs modernes

— Résistant aux timing attacks (pas de lookups dépendant des données)

— Utilisé dans TLS 1.3, WireGuard VPN, Google (HTTPS)

— Paramètres : clé 256 bits, nonce 96 bits, compteur 32 bits

3.3.4 AES-CTR mode

Construction : Utiliser AES comme PRG en mode compteur (voir section 6.3).

Propriétés :

— Très rapide avec accélération matérielle (AES-NI)

— Standard industriel

— Parallélisable

3.3.5 DRBG (Deterministic Random Bit Generator)

Standards NIST :

— **Hash_DRBG** : Basé sur SHA-256, SHA-512

— **HMAC_DRBG** : Basé sur HMAC

— **CTR_DRBG** : Basé sur AES-CTR

Utilisés pour génération de clés, nonces, IVs dans standards cryptographiques (TLS, SSH, etc.).

4 Stream Ciphers

4.1 Construction à partir de PRG

Définition

Stream Cipher depuis PRG
Soit $G : \{0,1\}^\lambda \to \{0,1\}^n$ un PRG. On définit :

- $= \{0,1\}^\lambda$: espace des clés
- $== \{0,1\}^n$: messages et ciphertexts
- $\mathsf{Enc}_k(m) = m \oplus G(k)$
- $\mathsf{Dec}_k(c) = c \oplus G(k)$

Intuition : Remplacer le OTP avec pad vraiment aléatoire par un pad pseudoaléatoire $G(k)$.

4.2 Sécurité

Théorème 4.1. *Si G est un PRG sécurisé, alors le stream cipher construit ci-dessus est sémantiquement sécurisé contre les attaques passives.*

Idée. Réduction : Si un adversaire distingue les chiffrements, alors on peut construire un adversaire $\mathcal{B}$ qui distingue $G(s)$ de l'aléatoire, contredisant la sécurité de G. (Preuve formelle détaillée dans exercices) □

4.3 Limitations

Avertissement

Stream ciphers : Pas de réutilisation de clé !
Comme le OTP, les stream ciphers ne doivent **JAMAIS** réutiliser la même clé k pour chiffrer deux messages différents :

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

Solution : Utiliser des nonces (numbers used once) - voir modes opératoires.

5 Block Ciphers

5.1 Abstraction : Permutation pseudoaléatoire (PRP)

Un block cipher est une permutation paramétrée par une clé.

Définition

Block Cipher (PRP)
Un block cipher sur $\{0,1\}^n$ est une famille de permutations $\{E_k : \{0,1\}^n \to \{0,1\}^n\}_{k\in}$ telle que :

1. E_k est une permutation pour tout k
2. E_k et E_k^{-1} sont calculables efficacement
3. E_k est indistinguable d'une permutation aléatoire (PRP security)

Notation :

— n : taille du bloc (ex : 128 bits pour AES)
— λ : taille de la clé (ex : 128, 192, 256 bits)
— $E_k(m)$: chiffrement du bloc m
— $E_k^{-1}(c)$: déchiffrement du bloc c

5.2 Jeu PRP-IND

Théorème

Jeu PRP-IND (Indistinguabilité PRP vs Permutation Aléatoire)

Challenger choisit $b \xleftarrow{\$} \{0,1\}$
if $b = 0$ **then**
$k \xleftarrow{\$}$, donne accès à l'oracle $E_k(\cdot)$ à
else
$\pi \xleftarrow{\$} \mathrm{Perm}(\{0,1\}^n)$ (permutation aléatoire), donne accès à $\pi(\cdot)$ à
end if
fait jusqu'à q requêtes à l'oracle
retourne $b' \in \{0,1\}$
gagne si $b' = b$

Avantage de l'adversaire :

$$\mathsf{Adv}_E^{\mathrm{PRP}}() = \left|\Pr[\text{ gagne}] - \frac{1}{2}\right|$$

E est une PRP sécurisée si $\mathsf{Adv}_E^{\mathrm{PRP}}() \leq \epsilon$ négligeable pour tout efficace.

Remarque 5.1. ***PRP vs PRF***
*Une **PRF** (Pseudorandom Function) n'exige pas que F_k soit une permutation (peut être une fonction quelconque).*
***PRP Switching Lemma** : Si E est utilisé pour chiffrer au plus q blocs avec $q \ll 2^{n/2}$, alors PRP $\approx$ PRF (différence négligeable).*
***Conséquence** : Pour AES ($n = 128$), tant que $q < 2^{64}$ blocs chiffrés avec la même clé, on peut traiter AES comme une PRF.*

6 Modes opératoires

Un block cipher chiffre des blocs de taille fixe (ex : 128 bits). Pour chiffrer des messages longs, on utilise des **modes opératoires**.

6.1 Mode ECB (Electronic Codebook) - INSÉCURISÉ

Principe : Chiffrer chaque bloc indépendamment.

$$c_i = E_k(m_i) \quad \text{pour } i = 1, \ldots, t$$

Avertissement

ECB est DANGEREUSEMENT INSÉCURISÉ
Problème : Blocs identiques → ciphertexts identiques. Révèle la structure du message !
Exemple célèbre : Chiffrer une image avec ECB conserve les contours visibles (voir notebook).

6.2 Mode CBC (Cipher Block Chaining)

Principe : Chaîner les blocs avec XOR.

Algorithm 1 CBC Encryption

Require : Message $m = m_1 \| m_2 \| \cdots \| m_t$, clé k, IV aléatoire
  $c_0 \leftarrow$ IV (Initialization Vector)
  for $i = 1$ to t **do**
    $c_i \leftarrow E_k(m_i \oplus c_{i-1})$
  end for
  return $c = c_0 \| c_1 \| \cdots \| c_t$

Déchiffrement CBC :

$$m_i = D_k(c_i) \oplus c_{i-1}$$

Propriété : CBC est CPA-sécurisé si IV est aléatoire et unique.

6.3 Mode CTR (Counter)

Principe : Transformer un block cipher en stream cipher.

$$c_i = m_i \oplus E_k(\text{nonce} \| i)$$

Avantages :

— Parallélisable (contrairement à CBC)
— Pas besoin de padding
— Accès aléatoire aux blocs
— CPA-sécurisé avec nonce unique

6.4 Mode OFB (Output Feedback)

Principe : Générer un keystream en chaînant les sorties du block cipher.
Déchiffrement OFB : Identique (régénérer le même keystream, XOR avec ciphertext)
Propriétés :

— Le keystream est généré **indépendamment** du message (contrairement à CBC)
— Peut pré-calculer le keystream avant de recevoir le message
— Pas besoin de padding (comme CTR)
— **Non parallélisable** (contrairement à CTR)
— Erreur de transmission : affecte seulement le bit correspondant (self-synchronizing)

Algorithm 2 OFB Encryption

Require : Message $m = m_1 \| m_2 \| \cdots \| m_t$, clé k, IV aléatoire

```
I_0 ← IV
for i = 1 to t do
    I_i ← E_k(I_{i−1})          // Générer keystream indépendamment du message
    c_i ← m_i ⊕ I_i
end for
return c = IV‖c_1‖···‖c_t
```

Sécurité : CPA-sécurisé si IV est unique et aléatoire.

Avertissement

Attention : Cycle du keystream
Si $I_i = I_j$ pour $i \neq j$, le keystream se répète ! Avec taille de bloc n bits, cycle attendu : $\approx 2^{n/2}$ blocs (paradoxe des anniversaires).
Conséquence : Pour AES-128 ($n = 128$), ne pas chiffrer plus de 2^{64} blocs avec la même clé.

Usage moderne : Rare (CTR préféré car parallélisable)

6.5 Comparaison des modes

Mode	CPA-sécurisé	Parallélisable	Padding	Erreur propagation	Usage
ECB	NON	Oui	Non	Limitée (1 bloc)	JAMAIS
CBC	Oui (IV aléat.)	Déchiff. seul.	Oui	Propagée (1 bloc)	Legacy
CTR	Oui (nonce uniq.)	Oui	Non	Limitée (1 bit)	Recommandé
OFB	Oui (IV uniq.)	Non	Non	Limitée (1 bit)	Rare

Recommandations pratiques :

- **Chiffrement moderne** : Utiliser CTR ou modes AEAD (AES-GCM, ChaCha20-Poly1305)
- **Legacy systems** : CBC acceptable si IV vraiment aléatoire + MAC séparé
- **ECB** : Ne JAMAIS utiliser (sauf cas très spécifiques comme wrapping de clés courtes)
- **OFB** : Remplacé par CTR dans pratiquement tous les usages

7 Advanced Encryption Standard (AES)

7.1 Historique

- **1997** : NIST lance concours pour remplacer DES
- **2000** : Rijndael (Daemen & Rijmen) sélectionné
- **2001** : Standardisé comme AES (FIPS 197)
- **Aujourd'hui** : Standard mondial de facto

7.2 Paramètres

- **Taille de bloc** : 128 bits (fixe)
- **Tailles de clé** : 128, 192, ou 256 bits
- **Nombre de rounds** : 10 (AES-128), 12 (AES-192), 14 (AES-256)

7.3 Structure (aperçu)

AES est un **Substitution-Permutation Network (SPN)** avec 4 opérations par round :

1. **SubBytes** : Substitution non-linéaire (S-box)
2. **ShiftRows** : Permutation des lignes de la matrice d'état
3. **MixColumns** : Transformation linéaire des colonnes
4. **AddRoundKey** : XOR avec la sous-clé du round

7.4 Détails mathématiques

7.4.1 Corps de Galois GF(2^8)

AES opère dans le corps fini GF(2^8) : 256 éléments représentés par des octets.
Représentation : Un octet $b_7b_6 \cdots b_1b_0$ représente le polynôme :

$$b_7x^7 + b_6x^6 + \cdots + b_1x + b_0 \in \mathbb{F}_2[x]$$

Opérations :

— **Addition** : XOR bit-à-bit

— **Multiplication** : Multiplication de polynômes modulo le polynôme irréductible $m(x) = x^8 + x^4 + x^3 + x + 1$ (0x11B en hexadécimal)

Exemple : $\{53\} \cdot \{CA\}$ en hexadécimal

$$\{53\} = x^6 + x^4 + x + 1$$
$$\{CA\} = x^7 + x^6 + x^3 + x$$
$$\{53\} \cdot \{CA\} = \ldots \bmod m(x) = \{01\}$$

7.4.2 S-Box (Substitution Box)

Construction en 2 étapes :
1. Inversion dans GF(2^8) :

$$b \mapsto b^{-1} \quad \text{(avec convention } 0^{-1} = 0)$$

2. Transformation affine :

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

où $c = (01100011)_2 = 0x63$.
Propriétés :

— Non-linéarité forte (résistance aux attaques linéaires et différentielles)

— Pas de points fixes : $S(a) \neq a$ pour $a \neq 0x52$

— Inversible : S^{-1} existe (pour déchiffrement)

7.4.3 MixColumns

Opération : Multiplication matricielle dans GF(2^8) pour chaque colonne de l'état.
Pour une colonne $[s_0, s_1, s_2, s_3]^T$, calculer :

$$\begin{bmatrix} s'_0 \\ s'_1 \\ s'_2 \\ s'_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

où les coefficients sont dans GF(2^8).
Propriété : Cette matrice est MDS (Maximum Distance Separable) : garantit diffusion optimale.

7.4.4 Key Schedule

Objectif : Dériver les round keys $k_0, k_1, \ldots, k_{N_r}$ depuis la clé maître.
Algorithme (pour AES-128, clé de 128 bits = 4 mots de 32 bits) :

- k_0 = clé maître
- Pour chaque round i :
  - Prendre dernier mot de k_{i-1}
  - **RotWord** : rotation circulaire d'un octet
  - **SubWord** : appliquer S-box à chaque octet
  - XOR avec constante de round $\text{Rcon}[i] = (x^{i-1}, 0, 0, 0)$ dans $\text{GF}(2^8)$
  - XOR avec premier mot de k_{i-1} pour obtenir premier mot de k_i
  - Générer autres mots de k_i par XOR successifs

Résultat : 11 round keys pour AES-128 (176 octets total)

7.5 Sécurité d'AES

- Aucune attaque pratique connue sur AES complet
- Meilleure attaque théorique : 2^{126} pour AES-128 (négligeable)
- AES-256 résistant aux ordinateurs quantiques (algorithme de Grover)
- Vulnérabilités possibles : implémentations (side-channels, timing attacks)

8 Sécurité CPA (Chosen Plaintext Attack)

8.1 Modèle de l'adversaire

Définition

CPA (Chosen Plaintext Attack)
L'adversaire peut :

- Choisir des plaintexts $m_1, m_2, \ldots, m_q$ de son choix
- Obtenir les ciphertexts correspondants $c_i = \mathsf{Enc}_k(m_i)$
- Tenter de briser le système (distinguer chiffrements, retrouver clé, etc.)

Contexte pratique : Attaquant peut injecter des messages dans un système (ex : email chiffré)

8.2 Jeu CPA-IND

Théorème

Jeu d'indistinguabilité CPA (IND-CPA)

Challenger génère $k \xleftarrow{\$}$
Phase 1 : interroge oracle $\mathsf{Enc}_k(\cdot)$ sur $m_1, \ldots, m_q$ (obtient $c_1, \ldots, c_q$)
Challenge : envoie m_0^*, m_1^* avec $|m_0^*| = |m_1^*|$
Challenger choisit $b \xleftarrow{\$} \{0,1\}$, envoie $c^* = \mathsf{Enc}_k(m_b^*)$
Phase 2 : continue à interroger $\mathsf{Enc}_k(\cdot)$ (sauf sur m_0^*, m_1^*)
retourne $b' \in \{0,1\}$

Avantage :

$$\mathsf{Adv}_{\Pi}^{\mathrm{CPA}}() = \left|\Pr[b' = b] - \frac{1}{2}\right|$$

Un système est **CPA-sécurisé** si $\mathsf{Adv}_{\Pi}^{\mathrm{CPA}}() \leq \epsilon$ négligeable.

8.3 Théorèmes de sécurité

Théorème 8.1. *CBC avec IV aléatoire est CPA-sécurisé si le block cipher sous-jacent est une PRP sécurisée.*

Théorème 8.2. *CTR avec nonce unique est CPA-sécurisé si le block cipher sous-jacent est une PRF sécurisée.*

Idée de la preuve pour CTR. Réduction par l'absurde. Supposons qu'un adversaire distingue les chiffrements CTR. On construit un adversaire $\mathcal{B}$ qui distingue le block cipher E_k d'une fonction aléatoire.

Étapes :

1. $\mathcal{B}$ reçoit accès à un oracle $\mathcal{O}$ qui est soit $E_k(\cdot)$ (clé k aléatoire), soit $f(\cdot)$ (fonction aléatoire)
2. $\mathcal{B}$ simule le jeu IND-CPA pour :
   — Pour chiffrer m, $\mathcal{B}$ choisit nonce aléatoire, interroge $\mathcal{O}(\text{nonce}\|i)$ pour chaque bloc, fait $c_i = m_i \oplus \mathcal{O}(\text{nonce}\|i)$
3. $\mathcal{B}$ retourne la même guess que

Analyse :

— Si $\mathcal{O} = E_k$, alors $\mathcal{B}$ simule parfaitement CTR avec E_k
— Si $\mathcal{O} = f$ aléatoire, alors le chiffrement est un OTP parfait (car $f(\text{nonce}\|i)$ est vraiment aléatoire pour chaque i)
— Donc $\mathsf{Adv}_{\mathrm{CTR}}^{\mathrm{CPA}}() \leq \mathsf{Adv}_{E}^{\mathrm{PRF}}(\mathcal{B})$

Si E est une PRF sécurisée, alors $\mathsf{Adv}_{E}^{\mathrm{PRF}}(\mathcal{B})$ est négligeable, donc CTR est CPA-sécurisé. □ □

Idée de la preuve pour CBC. Similaire à CTR mais plus technique car CBC n'est pas parallélisable.

Point clé : L'IV doit être **imprévisible** (aléatoire), pas seulement unique !

Contre-exemple : Si IV est prévisible (ex : compteur), CBC n'est PAS CPA-sécurisé.

Attaque : Adversaire demande chiffrement de $m_1 = 0^n$, obtient (IV_1, c_1). Ensuite, prédit IV_2 (car compteur), demande challenge sur $m_0^* = 0^n$ et $m_1^* = IV_1 \oplus IV_2$. Peut distinguer facilement !

Preuve complète : Voir Bellare-Rogaway "Introduction to Modern Cryptography" ou Katz-Lindell Chapitre 3. □ □

Remarque 8.1. ***Importance de la randomisation***

Tous les schémas CPA-sécurisés nécessitent de la randomisation (IV, nonce) pour chaque message. Sinon le chiffrement est ***déterministe*** *et donc pas CPA-sécurisé (adversaire chiffre lui-même les messages challenges et compare).*

9 Notebooks pratiques

Les notebooks suivants illustrent les concepts :

— $02_demo_stream_cipher.ipynb$: $StreamcipheravecChaCha20$

- $02_demo_aes_modes.ipynb$: $ComparaisonECB, CBC, CTR$
- $02_demo_cpa_attack.ipynb$: $AttaqueCPAsurECBmode$
- $02_exercices.ipynb$: $Exercicesguids$

10 Exercices

10.1 Exercices théoriques

1. Montrer que ECB n'est pas CPA-sécurisé
2. Prouver que CBC avec IV prévisible n'est pas CPA-sécurisé
3. Analyser la sécurité de CTR avec compteur qui part de 0 à chaque message
4. Démontrer la réduction PRG → stream cipher sécurisé

10.2 Exercices pratiques

Voir notebooks.

11 Conclusion

Ce chapitre a introduit les chiffrements symétriques modernes :

— **Sécurité computationnelle** : Compromis pratique vs sécurité parfaite
— **PRG** : Fondation théorique des stream ciphers
— **Block ciphers** : Permutations pseudoaléatoires (AES)
— **Modes opératoires** : ECB (insécurisé), CBC, CTR (recommandé)
— **Sécurité CPA** : Modèle d'attaque réaliste

Limitation importante : Les chiffrements de ce chapitre assurent seulement la **confidentialité**, pas l'**intégrité**. Un attaquant peut modifier les ciphertexts !

Le chapitre suivant introduira les **MAC** et l'**authenticated encryption** pour garantir à la fois confidentialité et intégrité.