Chapitre 4 : Cryptographie à Clé Publique Diffie-Hellman, RSA, ElGamal

Cours de Cryptographie

12 janvier 2026

Table des matières

1 Introduction : Le problème de la distribution des clés

Problème fondamental : La cryptographie symétrique nécessite une clé partagée secrète. Comment Alice et Bob établissent-ils une clé commune sans canal sécurisé préalable ?

Solution révolutionnaire (1976) : Diffie et Hellman inventent la cryptographie à clé publique.

Paradigme :

— Chaque utilisateur a une paire (pk, sk) : clé publique + clé secrète

— pk est publique (annuaire)

— sk est gardée secrète

— Chiffrement avec pk, déchiffrement avec sk

2 Fondations mathématiques

2.1 Groupes cycliques

Groupe cyclique
Un groupe $(G, \cdot)$ d'ordre q est cyclique s'il existe un générateur g tel que :

$$G = \{g^0, g^1, g^2, \ldots, g^{q-1}\}$$

Exemples :

— $(\mathbb{Z}_p^*, \cdot)$: entiers modulo p premier (ordre $p-1$)

— Courbes elliptiques sur corps finis

2.2 Problème du logarithme discret (DLP)

Discrete Logarithm Problem (DLP)
Entrée : G, g, h où $h \in G$
Problème : Trouver x tel que $g^x = h$
Hypothèse : Aucun algorithme efficace ne résout DLP pour des groupes bien choisis.

Groupes sécurisés :

— $\mathbb{Z}_p^*$ avec p premier de 2048-4096 bits

— Courbes elliptiques (Curve25519, secp256k1) avec 256 bits

2.3 Problème Diffie-Hellman (CDH, DDH)

Le protocole Diffie-Hellman repose sur deux hypothèses de difficulté distinctes.

2.3.1 Computational Diffie-Hellman (CDH)

CDH Problem
Entrée : G, g, g^a, g^b où $a, b \xleftarrow{\$} \mathbb{Z}_q$
Problème : Calculer g^{ab}
Hypothèse CDH : Aucun algorithme efficace ne peut résoudre CDH avec probabilité non-négligeable.

Relation avec DLP : Si on peut résoudre DLP (trouver a depuis g^a), on peut résoudre CDH. Donc : DLP $\leq$ CDH (DLP au moins aussi difficile).

Usage : CDH suffit pour la sécurité passive (eavesdropping) de DH.

2.3.2 Decisional Diffie-Hellman (DDH)

DDH Problem
Jeu : Distinguer les deux distributions suivantes :

- **Distribution réelle** : (g, g^a, g^b, g^{ab}) où $a, b \xleftarrow{\$} \mathbb{Z}_q$
- **Distribution aléatoire** : (g, g^a, g^b, g^c) où $a, b, c \xleftarrow{\$} \mathbb{Z}_q$ (indépendants)

Hypothèse DDH : Aucun algorithme efficace ne peut distinguer ces deux distributions avec avantage non-négligeable.

Formellement : Pour tout adversaire $\mathcal{A}$ efficace,

$$\left|\Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1]\right| \leq \epsilon$$

où ϵ est négligeable.

Relation entre hypothèses :

$$\text{DLP} \Rightarrow \text{CDH} \Rightarrow \text{DDH}$$

- Si on résout DLP, on résout CDH
- Si on résout CDH, on résout DDH (tester si $Z = g^{ab}$ en calculant g^{ab})
- **Réciproques inconnues** : On ne sait pas si CDH $\Rightarrow$ DLP ou DDH $\Rightarrow$ CDH

Importance de DDH : Nécessaire pour prouver la sécurité CPA de ElGamal et de Diffie-Hellman key exchange.

2.3.3 Groupes où DDH est facile

Attention : DDH ne tient PAS dans tous les groupes !
Exemple : $\mathbb{Z}_p^*$ avec p premier
DDH est facile via le **symbole de Legendre** : Pour p premier impair, on peut tester si un élément est un carré modulo p en temps polynomial.

Attaque : Si g est un générateur de $\mathbb{Z}_p^*$, alors :

- g^{ab} est un carré si et seulement si ab est pair
- g^c est un carré avec probabilité $1/2$ (si c aléatoire)

Donc on peut distinguer (g^a, g^b, g^{ab}) de (g^a, g^b, g^c) avec probabilité $\approx 3/4$.

Solution : Travailler dans un sous-groupe d'ordre premier q de $\mathbb{Z}_p^*$ (avec $p = 2q + 1$, nombre premier de Sophie Germain).

Groupes sûrs pour DDH :

- Sous-groupes premiers de $\mathbb{Z}_p^*$ (avec p choisi correctement)
- Courbes elliptiques sur corps finis (Curve25519, NIST P-256, etc.)

3 Diffie-Hellman Key Exchange

3.1 Protocole

Diffie-Hellman **Paramètres publics** : Groupe G d'ordre q, générateur g **Alice** : Choisit $a \xleftarrow{\$} \mathbb{Z}_q$, calcule $A = g^a$, envoie A **Bob** : Choisit $b \xleftarrow{\$} \mathbb{Z}_q$, calcule $B = g^b$, envoie B **Alice** : Calcule $K = B^a = g^{ab}$ **Bob** : Calcule $K = A^b = g^{ab}$ **Clé partagée** : $K = g^{ab}$

Sécurité : Basée sur l'hypothèse DDH (Decisional Diffie-Hellman)

3.2 Attaque Man-in-the-Middle

Problème : DH seul n'authentifie pas les participants !
Attaque :

— Eve intercepte $A = g^a$ d'Alice

— Eve envoie $E = g^e$ à Bob (se faisant passer pour Alice)

— Eve partage $K_1 = g^{ae}$ avec Alice et $K_2 = g^{eb}$ avec Bob

— Eve déchiffre et rechiffre tous les messages

Solution : Authenticated DH (avec signatures ou certificats)

4 Chiffrement à clé publique

4.1 Définition

Public Key Encryption
Algorithmes :

— $\mathsf{Gen}() \to (pk, sk)$: Génération de paire de clés

— $\mathsf{Enc}(pk, m) \to c$: Chiffrement avec clé publique

— $\mathsf{Dec}(sk, c) \to m$: Déchiffrement avec clé secrète

Correction : $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m$

4.2 Sécurité CPA pour chiffrement asymétrique

Différence avec symétrique : L'adversaire a accès à pk et peut donc chiffrer lui-même !
Jeu IND-CPA : Similaire au cas symétrique mais $\mathcal{A}$ connaît pk.

5 Chiffrement ElGamal

5.1 Construction

ElGamal Encryption
Paramètres : Groupe G d'ordre q, générateur g
Gen :
- Choisir $x \xleftarrow{\$} \mathbb{Z}_q$
- $pk = g^x$, $sk = x$

Enc$(pk, m \in G)$:
- Choisir $r \xleftarrow{\$} \mathbb{Z}_q$
- $c_1 = g^r$, $c_2 = m \cdot (pk)^r = m \cdot g^{xr}$
- Retourner (c_1, c_2)

Dec$(sk, (c_1, c_2))$:
- Calculer $s = c_1^{sk} = g^{rx}$
- Retourner $m = c_2/s = (m \cdot g^{xr})/g^{xr}$

Sécurité : IND-CPA sous hypothèse DDH
Limitation : Chiffre seulement des éléments de G (pas directement des bitstrings)

6 RSA

6.1 Arithmétique modulaire

Prérequis :
- Théorème d'Euler : $a^{\phi(n)} \equiv 1 \pmod{n}$
- Si $n = pq$ (produit de deux premiers), $\phi(n) = (p-1)(q-1)$
- Identité de Bézout, algorithme d'Euclide étendu

6.2 Chiffrement RSA (textbook - INSÉCURISÉ)

RSA Textbook (ne PAS utiliser !)
Gen :
- Choisir deux grands premiers p, q (ex : 1024 bits chacun)
- $n = pq$, $\phi(n) = (p-1)(q-1)$
- Choisir e tel que $\gcd(e, \phi(n)) = 1$ (souvent $e = 65537$)
- Calculer $d = e^{-1} \bmod \phi(n)$
- $pk = (n, e)$, $sk = (n, d)$

Enc(pk, m) : $c = m^e \bmod n$
Dec(sk, c) : $m = c^d \bmod n$
Correction : $c^d = (m^e)^d = m^{ed} = m^{1+k\phi(n)} = m \cdot (m^{\phi(n)})^k = m \pmod{n}$

AVERTISSEMENT : RSA textbook est **déterministe** donc PAS CPA-sécurisé !

6.3 RSA-OAEP (sécurisé)

Optimal Asymmetric Encryption Padding (Bellare-Rogaway 1994)

Principe : Ajouter du padding randomisé avant chiffrement RSA pour obtenir la sécurité CPA.

6.3.1 Construction OAEP

Paramètres :

— k : taille du modulus RSA (en bits, ex : 2048)

— k_0 : taille du paramètre aléatoire (ex : 256 bits)

— k_1 : taille de la redondance (ex : 128 bits)

— $G : \{0,1\}^{k_0} \to \{0,1\}^{n-k_0}$: fonction de hachage (oracle aléatoire)

— $H : \{0,1\}^{n-k_0} \to \{0,1\}^{k_0}$: fonction de hachage (oracle aléatoire)

— $n = k - k_1$: longueur du bloc OAEP

Algorithme OAEP-Encode(m) :

1. Choisir $r \xleftarrow{\$} \{0,1\}^{k_0}$ (randomness)
2. Calculer $s = (m \| 0^{k_1}) \oplus G(r)$ (masquer le message)
3. Calculer $t = r \oplus H(s)$ (masquer la randomness)
4. Retourner $s \| t$

Chiffrement RSA-OAEP(pk, m) :

1. $\hat{m} \leftarrow$ OAEP-Encode(m)
2. $c \leftarrow (\hat{m})^e \bmod n$ (RSA textbook sur le bloc paddé)
3. Retourner c

Déchiffrement RSA-OAEP(sk, c) :

1. $\hat{m} \leftarrow c^d \bmod n$ (RSA textbook)
2. Parser $\hat{m} = s \| t$ (découper en deux parties)
3. $r \leftarrow t \oplus H(s)$ (retrouver randomness)
4. $m \| \text{pad} \leftarrow s \oplus G(r)$ (retrouver message)
5. Vérifier que pad $= 0^{k_1}$ (sinon retourner $\perp$)
6. Retourner m

6.3.2 Propriétés de sécurité

Théorème 6.1 (Bellare-Rogaway 1994)**.** *Si RSA est une permutation à sens unique (OWF) et G, H sont des oracles aléatoires, alors RSA-OAEP est IND-CPA sécurisé.*

Modèle de l'oracle aléatoire : Hypothèse forte qui modélise les fonctions de hachage comme des fonctions vraiment aléatoires. Preuve ne tient pas dans le modèle standard, mais pratique acceptable.

Intuition de la sécurité :

— La randomness r assure que le chiffrement est **probabiliste** (même message $\Rightarrow$ ciphertexts différents)

— Le double masquage (Feistel network) assure que même si on connaît m, on ne peut pas prédire $\hat{m}$ sans connaître r

— La redondance 0^{k_1} empêche les manipulations du ciphertext (non-malléabilité partielle)

6.3.3 Standards et implémentation

PKCS#1 v2.2 (RFC 8017) spécifie RSA-OAEP avec :

— G et H basés sur MGF1 (Mask Generation Function) avec SHA-256 ou SHA-512

— Label optionnel L (souvent vide) pour lier le chiffrement à un contexte

— Tailles recommandées : $k_0 = 256$ bits (taille de SHA-256), $k_1 = 256$ bits

Taille maximale du message : Pour RSA-2048 avec SHA-256,

$$|m|_{\max} = k - 2k_0 - k_1 - 8 = 2048 - 2 \cdot 256 - 256 - 8 = 1272 \text{ bits} = 159 \text{ octets}$$

Usage pratique : Pour chiffrer des messages longs, utiliser **chiffrement hybride** :

1. Générer clé symétrique aléatoire k_{AES}
2. Chiffrer message avec AES-GCM : $c_{\text{sym}} = \text{AES-GCM}_{k_{\text{AES}}}(m)$
3. Chiffrer clé avec RSA-OAEP : $c_{\text{key}} = \text{RSA-OAEP}(pk, k_{\text{AES}})$
4. Transmettre $(c_{\text{key}}, c_{\text{sym}})$

Alternatives modernes : ECIES (Elliptic Curve Integrated Encryption Scheme) plus efficace que RSA-OAEP

7 Signatures numériques

7.1 Définition

Digital Signature
Algorithmes :

— $\mathsf{Gen}() \to (pk, sk)$

— $\mathsf{Sign}(sk, m) \to \sigma$: Signature

— $\mathsf{Vrfy}(pk, m, \sigma) \to \{0, 1\}$: Vérification

Sécurité : UF-CMA (Unforgeability under Chosen Message Attack)

7.2 RSA Signatures (avec hachage)

Sign(sk, m) : $\sigma = H(m)^d \bmod n$

Vrfy(pk, m, σ) : Vérifier $\sigma^e \stackrel{?}{=} H(m) \pmod n$

7.3 DSA (Digital Signature Algorithm)

Standard NIST (FIPS 186) basé sur le logarithme discret.

7.3.1 Construction DSA

Paramètres globaux :

— p : grand nombre premier (2048 ou 3072 bits)

— q : nombre premier divisant $p - 1$ (256 bits typiquement)

— g : générateur d'un sous-groupe d'ordre q dans $\mathbb{Z}_p^*$

— H : fonction de hachage (SHA-256)

Gen(1^λ) :

— Choisir $x \xleftarrow{\$} \mathbb{Z}_q$

— Calculer $y = g^x \bmod p$

— $pk = y$, $sk = x$

Sign(sk, m) :

1. Choisir $k \xleftarrow{\$} \mathbb{Z}_q^*$ (nonce, DOIT être aléatoire et unique !)
2. Calculer $r = (g^k \bmod p) \bmod q$
3. Calculer $s = k^{-1} \cdot (H(m) + x \cdot r) \bmod q$
4. Retourner $\sigma = (r, s)$

Vrfy(pk, m, σ) :

1. Parser $\sigma = (r, s)$
2. Vérifier $0 < r < q$ et $0 < s < q$ (sinon rejeter)
3. Calculer $w = s^{-1} \bmod q$
4. Calculer $u_1 = H(m) \cdot w \bmod q$ et $u_2 = r \cdot w \bmod q$
5. Calculer $v = (g^{u_1} \cdot y^{u_2} \bmod p) \bmod q$
6. Accepter si et seulement si $v = r$

Correction : Vérifier que

$$v = (g^{H(m)\cdot w} \cdot g^{xr\cdot w} \bmod p) \bmod q = (g^k \bmod p) \bmod q = r$$

car $w = s^{-1} = k \cdot (H(m) + xr)^{-1}$.

CRITIQUE : Gestion du nonce k

Le nonce k doit être :

— **Vraiment aléatoire** (pas pseudo-aléatoire prévisible)

— **Unique** pour chaque signature

— **Secret** (jamais révélé)

Attaque si nonce réutilisé : Si deux signatures (r_1, s_1) et (r_2, s_2) utilisent le même k :

— $r_1 = r_2$ (même g^k)

— De $s_1 - s_2 = k^{-1}(H(m_1) - H(m_2))$, on retrouve k

— De $s_1 = k^{-1}(H(m_1) + xr_1)$, on retrouve la clé secrète x !

Cas réels : PlayStation 3 (2010), Android Bitcoin wallets (2013).

7.3.2 ECDSA (Elliptic Curve DSA)

Variante de DSA sur courbes elliptiques. Même algorithme mais :

— Groupe : Points d'une courbe elliptique $E(\mathbb{F}_p)$ d'ordre premier q

— Générateur : Point de base G sur la courbe

— Clé publique : $Y = x \cdot G$ (multiplication scalaire)

— Signature : Même formule (r, s) mais avec opérations sur courbe

Avantages :

— Clés plus courtes : 256 bits (ECDSA) $\approx$ 3072 bits (DSA)

— Plus rapide (génération et vérification)

— Même niveau de sécurité avec moins de ressources

Courbes standard :

— **NIST P-256, P-384, P-521** : Standards FIPS, utilisés dans TLS

— **secp256k1** : Utilisée dans Bitcoin, Ethereum

— **Curve25519** : Moderne, résistante aux side-channels (mais pour ECDH, pas signatures)

Usage : TLS (certificats), Bitcoin (transactions), SSH, Git (commits signés)

7.4 EdDSA (moderne)

Ed25519 : Standard moderne, très rapide, résistant aux side-channels

8 Courbes elliptiques

8.1 Principe

Groupe : Points d'une courbe elliptique $y^2 = x^3 + ax + b$ sur un corps fini
Avantages :

— Clés plus courtes (256 bits ECC $\approx$ 3072 bits RSA)

— Opérations plus rapides

— Meilleure résistance aux attaques quantiques (Grover seulement, pas Shor pour log discret)

Courbes standard :

— Curve25519 (X25519 key exchange, Ed25519 signatures)

— secp256k1 (Bitcoin, Ethereum)

— NIST P-256, P-384, P-521

9 Notebooks pratiques

— $04_demo_diffie_hellman.ipynb : changedeclsDH$

- $04_demo_elgamal.ipynb : ChiffrementElGamal$

- $04_demo_rsa.ipynb : RSA(chiffrementOAEP + signatures)$

- $04_demo_ecdsa.ipynb : Courbeselliptiques(ECDH, ECDSA, Ed25519)$

- $04_exercices.ipynb : Exercicesguids$

10 Conclusion

Points clés :

— Cryptographie à clé publique résout la distribution des clés

— Basée sur problèmes difficiles (DLP, factorisation)

— Diffie-Hellman : échange de clés

— ElGamal, RSA : chiffrement asymétrique (avec padding !)

— DSA, RSA-PSS, EdDSA : signatures numériques

— Courbes elliptiques : efficaces et modernes

Le chapitre suivant explore les applications à la **communication anonyme** (Tor, mixnets).