Cryptographie

Clé, Confusion, Diffusion, Substitution, Transposition

Rappel rapide

- **Confusion** ⇔ obtenue principalement par les **substitutions non linéaires** (S-Boxes).
- **Diffusion** ⇔ obtenue par les **transpositions linéaires** (permutations, mélanges, XOR, rotations).
- **La clé**

  Elle est indispensable : sans clé, il n’y a pas de secret.

  Mais seule, elle ne suffit pas : il faut que l’algorithme exploite la clé efficacement.

  Une clé forte sans confusion/diffusion → **faible sécurité** (structure exploitable, analyse fréquentielle).

  La clé = **source du secret**, mais pas garante de la sécurité structurelle.

Clé, Confusion, Diffusion, Substitution, Transposition

Substitution

- C’est le cœur de la confusion : elles introduisent la non-linéarité.
- Sans non-linéarité, le chiffrement se réduit à une suite de transformations linéaires → cassable (cryptanalyse) par des méthodes algébriques (équations, matrices).
- Exemples : S-Box de DES, S-Box d’AES.
- Les substitutions sont vitales : elles empêchent de prédire la relation entre clair et chiffré.

Transposition (Permutation, Diffusion)

- Assure la diffusion.
- En mélangeant les bits ou les positions des caractères, elle fait en sorte qu’un petit changement (clair ou clé) se propage partout.
- Exemples : ShiftRows + MixColumns dans AES, P-Box dans DES.
- La transposition seule n’apporte pas de sécurité : c’est linéaire. Mais elle est indispensable pour amplifier la confusion.

Clé, Confusion, Diffusion, Substitution, Transposition

Conclusion

- La clé est la condition nécessaire → elle introduit le secret.
- La substitution (non-linéarité) est la plus importante pour la sécurité cryptographique : sans elle, tout chiffrement est vulnérable.
- La transposition (diffusion) est complémentaire : seule, elle ne protège pas, mais sans elle la confusion reste locale et insuffisante.

Donc :

- La clé = secret.
- La substitution = cœur de la sécurité.
- La transposition = garantit et renforce la robustesse.
- En résumé : les trois sont indispensables, mais les substitutions (non-linéarité) sont la partie plus importante de la cryptographique.

DES

DES

- Type : chiffrement par blocs symétrique.
- Taille du bloc : 64 bits.
- Clé effective : 56 bits (la clé est donnée sur 64 bits, mais 8 bits sont utilisés pour la parité).
- Structure : réseau de Feistel, avec 16 tours.
- But : transformer un texte clair en texte chiffré de façon réversible (déchiffrement avec la même clé).

Étapes principales de DES

1. Initial Permutation (IP)
   - Les 64 bits du texte clair sont réarrangés selon une table fixe.
   - Cette permutation n'ajoute pas de sécurité en soi, mais prépare les données pour la structure de Feistel.
2. Division en deux moitiés
   - Après l'IP, on sépare le bloc en deux parties :
     - Gauche : L_0 32) bits)
     - Droite : R_0 32) bits)

DES

3. Les 16 tours de Feistel
   - Chaque tour i ($1 \leq i \leq 16$) applique la règle :

$$L_i = R_{i-1}$$

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

```
Entrée :  L0        R0
          |         |
          |         +--> f(R0, K1)
          |               |
          |               v
          |          XOR (⊕)
          |               |
          v               v
Sortie :  R0      L0 ⊕ f(R0, K1)
```

   - K_i = sous-clé de 48 bits générée à partir de la clé principale.
   - f = fonction de chiffrement non linéaire :
     - Expansion : R_{i-1} (32 bits) → 48 bits (en dupliquant certains bits).
     - Mélange avec la sous-clé : XOR avec K_i.
     - Substitution (Confusion) : découpe en 8 blocs de 6 bits → passage par 8 S-boxes → 32 bits en sortie.
     - Permutation (Diffusion) : réarrangement des 32 bits.
   - C'est dans ces S-boxes que réside la non-linéarité essentielle de DES (confusion).

DES

4. Permutation finale (IP^{-1})
    - Après les 16 tours, on inverse la permutation initiale (IP^{-1}).
    - On obtient ainsi le texte chiffré de 64 bits.

Génération des sous-clés (Key Schedule)
- La clé de 64 bits → 56 bits (on enlève les bits de parité).
- Ces 56 bits sont divisés en deux moitiés (28 + 28).
- À chaque tour : décalages circulaires + sélection selon une table → sous-clé de 48 bits K_i.
- Total : 16 sous-clés $K_1, K_2, \ldots, K_{16}$.

Faiblesses :

- Clé de 56 bits → trop courte aujourd'hui : brute-force possible en quelques heures avec du matériel moderne.

3DES

2-clé 3DES (112 bits de sécurité)

- On utilise deux clés K_1 et K_2.
- L'algorithme est :
  - $C = E_{K_1}\left(D_{K_2}\left(E_{K_1}(M)\right)\right)$
  - On chiffre avec K_1,
  - on déchiffre avec K_2,
  - puis on rechiffre avec K_1.
- Déchiffrement au milieu → Pour assurer la compatibilité descendante avec du simple DES (si $K_1 = K_2$,ça équivaut à un seul DES).

3-clé 3DES (168 bits de sécurité théorique)

- On utilise trois clés différentes K_1, K_2, K_3:
  - $C = E_{K_3}\left(D_{K_2}\left(E_{K_1}(M)\right)\right)$
- Sécurité effective ≈ **112 bits** (et pas 168) à cause de l'attaque dite *Meet-in-the-Middle*.

Meet-in-the-middle

Meet-in-the-Middle

- Un attaquant qui connaît un couple (P, C) peut faire
- Quand un chiffrement applique deux (ou plusieurs) étapes de chiffrement indépendantes (p. ex. $C = E_{K_2}\left(E_{K_1}(M)\right)$,(on peut « couper » l'opération au milieu :
  - calculer toutes les valeurs intermédiaires possibles $X = E_{K_1}(M)$ pour tous les K_1 ,
  - stocker les résultats dans une table (2^{56} entrées)
  - calculer toutes les valeurs intermédiaires possibles $Y = D_{K_2}(C)$ pour tous les K_2
  - chercher des correspondances $X = Y$. Une correspondance donne une paire candidate (K_1, K_2).
    - → paire de clés candidates trouvée
      → vérification avec un autre couple (P,C)
- C'est un principe de temps / mémoire : on remplace une recherche exhaustive sur l'espace produit $K_1 \times K_2$ par deux recherches de taille $|\ K\ |$ chacune plus une recherche par table.
- Calculs $\approx 2^{56} + 2^{56} \approx 2^{57}$

AES – meilleur algorithme symétrique

AES (Advanced Encryption Standard) est le standard actuel recommandé par le NIST (USA) et utilisé mondialement.

- C'est un chiffrement par blocs de 128 bits, avec des clés de 128, 192 ou 256 bits.
- AES-128 est déjà très sûr, AES-256 est utilisé pour les applications nécessitant un niveau de sécurité maximal (ex. militaires, gouvernements).

- Très robuste : aucune attaque pratique ne permet aujourd'hui de casser AES.
- Rapide : accélération matérielle (instructions AES-NI dans les processeurs modernes).
- Standardisé et interopérable : utilisé dans TLS, VPN (IPsec, OpenVPN), disques chiffrés (BitLocker, VeraCrypt), etc.
- Flexibilité : utilisé en différents modes (CBC, GCM, CTR, etc.) selon le besoin.

Groupe Abélien ou "commutatif"

– Un ensemble abstrait G sur lequel on a défini une opération abstraite « $\otimes$ » avec certaines propriétés
  - élément identité : $\exists\ 1 \in G$, t.q. $\forall a \in G$, $a \otimes 1 = a$
  - Associativité : $\forall a, b, c \in G$, $a \otimes (b \otimes c) = (a \otimes b) \otimes c$,
  - Tout éléments à un inverse : $\forall a \in G$, $\exists\ a^{-1}$ t.q. $a \otimes a^{-1} = 1$
  - (Commutativité): $\forall a, b \in G$, $a \otimes b = b \otimes a$
    on dit alors que le groupe est "abélien" ou "commutatif"

1. $(\mathbb{Z}, +)$
   - Groupe des entiers avec l'addition.
   - Neutre : 0.
   - Inverse de $n = -n$.
   - C'est un groupe abélien infini.
2. $(\mathbb{Z}_n, +)$ (entier module n)
   - Exemple : $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$ avec addition modulo 6.
   - Neutre : 0.
   - Inverse de $k = n - k$.
   - C'est un groupe abélien fini.

Groupe cyclique

Un **groupe cyclique** est un groupe G qui peut être engendré par un seul élément g.
Tout **groupe cyclique** est **abélien** (mais l'inverse n'est pas vrai).

$$G = \langle g \rangle = \{g^k \mid k \in \mathbb{Z}\}$$

notation multiplicative

$$G = \langle g \rangle = \{kg \mid k \in \mathbb{Z}\}$$

notation additive

Exemples

- (Z6,+) : cyclique d'ordre 6, générateurs = 1, 5.
- $(\mathbb{Z}_8, +)$: générateurs = 1, 3, 5, 7.

Arithmétique modulaire

- S'applique aux nombres entiers non-négatifs seulement
- a modulo b est le reste entier de la division de a par b
- Deux entiers sont équivalents modulo n si leurs modules sont égaux,

$$x \equiv_n y \text{ si et seulement si } (x \bmod n) = (y \bmod n)$$

- Propriétés
  - Associativité: (a + (b + c)) mod n = ((a + b) + c) mod n
    (a * (b * c)) mod n = ((a * b) * c) mod n
  - Commutativité: (a + b) mod n = (b + a) mod n
    (a * b) mod n = (b * a) mod n
  - Distributivité: (a * (b + c)) mod n = ((a * b) + (a * c)) mod n
  - Existence d'identités: (a + 0)mod n = (0 + a) mod n = a
    (a *1) mod n = (1 * a) mod n = a
  - Existence d'inverses: (a + -a) mod n =0
    $(a * a^{-1})$ mod n = 1 si pgcd(a,n)=1

Le plus grand commun diviseur (pgcd) de deux nombres est le plus grand entier qui divise ces deux nombres

Petit Théorème de Fermat

Petit théorème de Fermat (résultat fondamental en en cryptographie - RSA)

- Soit p un nombre premier et a un entier **non divisible par** p. Alors :
  - $a^{p-1} \equiv 1 \pmod{p}$

  Version plus générale (valable pour tout entier a) :

  - $a^p \equiv a \pmod{p}$

Exemples

- $p = 7, a = 3$

$$3^6 = 729 \equiv 1 \pmod{7}.$$

- $p = 5, a = 2$

$$2^4 = 16 \equiv 1 \pmod{5}.$$

- $p = 11, a = 10$

$$10^{10} \equiv 1 \pmod{11}.$$

Petit Théorème de Fermat

Théorème d'Euler

Soit $n \in \mathbb{N}$ et $a \in \mathbb{Z}$ tel que pgcd(a, n) =1.
Alors :

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

où $\varphi(n)$ est la fonction indicatrice d'Euler, c'est-à-dire le nombre d'entiers compris entre 1 et n qui sont premiers avec n.

Exemple

- Soit $n = 10$, donc $\varphi(10) = 4$, il y 4 nombres <10 qui sont premiers avec 10 $\{1, 3, 7, 9\}$
- Prenons $a = 3$.

$$3^4 = 81 \equiv 1 \pmod{10}.$$

- Cas particulier
  - Si $n = p$ est premier, alors $\varphi(p) = p - 1$.
    On retrouve le **Petit Théorème de Fermat** :

$$a^{p-1} \equiv 1 \pmod p$$

Petit Théorème de Fermat

Exemple pratique (cryptographie)

Calculer $7^{222} \pmod{13}$.

Comme $p = 13, \varphi(13) = 12$ (Théorème d'Euler – p est premier)

Théorème Fermat : $7^{12} \equiv 1 \pmod{13}$ $(a^{p-1} \equiv 1 \pmod{p})$

- $222 = 18 \times 12 + 6$.
- $7^{222}(mod\ 13) = \left(7^{12}\right)^{18} \cdot 7^6 \pmod{13} \rightarrow 1^{18} \cdot 7^6 \equiv 12 \pmod{13}$
- $7^{222} \equiv 12 \pmod{13}$.

Petit Théorème de Fermat

Exemple : calculer 5^{1234} (mod 11)

Petit théorème de Fermat : $a^{p-1} \equiv 1 \pmod{p}$ dans cet exemple **,** $a = 5\ et\ p = 11$.

- Comme 11 est premier et $a = 5$, selon Fermat on a :
$$5^{10} \equiv 1 \pmod{11}$$
- Cela veut dire que **toute puissance de 5 dont l'exposant est un multiple de 10 vaut 1 modulo 11**
$$5^{10k} = \left(5^{10}\right)^k \equiv 1^k = 1 \pmod{11},\ \forall\ k \in \mathbb{N}.$$
- Donc, si on a un exposant énorme n, on peut écrire
$n = 10k + r.\ Pour\ n = 1234 = 123 * 10 + 4,$
$$5^n = 5^{10k+r} = (5^{10})^k \cdot 5^r = 5^{10*123+4} = \left(5^{10}\right)^k \cdot 5^r = (5^{10})^{123} * 5^4 = 1^{123} * 5^4 \equiv 5^4 \pmod{11}.$$

Car $(5^{10})^{123} \equiv 1^{123} = 1 \pmod{11}$

Donc, $5^n \equiv 5^r \pmod{11}$, si n = 10k + r → $5^{1234} \equiv 5^4 \pmod{11}$

Finalement, Calculer 5^4 (mod 11)

- $5^2 = 25 \equiv 25 - 2 \cdot 11 = 3 \pmod{11}$.
- $5^4 = \left(5^2\right)^2 \equiv 3^2 = 9 \pmod{11}$.

Petit Théorème de Fermat

En appliquant le **théorème de Fermat** (petit théorème) calculez 11^{4571} (mod 7)

On applique le petit théorème de Fermat avec $p = 7$ (p premier) et pgcd(11,7) =1 :

$$11^6 \equiv 1 \pmod 7.$$

On réduit l'exposant modulo 6 :

$$4571 = 6\times761+5 \Rightarrow 11^{4571} \equiv (11^6)^{761} * 11^5 \pmod 7.$$

Réduisons la base modulo 7 :

$$11 \equiv 4 \pmod 7.$$

Donc

$$11^5 \equiv 4^5 \pmod 7.$$

Calcule :

$4^2 = 16 \equiv 2, 4^4 \equiv 2^2 = 4, 4^5 = 4^4 \cdot 4 \equiv 4 \cdot 4 = 16 \equiv 2$ (mod7).

Petit Théorème de Fermat -utilisation

Cryptographie (RSA)

- Si $n = p \cdot q$ avec p, q premiers, alors $\varphi(n) = (p-1)(q-1)$.
- La clé publique e est choisie avec pgcd(e, $\varphi(n)$)=1.
  - Théorème d'Euler : Soit $n \in \mathbb{N}$ et $a \in \mathbb{Z}$ tel que pgcd(a, n) =1.

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

- La clé privée d est l'inverse de e modulo $\varphi(n)$.
- Le théorème d'Euler garantit que :

$$(M^e)^d \equiv M \pmod{n}.$$

Algorithme RSA

Fonctionnement

- Alice publie sa clé publique K_{pub}.
- Bob chiffre son message M pour envoyer à Alice

$$C = \text{Chiffrement}\left(M, K_{pub}\right)$$

- Alice déchiffre C avec sa clé privée K_{priv}

$$M = \text{Déchiffrement}\left(C, K_{priv}\right)$$

- Résultat : seul Alice peut lire le message, même si tout le monde connaît K_{pub}

Algorithme RSA

1. **Génération des clés**
   - Choisir deux nombres premiers p et q (p et q très grands).
   - Calculer n :

$$n = p \cdot q$$

   - Calculer l'indicatrice d'Euler :

$$\varphi(n) = (p-1)(q-1)$$

   - Choisir un entier e (clé publique) tel que :

$$1 < e < \varphi(n), \ \text{pgcd}(e, \varphi(n)) = 1$$

(souvent $e = 65537$ en pratique).

   - Calculer d (clé privée) → l'inverse modulaire de e modulo $\varphi(n)$:

$$e \cdot d \equiv 1 \pmod{\varphi(n)}$$

   - **Clé publique : (n, e) → à transmettre au besoin**
   - **Clé privée : (n, d) → jamais transmettre**

Algorithme RSA

2. Chiffrement

- Pour un message M (avec $0 \leq M < n$) :

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

3. Déchiffrement

- Pour retrouver le message :

$$M = C^d \pmod{n}$$

- Cela marche grâce au théorème d'Euler :

$$M^{ed} \equiv M \pmod{n}$$

RSA - Exemple

Étape 1 : Choisir deux nombres premiers p et q

- Exemple : $p = 17, q = 11$
- On calcule :
  - $n = p \cdot q = 17 \times 11 = 187$
  - $\varphi(n) = (p-1)(q-1) = 16 \times 10 = 160$ (le nombre d'entiers compris entre 1 et n qui sont premiers avec n, si p et q sont premiers)

Étape 2 : Choisir une clé publique (n, e)

- On choisit e tel que pgcd(e, $\varphi(n)$)=1.
- e = 7 (pgcd(7,160)=1)

Donc la clé publique est :

$$\mathbf{(n, e)} = (\mathbf{187}, \mathbf{7})$$

RSA - Exemple

Étape 3 : Calculer la clé privée d

- Il faut trouver d tel que : $e \cdot d \equiv 1 \pmod{\varphi(n)}$.
- $7d \equiv 1 \pmod{160}$.

En cherchant, on trouve :

- $\boldsymbol{d = 23}$, *car* $7 \times 23 = 161 \equiv 1 \pmod{160}$.
- Donc la **clé privée** est :

$$(\boldsymbol{n}, \boldsymbol{d}) = (\mathbf{187}, \mathbf{23})$$

RSA - Exemple

Étape 4 : Chiffrement

Supposons que Bob veuille envoyer un message à Alice

- Message = M = 88
- Clé publique d'Alice est : $(n, e) = (187\,, 7)$
- Le chiffrement RSA est :
  - $C = M^e \pmod{n} = 88^7 \pmod{187}$.

Calcul rapide (exponentiation modulaire) :

- $88^1 \equiv 88 \pmod{187}$
- $88^2 = 7744 \equiv 77 \pmod{187}$
- $88^4 = 77^2 = 5929 \equiv 33 \pmod{187}$
- $88^7 = 88^4 \cdot 88^2 \cdot 88 \equiv 33 \cdot 77 \cdot 88 \equiv 11 \pmod{187}$
- Le message chiffré est $C = 11$

- Bob envoie **11** à Alice.

RSA - Exemple

Étape 5 : Déchiffrement

Alice reçoit $C = 11$ (le message envoyé par Bob) et calcule :

- Clé privée d'Alice $(n, d) = (187, 23)$
- $M = C^d \pmod{n} = 11^{23} \pmod{187}$.

- Calcul par carrés successifs pour $11^{23} = 11^{16} \cdot 11^4 \cdot 11^2 \cdot 11$
  - $11^2 \equiv 121 \pmod{187}$
  - $11^4 \equiv 55 \pmod{187}$
  - $11^8 \equiv 33 \pmod{187}$
  - $11^{16} = 11^{4\,*\,4} \equiv 55^{\wedge 4} \equiv 154 \pmod{187}$
- Puis :
  - $11^{23} = 11^{16} \cdot 11^4 \cdot 11^2 \cdot 11$
  - $\equiv 154 \cdot 55 \cdot 121 \cdot 11 \equiv 88 \pmod{187}$
- On retrouve bien le message initial :

$$M = 88$$

Communication clé RSA publique

1. **La clé publique RSA**

- La clé publique n'est pas secrète → elle peut être envoyée librement.
- Dans une communication sécurisée (ex : HTTPS), elle est incluse dans un certificat numérique X.509 signé par une Autorité de Certification (CA).
- Elle est donc transmise au client (navigateur) quand on se connecte à un site web (ex : https://...).
- Le client utilise cette clé publique pour :
  - chiffrer une clé de session (dans TLS),
  - ou vérifier une signature du serveur.

2. **La clé privée RSA**

- La clé privée doit absolument rester secrète.
- Elle est stockée uniquement sur le serveur ou le périphérique qui l'utilise, par exemple :
  - Fichier protégé (server.key) sur un serveur web (Apache, Nginx, etc.), généralement chiffré par un mot de passe.
  - Stockage matériel sécurisé :
    - TPM (Trusted Platform Module) sur un ordinateur,
    - HSM (Hardware Security Module) pour les grandes entreprises/bancaires,
    - Smartcards ou tokens USB pour les utilisateurs.
  - Dans SSH, la clé privée de l'utilisateur est stockée dans ~/.ssh/id_rsa (protégée par un mot de passe).
- Jamais elle ne doit circuler sur le réseau.

Communication clé RSA publique

3. Communication RSA typique (HTTPS/TLS)
   - Le serveur envoie sa clé publique (dans le certificat).
   - Le client (navigateur) génère une clé de session symétrique (i.e. AES) et la chiffre avec la clé publique.
   - Le serveur déchiffre la clé de session avec sa clé privée RSA.
   - Toute la suite de la communication se fait avec la clé de session (cryptographie symétrique, plus rapide).

Clé RSA

La longueur de la clé publique RSA correspond en fait au grand nombre $n = p \times q$, produit de deux nombres premiers.

Tailles courantes des clés publiques RSA

- 1024 bits : Considérée comme insécure aujourd’hui.
- 2048 bits : Taille minimale recommandée actuellement (TLS, HTTPS).
- 3072 bits : Sécurité renforcée (équivalent ~128 bits symétrique).
- 4096 bits : Très robuste, mais plus lent.

Fonctionnement du chiffrement RSA

RSA applique la formule :

- $C = M^e \mod n$
  - M est le message clair (sous forme d'entier),
  - e est l'exposant public,
  - n est le module (produit de deux grands nombres premiers).

- Comme M doit être inférieur à n, on ne peut pas chiffrer un texte arbitraire directement.

Texte → blocs numériques

- Le texte est converti en octets (par exemple UTF-8).
- Ces octets sont découpés en blocs dont la taille est inférieure à clé publique en octets.
  - Exemple
    - Avec RSA 2048 bits (≈ 256 octets), la taille maximale d'un bloc est un peu moins de 256 octets
    - Chaque bloc est transformé en un entier M, puis chiffré avec la formule RSA.

Communication clé RSA publique

Public key example:

```
ssh-rsa
EXAMPLEzaC1yc2EAAAADAQABAAABAQCoYFOS10yNQ2AoRuvt2uM2LpuZXLGpNoHFxCAmXZjNIZ6t6s
sHCAWgiqzbp5fzRSZnPXjeuxQo2KsGkZCD6f81YHfEIBTSPWoiA6HPWA1AOR6K7E4ZGBkpYhOJKDK1
BYzCKUTgyRUvemmNmGme/c5O4ts50se0A/8m26YNt8TYgKqLV7mjl+Q1uMix0qS3wOim4x
+Iq5eV3cdTa0v0iuQJd01aXoCdJ1cdMW6qEDxZ5ILEMtle8FoLvvMe67JLqjCTxy8i/6x
+SiBWVITOgBKfeePPHsq2PceOQN/XfajeLd+CMAXYyRrvUo4HIiR443BJG1zevIvKYA7+yEXAMPLE
```

Private key example:

```
-----BEGIN RSA PRIVATE KEY-----
EXAMPLEBAAKCAQEAqGBTktdMjUNgKEbr7drjNi6bmVyxqTaBxcQgJl2YzSGererL
BwgFoIqs26eX80UmZz143rsUKNirBpGQg+n/JWB3xCAU0j1qIgOhz1gJQDkeiuxO
GRgZKWITiSgypQWMwilE4MkVL3ppjZhpnv3OTuLbOdLHtAP/JtumDbfE2ICqi1e5
o5fkNbjIsdKkt8DopuMfiKuXld3HU2tL9IrkCXdNWl6AnSdXHTFuqhA8WeSCxDLZ
XvBaC77zHuuyS6owk8cvIv+sfkogVlSEzoASn3njzx7Ktj3HjkDf132o3i3fgjAF
2Mka71KOByIkeONwSRtc3ryLymAO/smHHNQRzwIDAQABAoIBAGoipiu2uVOGd/OL
mSaKxpSd1olaq8atTCo8kcN9Vldf7OVWTnp1LQ7gu0uOnjLDkQyc7DcCGBgTU+NF
GKJ+es21vGkNi/JmsiMUxQetR8+K8dzCTgx1a07xurzHcP0ivXKajwde2ZLfB/Aw
dcu50zVYvLX7TtUDe++jn02gXF3X3q98lqWmSPV+dtlZPctQqcmemjQg3onUdpZo
4yrAKUKJdrchIMHhBD0jisom86Zl3jEPXRY7iuOfa1bB76cmErja18rijUhMH5Pn
mjAsbvZ0CTxU7QGx5yHnFtSK73oLN4LoYKek0TA7JARc4lp0MELtkOTn9mj2IeEw
h2yygPECgYEA37mi3uGVBBAVLEU3Z2sAS/thF0+L2y6qcuBxjY/HeyPnwvuXied0
xJhb9wPpODRTShDkKLfHPiVYD7H6bXZLetZfNLjIu/IKvseL85zCX8fWz6cJ6IeS
3QKRYu2VdpQW2prs+58QyKDlDqQ0hfE3dHZvSayLmm/9/sBZ24+G/WcCgYEAwKqb
yYkDOZtXIHZyTt1UUHvKFzo9LFuuMwlHQdNpvy2QbNNw4iE706DzVjy9FNuMXzIs
Skhhn7m+wredBP+r8udX3+gA1vY329wJ/+c7W8IPN21RiWIT4VtawmoHgMeJHOv4
4mdxqMo6L44Nkny/4KLtGAuZCUrJzoLr+d+Fn1kCgYEAyA7MIdo+0r8+77OFc6kv
PsKvc5TiT0FPkiI56IilrOvSl307aUncFODZe+23Y1cHE7g/DloohN4H/SD9+1xI
6rM/t3l1pvstuKPf9hw7hELDSDTqm1CAd7mQIJKrkLmkJh9bwzXeYEngClOz1AJ7
wF0X7x2oSJXU3zVKJRgXcgkCgYEAn504DxC5YUI2Piiirn9iWIMVe4S+JT+W46Uu
KXSSSNXgrqfE/zH1NHBE6A7NvrfcZQlV8/xfFEp3pS0kon2F4GiUPmUgPPYidLyo
dB8G6A+vN4YTZLOiMLLUT/gzWxbzmshLmpWEbgeLiNYwnElJVTrlHWSOVkplQfbo
tEvfkZECgYAayAwDXa2gbZBmqInwCTNJyqu8XW/Kc4JBT6mugXzQqxMr6ZnXM70h
Fq0EAT7kAHt4wKfZyPkcgrrmj0Mej6VoL2GlJejPykNa20nxrPIi8ecJDYhjiaIp
zoO5rFDVcZhMctewa70OL3c1q+nDGf7Sd9pqw0q31K6MiJwEXAMPLE==
-----END RSA PRIVATE KEY-----
```

RSA vs ECC

1. RSA (Rivest–Shamir–Adleman)
   - Basé sur la difficulté de factoriser de grands nombres premiers.
   - Clé typique : 2048 bits (minimum recommandé), 3072 ou 4096 bits pour plus de sécurité.
   - Usage : chiffrement, signature numérique (TLS/HTTPS, certificats X.509, etc.).
   - Avantage : largement supporté, robuste.
   - Inconvénient : clés longues et opérations plus lentes que les alternatives modernes.

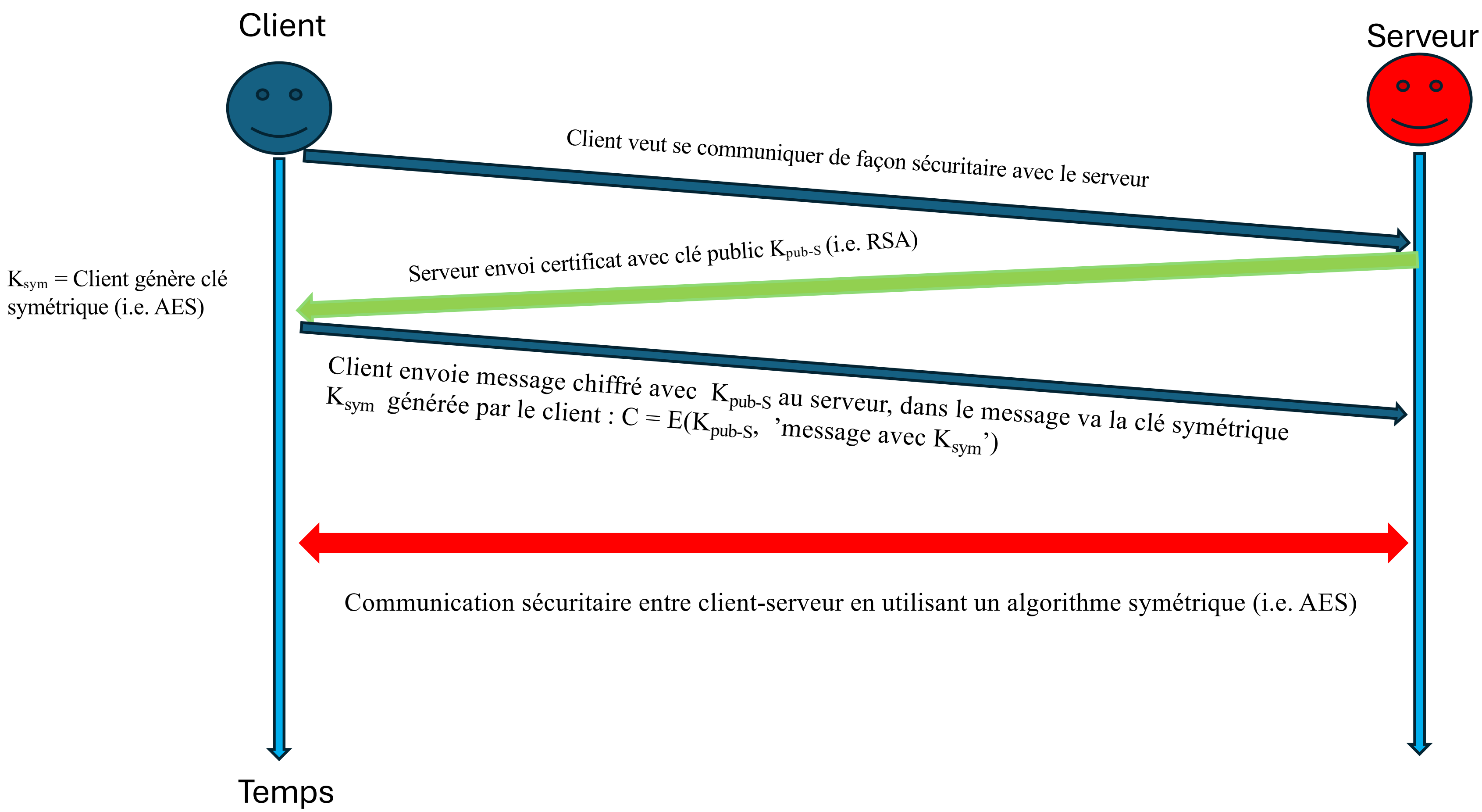
2. ECC (Elliptic Curve Cryptography)
   - ~20 % des certificats qu'ils émettent utilisent ECC pour la signature.
   - Basée sur le problème du logarithme discret sur les courbes elliptiques.
   - Clés beaucoup plus petites que RSA pour le même niveau de sécurité :
     - ECC 256 bits ≈ RSA 3072 bits en sécurité.
   - Usage : ECDSA (signature), ECDH (échange de clés), TLS modernes.
   - Avantage : plus rapide, moins de mémoire, idéal pour mobiles et IoT.
   - Inconvénient : implémentations plus complexes, certaines courbes non recommandées (préférer Curve25519, secp256r1).

Transaction sur Internet

1. Authentification du serveur (certificats HTTPS)
    - Ton navigateur reçoit un certificat numérique (X.509) émis par une autorité de certification (CA).
    - Ce certificat contient une clé publique (RSA ou ECC en général).
    - RSA ou ECC sert à vérifier que tu parles bien au bon serveur (ex. amazon.com).
    - Exemple : un site peut avoir un certificat RSA-2048 ou ECC (secp256r1).

2. Échange de clé (Key Exchange)
    - RSA peut être utilisé pour chiffrer une clé de session (symétrique, ex. AES).
    - Aujourd'hui, on préfère ECDHE (Elliptic Curve Diffie-Hellman Éphémère) car :
        - plus rapide,
        - offre la forward secrecy (si la clé privée du serveur est compromise plus tard, les anciennes sessions restent protégées).
    - Dans TLS 1.3, RSA n'est presque plus utilisé pour l'échange → c'est ECC (ECDHE) qui domine.

Transaction sur Internet



Groupe multiplicatif $\mathbb{Z}_p^*$ (PAS POUR L'INTRA)

Le groupe multiplicatif $\mathbb{Z}_p^*$ est :

$$\mathbb{Z}_p^* = \{1, 2, 3, \ldots, p-1\}$$

c'est-à-dire **tous les entiers de 1 à $p-1$** avec la multiplication modulo p.

Si p est premier :

- Fermeture : le produit de deux éléments reste dans $\mathbb{Z}_p^*$.
- Associativité : la multiplication mod p est associative.
- Élément neutre : 1 est l'élément neutre.
- Inverse : chaque $a \in \mathbb{Z}_p^*$ a un inverse $a^{-1} \pmod{p}$.
- Donc $\mathbb{Z}_p^*$ est bien un groupe abélien (commutatif).

Exemple
$p = 7$.

$$\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$$

$3 \cdot 5 \equiv 15 \equiv 1 \pmod{7} \rightarrow$ donc $3^{-1} = 5$. (!! Attention 3^{-1} veut dire l'inverse de 3, et non 1/3)
$2 \cdot 4 \equiv 8 \equiv 1 \pmod{7} \rightarrow$ donc $2^{-1} = 4$.
$6 \cdot 6 \equiv 36 \equiv 1 \pmod{7} \rightarrow$ donc $6^{-1} = 6$.

Groupe multiplicatif $\mathbb{Z}_p^*$ (PAS POUR L'INTRA)

Un élément $g \in \mathbb{Z}_p^*$ est générateur si et seulement si :

$$g^{\frac{p-1}{q}} \not\equiv 1 \pmod p \quad \text{pour tout facteur premier } q \text{ de } (p-1)$$

Un facteur premier d'un entier n est un nombre premier m qui divise n ($i.e.$ $n = 6, m = 3$)

Autrement dit :

- Factoriser $p - 1$.
- Vérifier que pour chaque facteur premier q, la puissance $g^{(p-1)/q}$ n'est jamais égale à 1 modulo p.

Générateurs

$\mathbb{Z}_p^*$ est un groupe cyclique. Il existe donc des générateurs g tels que :

$$\{g^1, g^2, \dots, g^{p-1}\} \equiv \mathbb{Z}_p^* \pmod p$$

Exemple

$p = 7 \rightarrow \mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$, d'ordre 6.

Les diviseurs premiers de 6 sont 2 et 3:

$$3^{6/2} = 3^3 = 27 \equiv 6 \not\equiv 1 \pmod 7$$
$$3^{6/3} = 3^2 = 9 \equiv 2 \not\equiv 1 \pmod 7$$

Donc 3 est un générateur.

$$3^1 = 3,\ 3^2 = 2,\ 3^3 = 6,\ 3^4 = 4,\ 3^5 = 5,\ 3^6 = 1 \pmod 7$$

$\rightarrow$ tout $\mathbb{Z}_7^*$ est couvert.

Groupe multiplicatif $\mathbb{Z}_p^*$ - logarithme discrète (PAS POUR L'INTRA)

Soit :

- un groupe fini (souvent $\mathbb{Z}_p^*$,le groupe multiplicatif mod p),
- un générateur g,
- un élément a de ce groupe.

On appelle logarithme discret de a en base g, l'entier x tel que :

$$g^x \equiv a \pmod p$$

On note souvent :

$$x = \log_g(a) \pmod p$$

Exemple

Prenons $p = 7$, donc $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$.

Générateur : $g = 3$.

Calculons les puissances de 3 :

$$3^1 \equiv 3 \pmod 7$$
$$3^2 \equiv 2 \pmod 7$$
$$3^3 \equiv 6 \pmod 7$$
$$3^4 \equiv 4 \pmod 7$$
$$3^5 \equiv 5 \pmod 7$$
$$3^6 \equiv 1 \pmod 7$$

Donc :

$\log_3(2) \equiv 2 \pmod 7$ $\quad x = 2, g = 3, a = 2,\ g^x \equiv a \pmod p \rightarrow x = \log_g(a) \pmod p$

$\log_3(5) \equiv 5 \pmod 7$ $\quad x = 5, g = 3,\ a = 5,\ g^x \equiv a \pmod p \rightarrow x = \log_g(a) \pmod p$

Algorithme El-Gamal

El-Gamal repose sur :

- un grand nombre premier p,
- un générateur g du groupe multiplicatif $\mathbb{Z}_p^*$,
- et la difficulté de résoudre :

$$g^x \equiv y \pmod{p}$$

(c'est le problème du logarithme discret).

1. **Génération des clés**

Choisir un grand premier p et un générateur g.
L'utilisateur choisit une clé privée :

$$x \in \{1, \ldots, p-2\}$$

Il calcule sa clé publique :

$$y = g^x \pmod{p}$$

Clé privée : x
Clé publique : (p, g, y)

Algorithme El-Gamal

2. **Chiffrement**

Pour envoyer un message m (tel que $m < p$) :
L'expéditeur choisit un nombre aléatoire $k \in \{1, \dots, p-2\}$.
Il calcule :

$$c_1 = g^k \pmod p$$
$$c_2 = m \cdot y^k \pmod p$$

Le chiffre est le couple :

$$(c_1, c_2)$$

3. **Déchiffrement**

Le destinataire connaît x (clé privée).
Il calcule :

$$m = c_2 \cdot \left(c_1^x\right)^{-1} \pmod p$$

Car :

$$c_1^x = \left(g^k\right)^x = g^{kx}$$
$$c_2 = m \cdot y^k = m \cdot \left(g^x\right)^k = m \cdot g^{kx}$$

donc :

$$c_2 \cdot \left(c_1^x\right)^{-1} = m$$

Algorithme El-Gamal

4. **Exemple**
Communication d'Alice vers Bob. Construction clé privée de Bob)

Paramètres et clés

$p = 23$ (premier)
générateur $g = 5$ (ici 5 est un générateur de $\mathbb{Z}_{23}^*$)
Clé privée de Bob :

$x = 6$ (L'utilisateur choisit une clé privée : $x \in \{1, \dots, p-2\}$)

Clé publique de Bob :

$$y = g^x \bmod_p = 5^6 \bmod 23.$$

Calcul rapide :

$$5^2 = 25 = 25 - 23 \equiv 2 \bmod 23$$
$$5^4 = 5^2 \cdot 5^2 \equiv 2^2 \equiv 4 \bmod 23$$
$$5^6 = 5^4 \cdot 5^2 \equiv 4 \cdot 2 = 8 \bmod 23$$

Donc $y = 8 \rightarrow 5^6 \equiv 8 \bmod 23$.

Clé publique de Bob $= (p, g, y) = (23, 5, 8)$
Clé privée xde Bob $= 6$.

Algorithme El-Gamal

Chiffrement (Alice envoie un message m à Bob) → Clé publique de Bob = $(p, g, y) = (23, 5, 8)$
Prenons message (encodé comme entier < p) : m = 15.
Alice choisit un nombre aléatoire (aléa) k = 10 (doit être choisi au hasard et différent à chaque chiffrement).

- $\mathbf{c_1 = g^k \,(mod\ p) = 5^{10}\,(mod\ 23).}$
  Calcul détaillé (exponentiation par carrés) :
  $$5^2 = 25 \equiv 25 - 23 = 2 \pmod{23}$$
  $$5^4 = \left(5^2\right)^2 \equiv 2^2 = 4 \pmod{23}$$
  $$5^8 = \left(5^4\right)^2 \equiv 4^2 = 16 \pmod{23}$$
  $$5^{10} = 5^8 \cdot 5^2 \equiv 16 \cdot 2 = 32 \equiv 9 \pmod{23}$$

  Donc $\mathbf{c_1 = 9}$.
- $c_2 = m \cdot y^k \bmod p = 15 \cdot 8^{10} \bmod 23.$
  Calcul détaillé :
  $$8^2 = 64 \equiv 64 - 2 \cdot 23 = 64 - 46 = 18 \pmod{23}$$
  $$8^4 = \left(8^2\right)^2 = 18^2 = 324 \equiv 324 - 14 \cdot 23 = 324 - 322 = 2 \pmod{23}$$
  $$8^8 = \left(8^4\right)^2 = 2^2 = 4 \pmod{23}$$
  $$8^{10} = 8^8 \cdot 8^2 \equiv 4 \cdot 18 = 72 \equiv 72 - 3 \cdot 23 = 72 - 69 = 3 \pmod{23}$$

  Donc $8^{10} \equiv 3$ et $\mathbf{c_2 = 15 \cdot 8^{10}\,(mod\ 23) \equiv 15 \cdot 3 = 45 \equiv 22 \pmod{23}}$.
- Le texte chiffré envoyé par Alice est donc le couple :
  $$(c_1, c_2) = (9, 22).$$

Algorithme El-Gamal

Déchiffrement (Bob récupère m)

- Bob reçoit $(c_1, c_2) = (9, 22)$ et utilise sa clé privée $x = 6$.
- Il calcule $\mathbf{c_1^x \bmod p = 9^6 \bmod 23}$:

$$9^2 = 81 \equiv 12$$
$$9^4 \equiv 12^2 = 144 \equiv 6$$
$$9^6 = 9^4 \cdot 9^2 \equiv 6 \cdot 12 = 72 \equiv 3 \pmod{23}$$

Donc $c_1^x \equiv 3$.

Il calcule l'inverse modulaire de $c_1^x = 3 \bmod 23$: $3^{-1} \equiv 8$ (car $3 \cdot 8 = 24 \equiv 1 \bmod 23$).

Il récupère :

$$m \equiv c_2 \cdot \left(c_1^x\right)^{-1} \bmod 23 \equiv 22 \cdot 8 = 176 \equiv 15 \pmod{23}.$$

On retrouve bien le message initial $m = 15$.

Déchiffrement

Le destinataire connaît x (clé privée).

Il calcule :

$$m = c_2 \cdot \left(c_1^x\right)^{-1} \pmod p$$

Car :

$$c_1^x = \left(g^k\right)^x = g^{kx}$$
$$c_2 = m \cdot y^k = m \cdot \left(g^x\right)^k = m \cdot g^{kx}$$

donc :

$$c_2 \cdot \left(c_1^x\right)^{-1} = m$$

Résumé chiffré

Paramètres : $p = 23,\ g = 5$

Bob : clé privée $x = 6$, clé publique $y = 8$

Alice : message $m = 15$, aléa $k = 10$

Chiffre : $(c_1, c_2) = (9, 22)$

Déchiffrement : $m = 15$

Corps de Galois (PAS POUR l'INTRA)

Un **corps** est un ensemble muni de deux opérations :

- une addition,
- une multiplication,
- qui respectent les propriétés usuelles (associativité, commutativité, existence d'un élément neutre et d'inverses, distributivité).
- Exemples :
  - $\mathbb{R}$,
  - mais aussi des ensembles finis comme $\mathbb{Z}_p$ (entiers modulo un nombre premier p).

Corps finis ou Corps de Galois : un corps qui contient un nombre fini d'éléments.

$$GF(q) = GF(p^n)$$

- où $q = p^n$ pour un nombre premier p et un entier $n \geq 1$.
- p est appelé la caractéristique du corps,
- n est le degré de l'extension.

Corps de Galois (PAS POUR l'INTRA)

Exemples

- $\mathrm{GF}(2) = \{0, 1\}$ avec addition et multiplication modulo 2.
  $1 + 1 = 0$,
  $1 \cdot 1 = 1$.
- $\mathrm{GF}(2^3)$→ ce corps contient $2^3 = 8$ éléments.
  On le construit en utilisant des polynômes modulo un polynôme irréductible de degré 3 sur $\mathrm{GF}(2)$,
  par exemple $x^3 + x + 1$.

- $\mathrm{GF}(2^n)$ → Utilisé par ECC, $\mathrm{GF}(2^8)$ utilisé par AES.
  - On part du corps de base $\mathrm{GF}(2) = \{0, 1\}$.
  - Les éléments de $\mathrm{GF}(2^n)$ sont représentés par des polynômes de degré < n à coefficients dans $\{0, 1\}$.
    Il y a 2^n polynômes possibles, donc 2^n éléments.
  - Les opérations (addition et multiplication) se font modulo un polynôme irréductible de degré n sur $\mathrm{GF}(2)$.
    Addition = addition coefficient par coefficient modulo 2 (XOR).
    Multiplication = produit des polynômes, puis réduction modulo ce polynôme irréductible.

Corps de Galois – exemple (PAS POUR l'INTRA)

Exemples

$\mathrm{GF}(2^3)$ (8 éléments)

- On choisit le polynôme irréductible $\mathrm{P(x)} = \mathrm{x}^3 + \mathrm{x} + 1$.
- Les éléments de $\mathrm{GF}(2^3)$ sont :

$$\{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}$$

- Addition :

$$(\mathrm{x}^2 + 1) + (\mathrm{x} + 1) = \mathrm{x}^2 + \mathrm{x} \text{ (car } 1 + 1 = 0 \quad \text{(XOR))}.$$

- Multiplication (exemple) :

$$(\mathrm{x} + 1) \cdot (\mathrm{x}^2 + 1) = \mathrm{x}^3 + \mathrm{x}^2 + \mathrm{x} + 1.$$

Dans **GF(2³)**,

- Les éléments sont bien représentés par des **polynômes** de degré inférieur à 3, avec des coefficients dans $\mathrm{GF}(2) = \{0, 1\}$.
- Chaque élément est un polynôme $a_0 + a_1x + a_2x^2$ avec $a_i \in \{0, 1\}$.
- Les opérations (addition et multiplication) se font modulo 2 sur les coefficients, et modulo un polynôme irréductible de degré 3

ECC (Elliptic Curve Cryptography)

Une **courbe elliptique** sur un corps fini définie par l'équation :

$$y^2 = x^3 + ax + b \pmod p$$

- a et b sont des constantes qui définissent la courbe.
- p est un nombre premier (dans le cas d'un corps fini).
- Les points (x, y) qui satisfont cette équation, plus un point à l'infini O, constituent un groupe abélien.
- On définit une addition de points et une multiplication par un entier qui servent à la cryptographie.

Principes de la cryptographie ECC

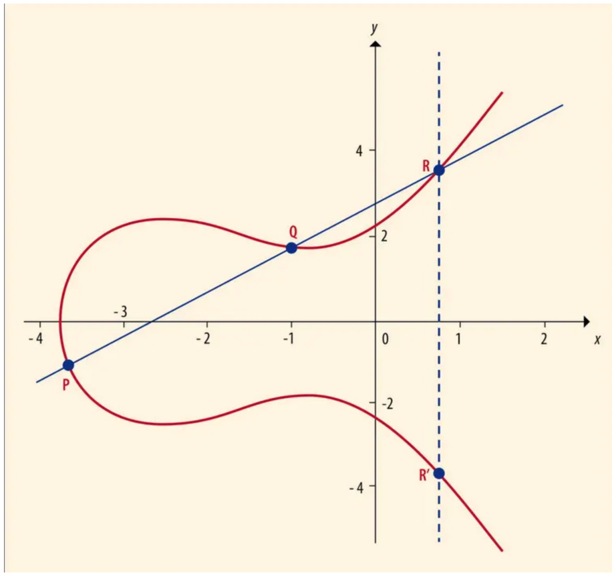
- La sécurité repose sur le problème du logarithme discret sur la courbe elliptique (ECDLP) :
    - Si P est un point de la courbe et k un entier, calculer $Q = kP$ est facile.
    - Retrouver k à partir de Q et P est très difficile, même avec des ordinateurs puissants.

Pour une courbe $y^2 = x^3 + ax + b$ sur un corps fini chaque point $P = (x_1, y_1)$ et $Q = (x_2, y_2)$ peut être additionné :

On note $R = P + Q = (x_1, y_1) + (x_2, y_2) = (x_3, y_3)$.

Somme : **La somme de deux points sur une courbe elliptique est définie géométriquement** : pour additionner deux points P et Q, on trace la droite qui les relie. Cette droite coupe la courbe en un troisième point, R. Le point R', symétrique de R par rapport à l'axe des abscisses, est alors le résultat de l'addition P + Q. Cette opération a des propriétés similaires à l'addition des nombres et est fondamentale en cryptographie, où elle permet de multiplier un point par un entier de manière répétée.

ECC (Elliptic Curve Cryptography)



Cette courbe elliptique est la représentation graphique dans le plan de l'équation $y^2 = x^3 + 5x^2 + 6x + 5$. Une courbe elliptique vérifie les propriétés de l'« addition » géométrique. Tous les points de la courbe sont la « somme » de deux autres points. Dans cet exemple, P + Q = R et P + Q = R'.

Hachage cryptographique, ‘message digest’, ‘MAC message authentication Code’

Objectif → Intégrité (pas confidentialité)

- S’assurer qu’un message n’a pas été modifié de façon non autorisée une fois qu’il a été terminé par son auteur légitime.

- Un hachage cryptographique transforme une entrée de taille arbitraire (message, fichier, mot de passe) en une empreinte fixe (suite de bits/hex) appelée digest ou hash. Le processus est déterministe : la même entrée → toujours la même empreinte.

Hachage cryptographique - propriétés

Fonctions de hachage cryptographique *h*

– Une fonction *h*( ) est dite de hachage cryptographique si à partir d'un message *x* elle produit un *haché* (ou *hachage*) *h*(*x*), avec ces propriétés

1. **(à sens unique)** : il est très difficile de trouver un *x* à partir d'un *h* donné, tel que $h = h(x)$
2. **(absence de collision)** : il est très difficile de trouver deux messages *x* et *y*, tel que $h(x) = h(y)$
3. **(effet d'avalanche)** : il est très difficile de trouver à partir d'un *x* donné de trouver un autre *x'* « similaire » tel que $h(x) = h(x')$

4. Pas d'information réversible : le hash ne doit pas révéler de caractéristiques utiles sur l'entrée.

Note : aucun algorithme n'est parfait — on parle de difficulté computationnelle. Des méthodes plus anciens (MD5, SHA-1) ont été cassés pour la collision.

- Exemple de cas réel : le malware Flame (2012) signature frauduleuse via MD5 :le malware Flame a abusé d'une signature frauduleuse liée à MD5 pour faire apparaître certaines composantes comme signées par Microsoft.

Hachage cryptographique - propriétés

- Fonctions obsolètes
  - MD4 (128 bit)
    - Conçu par Rivest (de RSA)
    - Ressemble un peu à DES
      - Plusieurs rondes de coupage, transposition, permutation, et autre opérations binaires.
  - MD5 (128 bit)
    - Version amélioré de MD4
    - Usage très répandu
    - Utilisé par le programme linux md5sum
    - **Collisions en quelques heures**!
  - SHA-1 (160 bit)
    - Conçue par la NSA
    - Collisions possibles
    - Remplacé officiellement depuis 2011
- Fonctions recommandées
  - SHA-2
    - Famille de 5 fonctions introduites en 2001
    - Conçue par la NSA
    - Taille de haché : 224, 256, 384, 512
    - Similaire en structure à MD5 et SHA1
    - Aucune vulnérabilité connue (2020)
  - SHA-3
    - Compétition du NIST en 2006
    - Besoin d'avoir une famille de fonction différente de MD5/SHA1/SHA2 (au cas où…)
    - Algorithme KECCAK choisi (Bertoni, Daemen, Peeters, van Aasche)
    - Taille de haché : 224, 256, 384, 512

HMAC

L'idée de HMAC (Hash-Based Message Authentication Code) est d'assurer à la fois :

- L'intégrité : vérifier que le message n'a pas été modifié.
- L'authenticité : garantir que le message provient bien de l'expéditeur légitime (qui connaît la clé secrète).
- → Utilisation: API REST (Authorization headers), TLS, VPNs, signed cookies, etc.

Principe de fonctionnement

- On prend une fonction de hachage cryptographique (par ex. SHA-256).
- On combine le message avec une clé secrète partagée entre l'expéditeur et le destinataire.
- Le résultat est un code d'authentification (le HMAC) transmis avec le message ('hash', 'digest').
- Le destinataire, connaissant aussi la clé, peut recalculer le HMAC et vérifier qu'il correspond.

Construction → HMAC utilise deux étapes avec la clé :

- On mélange la clé avec un bloc appelé ipad (inner pad) et on applique la fonction de hachage avec le message.
- On mélange la clé avec un bloc appelé opad (outer pad), puis on applique encore la fonction de hachage sur le résultat précédent.
- Formellement :

$$\mathrm{HMAC}(K, M) = H((K \oplus \mathrm{opad}) \;||\; H\big((K \oplus \mathrm{ipad}) \;||\; M\big))$$

où K est la clé, M le message, et H la fonction de hachage,
opération de XOR (OU exclusif, $\oplus$) appliquée octet par octet

HMAC- propriétés

- Protocole standardisé (standard HMAC)
  1. Alice calcule HMAC (K, x) où
     - HMAC (K,x) = h($K' \oplus opad$) || h($K' \oplus ipad$) || x)

       et
     - K' est la clé dérivée
       – $H(K)$, si K est plus grande que la taille du bloc
         (déterminée par la fonction de hachage)
       – K , dans le cas contraire
     - *opad* est une constante de la taille du bloc répétant le byte `0x5C`
     - *ipad* est une constante de la taille du bloc répétant le byte `0x36`
  2. Bob calcule et vérifie HMAC (K, x') = HMAC (K, x)

Note :

- Opérateur || → c'est l'opérateur de concaténation binaire (m1 ‖ m2 signifie « les bits ou octets de m1 suivis, sans séparateur, des bits ou octets de m2 »)
- où k' est la clé k normalisée à la longueur du bloc B :
  si len(k) > B alors k' = H(k)
  sinon k' = k paddé par des zéros jusqu'à
  B octets
  ipad = 0x36 répété B fois, opad = 0x5c répété B fois.
- H(…) est la fonction de hachage cryptographique complète utilisée comme boîte noire. Exemples : SHA-256, SHA-512, SHA-3,…
- H(…) désigne la fonction de compression interne → H est obtenu en appliquant h de façon itérative (sur chaque bloc)

Signature numérique avec clé publique

Une signature numérique est un mécanisme cryptographique qui permet à une entité de prouver l'authenticité d'un message ou d'un document, à l'aide de la cryptographie asymétrique (paire de clés privée/publique).

Principe général : On génère une paire de clés asymétriques :

Clé privée : connue uniquement du signataire.

Clé publique : distribuée à tous les vérificateurs.

On utilise une fonction de hachage et un algorithme de signature (RSA, DSA, ECDSA, EdDSA…).

Étapes

1. Hachage : calculer un digest du message → h = H(message).
2. Signature : le signataire chiffre ce 'digest' avec sa clé privée → signature = Sign(clé-privée, h).
3. Transmission : on envoie (message, signature) au destinataire.
4. Vérification : le destinataire recalcule h' = H(message) et vérifie que Verify(clé-publique, h', signature) est vrai.

Exemple avec RSA

- Signature : $s = (H(m))^d \bmod n$ (avec la clé d privée RSA).
- Vérification : vérifier que $s^e \bmod n = H(m)$ (avec la clé publique e).