Cryptographie Bachelor 3

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

Printemps 2025

Organisation :

Volume horaire : 12h de cours, 12h de TD
Notation : Examen final de 2h30 le 27 mars 9h15-11h45.
Matériel autorisé : cheat sheet (feuille A4 nominative)

Contenu du cours

Retour sur les fondamentaux, protocole TLS
Chiffrement authentifié
Signatures
KEM et cryptographie post-quantique

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

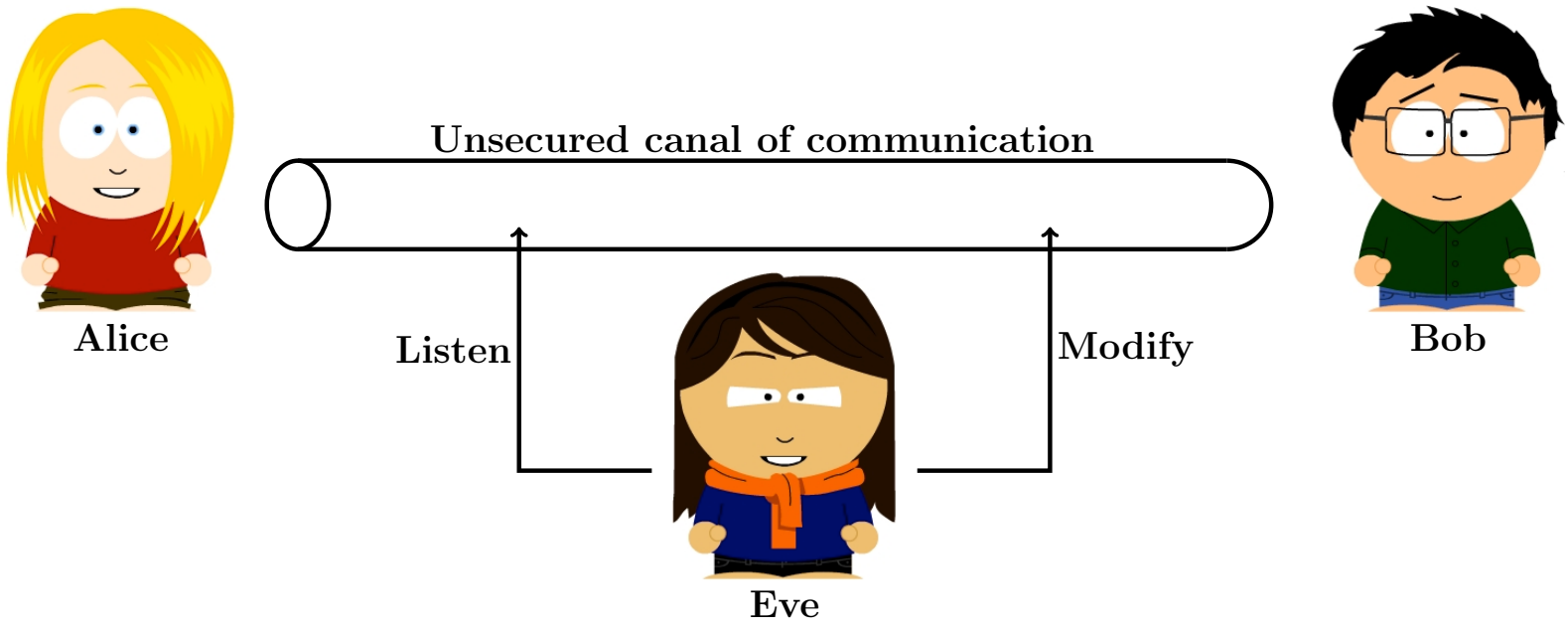
La cryptographie est presque aussi vieille que l'écriture, mais la cryptographie dite moderne n'a même pas 60 ans !

- 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

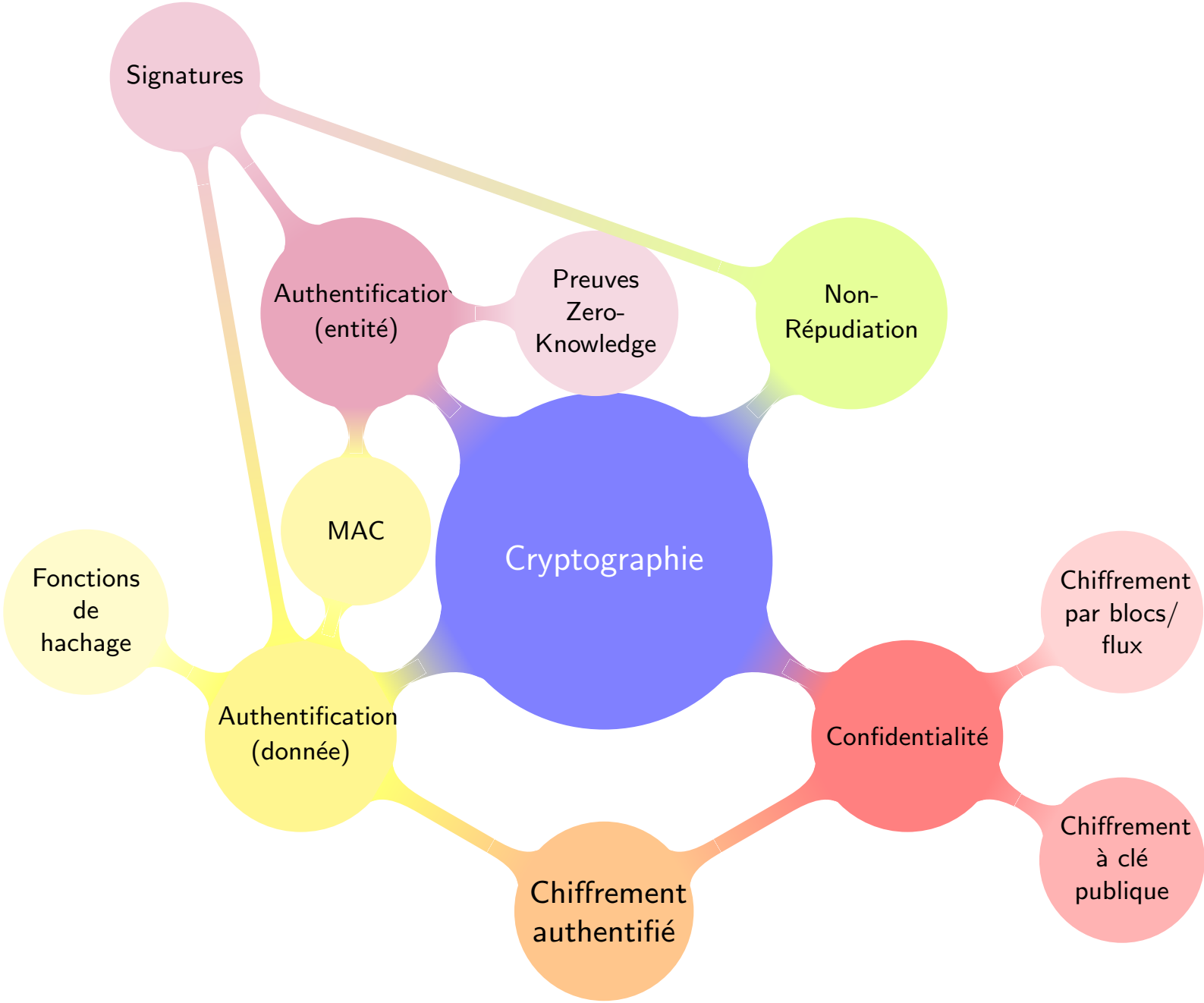
La cryptographie : un empilement de couches

- 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



Chapitre I

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

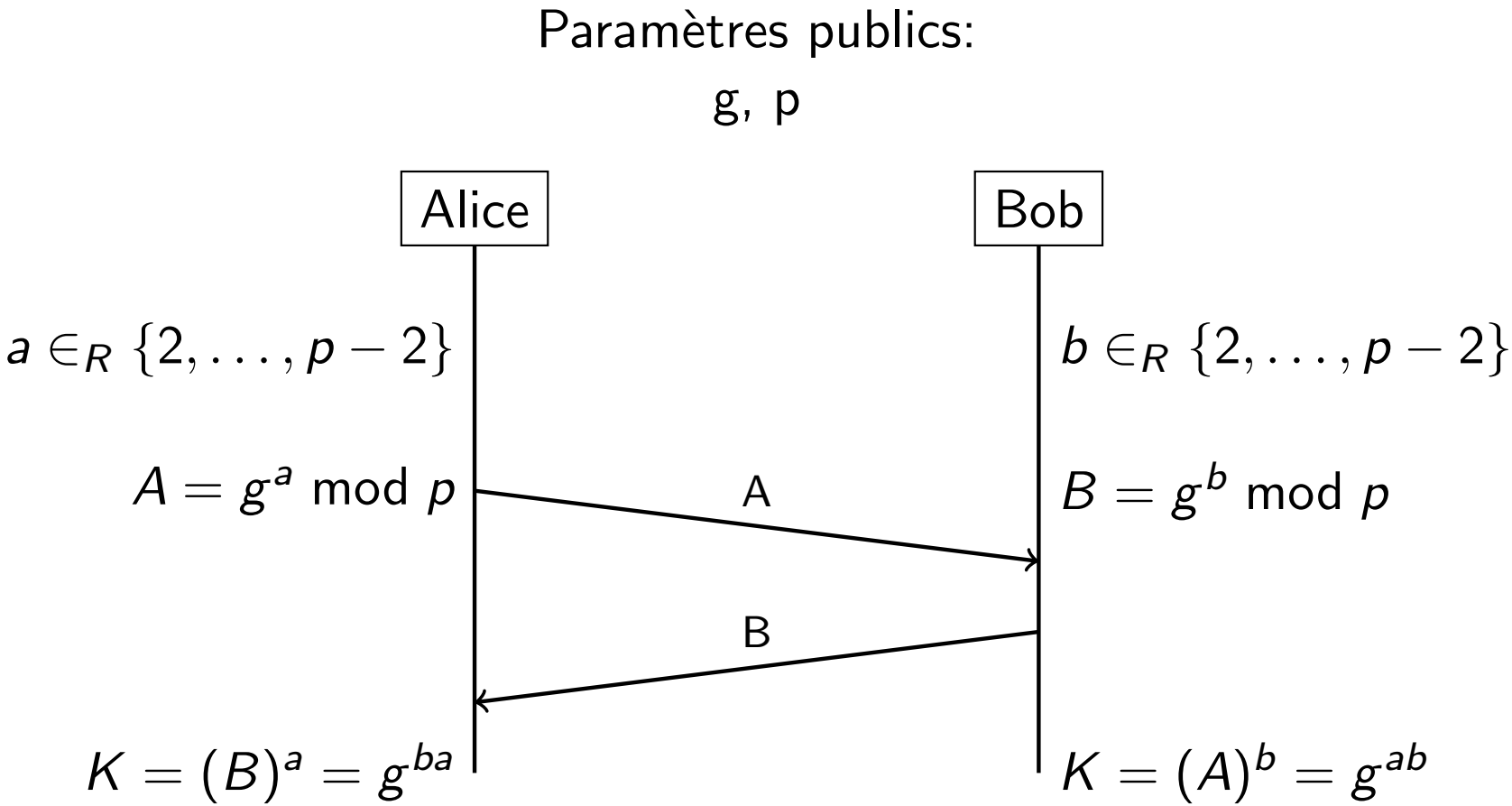
La sécurité dans TLS

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 ?

Rappel rapide sur Diffie-Hellman (1976)



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 : comment vérifier 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 (RDV au chap III).

Chiffrement Authentifié (AEAD)

Pourquoi l'intégrité ?

Attaques par rejeu (replay attack)

- Spoofing GPS : si je capte un signal GPS, je peux le rediffuser avec un retard. La cible captera ce signal et aura une fausse idée de sa position due au retard.
- Et si les données sont chiffrées ?
- Sur le réseau d'une chaîne de fabrication, j'intercepte un message chiffré correspondant à une commande (ex : actionner un bras). Il me suffit de copier et d'envoyer un boucle ce message chiffré pour déclencher l'action indéfiniment, potentiellement jusqu'à causer des dommages !
- Si un capteur envoie une donnée chiffrée C (ex : pression, température), je peux créer un faux chiffré $C' = C \oplus d$. Lors du déchiffrement, le logiciel de contrôle lira des valeurs (probablement) aberrantes et déclenchera une action (arrêt d'urgence, dispositif anti-incendie, etc.)

Le chiffrement authentifié

Idée : il suffit de chiffrer et de calculer un MAC pour obtenir un chiffrement authentifié !

Question : oui mais dans quel ordre ?

Le chiffrement authentifié

Idée : il suffit de chiffrer et de calculer un MAC pour obtenir un chiffrement authentifié !

Question : oui mais dans quel ordre ? **Options :**

- Encrypt-and-MAC (les deux en même temps)
- MAC-then-encrypt (MAC sur le clair puis chiffrement)
- **Encrypt-then-MAC** (spoiler : c'est LA bonne approche)

Encrypt and MAC (SSH)

Concept: le chiffré authentifié est $ct = C||tag$, avec

- $C = \text{Encrypt}(K_C, \text{message})$
- $\text{tag} = \text{MAC}(K_I, \text{message})$

Point faible: le MAC ne protège pas le chiffré !

Remarque : les clés pour la confidentialité et l'intégrité sont différentes (K_C vs K_I), c'est une bonne pratique d'avoir une clé par usage.

MAC then encrypt (SSL/TLS < 1.2)

Concept: le chiffré authentifié est C, avec

- tag $=$ MAC(K_I, message)
- $C=$ Encrypt(K_C, message $||$ tag)

Point faible: pour pouvoir vérifier l'intégrité (le MAC) il faut d'abord déchiffrer. Est-ce grave ?

Bonne pratique : on préfère pouvoir rejeter un message le plus vite possible en faisant le moins d'opérations dessus (réduire la surface d'attaque). Ici, **c'est encore plus grave!**

Padding Oracle attacks

En bon français, attaques par oracle de remplissage.

Les chiffrements par blocs travaillent sur des messages dont la taille est un multiple de la taille du bloc (16 octets pour AES). Le cas échéant, on doit ajouter des octets de remplissage (padding).

L'oracle : après déchiffrement, le serveur vérifie la validité du padding et sa réponse permet à l'interlocuteur de savoir si le padding est valide ou non.

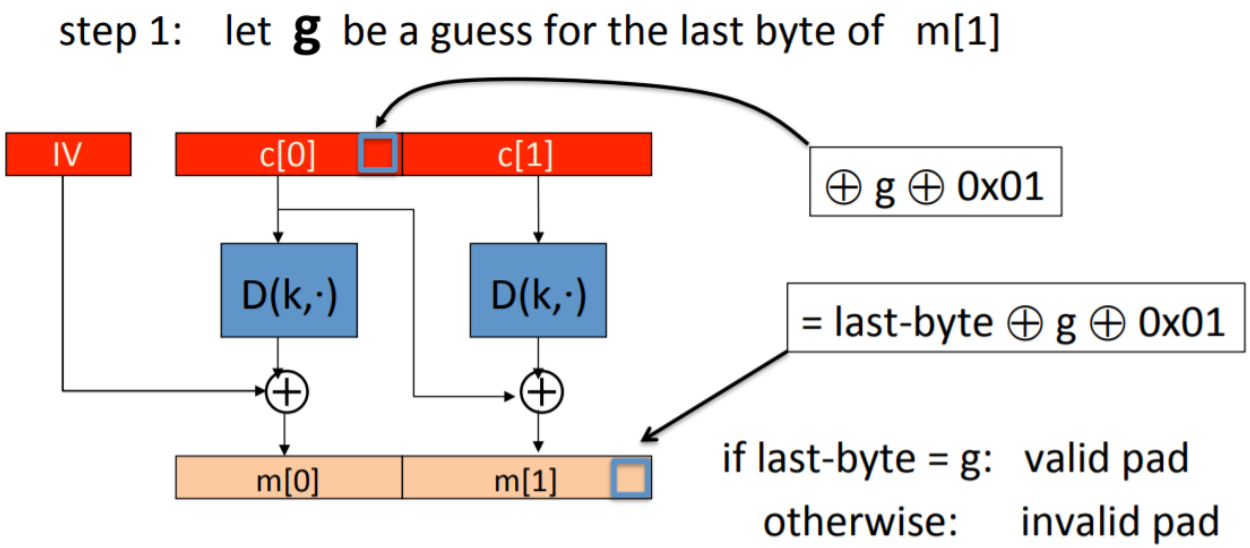
L'attaque : cette réponse suffit à retrouver le message octet par octet même sans connaître la clé.

Pourquoi ça fonctionne ?

L'attaquant intercepte deux blocs chiffrés valides C_1 et C_2.

Il envoie des faux chiffrés $C_1'||C_2$ en modifiant le dernier octet de C_1', jusqu'à ce que le serveur ne lui renvoie pas d'erreur de padding.

À ce moment il sait alors que le dernier octet de $Dec(K, C_2)$ est de telle sorte qu'on obtient l'octet $0\text{x}01$ en le XORant avec C_1'.



Il connaît donc un octet du déchiffrement de C_2, et il peut continuer de proche en proche (en visant un padding $0\text{x}02\ 0\text{x}02$).

Le paradigme encrypt-then-MAC

Concept: le chiffré authentifié est $C||\text{tag}$, avec

- $C=$ Encrypt(K_C, message)
- tag $=$ MAC(K_I, C)

Bonne pratique : le destinataire vérifie le MAC **avant** de tenter le déchiffrement. Si la vérification échoue, le paquet est rejeté avant même de tenter un déchiffrement.

Justification : il faut minimiser le nombre d'opérations qu'une personne non authentifiée peut faire faire à votre machine. C'est de la **défense en profondeur**.

L'AEAD (Authenticated Encryption with Associated Data)

Principe : solution moderne avec interface unifiée, plus simple et moins de risque d'erreurs (ex: utiliser K_I pour chiffrer, faire les opérations dans le mauvais ordre, etc.)

Interface type pour l'AEAD

$\mathrm{Encrypt}$(Clé, Nonce, Clair, Données Associées) = (Chiffré, tag)
$\mathrm{Decrypt}$(Clé, Nonce, Chiffré, Données Associées, tag) = (Clair, Données Associées)

Composants

- Clé : la clé secrète
- Nonce : numéro unique (public mais unique!)
- Message : données chiffrées et authentifiées
- Données Associées : données authentifiées mais non chiffrées (en-tête)
- tag : MAC couvrant le clair et les données associées

Données associées, késako ?

Question : pourquoi ne pas tout mettre dans le clair ?

Certaines métadonnées doivent être lisibles sans déchiffrement (adresse du destinataire, le choix de l'algorithme de chiffrement, la taille du paquet, etc.) mais ne doivent pas pouvoir être falsifiées.

Exemple:

- Message = Header || Payload || Tag
- Header = ID Source || ID Destination || Message Number
- Payload = message chiffré

Avec Header mis dans le champ données associées, le routeur peut lire Header mais personne ne peut modifier le Header sans invalider le tag.

Attaque par découpage (Splicing Attack)

Si l'attaquant intercept deux messages

- M1= Header 1 || Payload 1 || Tag 1
- M2= Header 2 || Payload 2 || Tag 2

Un attaquant peut-il réutiliser le Tag ?

- Sans AD, il peut envoyer M= Header 2 || Payload 1 || Tag 1 et la vérification sera valide. Exemple typique : intercepter un message et renvoyer ce message à un autre destinataire.
- Avec AD le Tag englobe le header donc Tag 1 n'est valide que pour Header 1 || Payload 1, le message M sera donc rejeté.

Exemple : si je remarque que la porte 1 s'ouvre après qu'une payload chiffrée ait été envoyé au contrôleur 1, je peux tenter d'envoyer la même payload avec un header adressé au contrôleur 2. Sans AD, cela peut me permettre d'ouvrir la porte 2.

Le processus de déchiffrement

Entrée : le destinataire reçoit le message $AD||C||T$ avec C le chiffré, AD les données associées et le tag T. Le destinataire connaît par ailleurs la clé K

Dérivation des clés (optionnelle): le programme calcule K_I et K_C deux clés dérivées de la clé K via une **Key Derivation Function** (KDF): $(K_C, K_I) = KDF(K, FixedInfo, sel)$

Déchiffrement:
Le destinataire recalcule $t = MAC(K_I, AD||C)$, deux options :

- Si $t \neq T$ alors le programme retourne une erreur.
- Si $t = T$, le programme calcule $M = \mathrm{Decrypt}(K_C, C)$ et retourne $AD||M$.

Dérivation de clés

Question : il faut une clé par usage en théorie, je dois vraiment envoyer toutes ces clés ?

Solution : utiliser un mécanisme qui transforme une seule clé K (dite Master Key) en plein de clés dérivées $K_1, \ldots, K_n$.

Contraintes :

- Il doit être impossible de trouver K, même en connaissant toutes les K_i
- Il doit être impossible de trouver K_i, même en connaissant toutes les autres clés K_j pour $j \neq i$
- Les K_i doivent être indistinguables de l'aléatoire, et ce même si K n'est pas aléatoire (intérêt : si K est un mot de passe)

En pratique c'est de la crypto symétrique (MAC ou chiffrements par blocs). **Textes normatifs :** NIST SP800-56 & SP800-108.

Le nonce

Attention : nonce $\neq$ IV. Un nonce doit être **unique** alors que pour le mode CBC l'IV doit juste être imprévisible.

Règle d'or : Un couple (clé, nonce) ne doit **jamais** être réutilisé pour chiffrer deux messages différents.

Ou sinon... cela cause une défaillance catastrophique du système.

Exemple historique

Les cryptanalystes de Bletchley Park ont compris le fonctionnement d'Enigma en janvier 1942 sans en avoir vu un seul exemplaire.

Comment ? En août 1941, un opérateur a renvoyé un message qui n'avait pas été reçu correctement en le chiffrant avec la même clé (la pratique était déjà interdite à l'époque dans les procédures). De plus l'opérateur a légèrement modifié le second message (utilisation d'abbréviations).

Conséquence : à partir des deux chiffrés, John Tiltman a reconstitué le texte clair mais aussi le mécanisme de chiffrement. Son collègue W. Tutte a ensuite pu reconstituer toute la machine à partir de cela.

Source : Wikipedia

Exemple d'attaque moderne

Chiffrement authentifié AES-GCM

Pour le chiffrement, AES en Mode Compteur

- KeyStream = AES(Clé, Nonce || Compteur)
- Chiffré = Clair $\oplus$ KeyStream

Pour l'authentification, hachage polynomial (GHASH).

Si je réutilise (Clé, Nonce) pour chiffrer M_1 et M_2, le KeyStream sera identique donc en xorant les chiffrés on a
$C_1 \oplus C_2 = (M_1 \oplus \text{KeyStream}) \oplus (M_2 \oplus \text{KeyStream}) = M_1 \oplus M_2$

Conséquence : si les messages sont structurés je peux en déduire des informations, si jamais je connais M_1 (par exemple si je "demande" à un serveur de chiffrer M_1) je déduis M_2.

Mais il y a pire...

Intégrité dans AES-GCM

Calcul du tag

Tag=GHASH(H,AD,C)$\oplus$ AES(K,N_0), où

- H=AES(K,0) est une clé dérivée de la clé K
- N_0 est dérivé du nonce
- GHASH correspond à une multiplication polynomiale

En cas de réutilisation de nonce : AES(K,N_0) identique.
Donc $T_1 \oplus T_2$ =GHASH(H,A_1, C_1)$\oplus$GHASH(H,A_2, C_2).
Comme GHASH est une fonction linéaire on en déduit l'équation

$$\text{(E1):}\ T_1 \oplus T_2 = \text{GHASH(H},A_1 \oplus A_2, C_1 \oplus C_2)$$

Conclusion : j'ai juste à résoudre l'équation polynomiale (E1) pour retrouver la clé H car je connais les associated data A_1 et A_2 et les chiffrés C_1 et C_2.

Conséquence : attaque sur l'intégrité

Une fois que l'attaquant connaît la clé H il peut calculer des tags valides pour n'importe quel message.

Du point de vue sécurité, on retombe dans le cas où le chiffrement n'est pas authentifié.

L'attaquant peut donc (liste non exhaustive) :

- Rejouer des messages à volonté
- Envoyer des chiffrés créés de toutes pièces
- Envoyer des chiffrés avec des en-têtes modifiées
- Se faire passer pour n'importe quelle composante du système

Remarque : ce n'est pas qu'un exemple théorique, aujourd'hui beaucoup d'objets sont un assemblage de puces reliées par un bus (voitures, satellites) ou par Wi-Fi/bluetooth (domotique).

Gestion des nonces en pratique

Question : comment garantir l'unicité des nonces ?

- **Utiliser des compteurs**
  - **Stratégie :** Le premier nonce est 0, on incrémente à chaque message
  - **Avantage :** Simple à implémenter, l'unicité est garantie
  - **Difficulté :** gestion fiable de l'état (redémarrage, systèmes distribués ou redondants)
- **Nonces aléatoires**
  - **Stratégie :** Générer un nonce au hasard pour chaque message
  - **Avantage :** Pas de gestion d'état
  - **Difficulté :** Risque de collision (paradoxe des anniversaires)

Une autre option ?

Il existe des AEAD qui sont résistants à une mauvaise utilisation des nonces (misuse-resistant).

Exemples : SIV (RFC 5297), AES-GCM-SIV (RFC 8452)

Principe : le nonce n'est pas directement utilisé, on dérive une clé interne à partir du nonce ET du message.

Avantage : en cas de réutilisation d'un nonce sur deux messages M et M', la fuite d'information est de pouvoir déterminer si M=M'. Pas de risque sur la confidentialité ou l'intégrité.

Défaut : l'algorithme est plus lent car il traite les données deux fois.

Les normes cryptographiques

Le choix des paramètres, la bonne implémentation et des séquences étalon sont fournies par des normes, souvent internationales. L'ANSSI et le BSI (France et Allemagne) en proposent mais le NIST fait souvent autorité.

- Pour le chiffrement : FIPS197 (AES), SP800-232 (Ascon)
- Pour l'intégrité : SP800-224 (HMAC), SP800-185 (KMAC)
- Pour les modes : SP800-38 A (5 modes pour la confidentialité), B (CMAC), C et D (CCM/GCM pour AEAD)
- Pour les KDF : SP800-108 et SP800-56C

Les limites de la cryptographie seule

La cybersécurité est un écosystème complexe où de nombreuses contraintes et de nombreux acteurs doivent cohabiter. La cryptographie n'est qu'un aspect parmi tant d'autres.

- **Gestion des clés (Key management) :** distribuer, mettre à jour et stocker les clés secrètes est un enjeu majeur imposant des solutions souvent complexes.
- **Sécurité logicielle :** une cryptographie mal implémentée ou des bugs dans l'interface peuvent réduire la sécurité à néant. Ces bugs ne sont pas toujours simples à détecter !
- **Sécurité matérielle :** attaque physique contre le serveur, attaques par canaux auxiliaires, attaques sur la mémoire
- **Culture de sécurité :** l'utilisateur est-il conscient, formé, prudent ? Peu importe la cryptographie si son mot de passe est t0t0123 ou s'il est écrit sur un post-it.

Chapitre III

Signatures et PKI

La cryptographie asymétrique (rappel)

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.

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.

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 propre clé publique dans l'annuaire en prétendant m'appeler Alice ?

Construire un écosystème de confiance

- **Composants essentiels**
  - **PKI (Public Key Infrastructure) :** architecture de confiance
  - **Certificats X.509 :** lien de confiance entre une entité et sa clé publique, signée par une autorité de certification (CA)
  - **Chaîne de confiance :** validation récursive des certificats jusqu'à la racine (Root CA)
- **Mécanisme de révocation**
  - **CRL (Certificate Revocation List) :** Listes diffusées périodiquement. Problèmes : fraîcheur et volume.
  - **OCSP (Online Certificate Status Protocol) :** Interrogation en temps réel de l'état d'un certificat.
    Problèmes : centralisation et vie privée.
  - **OCSP Stapling :** Le serveur web agrafe une réponse OCSP signée à son certificat pour améliorer les performances et la confidentialité

Modèle de sécurité formel : EUF CMA

- **Acteurs :** challenger C (honnête) et adversaire A
  - **Setup :** C génère une paire de clés (pk,sk) et donne pk à A
  - **Oracle d'accès :** A peut demander à C de signer un nombre polynomial de messages (choisis par A). C retourne les signatures σ_i des messages m_i.
  - **Forge :** A produit une paire m, σ
- **Conditions de victoire :** A gagne le jeu si
  - Verify(pk,m, σ) renvoie True (σ est une signature valide de m)
  - m ne fait pas partie des m_i (sinon trop facile et inutile)

Un schéma de signature est **EUF-CMA** si la probabilité de victoire de tout adversaire A en temps polynomial est négligeable.

EUF CMA = Existential UnForgeability under Chosen Message Attack : un adversaire ne peut pas forger (falsifier) une signature même avec des signatures valides pour des messages de son choix. C'est la notion de sécurité standard pour les schémas modernes.

Exemples de signatures modernes

La cryptographie moderne repose (en partie, pas uniquement) sur l'existence de fonction à sens unique qui elles-mêmes proviennent de problèmes mathématiques difficiles à résoudre.
Quelques exemples :

- DSA (factorisation), standard en cours d'abandon
- ECDSA (logarithme discret), la référence actuelle
- Nouveaux standards post-quantiques :
  - ML-DSA aka Dilithium (réseaux euclidiens), standard FIPS 204
  - SLH-DSA aka SPHINCS+ (fonctions de hachage), FIPS 205
  - FN-DSA aka Falcon (réseaux structurés), futur FIPS 206
  - Et d'autres à venir sans doute.

Pourquoi de nouveaux standards ? Pour lutter face à la menace de l'ordinateur quantique potentiellement capable de calculer facilement des factorisations et des logarithmes discrets.

Pourquoi ECDSA ?

Pour la factorisation et le logarithme discret sur les corps finis, on connaît des attaques dites sous-exponentielles. Pour le logarithme discret sur les courbes elliptiques, la meilleure attaque est en $O(\sqrt{p})$ où p est la taille du corps de base.

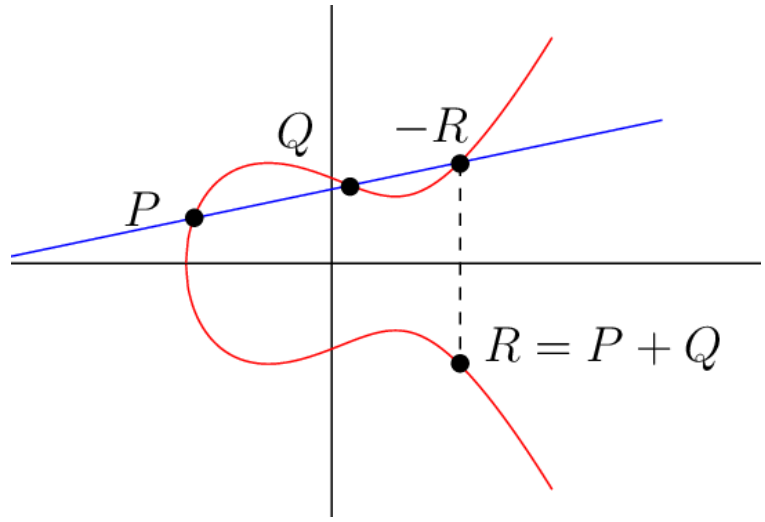
Conséquences :

- Pour un même niveau de sécurité, les courbes elliptiques offrent un gain de taille d'un facteur 10 environ (typiquement 256 vs 2048 bits)
- L'arithmétique étant plus simple, les calculs sont également plus rapides (à nuancer selon les implémentations)

Courbes elliptiques

Définition : une courbe elliptique E est définie par une équation de la forme $y^2 = x^3 + ax + b$ sur $\mathbb{F}_{p^n}$ (avec $p > 3$ et $4a^3 + 27b \neq 0$).

On note $E(\mathbb{F}_q)$ l'ensemble des points de E sur $\mathbb{F}_q$. C'est un groupe abélien avec une opération d'addition décrite par la figure.



- Le neutre est le point à l'infini
- Si $P = (x, y)$, $-P$ est $(x, -y)$

On prend un point $G \in E(\mathbb{F}_q)$ qui engendre un groupe cyclique dont l'ordre possède un grand facteur premier.
Pourquoi ? L'algorithme de Pohlig-Hellman montre que le DLP sur $\mathbb{Z}/\prod_i p_i^{n_i}\mathbb{Z}$ est aussi dur que le DLP dans chacun des $\mathbb{Z}/p_i\mathbb{Z}$.

Paramètres pour ECDSA

Créer une instance ECDSA c'est choisir les paramètres (q, a, b, G, n, h) suivants :

- q la taille du corps $\mathbb{F}_q$ (en général on prend q premier)
- a et b les coefficients de la courbe E, choisis principalement pour assurer que le G ci-dessous existe
- $G \in E(\mathbb{F}_q)$ engendrant un groupe d'ordre ayant un grand facteur premier (sécurité vs Pohlig-Hellman)
- n l'ordre de $\langle G \rangle$, pour simplifier on préfère $n < q$
- h le cofacteur tel que $\#E(\mathbb{F}_q) = nh$, on préfère $h = 1$ pour simplifier l'implémentation

Remarques : le choix d'une courbe est sensible !
Pour optimiser les opérations, on choisit parfois des q particuliers comme $q = 2^{255} - 19$ (c'est un nombre premier) et on préfère parfois avoir $h = 8$ ou 16 pour accélérer l'addition sur E.

Génération de clés

On dispose d'une instance ECDSA issue d'un standard fiable, donc d'un sextuplet (q, a, b, G, n, h).

- On tire un entier d au hasard (uniformément) dans $[1, n-1]$, ce sera la **clé secrète**
- On calcule un point P de E en faisant l'opération $P = d \times G = G + G + \cdots + G$, n fois. Ce sera la **clé publique**

Fonction à sens unique : calculer $P = dG$ à partir de d et G est facile (addition) mais retrouver d à partir de P revient à résoudre le problème ECDLP.

Signature d'un message

Objectif : prouver que l'on connaît d, mais sans le révéler !

- Calculer $h = \mathrm{HASH}(m)$ un haché de taille n
- Générer un **nonce secret** $1 \leq k \leq n-1$
- Calculer $kG = (x_1, y_1)$
- Convertir G en entier $r = x_1 \bmod n$ (lie le nonce à la signature)
- Calculer $s = k^{-1}(h + rd) \bmod n$ (mélange le message, la clé secrète et le nonce)

Partant de la clé publique P et de ma signature (r, s), comment savoir si le message m est authentique sans connaître ni k ni d ?

Idée : pour d ça ira car on connaît au moins $dG = P$ la clé publique. Mais pour k ?

Vérification d'une signature ECDSA

Essayons d'exprimer k en fonction de données connues. Par définition, on a $sk = h + rd \bmod n$, donc $k = s^{-1}(h + rd)$.

On ne connaît pas d mais on connaît $dG = P$, la clé publique !

Ainsi $kG = s^{-1}(hG + rdG) = s^{-1}(hG + rP)$, avec toutes les informations publiques on connaît kG même sans connaître k.

Algorithme de vérification

Entrée : signature (r, s), clé publique P (et les paramètres de E)

- Calculer $w = s^{-1}$ puis $u_1 = hw$ et $u_2 = rw$
- Calculer le point $R = u_1 G + u_2 P$ en déduire son abscisse x_R
- Si $x_R \bmod n = r$ alors retourner "vrai", sinon retourner "faux"

Encore une histoire de nonce !

Le nonce k doit être unique, imprévisible et secret :

- Secret : car l'équation $s = k^{-1}(h + rd) \bmod n$ permet de retrouver d si on connaît k.
- Unique : l'équation ci-dessus avec s_1 et s_2 entraîne $k = (s_1 - s_2)^{-1}(h_1 - h_2)$ et permet ensuite de calculer d.
- Imprévisible ? C'est plus subtil.

Nonce biaisé : si k est généré de manière prévisible on peut utiliser des techniques permettant de brute-forcer la clé.

Exemple : dans Bitcoin, la moitié des bits de k provenaient de h et l'autre moitié de la clé privée d.

Plus généralement si les premiers bits de deux nonces sont identiques, on peut en déduire les premiers bits de la clé.

Des nonces déterministes (RFC 6979)

Problème : générer un aléa de qualité n'est pas toujours simple, et peu coûter cher pour des applications de type IoT.

Idée : Calculer k à partir de h et d, mais sans faire la même erreur que Bitcoin ! Par exemple en calculant $k = \mathrm{HMAC}(d, h)$, car la fonction HMAC rend k imprévisible, même si on ne change qu'un bit de h.

Remarques :

- Plus besoin d'aléa externe pour signer (mais quand même besoin d'aléa pour générer d initialement)
- Le seul moyen de réutiliser k c'est de signer deux fois le même message, mais ça ne donne aucune info : dans ce cas $s_1 = s_2$ donc on ne peut pas écrire $k = (s_1 - s_2)^{-1}(h_1 - h_2)$

Une attaque : jailbreak PS3 en 2010

Contexte : Sony utilisait ECDSA pour signer ses éxécutables et contrôler le code qui tournait sur la PS3.

Problème : l'implémentation d'ECDSA ne respectait pas les règles de sécurité en matière de nonce, il a donc été possible de retrouver la clé secrète de Sony et donc de faire tourner n'importe quoi sur la PS3.

Cause profonde : ce n'était pas un bug, le code utilisait un nonce k codé en dur.

Recommandation : les implémentations de primitives cryptographiques doivent être développées ou a minima auditées par des gens qui s'y connaissent !

Autres type d'attaque

- Attaques par canaux auxiliaires (rayonnement EM, temps d'éxécution, consommation électrique). Cible de choix = le calcul rapide de dG s'il est fait naïvement.
- Attaques par fautes : on fait manipuler au serveur des données truquées (signer le message 0, utiliser un point P qui vient d'une autre courbe, etc)
- Attaques sur le générateur d'aléa : pour forcer deux nonces identiques ou très proches

Important : le grand nombre d'attaques ne signifie pas qu'ECDSA est un mauvais algorithme, sa grande popularité justifie que de nombreux chercheurs et hackers s'y intéressent. Les failles sont exclusivement liées à des bugs ou des mauvais usages.

Exemple de mauvaises pratiques

- **Réimplémenter soi-même** la crypto ou l'algèbre sous-jacente. Risque = bugs ou vulnérabilité side-channels
- **Ignorer les standards**, générer sa propre courbe. Risque : tomber sur une courbe où ECDLP est facile.
- **Ne pas valider** les paramètres. Risques = attaques par courbe invalide, choix de $d = 0$ ou $s = 0$
- **Mal gérer les secrets**. Risques : réutiliser la clé pour deux usages, avoir la clé dans le code source ou dans un répertoire non chiffré
- **Tronquer les signatures ou les hash**. Cela réduit les garanties de sécurité.

Choisir une courbe elliptique

Les courbes doivent venir d'un expert (beaucoup de critères mathématiquement subtils à garantir), souvent d'une institution.

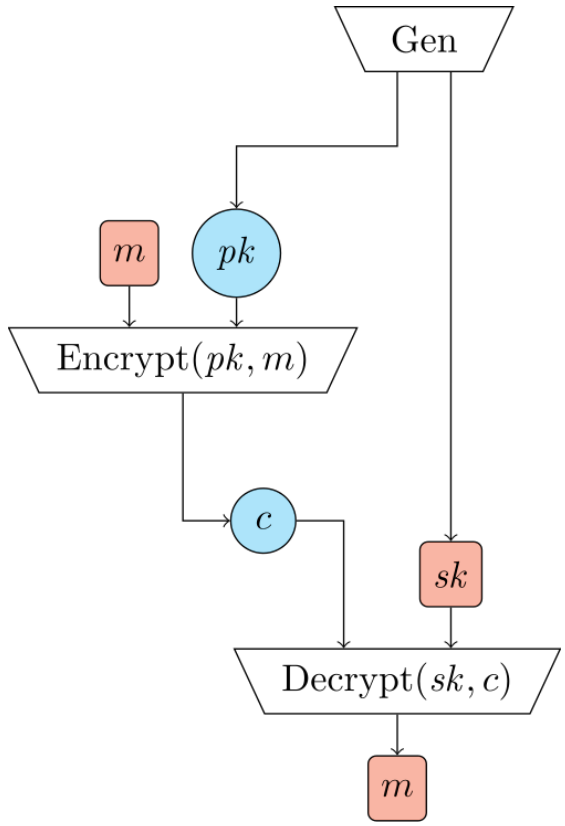
- Le NIST américain propose les courbes NISTp256, p384 et p521 qui sont les plus utilisées notamment sur le web. Attention : leur génération est opaque, on ne sait pas d'où viennent a, b et G. Il y a déjà eu des cas où certaines courbes ont été truquées (Affaire Dual EC-DRBG)
- Courbes Brainpool : standardisées par l'Allemagne et générées de manière plus transparente
- Courbe FRp256 de l'ANSSI, utilisée dans les cartes d'identité
- Courbe 25519, conçue par Daniel J. Bernstein, un cryptologue reconnu. Ses paramètres sont produits de manière transparente et elle est conçue pour résister à de nombreuses attaques, en se basant sur le retour d'expérience.

Encapsulation de clés (KEM)

Key Exchange (KEX) vs Key Encapsulation (KEM)

Diffie-Hellman : Alice et Bob échangent leurs clés publiques et utilisent leurs clés secrètes pour en déduire un secret partagé. Ce secret est conçu en combinant des informations de Bob et d'Alice.

Encapsulation de clés :
L'émetteur génère une clé K qu'il **encapsule** en utilisant de l'aléa et la clé publique du destinataire. Le destinataire **décapsule** la clé K en utilisant sa clé secrète.



Les routines d'un KEM

- **KeyGen**(alea_{kg}) utilise un aléa et renvoie la paire de clés privée/publique (sk, pk)
- **Encaps**(pk, alea_{e}) utilise la clé publique et de l'aléa, renvoie le chiffré ct et la clé de session Ks
- **Decaps**(sk, ct) utilise la clé secrète et le chiffré, renvoie la clé de session Ks

La personne qui encapsule n'a pas besoin de sa propre clé.
La personne qui décapsule n'a besoin d'aléa que pour **KeyGen**.

Sécurité IND-CPA d'un KEM

- **Acteurs :** challenger C (honnête) et adversaire A
- **Setup :** C génère une paire de clés (pk,sk) et donne pk à A
- **Choix des messages :** A produit m_0 et m_1 et les envoie à C. C tire un bit b au hasard et renvoie $ct = \mathrm{Encaps}(pk, m_b)$
- **Jeu :** A doit deviner la valeur de b

Avantage : l'avantage de A est la différence entre la probabilité de succès de A et $1/2$ (cas où A choisit au hasard).

Un KEM est **IND-CPA** si l'avantage de tout adversaire A en temps polynomial est négligeable (inférieur à $2^{-\lambda}$).

IND CPA = Indistinguishability under Chosen Plaintext Attack : un adversaire polynomial ne peut pas distinguer un chiffré. C'est une notion de sécurité standard pour les schémas modernes.

Sécurité IND-CCA d'un KEM

- **Acteurs :** challenger C (honnête) et adversaire A
- **Setup :** C génère une paire de clés (pk,sk) et donne pk à A
- **Oracle de chiffrement/déchiffrement 1 :** A peut soumettre un nombre polynomial de requêtes de (dé)chiffrement.
- **Choix des messages :** A produit m_0 et m_1 et les envoie à C. C tire un bit b au hasard et renvoie $ct = \mathrm{Encaps}(pk, m_b)$
- **Oracle de chiffrement 2 :** A peut soumettre un nombre polynomial de requêtes de chiffrement.
- **Jeu :** A doit deviner la valeur de b

Avantage : l'avantage de A est la différence entre la probabilité de succès de A et $1/2$ (cas où A choisit au hasard).

Un KEM est **IND-CCA1** si l'avantage de tout adversaire A en temps polynomial est négligeable (inférieur à $2^{-\lambda}$). On parle de **IND-CCA2** (adaptative CCA) si l'attaquant peut aussi faire des requêtes de déchiffrement avant de deviner b (dans l'étape 2).

IND CCA = Indistinguishability under Chosen Ciphertext Attack.

Relation entre KEM et PKE

Les deux notions sont proches on peut passer de l'une à l'autre :

- PKE $\rightarrow$ KEM : il suffit de générer une clé au hasard et de chiffrer cette clé (la clé devient le message)
- KEM $\rightarrow$ PKE : il suffit d'utiliser Ks dans un chiffrement authentifié pour chiffrer le message.

La notion de KEM est plus moderne et préférée, notamment dans le contexte de cryptographie **hybride** mélangeant un KEM classique et un KEM post-quantique via un **KEM combiner**.

KEM combiner : mécanisme dérivant une clé unique K à partir des Ks de deux KEM, de sorte qu'il soit impossible d'obtenir K sans avoir **les deux** Ks.

La menace quantique

Rappel : en cryptographie la sécurité est *calculatoire*, on résiste à un attaquant polynomial. Cette notion est évidemment dépendante des outils de l'attaquant.

Nouvelle menace : en 1995, Peter Shor propose un algorithme **polynomial** permettant de factoriser et de calculer des logarithmes discrets. Cela remet en cause la sécurité des schémas actuels (RSA, (EC)DSA, (EC)DH).

Et la cryptographie symétrique ? L'algorithme de Grover divise par deux le niveau de sécurité, il suffit de doubler la taille des clés pour s'en prémunir (le NIST considère que ce n'est pas une menace en pratique).

La cryptographie post-quantique

La compétition du NIST de 2015 : le NIST a lancé un appel pour standardiser des alternatives (KEM et signatures) ne reposant ni sur le logarithme discret ni sur la factorisation.

Exemples de problèmes difficiles :

- Calculer un vecteur court dans un réseau (ML-KEM, ML-DSA, FN-DSA)
- Inverser des fonctions de hachages (SLH-DSA, XMSS)
- Décoder des codes aléatoires (HQC, WAVE)
- Résoudre des équations quadratiques (UOV, MAYO)
- et quelques autres...

Un changement de paradigme

Attention : La transition post-quantique ne se limite pas à remplacer chaque algorithme par un analogue post-quantique

Pas de solution idéale : compromis entre

- Performance (vitesse, taille, contraintes hardware)
- Résistance aux attaques side-channel, facilité d'implémentation
- Maturité et confiance

Défis politiques :

- Faire un inventaire complet de la cryptographie (pas si facile)
- Identifier des cas d'usages prioritaires (confidentialité $>$ authenticité)
- Pour chaque cas, prendre un remplaçant adapté

Quel algorithme utiliser ?

Règle générale: réseaux structurés

- Kyber aka ML-KEM (NIST FIPS 203) pour chiffrer
- Dilithium aka ML-DSA (NIST FIPS 204) pour signer

Options de secours / plus fortes :

- HQC (codes) futur standard NIST, compétitif
- FrodoKEM (non structuré), moins compétitif mais simple à implémenter, soutenu par BSI et ANSSI
- SPHINCS+ aka SLH-DSA (hash-based), NIST FIPS 205, énorme mais forte sécurité

Cas d'usage spécifiques :

- Falcon (future FIPS 206) si bande-passante limitée, mais risque implementation / side-channel
- XMSS (hash-based) pour signer du software/firmware

Principales menaces contre le post-quantique

- Récolte et déchiffre (vise la crypto classique et tout algorithme post-quantique mal pensé ou mal implémenté). Pas de remède une fois le mal fait.
- Manque de maturité scientifique : deux candidats prometteurs au NIST (dont un finaliste) cassés en quelque jours avec un PC **classique**.
- Manque inquiétant d'attaques quantiques : beaucoup de cryptanalyse depuis 2015 mais $> 90\%$ classique. Pourtant la compétition du NIST vise la résistance au quantique...

Défense en profondeur :

- Hybride classique + post-quantique (pour un temps)
- Crypto-agilité pour faire des patches rapides
- Autre type de garanties : hybride quantique & post-quantique