Récap Cryptographie

Confidentialité

Garantit qu'une partipant ne connaissant pas la clé de déchiffrement **n'apprend aucune information sur le message clair**.

Chiffrement symmétrique

`Chiffrement(clé, IV, clair) = chiffré`
`Déchiffrement(clé, IV, chiffré) = clair`

Contraintes

`clé`	générée par un générateur d'aléa **cryptographiquement sûr** OU dérivé à partir d'un mot de passe ou d'un secret Diffie-Hellman par une **fonction de dérivation de clé**
`IV`	**jamais réutilisé** pour chiffrer **deux messages avec la même clé**, **imprédictible** si mode CBC utilisé, peut être séquentiel sinon

Modes de chiffrement par bloc

`ECB`	chiffre chaque bloc indépendamment, **à proscrire**
`CTR`	génère un flux pseudo-aléatoire comme du chiffrement par flux
`CBC`	effectue un XOR entre le chiffré du bloc précédent et le bloc suivant. IV **imprédctible** impérativement

Conseillés : aucun, préférer le chiffrement authentifié.

Chiffrement asymétrique

`Chiffrement(clé_publique, clair) = chiffré`
`Déchiffrement(clé_privée, chiffré) = clair`

⚠️ Choisir un padding adapté (OAEP ou sinon PKCS v1.5) avant chiffrement.
Deux choix : RSA (plus répandu) ou El-Gamal

Intégrité

Garantit qu'un participant ne connaissant pas la clé de génération de motif d'intégrité **ne peut pas émettre de messages valides**.

Chiffrement intègre

`Chiffrement(clé, IV, clair) = tag||chiffré`
`Déchiffrement(clé, IV, tag, chiffré) = clair ou erreur de déchiffrement`

⚠️ Pour certaines bibliothèques, le chiffré contient déjà le tag pour le déchiffrement

⚠️ Certaines bibliothèques exposent la possibilité de protéger en intégrité des donnés en clair, qui doivent être fournies **à l'identique** lors du déchiffrement.

Algorithmes de chiffrement intègre

`AES-GCM`	Combine un mode AES-CTR avec une garantie d'intégrité. L'IV est sur 96 bits et doit être unique. Si beaucoup de données à chiffrer : utiliser des IVs successifs, sinon générer aléatoirement
`ChaCha20-Poly1305`	Chiffrement par flot avec une garantie d'intégrité. Mêmes contraintes que `AES-GCM`
`XSalsa20`	Variante avec un IV de 192 bits, adapté si IV générés aléatoirement

(H)MAC (symétrique)

`Authentifier(clé, message) = tag`
`Vérifier(clé, message, tag) = oui / non`

Signature (asymétrique)

`Signer(clé_privée, message) = signature`
`Vérifier(clé_publique, message, signature) = oui / non`

⚠️ Choisir un padding adapté (PSS ou sinon PKCS v1.5) si signature RSA.

⚠️ ECDSA requiert en plus un IV unique. Il ne doit pas être réutilisé.

⚠️ Choisir une fonction de hachage adaptée avant signature et potentiellement padding (SHA2, sinon SHA3 ou Blake2 ou Blake 3, avec au moins 256 bits de sortie).

Deux choix : ECDSA (conseillé) avec une courbe sûre, sinon RSA-2048 minimum.

Stockage de mot de passe

`Dérivation(sel, mot_de_passe) = condensat`
`Vérification(sel, condensat, mot_de_passe) = oui / non`

⚠️ Le sel doit être différent pour chaque mot de passe à stocker

⚠️ Mêmes fonctions pour la dérivation de clé (sans sel obligatoire)
Conseillés : argon2id, sinon balloon, sinon PBKDF2

Partage de secret Diffie-Hellman

`Dérivation`$(K^1_{\text{pub}}, K^2_{\text{priv}})$ = **`Dérivation`**$(K^2_{\text{pub}}, K^1_{\text{priv}})$

Conseillé : Diffie-Hellman sur courbe elliptique (X25519, P521, P384, P-256), sinon sur corps premier suffisamment grand (1024 bits minimum)