Séance 11 et 12 - Travaux Dirigés et Pratiques
Signatures et Certificats

Yann Rotella

2026

Questions et Réponses

C'est la dernière séance de travaux dirigés. Bien que toute l'année a pu être utile pour des questions et des réponses. C'est votre dernier moment pour répondre et avoir des précisions sur l'ensemble du cours, c'est à dire :

1. Histoire de la cryptographie (jusqu'à 1950)
2. Théorie de Shannon, concept de sécurité
3. Définitions de Sécurité et Primitives
4. Stratégies de Conceptions (des Chiffrements par Bloc)
5. Cryptanalyse des Chiffrements par Bloc
6. Fonctions de Hachage Cryptographiques
7. Modes Opératoires, MACs et AEAD
8. Arithmétique, Diffie-Hellman, El-Gamal et Réductions
9. Arithmétique, RSA, RSA-OAEP
10. Signatures numériques et authentification
11. Certificats, protocoles et confiance
12. Aperçu de Cryptographie Avancée

Signatures : théorie

Exercice 1. *Attaque sur la signature RSA seule.*
Supposons que n soit un entier produit de deux nombres premiers distincts p et q. Soit e un entier premier avec $\phi(n)$. Alice utilise la signature RSA avec (n, e) comme clé publique. On note d sa clé privée.

(1) Oscar récupère les signatures valides s_1 et s_2 de deux messages m_1, $m_2 \in \mathbb{Z}_n$, signés par Alice. Montrer comment Oscar peut construire la signature valide d'un autre message.

(2) Montrer comment Oscar peut construire un message (possiblement sans sens) et sa signature valide, sans interaction avec Alice.

(3) Expliquer comment empêcher cela.

Exercice 2. *Aspects calculatoires.*
Un aspect important à étudier dans le cas des signatures numériques est la quantité de calcul nécessaire pour (i) signer et (ii) vérifier une signature. On étudie ici la complexité calculatoire du protocole de la signature RSA.

(1) Combien de multiplications avons nous besoin en moyenne afin de (i) signer un message avec un exposant aléatoire et (ii) vérifier la signature avec l'exposant $e = 2^{16}+1$. On suppose que le module n est de ℓ bits et que l'algorithme *square and multiply* est utilisé pour l'exponentiation modulaire. Dériver des expressions générales en fonction de ℓ.

(2) Quelle action est plus longue, la signature ou la vérification ?

(3) On dérive maintenant des estimations pour la vitesse des implémentations logicielles. On adopte le modèle suivant : L'ordinateur opère sur des données de 32 bits. Par conséquent, chaque variable, en particulier le module n et x (base de l'exponentiation) sont représentés comme des chaînes de $m = \lceil \ell/32 \rceil$ éléments. On suppose qu'une multiplication ou un élévation au carré de deux de ces variables modulo n nécessitent m^2 unités de temps (une unité de temps est un cycle d'horloge multipliée par une constante plus grande de 1 qui dépend de l'implémentation). Noter qu'on ne multiplie jamais avec les exposants d et e. Ceci signifie, que la longueur en bits de l'exposant n'influence pas le temps d'une opération individuelle (multiplication ou élévation au carré). Combien de temps a-t-on besoin pour effectuer ou vérifier une signature si l'unité de temps d'un ordinateur est 100 nsec et n est de 512 bits ? Combien de temps ça prend si n est de 1024 bits ?

Exercice 3. *Confidentialité et authenticité.*
Supposons qu'on souhaite protéger des regards indiscrets le contenu du message signé. Expliquer comment atteindre la confidentialité du message en prouvant en même temps son authenticité. Décrire en détail les calculs à effectuer en cas d'utilisation du système RSA.

Application de la cryptographie

Nous avons joué avec plusieurs algorithmes, nous en avons implémenter une partie, mais tous ceux-ci sont déjà implémentés dans des librairies !

La cryptographie est un sujet sensible il ne faut pas l'utiliser sans avoir un code bien propre, vérifié et testé. On privilégiera la cryptographie des librairies déjà implémentée !

GPG et Open SSL

GPG

GNU Privacy Guard est une implémentation publique du standard OpenPGP (RFC 4880), il est principalement utilisé pour le chiffrement et déchiffrement des messages et garantir authenticité et intégrité, en utilisant toutes les méthodes vues dans ce cours.

OpenPGP a notamment gagné en popularité suite aux révélations de Snowden (celui-ci chiffrait ses communications avec des journalistes avec ce type de cryptographie).

(1) Envoyez un message chiffré en utilisant GPG à votre chargé de TD.

OpenSSL

`SSL` pour Secure Socket Layer (et maintenant renommé `TLS` - Transport Layer Security - pour les récentes versions) a été intialement développé par Netscape Communications Corporations pour permettre une connexion sécurisée client - serveur. `TLS` correspond au `s` de `https` le protocole suit les grandes lignes vues en cours : une poignée de main (handshake) avec possibilité de vérfier un certificat, un protocole d'échange de clefs (associées aux certificats) pour générer une clef de session, et ensuite une communication symétrique avec un mode AEAD.

Dans `SSL` les certificats suivent la norme X509 décrite en cours.

`OpenSSL` est une boîte à outils cryptographique qui contient tous vos algorithmes préférés ainsi que le protocole décrit ci-dessus avec une interface en ligne de commande de votre terminal. Il est donc en particulier réalisable de chiffrer, déchiffrer avec des schémas asymétriques, générer des clefs, des certificats,

chiffrer déchiffrer et authentifier avec des chiffrements par bloc et des modes opératoires bien connus (et sécurisés !)

(2) Vérifie que tu as `OpenSSL`. Passe en ligne de commande
`man openssl`
Regarde les différentes commandes. Tu les connais ? Va jusqu'à la partie « Encryption, Decryption and Encoding Commands ».

OpenSSL et cryptographie symétrique

(3) Tu veux chiffrer tous tes fichiers super secrets, mais les garder sur ton ordinateur ? As-tu besoin de chiffrement authentifié ? Choisis un bon algorithme et un mode et garde la commande associée.

(4) Tape

```
openssl enc -ALGO -in CLEAR_FILE -out ENCRYPTTED_FILE
```

Que te répond `OpenSSL` ? Peux-tu expliquer ?

(5) Si tu veux une vraie sécurité, de toute façon, il faut une clef et non un mot de passe. `OpenSSL` va ici `https://docs.openssl.org/master/man1/openssl-rand/` et génère une clef secrète de 256 bits dans un fichier nommé secretkey !

(6) `openssl enc -aes-256-ctr -in CLEAR_FILE -out ENCRYPTTED_FILE -K YOUR_SECRETKEY_IN_HEXA`
Que se passe t'il ? Trouve une solution pour chiffrer.

(7) Pour déchiffrer il faut faire l'option `-d` : déchiffre ton fichier chiffré dans un autre fichier, puis vérifier que tout est bon en faisant la commande

```
diff FILE1 FILE2
```

Génération d'une paire de clés RSA

On va générer des clés RSA. Des commandes sont déjà toutes faites et produisent des clés dans un fichier en format PEM (Privacy Enhanced Mail). Attention : ton système d'exploitation peut être embêtant avec les extensions de ce type (et te demande un mot de passe à chaque fois pour voir ces fichiers. C'est normal : ce sont des clefs !

(8) `openssl genrsa -out myKey.pem SIZE`
Pense à choisir une taille cryptographiquement adaptée : Attention, il s'agit de RSA et non d'AES !

(9) Ouvre ton fichier. Ce n'est pas très lisible et puis on ne voit pas grand chose...

```
openssl rsa -in myKey.pem -text
```

(10) Pourquoi l'exposant public e dans RSA est ici 65537 ?

(11) Si tu as fait l'exercice 6 du TD 8, tu peux dire à quoi correspondent `exponent1`, `exponent2` et `coefficient` ?

(12) Il est possible de chiffrer sa clef secrète au moyen d'une clef symétrique (dérivée par un mot de passe avec une KDF). Pour cela réalisez la commande

```
openssl rsa -in myKey.pem -aes128 -out myKey.pem
```

(13) Vérifie que tu peux revoir ta clef en tapant la commande précédente d'affichage.

(14) Dans ta clef, tu as les choses publiques et privées mélangées, mais ta clef publique doit être exportée.

```
openssl rsa -in myKey.pem -pubout -out publicKey.pem
```

(15) Regarde ta clef avec la commande précédente en rajoutant l'option `-pubin`

Signatures avec ECDSA

Comme vu en cours, pour signer des « gros »fichiers et pour éviter une malléabilité des chiffrements et déchiffrements asymétriques, on utilise le paradigme *Hash and Sign*. Celui-ci est inclu en option, selon l'algorithme asymétrique dans la commande `pkeyutl`.

Par ailleurs, nous avons mentionné que les algorithmes de type DSA (ou ElGamal) sont en réalité sur des groupes provenant de courbes elliptiques (hors programme). Ceci amène à l'utilisation aujourd'hui d'ECDSA (pour Elliptic Curve DSA). Nous n'allons pas rentrer dans les détails, mais il s'agit d'un algorithme recommandé.

(16) Commence par taper la commande suivante :

```
openssl ecparam -list_curves
```

Que vois-tu ? Que veux dire prime field d'après toi ? On va utiliser `sect571r1`

(17) Génère ta clef :

```
openssl ecparam -genkey -out ecPrivateKey.pem -name sect571r1
```

(18) Sors ta clef publique :

```
openssl ec -in ecPrivateKey.pem -pubout -out ecPublicKey.pem
```

(19) Signe un fichier avec la commande `openssl pkeyutl` dont la description peut être trouvée ici : `https://docs.openssl.org/master/man1/openssl-pkeyutl/`

(20) Vérifie ton fichier. Il faudra rajouter l'option `-sigfile` et `-pubin`.

(21) Récupère les fichiers suivants : `confiance1.txt` et `confiance2.txt`, associés avec des signatures `sign1` et `sign2`. Ainsi que la clef publique `publicKeyRotella.pem`. Quel message est signé par ma clef publique ?

Certificats

On va maintenant établir un certificat en « jouant » le rôle factice d'une autorité de certification. Un certificat associe des données d'identification et une clef publique.

(22) Récupère le fichier `req.cnf` puis réalise une requête en faisant la commande

```
openssl req -config req.cnf -new -key ecPrivateKey.pem -out maRequete.pem
```

(23) Consulte la requete avec la commande suivante :

```
openssl req -config req.cnf -in maRequete.pem -text
```

(24) Regardez tout ce qu'il y a dans cette requête. La clé privée est-elle présente ?

Pour éviter de démultiplier les clefs secrètes, nous allons simuler l'autorité. L'autorité sera alors jouée par votre chargé de TD. Le certificat de l'autorité est celui de votre enseignant préféré.

(25) Visualisez le certificat avec

```
openssl x509 -in CACertif.pem -text
```

Que voyez-vous ? Qui a signé ce certificat ? Qui est le sujet ?

(26) Votre chargé de TD vous transmet un nombre aléatoire à signer avec la clef publique que vous voulez faire avlider par l'autorité. Transmettez votre requête par mail, avec la clef publique ainsi que la signature d'un fichier texte classique contenant le nombre aléatoire donné par votre chargé de TD.

(27) Donnez votre certificat à quelqu'un d'autre du TD et demandez-lui de vérifier qu'il est correct avec la commande suivante.

```
openssl verify -CAfile CACertif.pem monCertif.pem
```

Vous voilà certifié en cryptographie par votre enseignant !
Bonnes révisions !
Puis bonnes vacances d'été !