Rapport Projet Cryptographie – Algo vérification Certificat

By : Emmanuel Bert & Duncan Sayn

1°) Introduction :

Par définition, les certificats ont été créés pour répondre à un besoin de sécurisation des communications entre un client et un serveur web. Leur vérification est donc nécessaire pour assurer une communication de confiance.

Les certificats sont gérés sous forme de chaine de certificats, dont chaque entité est signée par l'entité qui l'a précédé. Cela permet d'établir une relation de confiance entre le haut de la chaine, comme une autorité de certification, et le bas de la chaine, souvent un serveur web ou une application.

Mais pour cela, des vérifications doivent être faites comme :

- L'analyse de l'authenticité de chaque certificat de la chaine, en faisant une vérification de l'entité qui a émis le certificat et en vérifiant la validité de sa signature.
- L'analyse de l'intégrité de la chaine de certificats, cela en vérifiant les signatures numériques de chaque certificat pour garantir que rien n'a été altéré ou falsifié depuis l'émission du certificat.
- L'analyse de la validité de la chaine de certificat, en vérifiant la date de début et de fin de validité puisque chaque certificat a une durée de validité limitée. Cela permet lors de la redemande de certificat de vérifier que le client est toujours digne de confiance.
- L'analyse de la chaine permet aussi de reconnaitre une autorité de confiance ou une ancre de confiance, pour établir une confiance utilisée pour la sécurité des communications en ligne.
- On peut aussi surveiller les révocations de certificats car ces derniers peuvent être révoqués avant leur fin de validité, dans le cas d'un quelconque incident de sécurité par exemple. Cela est fait en analysant les CRL, ou listes de révocations de certificats, ou encore des OCSP, soit les points de distribution de certificats révoqués. Ce pour éviter l'utilisation des certificats de façon non-autorisée.

- De plus, faire des vérifications de chaine de certificats permet d'éviter le Man-in-the-Middle.

Avec ces vérifications, on a donc l'assurance de garantir la confiance, l'authenticité et la sécurité des communications et des transactions en ligne.

C'est la raison pour laquelle il est intéressant de se lancer dans un projet de ce genre. Voici ce que nous avons fait pendant le temps que nous avions pour le réaliser.

II°) Le choix des outils

Le choix du langage :

Tout d'abord, nous avons choisi de réaliser notre algorithme en Python, dans l'idée de nous simplifier la compréhension des ressources que nous avions, dû au fait de notre plus grande expérience avec le python que le Java, proposé par défaut pour ce pojet. Néanmoins, nous nous sommes vite confrontés à un obstacle de vérification de la clef public du certificat.

Dès lors, et d'après les conseils de camarades qui ont eu plus de facilité à gérer ça, nous avons décidé de redémarrer le projet en java, dans l'idée que les ressources que nous pouvions utiliser pour faire ce projet nous étaient fournies et aussi car nous pourrions demander de l'aide au professeur qui acceptait de nous supporter à condition de l'utilisation de ce langage. C'est en arrivant à faire plus que nous sommes resté sur du Java pour le reste du projet.

Le choix des outils :

Pour réaliser ce projet, nous avons commencé par travailler avec Virtual Studio Code, l'IDE que nous utilisons le plus. Mais très vite, et en parti en passant sur Java comme langage de développement, nous avons changé d'IDE pour Intellij IDEA. C'est principalement au besoin d'utiliser la librairie Bouncycastle que nous avons changé.

Le choix des librairies :

Nous avons utilisé BouncyCastle comme bibliothèque supplémentaire dans notre algorithme ValidateCertChain.

III°) Etapes réalisées du projet :

Nous avons pu réaliser l'ensemble des étapes du projet dans notre algorithme à l'exception de la vérification de signature ECDSA et des étapes bonus.

Pour chaque parties, la partie 1 :

- Nous avons tout d'abord pris en compte le nombre d'argument en entrée, en vérifiant qu'il n'y en avait uniquement que 2. Sinon, on indique :

```
Usage: validate-cert <format> <certfile>
```

Et quittons le programme. Sinon, nous le continuions.

Ensuite, nous vérifions si le deuxième argument est un DER ou PEM. Si l'on a du DER, nous chargions le certificat. SI nous avions un PEM, nous le chargions aussi avec une conversion de PEM vers DER. Si nous nous trouvions dans un autre cas, alors nous indiquions à l'utilisateur :

```
Format de certificat non valide
```

- Ensuite, nous récupérons la clef publique grâce à la méthode getPublicKey().
  Puis, nous vérifions la signature grâce à la méthode verifiy().
  Si tout à bien fonctionné, nous devrions avoir comme retour :

```
La signature du certificat a été vérifiée avec succès.
```

Si ce n'est pas le cas, alors une erreur sera retournée en fonction de ce qui n'a pas été et le programme est fermé.

- Dès lors, nous affichons le sujet et l'émetteur avec les méthodes getSubjectX500Principal() pour le sujet et getIssuerX500Principal() pour l'emetteur.
- Ensuite, pour vérifier l'extension KeyUsage, on utilise la méthode getKeyUsage().
  Pour tester le resultat, on regarde si la keyUsage existe et est positive.
  Si c'est le cas, on retourne

```
Extension KeyUsage présente.
```

Sinon, on indique

```
Extension KeyUsage non présente.
```

- Puis, nous vérifions la période de validité à l'aide de la méthode checkValidity().
  Plus en détail, nous utilisons getNotAfter() pour vérifier la fin de validité et getNotBefore() pour la date de mise de validité. On la compare alors à la date actuelle récupérée grâce à la classe Date() dans une instance now. Dès lors, on teste si getNotBefore() de notre certificat est considéré avant la date actuelle grâce à la méthode before(). Si c'est le cas, est retourné à l'écran

```
Le certificat n'est pas encore valide.
```

Sinon, on vérifie alors si getNotAfter() de notre certificat est considéré après la date actuelle avec after() et si c'est le cas, on retourne à l'écran

```
Le certificat a expiré.
```

Si aucun des deux tests n'est validé, alors on retourne à l'utilisateur

```
Le certificat est valide.
```

- Tout d'abord, on extrait le nom de l'algorithme de signature avec getSigAlgName() que l'on indique ensuite à l'utilisateur via ce message

```
- Algorithme de signature : " + algorithm
```

algorithm étant notre variable contenant le résultat de getSigAlgName()
Puis, pour vérifier sa signature, on extrait la signature du certificat avec la méthode getSignature(). Ensuite, on crée une instance de la classe Signature. On initialise la vérification de la clef publique avec initVerify() pour indiquer quelle clef publiue utiliser pour le test. On mets alors à jour les données de l'objet Signature avec ce qu'on extrait de la partie To Be Signed à partir duquel à été calculé la signature. On utilise alors la méthode vérify() pour vérifier la signature extraite du certificat.
Dès lors, on vérifie si le retour est correct et dans ce cas on retourne

```
La signature du certificat est valide.
```

Sinon,

```
La signature du certificat est invalide.
```

Maintenant, voyons pour la partie 2 :

Tout d'abord, nous vérifions bien qu'au lancement du programme, avons bien le bon nombre d'arguments. Sinon, nous affichons

```
Usage: validate-cert-chain <format> <certfile1> <certfile2> ...
```

Et quittons le programme.

Ensuite, nous chargons chaque certificat dans une liste de type X509Certificate.

- Dès lors, nous commençons par vérifier la validité de la chaine en vérifiant la clef publique de chaque certificat et en s'assurant que le sujet du certificat est l'emetteur du certificat précédent. Si le moindre test ne passe pas, on retourne alors

```
La chaîne de certificats est invalide.
```

Si par contre, aucun echec n'est rapporté, alors on retourne

```
La chaîne de certificats est valide.
```

- Pour vérifier l'extension BasicConstraints, on récupère la valeur avec getExtensionValue(). Si l'on récupère quelque chose, alors on retourne

```
L'extension BasicConstraints est prÃ©sente dans le certificat.
```

Sinon, on aura

```
L'extension BasicConstraints est absente dans le certificat.
```

Enfin, pour la partie 3 :

Pour faire la vérification du status de révocation, nous extrayons d'abord le CRL avec notre méthode extractCrlDistributionPoints() en ne gardant que les données qui nous interessent. On teste alors si la révocation a eu lieu en utilisant la CRL. Si le résultat est vrai, on retourne

```
Le certificat a été révoqué
```

Sinon, on imprime à l'écran :

```
Le certificat n'a pas été révoqué.
```

IV°) Autres choix techniques et structure de programme

De manière générale, nous avons programmé chaque fonctionnalité sur une suite chronologique des évènements dans notre fonction principale. Quelques fois des fonctions spécifiques à une tâches on été utilisé pour alléger le contenu de la fonction principale. Mais dans la majorité des cas, c'est dans la fonction principale que l'on retrouve tous les retours de tests effectués.

V°) Les difficultés rencontrées

Nous avons, tout d'abord lors de notre phase de travail en Python, eu beaucoup de difficulté à comprendre la documentation que nous avions, ce qui souvent conduisait à ne pas réussir ce que nous essayions de faire.

Ensuite, dès que nous avons commencé à travailler en Java, nous avons eu quelques difficultés avec le langage n'étant pas encore bien familier avec.

VI°) Les axes d'amélioration

Tout d'abord, avec du temps en plus, nous aurions éventuellement pu terminer les éléments de la liste des livrables, bonus compris.

Ensuite, faute de temps, nous nous sommes concentrés sur la réussite des tests, en considérant qu'une erreur de l'algorithme signifiait que quelque chose n'avait pas été dans les vérifications de certificat. Evidemment, nous n'avons considéré cela que lorsque nous faisions nos tests sur les cas de certificats non sécurisés. Aucun des tests réalisés sur des certificats considérés sécurisés n'ont été laissé comme erreur de programme en ce qui concerne les tâches réalisées. Nous voyons donc comme autre axe d'amélioration une implémentation d'exceptions pour tous les cas du processus de vérification qui sont censé échouer.

VII°) Ressources utilisées

Pour réaliser ce projet, nous nous sommes principalement servi des propositions que nous faisait chat gpt. Cela nous a permis de facilité nos recherches puisqu'il nous proposait ce qui lui semblait le plus adapté et ensuite, cela nous a permis de progresser en Java car nous avons pris le temps de bien comprendre ce qu'il nous proposait pour compléter notre algorithme.

De plus, nous avons aussi collaboré avec d'autres étudiants lorsque nous rencontrions un obstacle qu'ils n'avaient pas rencontré et inversement, cela donnait l'occasion d'échanger des conseils, notamment lorsque nous avons décidé de nous orienter vers Java plutôt que Python pour réaliser ce projet.