VOLPELLIERE Anthony

DANIEL Aymeric

Introduction

L'importance et l'intérêt des différentes vérifications à faire sur une chaîne de certification résident dans la garantie de la sécurité et de la fiabilité des communications numériques. Une chaîne de certification, composée de plusieurs certificats numériques, établit une ligne de confiance entre l'émetteur et le destinataire, assurant ainsi l'authenticité, l'intégrité et parfois la confidentialité des données échangées.

La vérification de cette chaîne est cruciale pour :

- **Authentifier les parties en communication** : Elle permet de confirmer l'identité des parties impliquées, réduisant ainsi le risque d'attaques de type man in the middle où un attaquant pourrait se faire passer pour une personne de confiance.
- **Assurer l'intégrité des données** : En vérifiant que le certificat est valide et n'a pas été révoqué, on s'assure que les données n'ont pas été modifiées durant leur transmission.
- **Renforcer la confiance dans les transactions électroniques** : Les vérifications contribuent à un environnement numérique plus sûr, où les utilisateurs peuvent effectuer des transactions ou échanger des informations sensibles en tout sérénité.
- **Conformité aux normes et régulations** : Vérifier la chaîne de certification peut être exigence réglementaire pour protéger les données personnelles et sensibles des utilisateurs.

Choix d'Outils, Langage et Librairies

Pour ce projet, Java a été choisi comme langage de programmation principal pour sa riche bibliothèque standard, idéale pour le traitement des certificats numériques. La bibliothèque Bouncy Castle a été utilisée pour son support étendu des opérations cryptographiques, facilitant la manipulation avancée des certificats, la vérification des signatures et l'implémentation des protocoles de vérification de révocation comme OCSP.

Étapes Implémentées

- **Validation de la chaîne de certificats :** Le code implémente la logique pour charger une série de certificats et valider la chaîne, en s'assurant que chaque certificat est correctement signé par l'autorité de certification (CA) précédente dans la chaîne.

- **Vérification de la signature et de l'algorithme :** Pour chaque certificat, le code vérifie que la signature est valide et que l'algorithme de signature est sécurisé.

- **Période de validité :** Le code vérifie que chaque certificat dans la chaîne est dans sa période de validité.

- **Utilisation de la clé :** Le code vérifie les extensions d'utilisation de la clé pour s'assurer que le certificat peut être utilisé pour les objectifs prévus.

- **Points de distribution de la liste de révocation (CRL) :** Le code extrait et affiche les URL des points de distribution CRL pour chaque certificat.

- **Vérification CRL et OCSP :** Le code implémente la logique pour vérifier si le certificat a été révoqué en utilisant les listes de révocation de certificats (CRL) et le protocole de vérification de statut de certificat en ligne (OCSP).

Explication plus détaillé :

Vérification de la Période de Validité

La méthode `checkValidityPeriod` s'assure que le certificat est valide au moment de la vérification, en appelant `cert.checkValidity()`. Cette méthode lève une exception si le certificat a expiré ou s'il n'est pas encore valide. En cas de succès, elle affirme que le certificat est dans sa période de validité.

Affichage de l'Utilisation de la Clé

La méthode `checkAndDisplayKeyUsage` extrait et affiche les usages de la clé du certificat, tels que la signature numérique, le chiffrement de clé, etc. Cela est réalisé en récupérant un tableau de booléens via `cert.getKeyUsage()`, où chaque indice correspond à un usage de clé spécifique. Si un usage de clé est actif (valeur `true`), son nom est récupéré d'une `Map` pré-définie et affiché. Cette étape est importante pour confirmer que le certificat peut être utilisé pour les fonctions prévues par sa politique de sécurité.

Vérification de la Signature et de l'Algorithme

La méthode checkSignatureAndAlgorithm vérifie que la signature du certificat est valide en utilisant la clé publique de l'émetteur du certificat. Elle prend en compte différents algorithmes de signature, tels que RSA et ECDSA. A la fin de la fonction la verification de signature classique indépendant de l'algorithme est aussi présente.

Chargement des Certificats

La méthode loadCertificate permet de charger un certificat à partir d'un fichier, en supportant les formats DER et PEM. Cependant en testant avec des fichiers au format PEM, ils sont tout de même interprétés comme format DER.

Vérification des Signatures RSA et ECDSA

Les méthodes verifyRSASignature et verifyECDSASignature implémentent la logique pour vérifier les signatures RSA et ECDSA des certificats, respectivement.

- **RSA :** La vérification RSA implique le déchiffrement de la signature avec la clé publique de l'émetteur et la comparaison du résultat avec un hash du contenu du certificat.

- **ECDSA :** La vérification ECDSA est plus complexe en raison de la nature des courbes elliptiques. Cette méthode décode la signature ECDSA, extrait les points de la clé publique de l'émetteur, et utilise Bouncy Castle pour vérifier la signature contre le hash du contenu du certificat.
-

Décodage des Signatures ECDSA

La méthode decodeECDSASignature illustre comment décoder une signature ECDSA à partir de son format ASN.1 DER en utilisant Bouncy Castle. C'est important pour la vérification des signatures ECDSA, car elle permet de transformer la signature codée dans le certificat en une forme directement utilisable pour la vérification.

Contraintes de Base

La méthode verifyBasicConstraints vérifie si le certificat est autorisé à agir comme une autorité de certification (CA), en examinant l'extension des contraintes de base. Si la valeur est très grande c'est qu'il s'agit du rootCA, si elle vaut 0 c'est une CA intermediaire et sinon ce n'est pas une CA.

Extraction des Points de Distribution de la Liste de Révocation (CRL)

La méthode getCrlDistributionPoints extrait les URL des points de distribution CRL à partir d'un certificat. C’est essentiel pour la vérification de la révocation des certificats, car ces URL fournissent l'accès aux listes de révocation qui indiquent les certificats révoqués. Nous utilisons Bouncy Castle pour analyser les extensions de certificat et extraire ces informations.

Vérification CRL

La méthode verifyCRL implémente la logique de vérification de la révocation des certificats en utilisant les CRL. Elle gère le téléchargement des CRL à partir des URL extraites, la mise en cache pour éviter les téléchargements répétés, et la vérification de l'état de révocation du certificat concerné.

Vérification OCSP

La vérification OCSP confirme le statut de révocation d'un certificat en temps réel, offrant une alternative plus rapide et souvent plus à jour par rapport aux listes de révocation de certificats (CRL).

Extraction de l'URL OCSP

extractOcspUrl extrait l'URL du serveur OCSP à partir de l'extension Authority Information Access (AIA) du certificat. Cette étape dirige la requête vers le bon serveur OCSP capable de fournir le statut de révocation pour le certificat spécifique.

Génération de l'ID de Certificat pour la Requête OCSP

generateCertificateID crée un identifiant unique pour le certificat en question, en utilisant le numéro de série du certificat et les informations de l'émetteur. Cet identifiant est nécessaire pour formuler la requête OCSP, permettant au serveur OCSP d'identifier précisément le certificat dont le statut de révocation est demandé.

Tests ET Résultats :

ECDSA :

```
########## Certificate 1 ##########

Curve: secp384r1 [NIST P-384] (1.3.132.0.34)
ECDSA Signature verifies: SUCCESS
Signature algorithm: SHA384withECDSA
Signature verifies: SUCCESS
Certificate is within its valid period.
Key Usage:
- keyCertSign
- cRLSign

This certificate is a CA with basic constraints of : 2147483647
No CRL Distribution Points available.
```

```
########## Certificate 2 ##########

Curve: secp384r1 [NIST P-384] (1.3.132.0.34)
ECDSA Signature verifies: SUCCESS
Signature algorithm: SHA384withECDSA
Signature verifies: SUCCESS
Certificate is within its valid period.
Key Usage:
- digitalSignature
- keyCertSign
- cRLSign

This certificate is a CA with basic constraints of : 0
CRL Distribution Points:
- http://crl.usertrust.com/USERTrustECCCertificationAuthority.crl
CRL loaded from cache
CRL : Certificate is not revoked by CRL
OCSP : Certificate is not revoked
```

```
########## Certificate 3 ##########

Curve: secp256r1 [NIST P-256,X9.62 prime256v1] (1.2.840.10045.3.1.7)
ECDSA Signature verifies: SUCCESS
Signature algorithm: SHA256withECDSA
Signature verifies: SUCCESS
Certificate is within its valid period.
Key Usage:
- digitalSignature

This certificate is not a CA.
CRL Distribution Points:
- http://crl.sectigo.com/SectigoECCExtendedValidationSecureServerCA.crl
CRL loaded from cache
CRL : Certificate is not revoked by CRL
OCSP : Certificate is not revoked

Process finished with exit code 0
```

RSA avec un certificat expiré:

```
########## Certificate 1 ##########

RSA Signature verifies: SUCCESS
Signature algorithm: SHA256withRSA
Signature verifies: SUCCESS
Certificate is within its valid period.
Key Usage:
- keyCertSign
- cRLSign

This certificate is a CA with basic constraints of : 2147483647
No CRL Distribution Points available.
```

```
########## Certificate 2 ##########

RSA Signature verifies: SUCCESS
Signature algorithm: SHA256withRSA
Signature verifies: SUCCESS
Certificate is within its valid period.
Key Usage:
- digitalSignature
- keyCertSign
- cRLSign

This certificate is a CA with basic constraints of : 0
CRL Distribution Points:
- http://x1.c.lencr.org/
CRL downloaded and cached
CRL : Certificate is not revoked by CRL
OCSP URL not found in certificate. Skipping OCSP verification.
```

```
########## Certificate 3 ##########

RSA Signature verifies: SUCCESS
Signature algorithm: SHA256withRSA
Signature verifies: SUCCESS
Certificate has expired: NotAfter: Wed Mar 20 13:13:48 CET 2024
Key Usage:
- digitalSignature
- keyEncipherment

This certificate is not a CA.
No CRL Distribution Points available.

Process finished with exit code 0
```

Autres Choix Techniques et Structure du Programme

Le programme est structuré de manière modulaire, avec des méthodes distinctes pour chaque aspect de la validation des certificats. La gestion des erreurs a été intégrée pour assurer la robustesse du programme.

Difficultés Rencontrées

La principale difficulté rencontrée au cours de ce projet a été dans la gestion des formats de données complexes, notamment ASN.1. Cette complexité a été accentuée par la nécessité de consulter diverses RFC pour identifier les formats appropriés à utiliser. De plus, un défi significatif a été de déterminer les méthodes spécifiques à appliquer à chaque étape du processus de vérification. Comprendre quand et comment utiliser correctement ces méthodes a requis une analyse approfondie et une compréhension détaillée.

Améliorations Possibles

Parmi les améliorations envisageables, on peut citer :

- **Gestion des erreurs :** Améliorer la résilience du programme face aux erreurs.
- **Interface utilisateur :** Développer une interface graphique pour faciliter l'utilisation.
- **Tests unitaires :** Tester les différents cas plus rapidement a l'aide de tests unitaires.
- **Prise en charge plus large des formats de certificats et des algorithmes de signature**

Ressources Utilisées

- **Documentation officielle de Java :** Pour les bases du langage et la manipulation des certificats.
- **Site officiel de Bouncy Castle :** Pour les détails sur l'utilisation de la bibliothèque dans les opérations cryptographiques.
- **IA : Copilot, ChatGPT :** Pour mieux comprendre la logique que le code doit avoir par moment.

- https://docs.oracle.com/javase/8/docs/api/java/security/cert/X509Certificate.html
- https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html

- https://people.eecs.berkeley.edu/~jonah/lcrypto/org/bouncycastle/math/ec/ECPoint.html
- https://javadoc.io/static/org.bouncycastle/bcprov-jdk15on/1.62/org/bouncycastle/crypto/signers/ECDSASigner.html

- http://www.java2s.com/example/java-api/java/security/cert/x509certificate/getextensionvalue-1-1.html