CRYPTOGRAPHIE JAVA

Teodor VRABIE[1], Alexandr BONTA[1*]

[1]*Université Technique de Moldavie, Faculté d'Ordinateur, Informatique et Microélectronique, gr. FI-181, Chişinău, Moldavie*

*Auteur correspondant: Bonta Alexandr, alexandr.bonta@ee.utm.md

Résumé. *Les API de Cryptographie Java offrent la possibilité de crypter et de décrypter les données en java, ainsi que de gérer les clés, les signatures et d'authentifier les messages, de calculer les hachages cryptographiques, etc. Extension de Cryptographie Java (ECJ) fait partie de la plate-forme Java. Architecture de Cryptographie Java (ACJ) est structurée autour de plusieurs classes de base et d'interfaces à usage général.*

Mots clés : *crypter, décrypter, architecture, clés, authentification, signature*

Introduction

Les API de Cryptographie Java permettent de crypter et de décrypter les données en Java, [1] de gérer les clés, les signatures et d'authentifier les messages, de calculer les hachages cryptographiques, etc. L'API de Cryptographie Java est fournie par ce que l'on appelle l'Extension de Cryptographie Java (ECJ). ECJ fait partie de la plate-forme Java. Architecture de Cryptographie Java (ACJ) est le nom de la conception interne de l'API de Cryptographie Java. Le ACJ est structuré autour de plusieurs classes de base et d'interfaces à usage général. La fonctionnalité réelle de ces interfaces est fournie par les fournisseurs. On peut également implémenter et connecter les propres fournisseurs.

Classes et interfaces de base

L'API de Cryptographie Java comprend les packages Java suivants :

- java.security
- java.security.cert
- java.security.spec
- java.security.interfaces
- javax.crypto
- javax.crypto.spec
- javax.crypto.interfaces

Les principales classes et interfaces de ces packages :

- Provider
- SecureRandom
- Cipher
- MessageDigest
- Signature
- Mac
- AlgorithmParameters
- AlgorithmParameterGenerator
- KeyFactory
- SecretKeyFactory
- KeyPairGenerator
- KeyGenerator

- KeyAgreement
- KeyStore
- CertificateFactory
- CertPathBuilder
- CertPathValidator

CertStore **Provider (fournisseur de cryptographie)**

La classe Provider (java.security.Provider) est la classe centrale de l'API de cryptage Java [2]. Pour utiliser l'API de chiffrement Java, vous devez installer un fournisseur de chiffrement. Le SDK Java est fourni avec son propre fournisseur de cryptographie.

Cipher (Шифр)

La classe Cipher (javax.crypto.Cipher) représente un algorithme cryptographique. Le chiffrement peut être utilisé à la fois pour le chiffrement et le déchiffrement des données.

Keys (Clés)

Vous avez besoin d'une clé pour crypter ou décrypter les données. Il existe deux types de clés, selon le type d'algorithme de chiffrement utilisé :

Clés symétriques :

Les clés symétriques sont utilisées pour les algorithmes de chiffrement symétriques. Les algorithmes de chiffrement symétriques utilisent la même clé pour le chiffrement et le déchiffrement.

Clés asymétriques :

Les clés asymétriques sont utilisées pour les algorithmes de cryptage asymétrique. Les algorithmes de cryptage asymétrique utilisent une clé pour le cryptage et une autre pour le décryptage. Les algorithmes de cryptage à clé publique et privée sont des exemples d'algorithmes de cryptage asymétrique.

Sécurité des clés

Les clés doivent être difficiles à deviner pour qu'un attaquant ne puisse pas facilement deviner la clé de chiffrement. L'exemple de la section précédente sur la classe Cipher utilisait une clé très simple codée en dur. En pratique, vous ne devriez pas faire cela. Il est souhaitable que la clé se compose d'octets aléatoires.

Génération de clés

On peut utiliser la classe KeyGenerator de Java pour générer des clés de chiffrement aléatoires.

Un exemple de son utilisation :

```
KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
SecureRandom secureRandom = new SecureRandom();
int keyBitSize = 256;
keyGenerator.init(keyBitSize, secureRandom);
SecretKey secretKey = keyGenerator.generateKey();
```

Générer une paire de clés

Les algorithmes de cryptage asymétrique utilisent une clé publique et une paire de clés privées pour crypter et décrypter les données. Vous pouvez utiliser KeyPairGenerator (java.security.KeyPairGenerator) pour générer une paire de clés asymétriques.

Un exemple de son utilisation :

```
SecureRandom secureRandom = new SecureRandom();
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");
KeyPair keyPair = keyPairGenerator.generateKeyPair();
```

Magasin de clés (Key Store)

Java KeyStore est une base de données pouvant contenir des clés. Le Java KeyStore est représenté par la classe KeyStore (java.security.KeyStore) [3]. Le magasin de clés peut contenir les types de clés suivants :

- Clés privées
- Clés publiques et certificats
- Clés secrètes

Outil de gestion des clés (Keytool)

Java Keytool — c'est un outil de ligne de commande qui peut fonctionner avec les fichiers Java KeyStore. Keytool peut générer des paires de clés vers le fichier KeyStore, exporter des certificats et importer des certificats vers KeyStore et certaines autres fonctions. Keytool est livré avec une installation Java.

Messages de résumé (MessageDigest)

Un résumé de message est une valeur de hachage calculée à partir des données du message. Si au moins un octet des données chiffrées change, le résumé de message calculé à partir des données changera également.

Lorsque vous recevez des données chiffrées, vous les déchiffrez, calculez le résumé de message à partir de celles-ci et comparez le résumé de message calculé avec le résumé de message envoyé avec les données chiffrées. Si deux résumés de messages sont identiques, il y a une forte probabilité (mais pas 100%) que les données n'ont pas été modifiées.

Code d'authentification du message (MAC)

La classe Mac Java est utilisée pour générer un MAC (Message Authentication Code) à partir d'un message [4]. MAC est similaire au résumé de message, mais utilise une clé supplémentaire pour chiffrer le résumé de message. Ce n'est qu'avec les données brutes et la clé que vous pouvez vérifier le MAC. Ainsi, MAC est un moyen plus sûr de protéger un bloc de données contre la modification que le résumé de message.

Signature

La classe Signature (java.security.Signature) est utilisée pour signer numériquement les données. Lorsque les données sont signées, une signature numérique est créée à partir de ces données. Ainsi, la signature est séparée des données.

Une signature numérique est créée en créant un résumé de message [5] (hachage) à partir des données et en chiffrant ce résumé de message avec la clé privée de l'appareil, de la personne ou de l'organisation qui doit signer les données. Le résumé d'un message chiffré est appelé une signature numérique.

Vérification de la signature :

Pour vérifier la signature, vous devez initialiser l'instance de signature en mode vérification en appelant la méthode initVerify (...), en passant en paramètre la clé publique qui sert à vérifier la signature

Conclusion

Il est probablement évident à partir de cet article que la cryptographie est un vaste sujet.

Il existe de nombreux problèmes intéressants que la cryptographie peut aider à résoudre, et de nombreuses façons d'utiliser le JCE. Il est également important de tenir compte de la situation actuelle et du degré de sécurité requis. Selon la situation, l'une des autres API liées à la sécurité peut être plus appropriée que de plonger directement dans la cryptographie brute à l'aide du JCE, ou peut-être que JCE peut être combiné avec d'autres API pour résoudre le problème.

Plus important encore, nous devons réaliser que la sécurité ne fonctionne que lorsqu'elle est de bout en bout, ce qui signifie que toutes les étapes d'un processus doivent être sécurisées. Par exemple, quelle que soit la qualité de l'algorithme utilisé pour crypter les données sur disque, si ces données sont transmises sur un réseau non sécurisé, l'algorithme de cryptage sur disque peut ne pas avoir d'importance.

Références Web:

1. JAKOB JENKOV. *Java Cryptography* [online]. 23.01.2018. [accéder 02.03.2021]. Disponible: http://tutorials.jenkov.com/java-cryptography/index.html
2. PANKAJ KUMAR. *J2EE Security for Servlets, EJBs, and Web Services* [online]. 2003. [accéder 02.03.2021]. Disponible: https://www.informit.com/articles/article.aspx?p=170967&seqNum=7
3. *Java™ Platform* Standard Ed. 7. [online]. Class KeyStore. [accéder 03.03.2021]. Disponible: https://docs.oracle.com/javase/7/docs/api/java/security/KeyStore.html
4. *Les codes d'authentification de message – MAC.* [online]. [accéder 03.03.2021]. Disponible: http://www.bibmath.net/crypto/index.php?action=affiche&quoi=moderne/mac
5. SERALAHTHAN VIVEKAANANTHAN. *Java Cryptographic Hash Functions* [online]. 21.07.2019. [accéder 03.03.2021]. Disponible: https://techexpertise.medium.com/java-cryptographic-hash-functions-a7ae28f3fa42