Recherche dans DEJ avec Google Rechercher

22. La sécurité

Niveau : Confirmé

Depuis sa conception, la sécurité dans le langage Java a toujours été une grande préoccupation pour Sun.

Avec Java, la sécurité revêt de nombreux aspects :

- les spécifications du langage disposent de fonctionnalités pour renforcer la sécurité du code
- la plate-forme définit un modèle pour gérer les droits d'une application
- les API JCA/JCE permettent d'utiliser des technologies de cryptographie
- l'API JSSE permet d'utiliser le réseau au travers des protocoles sécurisés SSL ou TLS
- l'API JAAS propose un service pour gérer l'authentification et les autorisations d'un utilisateur

Ces deux premiers aspects ont été intégrés à Java dès sa première version.

Ce chapitre contient plusieurs sections :

Ce chapitre est très incomplet : la suite de ce chapitre sera développée dans une version future de ce document

22.1. La sécurité dans les spécifications du langage

Les spécifications du langage apportent de nombreuses fonctionnalités pour renforcer la sécurité du code aussi bien lors de la phase de compilation que lors de la phase d'exécution :

- typage fort (toutes les variables doivent posséder un type)
- initialisation des variables d'instances avec des valeurs par défaut
- modificateur d'accès pour gérer l'encapsulation et donc l'accessibilité aux membres d'un objet
- les membres final
- ...

22.1.1. Les contrôles lors de la compilation

22.1.2. Les contrôles lors de l'exécution

La JVM exécute un certain nombre de contrôles au moment de l'exécution :

- vérification des accès en dehors des limites des tableaux
- contrôle de l'utilisation des casts
- vérification par le classloader de l'intégrité des classes utilisées
- ...

22.2. Le contrôle des droits d'une application

Un système de contrôle des droits des applications a été intégré à Java dès sa première version notamment pour permettre de sécuriser l'exécution des applets. Ces applications téléchargées sur le réseau et exécutées sur le poste client doivent impérativement assurer aux personnes qui les utilisent qu'elles ne risquent pas de réaliser des actions malveillantes sur le système dans lequel elles s'exécutent.

Le modèle de sécurité relatif aux droits des applications développées en Java a évolué au fur et à mesure des différentes versions de Java.

22.2.1. Le modèle de sécurité de Java 1.0

Le modèle proposé par Java 1.0 était très sommaire puisqu'il ne distinguait que deux catégories d'applications :

- les applications locales
- les applications téléchargées sur le réseau

Le modèle est basé sur le "tout ou rien". Les applications locales ont tous les droits et les applications téléchargées ont des droits très limités. Les restrictions de ces dernières sont nombreuses :

- impossibilité d'écrire sur le disque local
- impossibilité d'obtenir des informations sur le système local
- impossibilité de se connecter à un autre serveur que celui d'où l'application a été téléchargée
- ...

La mise en oeuvre de ce modèle est assurée par le "bac à sable" (sandbox en anglais) dans lequel s'exécutent les applications téléchargées.

22.2.2. Le modèle de sécurité de Java 1.1

Le modèle proposé par la version 1.0 était très efficace mais beaucoup trop restrictif surtout dans le cadre d'une utilisation personnelle telle que celle des applications pour un intranet par exemple.

Le modèle de la version 1.1 propose la possibilité de signer les applications packagées dans un fichier .jar. Une application ainsi signée possède les mêmes droits qu'une application locale.

22.2.3. Le modèle Java 1.2

Le modèle proposé par la version 1.1 a apporté des débuts de solution pour attribuer des droits à certaines applications. Mais ce modèle manque cruellement de souplesse puisqu'il s'appuie toujours sur le modèle "tout au rien".

Le modèle de la version 1.2 apporte enfin une solution très souple mais plus compliquée à mettre en oeuvre.

Les droits accordés à une application sont rassemblés dans un fichier externe au code qui se nomme politique de sécurité. Pour les différentes applications, l'ensemble des fichiers se situe dans le répertoire lib/security du répertoire où est installé le JRE. Par convention, ces fichiers ont pour extension .policy.

22.3. La cryptographie

Le mot cryptographie est dérivé des mots grecs kryptos (caché) et graphie (écriture). La cryptologie est la science qui étudie la transformation d'un texte en clair (plain text) en un texte chiffré difficile à comprendre (ciphered text).

La cryptographie permet de stocker ou d'échanger des données de façon plus ou moins sécurisée.

La cryptographie utilise deux opérations :

- le chiffrement : cette opération consiste à transformer des données en clair en des données chiffrées qui soient difficiles voire impossible à exploiter par un tiers ne pouvant effectuer l'opération de déchiffrement
- le déchiffrement : c'est l'opération inverse qui permet le retour des données en clair à partir de données chiffrées

La cryptographie est l'art de sécuriser un ensemble de données : elle est principalement utilisée pour créer une valeur de hachage d'un message ou pour chiffrer/déchiffrer un message. Dans les deux cas, des algorithmes mathématiques complexes voire très complexes sont utilisés.

La cryptographie est un élément très important de la sécurité des échanges notamment au travers des réseaux : elle est par exemple utilisée pour mettre en oeuvre les signatures digitales, les certificats, l'authentification de messages, le chiffrement/déchiffrement des messages, ...

La cryptographie est utilisée pour mettre en oeuvre plusieurs fonctionnalités notamment :

- confidentialité : les données peuvent être chiffrées pour garantir leur caractère privé
- intégrité des données : un algorithme mathématique permet de calculer une valeur de hachage et ainsi de vérifier l'intégrité d'un message
- authentification : les signatures numériques permettent de garantir que les données proviennent du bon émetteur

Une clé est un ensemble de données utilisée lors du chiffrement ou du déchiffrement de données afin de rendre ces opérations plus résistantes.

La cryptographie est assez ancienne : elle est utilisée depuis très longtemps avec des algorithmes initialement basiques. Par exemple, à l'époque gallo-romaine, l'empereur Jules César utilisait un système de chiffrement symétrique reposant sur un décalage alphabétique : ces principes de substitutions étaient utilisés pour crypter des messages.

Texte en clair	B	O	N	J	O	U	R

Décalage	+3	+3	+3	+3	+3	+3	+3
Texte chiffré	E	R	Q	M	R	X	U

Cette substitution est assez facile à casser, notamment par ce que chaque lettre en clair correspond à une même lettre chiffrée.

Il existe des versions plus complexes qui font varier la valeur de décalage pour chaque lettre.

Texte en clair	B	O	N	J	O	U	R
Décalage	+1	+2	+3	+1	+2	+3	+1
Texte chiffré	C	R	Q	K	Q	X	S

Dans l'exemple ci-dessus, la lettre O peut être cryptée en R ou Q selon sa position dans le texte.

Les algorithmes de substitution sont facilement cassables avec des machines.

Il est possible d'utiliser une clé de substitution qui associe à chaque caractère un autre caractère. L'inconvénient est que la clé doit être connue lors du chiffrement et du déchiffrement, ce qui nécessite d'une manière ou d'une autre de réaliser un échange de cette clé.

Caractère	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Clé	L	O	G	P	R	T	C	M	A	Q	S	U	Z	H	D	I	W	B	V	J	Y	N	E	K	X	F

En utilisant la clé, il est possible de chiffrer le message.

Texte en clair	B	O	N	J	O	U	R
Texte chiffré	O	D	H	Q	D	Y	B

Plusieurs types de modèles sont utilisés en cryptographie :

- Symétrique : utilise une clé unique qui est utilisée par l'émetteur et le récepteur. Ils doivent donc la partager entre-eux
- Asymétrique : utilise une clé publique et une clé privée
- Hybride : il utilise les modèles cryptographiques symétrique et asymétrique en combinant leurs avantages

Il existe deux grandes familles d'algorithmes de chiffrement :

- le chiffrement symétrique : une seule clé secrète est utilisée pour le chiffrement et le déchiffrement. L'émetteur et le récepteur doivent disposer de la même clé pour réaliser le chiffrement et le déchiffrement. La difficulté est de conserver secrète la clé qui doit être échangée.
- le chiffrement asymétrique : une clé publique et une clé privée sont utilisées pour le chiffrement et le déchiffrement. Les données sont chiffrées avec une clé et déchiffrées avec l'autre.

Le chiffrement symétrique utilise la même clé pour chiffrer et déchiffrer les données.

Ce procédé présente plusieurs inconvénients :

- chaque émetteur doit utiliser une clé différente avec chacun de ses correspondants
- l'échange de la clé doit se faire de manière sécurisée
- toute personne qui possède la clé de l'émetteur peut se faire passer pour lui : la non répudiation n'est pas garantie

Le chiffrement asymétrique utilise deux clés distinctes qui sont liées entre elles : la clé utilisée pour chiffrer peut être utilisée pour déchiffrer ce qui a été chiffré avec l'autre clé et vice versa. Une des deux clés conservée par l'émetteur est nommée clé privée, l'autre clé diffusée est nommée clé publique.

	Avantages	Inconvénients
Chiffrement symétrique	Rapide	Echange de la clé Ne peut pas être utilisé pour les signatures électroniques
Chiffrement asymétrique	Utilise deux clés Utilisable pour les signatures électroniques	Lent

Avec un chiffrement asymétrique, les deux clés sont liées mais il n'est pas possible de retrouver une clé à partir de l'autre.

Les algorithmes de chiffrements asymétriques sont généralement plus lents que les symétriques et nécessitent de nombreuses ressources lors de leurs calculs.

La cryptanalyse est la science qui étudie comment casser un algorithme de chiffrement : son but est donc de décrypter les données encodées avec un algorithme sans avoir les informations utiles pour réaliser l'opération. Pour cela, elle emploie de nombreuses techniques notamment la force brute, la factorisation, la cryptanalyse linéaire, la cryptanalyse différentielle, ...

Certains algorithmes ne possédant pourtant pas de failles connues ne sont plus utilisés car la puissance de calcul des machines actuelles permet de trouver leur clé (exemple : l'algorithme DES)

Il ne faut considérer aucun algorithme comme fiable et ils sont tous potentiellement vulnérables, si ce n'est maintenant, ce sera plus tard. La vulnérabilité d'un algorithme dépend du temps et de la puissance du hardware. Par exemple, les algorithmes DES et MD5 fréquemment utilisés sont maintenant considérés comme non sûrs et ne doivent donc plus être utilisés pour des besoins critiques.

22.3.1. Le modèle symétrique

Avec les algorithmes à clés symétriques, la même clé est utilisée pour les opérations de chiffrement et de déchiffrement. Ce type d'algorithme porte aussi le nom d'algorithme à clé secrète. Ce type de chiffrement est historiquement le plus ancien puisqu'il était déjà utilisé par les égyptiens et les romains plusieurs milliers d'années avant J.C.

La taille de la clé influe sur la robustesse de l'algorithme.

L'émetteur et le récepteur doivent partager la même clé. La clé ne doit être connue que de l'émetteur et du récepteur. Il faut donc garantir que l'échange de cette clé se soit fait de manière sécurisée.

Son principal défaut concerne donc l'échange de la clé entre l'émetteur et le récepteur. Hormis par un moyen physique, il n'est pas possible d'échanger électroniquement la clé de façon sécurisée sans la chiffrer.

Chaque paire d'acteurs (émetteur/receveur) doit avoir une clé différente sinon tous les acteurs qui partagent la même clé seront capables de déchiffrer les données. Pour des échanges sécurisés avec plusieurs acteurs indépendants, il faut une clé distincte ce qui peut rendre le nombre de clés à gérer important.

Son grand intérêt est d'être rapide lors de l'exécution des opérations car les algorithmes mathématiques sont relativement simples.

L'algorithme peut s'appliquer selon deux approches :

- Bloc (Block cipher)
- Flux (Stream cipher)

Il existe de nombreux algorithmes de chiffrements symétriques :

Algorithme	Description
DES	DES est l'acronyme de Data Encryption Standard. Cet algorithme a été inventé en 1977 par IBM et adopté en 1978 par le NBS (National Bureau of Standards). Il a été utilisé par les administrations fédérales américaines pour chiffrer les données sensibles. Les données sont chiffrées par blocs de 64 bits (block cipher). Il utilise une clé sur 56 bits ce qui le rend vulnérable aux attaques par force brute. Cet algorithme n'est donc plus considéré comme sûr.
Triple DES (3DES)	Pour palier la faiblesse de DES, l'algorithme Triple DES applique trois fois l'algorithme DES avec deux ou trois clés différentes. Ceci rend cet algorithme plus résistant aux tentatives de déchiffrement par force brute.
Blowfish	Blowfish utilise une clé de longueur variable de 32 à 448 bits et est plus rapide et plus sûr que DES (block cipher)
AES / Rijndael	(Advanced Encryption Standard) : 128/192/256 bits, plus rapide et plus sûr que le Triple DES, finalisé en 2000 (block cipher) AES est l'acronyme de Advanced Encryption Standard. Cet algorithme a été inventé en 2000 par Vincent Rijmen et Joan Daemen pour remplacer le DES.

	Il est utilisé par les administrations américaines pour chiffrer les données sensibles. AES peut utiliser plusieurs longueurs de clés notamment 128 bits, 192 bits et 256 bits selon le degré de sécurité souhaité.
IDEA	(International Data Encryption Algorithm) : 128 bits IDEA est l'acronyme d'International Data Encryption Algorithm. Cet algorithme a été inventé en 1991 par J. Massey et X. Lai. Son principe de fonctionnement est similaire à DES mais il utilise une clé plus longue sur 128 bits.
RC2, RC4, RC5, RC6 (Rivets Cipher 4)	128bits, taille variable (stream cipher) RC2, RC4 et RC5 sont des algorithmes créés par Ronald Rivest. RC est l'acronyme de "Ron's Code" ou "Rivest's Cipher". Ces algorithmes permettent de choisir la longueur de la clé (jusqu'à 1024 bits).

Le modèle symétrique possède plusieurs limitations :

- Le partage de la clé peut être une vulnérabilité
- La gestion de la clé est problématique : diffusion, révocation, multiplication des clés requises, ...

22.3.2. Le modèle asymétrique

Ce modèle repose sur des algorithmes mathématiques complexes qui utilisent deux clés distinctes, une dite publique et une autre dite privée :

- une clé publique : cette clé peut être diffusée à tout le monde
- une clé privée : cette clé ne doit être connue que du destinataire

Un grand nombre aléatoire est utilisé pour générer les deux clés. Les deux clés utilisées sont différentes et il est impossible de déduire une clé à partir de l'autre. Par contre, l'usage des deux clés est réversible : l'une peut être indifféremment la clé publique ou la clé privée et vice versa.

Chaque clé joue un rôle particulier : une clé est utilisée pour chiffrer les données et l'autre clé est pour déchiffrer.

La clé publique est utilisée pour chiffrer les données qui ne seront alors déchiffrable qu'avec la clé privée. Inversement, un message chiffré avec la clé privée ne pourra être déchiffré qu'avec la clé publique. La clé publique peut être diffusée à tout le monde mais la clé privée doit être gardée secrète.

L'utilisation de ce modèle nécessite plusieurs étapes :

1. La génération des deux clés
2. L'envoie à l'expéditeur de l'une des deux clés comme clé publique
3. L'expéditeur chiffre les données avec la clé publique et l'envoie au destinataire
4. Le destinataire déchiffre les données avec la clé privée

A et B possèdent chacun leur clé privée. Ils se sont échangé leurs clés publiques respectives.

A utilise sa clé privée pour envoyer un message chiffré à B qui utilise la clé publique correspondante pour déchiffrer le message.

L'échange de clés publiques est facile puisque cette clé est inutilisable sans la clé privée correspondante qui elle n'est pas échangée.

Ce type d'algorithme porte aussi le nom d'algorithme à clé publique. Le développement de ces algorithmes est assez récent puisqu'ils ont débuté dans les années 1970. Ils permettent d'utiliser la cryptographie pour assurer des fonctionnalités de type confidentialité et authentification.

Il existe plusieurs algorithmes qui mettent en oeuvre ce modèle dont :

Algorithme	Description
RSA	RSA est l'acronyme du nom de ses trois inventeurs (Ronald Rivest, Adi Shamir et Leonard Adleman). Le brevet relatif à RSA a expiré en septembre 2000. jusqu'à 2048 bits, publié en 1978

	La clé publique est le résultat de la multiplication de deux très grands nombres premiers. La clé privée dépend de ces deux valeurs mais il est extrêmement difficile de les recalculer à partir de la clé privée.
DSA	DSA est l'acronyme de Digital Signature Algorithm
Diffie-Hellman	Diffie-Hellman est un algorithme de chiffrement asymétrique créé par Whitfeld Diffie et Martin Hellman. L'algorithme a fait l'objet d'un dépôt de brevet en 1977 qui a expiré en 1997. C'est un protocole pour l'échange de clés qui est vulnérable
ElGamal	L'algorithme ElGamal est une variante de Diffie-Hellman inventée par Taher Elgamal.
Elliptic Curve Cryptography (ECC)	La vitesse de l'AES est utilisée avec les clés RSA : le chiffrage est efficace car plus rapide avec une taille de clé réduite Améliore grandement la vitesse de traitement par rapport à l'encryptage standard avec des clés publiques Plusieurs implémentations open source (BouncyCastle, OpenSSL, ...) Implémentations de plusieurs algorithmes dans Java 7

Ce type de modèle est utilisé dans la signature digitale :

- l'émetteur chiffre son message avec sa clé privée
- le receveur déchiffre le message chiffré avec sa clé publique

Ceci permet d'authentifier l'émetteur car c'est le seul qui possède la clé correspondant à la clé publique mais ne permet pas la confidentialité des données car la clé publique est diffusée.

Les algorithmes pour le chiffrement symétrique présentent plusieurs inconvénients :

- ils reposent sur des algorithmes mathématiques complexes qui nécessitent donc des ressources notamment en CPU lors des opérations de chiffrement/déchiffrement.
- la clé doit être échangée : durant cet échange, la clé peut être obtenue par un tiers qui pourra alors déchiffrer les messages
- il faut utiliser une clé différente entre un émetteur et ses différents récepteurs. Le nombre de clé à gérer peut alors devenir important

Le chiffrement asymétrique ne garantit pas l'intégrité du message reçu. A chiffre un message avec sa clé publique et envoie le message chiffré à B. C intercepte le message et le remplace par un autre message chiffré avec la même clé publique que A. B reçoit le message et peut le déchiffrer avec la clé privée.

Pour garantir l'intégrité du message, il est nécessaire de le signer. La signature électronique utilise la clé privée de A pour générer une valeur de hachage du message qui sera alors envoyée chiffrée.

Il est aussi possible de signer un message sans le chiffrer.

22.4. JCA (Java Cryptography Architecture) et JCE (Java Cryptography Extension)

Deux API fournies depuis Java 1.4 permettent la mise en oeuvre de la cryptographie :

- JCA (Java Cryptography Achitecture) qui définit l'architecture générale du framework et les fonctionnalités cryptographiques de base (fonctions de hachage, signatures numériques, clés, certificats, ...)
- JCE (Java Cryptography Extension) qui fournit des fonctionnalités cryptographiques de haut niveau (chiffrement/déchiffrement avec algorithmes symétriques/asymétriques, authentification de messages (HMAC), ...)

Historiquement, seul l'API JCA était fournie dans le JDK car le JCE était soumis à des restrictions liées à la législation américaine. Depuis l'assouplissement de ces restrictions, les deux API sont fournies dans le JDK.

Historiquement, les API de cryptographie de Java étaient séparées en deux parties afin de permettre une restriction de diffusion :

- le package java.security contient des classes qui ne possèdent pas de restriction de diffusion
- le package javax.crypto contient des classes qui ont été pendant un moment diffusées sous restriction

JCE est une API qui propose de standardiser l'utilisation de la cryptographie en restant indépendant des algorithmes utilisés. Elle prend en compte le cryptage/décryptage de données, la génération de clés et l'utilisation de la technologie MAC (Message Authentication Code) pour garantir l'intégrité d'un message.

JCE a été intégrée au JDK 1.4. Auparavant, cette API était disponible en tant qu'extension pour les JDK 1.2 et 1.3.

Pour pouvoir utiliser cette API, il faut obligatoirement utiliser une implémentation développée par un fournisseur (provider). Avec le JDK 1.4, Sun fournit une implémentation de référence nommée SunJCE.

L'API JCA est contenue dans le package java.security et l'API JCE est contenue dans le package javax.crypto.

JCA	JCE
java.security	javax.crypto
java.security.acl	javax.crypto.interfaces
java.security.cert	javax.crypto.spec
java.security.interfaces	
java.security.spec	

Ces API fournissent différentes fonctionnalités nommées services. Ces services peuvent offrir la même typologie de fonctionnalités mais leurs implémentations utilisent des algorithmes différents.
L'API utilise des fabriques pour permettre d'obtenir une instance de ces services pour un algorithme et éventuellement un fournisseur donné.

La cryptographie est généralement limitée par les législations de nombreux pays. De ce fait, les algorithmes fournis par Java sont répartis en deux groupes :

- strong : inclus dans tous les JRE
- limited : téléchargeables séparement selon les restrictions des lois américaines

La configuration utilisable dans le monde entier est fournie avec les JRE dans le fichier default_local.policy contenu dans le fichier lib/security/local_policy.jar

Exemple :

```
01. // Some countries have import limits on crypto strength. This policy file
02. // is worldwide importable.
03.
04. grant {
05.     permission javax.crypto.CryptoPermission "DES", 64;
06.     permission javax.crypto.CryptoPermission "DESede", *;
07.     permission javax.crypto.CryptoPermission "RC2", 128,
08.           "javax.crypto.spec.RC2ParameterSpec", 128;
09.     permission javax.crypto.CryptoPermission "RC4", 128;
10.     permission javax.crypto.CryptoPermission "RC5", 128,
11.           "javax.crypto.spec.RC5ParameterSpec", *, 12, *;
12.     permission javax.crypto.CryptoPermission "RSA", *;
13.     permission javax.crypto.CryptoPermission *, 128;
14. };
```

Le chapitre «JCA (Java Cryptography Architecture)» détaille l'utilisation de cette API.

Le chapitre «JCE (Java Cryptography Extension)» détaille l'utilisation de cette API.

22.5. JSSE (Java Secure Sockets Extension)

JSSE permet de mettre en oeuvre TLS (Transport Layer Security). TLS est le successeur de SSL (Secure Sockets Layer). SSL et TLS permettent de sécuriser les échanges sur le réseau en utilisant la cryptographie asymétique. SSL et TLS sont indépendants du protocole utilisé.

Plusieurs versions de TLS ont été publiées :

- v 1.0 définie dans la RFC 2246 en 1999
- v 1.1 définie dans la RFC en 2006
- v1.2 définie dans la RFC 5246 en 2008

Par abus de langage, SSL est souvent utilisé comme synonyme de TLS.

Les classes et interfaces de cette API sont regroupées dans les packages javax.net et javax.net.ssl.

22.6. JAAS (Java Authentication and Authorization Service)

Les classes et interfaces de cette API sont regroupées dans le package javax.security.auth

Cette API a été intégrée au JDK 1.4.