UNIVERSITÉ
PARIS-EST CRÉTEIL
VAL DE MARNE

Faculté des sciences et technologies
Master 2 Sécurité des systèmes informatiques

Compte rendu TP
Signature Aveugle

Mamadou Aliou DIALLO

Chargé du cours : Mr Habib BOUDJEMA

Introduction :

Le but du TP est d'implémenter en cryptographie java le système de signature aveugle.
Connu sous le nom de **Blind Signature**, la signature aveugle consiste à se faire signer un message ou un document sans que l'autorité signataire ne puisse prendre connaissance du contenu au moment de la signature.

Le propriétaire du document et l'autorité signature sont deux entités distinctes.
Ce type de signature est souvent utilisé dans les applications telles que le vote électronique, la monnaie numérique.
La vérification de la signature peut être faite publiquement à partir du message, document original.

Rappelons que nous avons besoin du provider **BouncyCastle** qui fournit des **API** cryptographiques pour la signature aveugle. Nous utiliserons des algorithmes à clés publique avec des facteurs aveuglants. Le code source est disponible dans le dossier **Blinding_Signature** développé avec **Eclipse.**

Les différentes étapes du protocole de signature aveugle.

1 - Création d'une paire de clés privée/publique avec RSA.

On génère une paire de clés de 2048 bits avec la méthode **generateKeyPair()** de **KeyPairGenerator.**

2 - Stockage des clés dans des fichiers.

Le choix de le faire sous forme d'objets dans un fichier est privilégié avec le modulus et l'exposant public/privé. La méthode **getKeySpec(clé, class)** de **KeySpec,** les classes **RSAPublicKeySpec** et **RSAPrivateKeySpec** ainsi que la méthode **writeObject(Biginteger)** de **ObjectOutputStream** qui prend le descripteur du fichier **FileOutputStream** ou sera enregistré la clé.

3 - Chargement et reconstruction des clés créées dans notre programme.

On fait l'opération inverse pour retrouvera les clés en fournissant les objets contenu dans le fichier **.key** avec **RSAPublicKeySpec(modulus, exponent).** La méthode **generatePublic/Private(RSAPublicKeySpec).**

4 - Brouillage du message à signer.

Pour cela nous avons besoin des paramètres des clés qu'on obtient pour la clé publique avec **RSAKeyParameters(false, PubModulus, PubExponent).**
False pour spécifier que c'est n'est pas une privateKey ; **true** pour la clé privée.

On génère un **blindingFactor** avec **generateBlindingFactor** pour la clé fourni en paramètre d'init de la classe **RSABlindingFactorGenerator** qui est adapté pour la signature aveugle**.**

On construit le **RSABlindingParameters** qu'on fournira en second paramètre du 'init **RSABlindingParameters(PublicKey, blindingFactor)**

RSABlindingEngine est utilisé pour brouiller le message sans que le signataire ne devine le message via la méthode **processBlock(tobeBlinded, 0, tobeBlinded.length);**

5 - Signature du message brouillé précédemment avec la clé privée.

Pour cette fonctionnalité il faut qu'on utilise la fonction de la classe **RSAEngine processBlock(blinded, 0, blinded.length)**.

6 - Débrouillage du message signé.

Comme pour le brouillage on fournit le **RSAKeyParameters** de la clé privé au premier argument du **RSABlindingParameters.** On initialise le **RSAEngine** à **false.**

7 - Vérification à partir de la clé publique que la signature résultante après le débrouillage correspond bien au message initial.

Comme la signature on utilise **processBlock(blinded, 0, blinded.length)** de la classe **RSAEngine** en l'initialisant à **false** avec **RSAKeyParameters** de la clé publique.

Conclusion :

Durant ce TP nous avons appris à utiliser une nouvelle technique de signature garantissant l'anonymat et l'intimité. La non traçabilité est assurée car l'autorité signataire ne pourra pas à posteriori certifier que le contenu d'un message ou document divulgué est bien celui qu'il a signé.