TP n°10

Chiffrement

Le but de ce TP est de tester les différentes implémentations en *Java* du chiffrement symétrique et du chiffrement asymétrique. L'objectif est de mettre en place les éléments nécessaires pour la création de certificats pour le prochain TP.

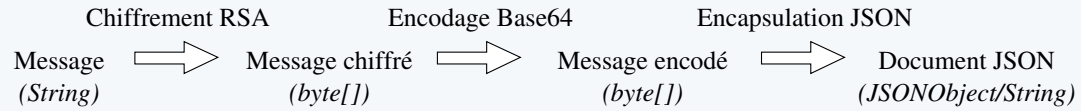
1 Chiffrement asymétrique

Nous souhaitons développer deux applications *Java*, que nous appellerons client et serveur, communiquant en HTTP avec des messages JSON dont les données sont chiffrées avec RSA. Au démarrage, chaque application vérifie l'existence des fichiers correspondant à sa clé publique et à sa clé privée. Si les deux fichiers n'existent pas, les clés sont générées et les fichiers créés.

Le client envoie ensuite sa clé publique au serveur et celui-ci répond en envoyant la sienne, ainsi qu'un challenge : il envoie une chaîne de caractères chiffrée au client que celui-ci doit déchiffrer. Le client envoie ensuite la chaîne déchiffrée que le serveur peut comparer à la chaîne originale. Une fois ces échanges terminés, les deux entités possèdent la clé publique de leur interlocuteur et peuvent les utiliser pour chiffrer les données.

Toutes les informations nécessaires aux applications sont spécifiées dans les fichiers de configuration : URL du serveur (pour le client), nom des fichiers contenant les clés, etc.

Pour éviter des problèmes dans vos applications, respectez le schéma suivant :



Il est conseillé de partir des classes fournies dans les fiches et d'ajouter de nouvelles méthodes (chiffrement/déchiffrement, encodage/décodage, etc.).

Questions

1. En vous aidant des fiches fournies en annexes, mettez en place la lecture du fichier de configuration pour les deux applications, ainsi que la vérification sur les clés (avec création si nécessaire).
2. Le client doit envoyer sa clé publique au serveur dans un message au format JSON. Pourquoi la clé ne peut-elle pas être directement placée dans du JSON ?
3. En utilisant l'encodage en base 64, faites que le client envoie sa clé au serveur. Vous afficherez la clé sur les deux applications pour s'assurer qu'elle est reçue correctement. Le serveur répond au client dans un premier temps avec un message au format JSON contenant un code d'état.
4. Le serveur doit générer un challenge aléatoire qu'il chiffre et qu'il envoie au client. Quelle clé doit être utilisée ?
5. Modifiez la réponse du serveur et vérifiez que le client peut déchiffrer le challenge.

Pour générer le challenge aléatoire, vous pouvez utiliser la classe `UUID` et la méthode `randomUUID`.

6. Si le challenge est correctement déchiffré, le client doit l'envoyer au serveur qui pourra ainsi constater que le client a bien réussi le challenge. Modifiez vos applications.

Le serveur doit conserver l'ensemble des informations sur les clients. Il est conseillé d'utiliser une `HashMap`, placé en attribut de classe, dont les clés sont des adresses (de client) `InetSocketAddress`. L'objet associé dans la `HashMap` contient les informations nécessaires pour chaque client : challenge, clé, étape en cours.

2 Chiffrement symétrique

Nous reprenons les applications précédentes. Nous souhaitons réaliser des échanges chiffrés entre le client et le serveur, en utilisant cette fois-ci un chiffrement symétrique. La clé est générée par le serveur et envoyée chiffrée au client, une fois le challenge réussi.

Questions

1. Commencez par tester le chiffrement symétrique, sans le chiffrement asymétrique : la clé est reçue par le client puis on procède à un simple échange.
2. Par quelle clé, la clé symétrique doit-elle être chiffrée ?
3. Modifiez les deux applications en couplant les deux méthodes de chiffrement.