Développement d'une application de messagerie instantanée

1. Utilisation du projet

1. Placez-vous à la racine du projet et lancer le serveur:

```
java -jar MessengerServer-assembly-0.1.0-SNAPSHOT.jar
```

2. Dans un navigateur, ouvrez l'URL: `http:/localhost:8080/`
3. Vous avez accès à deux applications:
   - Votre future application de messagerie (nommée `etu` dans la suite)
     - Elle se trouve dans le répertoire `src/main/webapp/etu` .
     - Les interfaces `filter` et `intruder` sur l'application `etu` .
   - L'application que vous avez attaquée au TP précédent (nommée `ens` dans la suite)
     - Elles se trouve dans le répertoire `src/main/webapp/ens` .
     - Les interfaces `filter` et `intruder` sur l'application `ens` .
     - Une calculette crypto
4. Pour les applications `etu` et `ens` vous pouvez vous connecter sous 3 identités: alice@univ-rennes.fr, bob@univ-rennes.fr ou charly@univ-rennes.fr. Ceci est fait pour vous permettre de débugger votre future application sans avoir à passer par le CAS de l'Université de Rennes.

2. Marche à suivre

Dans le répertoire `src/main/webapp/etu` , se trouve un premier squelette de votre future application. Pour l'instant, cette application ne fait rien à part afficher `Dummy message` à chaque rafraîchissement automatique de la page. Voici la structure de cette application:

- Le fichier `index.html` contient la page d'accueil de l'application.
- Le répertoire `src/` contient le code Typescript de l'application.
- Le répertoire `JS/` contient le code Javascript de l'application qui est appelé dans `index.html` .

Dans l'état, `src` contient 2 fichiers: `libCrypto.ts` et `messenger.ts` . On rappelle qu'une façon simple de produire un fichier `JS/messenger.js` depuis ces deux fichiers est d'utiliser la directive de compilation:

```
tsc --inlineSourceMap  true -outFile JS/messenger.js src/libCrypto.ts src/messenger.ts --target es2015
```

3. Plus de fonctions cryptographiques

Pour implanter votre protocole, vous aurez peut-être besoin de plus de fonctions cryptographiques que les chiffrements/déchiffrements avec les clés RSA. En plus de proposer le chiffrement/déchiffrement avec des clés RSA, la librairie `libCrypto.ts` permet de:

1. Signer/vérifier la signature d'un message avec des clés RSA
2. Générer des nonces
3. Générer des clés symétriques AES
4. Chiffrer/déchiffrer des messages avec des clés symétriques AES
5. Calculer le hash SHA256 d'un message

Vous trouverez des exemples d'utilisation de tout ceci dans le code Typescript de la calculette:

`src/main/webapp/ens/src/calculette.ts` .

4. ... plus réalistes

4.1. Chiffrement/déchiffrement et signature/vérification de signature

En pratique, il est déconseillé d'utiliser le même couple de clés RSA pour chiffrer/déchiffrer et pour signer/vérifier une signature. En conséquence, chaque utilisateur (ici alice@univ-rennes.fr, bob@univ-rennes.fr, et charly@univ-rennes.fr) dispose de **2 paires de clés RSA**. Une paire pour le chiffrement/déchiffrement et une paire pour la signature/vérification de signature. Ca sera également le cas pour tous les utilisateurs quand votre application sera déployée sur le web. Si vous lisez le code de `src/main/webapp/etu/src/messenger.ts` vous trouverez une fonction `fetchKey` qui va gérer cela pour vous:

```
function fetchKey(user: string, publicKey: boolean, encryption: boolean): Promise<CryptoKey>
```

Telle que

- `fetchKey("alice@univ-rennes.fr",true,true)` donne la clé publique d'Alice à utiliser pour le chiffrement
- `fetchKey("alice@univ-rennes.fr",false,true)` donne la clé privée d'Alice à utiliser pour le déchiffrement
- `fetchKey("alice@univ-rennes.fr",true,false)` donne la clé publique d'Alice à utiliser pour la vérification d'une signature
- `fetchKey("alice@univ-rennes.fr",false,false)` donne la clé privée d'Alice à utiliser pour la signature

Par contre, ce qui ne change pas c'est que seule l'utilisatrice alice@univ-rennes.fr peut obtenir ses clés privées.

4.2. Les signatures RSA en vrai

Dans le cours et en TD, on a utilisé la notation $\{m\}_{K_A^{-1}}$ pour représenter un message m signé avec la clé privée de A. En réalité, l'opération de **vérification** d'une signature RSA nécessite d'avoir:

- Le message signé $\{m\}_{K_A^{-1}}$
- La clé publique K_A
- Le message en clair m

Le résultat de la vérification est un booléen qui est vrai si $\{m\}_{K_A^{-1}}$ est bien le résultat de la signature de m par K_A^{-1}. Ainsi, en pratique, si vous envoyez un message signé vous devrez également envoyer le message en clair pour que la signature puisse être vérifiée. Ceci revient à envoyer le couple $m, \{m\}_{K_A^{-1}}$.

5. Points de vigilance

Tout est modifiable mais voici quelques points de vigilance à garder à l'esprit pour que votre application soit bien fonctionnelle quand nous la déploierons sur le réseau.

1. Toute votre application devra être contenue dans le répertoire `etu` .
2. Tout est modifiable, mais le point d'entrée de votre application devra **obligatoirement** se trouver dans le fichier `etu/index.html` .
3. Lisez bien les consignes dans le fichier `src/messenger.ts` . En particulier, lors des interactions avec le serveur, votre application doit transmettre 2 identités au serveur:
   i. `globalUserName` qui est le nom de l'**utilisateur** de l'application. Quand votre application sera déployée, cette identité sera l'identité CAS de l'utilisateur et sera obtenue directement par le serveur. Pendant le développement, cette identité sera `alice@univ-rennes.fr` , `bob@univ-rennes.fr` , ou `charly@univ-rennes.fr` .
   ii. `ownername` est le nom du propriétaire/développeur de l'application. Cette valeur est utilisée pour distinguer les messages émis par différentes applications. Par exemple, les messages émis par l'application `etu` ne seront pas visibles par l'application `ens` et réciproquement. Concrètement, pour l'instant, la valeur de `ownername` est `etu` pour votre application. Cependant, quand nous déploierons toutes les applications de tous les binômes étudiants (dans des répertoires avec des noms différents), nous aurons des `ownername` différents suivant les applications.