Rapport du projet « Encapsulation cryptographique dans un protocole de communication chiffré et signé »

# Principe

### Enregistrement auprès de l’autorité de Certification

**2) Génération des clés**

**Clé privée Alice**

**Clé publique Alice**



login: Alice

IP :

Clé pub :

**Clé de session Alice**



**Autorité de Certification**

1. **Diffusion de la clé publique**

**(Clé privée Alice) chiffrée**

1. **Délivrance de la clé privée**

**Registre de certificats**

1. **Alice fait une demande de certificat**

**Alice**

1. Alice fait une demande de certificat auprès de l’Autorité de Certification (AC). Elle envoie sa demande accompagnée d’un jeton (soit une clé de session de 128 bits, chiffrée avec la clé publique de l’AC) pour permettre à l’AC de lui répondre.
2. L’Autorité de Certification génère une paire de clé RSA, chacune de longueur 1024 bits.
3. L’AC délivre la clé privée au demandeur, soit Alice. La clé privée est chiffrée à l’aide de la clé de session d’Alice en utilisant l’algorithme Blowfish.
4. L’AC publie le certificat dans son registre de certificats. Dans celui-ci, il y a le login, l’IP et la clé publique du demandeur de certificat.

### Alice veut récupérer le certificat de Bob (ou inversement)

### http://lesactualitesdudroit.20minutes-blogs.fr/media/00/02/165028087.jpg

1. **Déchiffrement et vérification clé publique Bob**

login: Alice

IP :

Clé pub :



1. **Renvoi clé publique Bob signée par le CA**

**Message (CertificatBob, signature CA )**

1. **Alice demande certificat de Bob**

**2) Récupération certificat Bob**

**Clé privée Alice**

**Autorité de Certification**

**Registre de certificats**

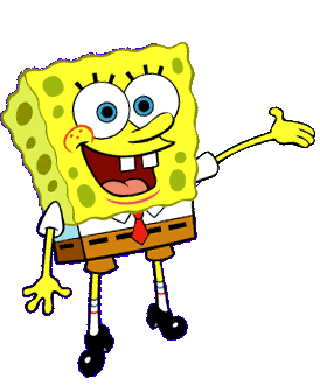
**Alice**

1. Alice contacte l’Autorité de Certification(AC) afin de récupérer les informations nécessaires pour communiquer avec Bob. Il s’agit de la clé publique de Bob, associée à son nom, ainsi que de son adresse IP.
2. L’AC récupère le certificat de Bob dans son registre de certificats.
3. L’AC crée un message chiffré contenant le certificat de Bob et la signature de l’AC. Cette dernière est en fait le hash du certificat de Bob chiffré avec la clé privée de l’AC. Il l’envoie à Alice.
4. Alice doit vérifier que le message vient bien de l’AC et contient bien la clé publique de Bob. Pour cela, elle déchiffre la signature de l’AC avec la clé publique de l’AC et obtient ainsi le hash du certificat de Bob. Elle vérifie le certificat de Bob avec ce hash. L’algorithme utilisé pour hasher est SHA-1.

### Alice envoie un message à Bob

### http://lesactualitesdudroit.20minutes-blogs.fr/media/00/02/165028087.jpg

**2) Création du message à envoyer à Bob**





1. **Envoi du message EnvXML**
2. **Récupération du certificat de Bob**



**Bob**

**Alice**

1. Alice récupère le certificat de Bob, suivant le protocole détaillé précédemment.
2. Alice crée le message à envoyer à Bob, constitué de :
   1. Une clé de session de 128 bits, chiffrée avec la clé publique de Bob (jeton)
   2. Le contenu du message, chiffré avec cette clé de session en utilisant Blowfish
   3. La signature d’Alice : Message chiffré + jeton, hashé, puis chiffré avec la clé privée d’Alice

3- Alice envoie ces trois éléments à Bob sous forme d’un message XML :

<EnvXml>

<From> emetteur </From>

<To> destinataire </To>

<Sign> signature </Sign>

<MsgXml>

<Token> jeton </Token>

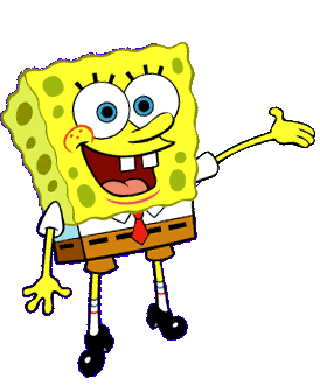
<MsgCrypt> message chiffrée avec la clé de cession </MsgCrypt>

</MsgXml>

</EnvXml>

### Bob reçoit le message d’Alice

**Bob**



**Alice**

1. **Déchiffrement et vérification de l’enveloppe XML**

**Message (CertificatAlice, signature CA )**

1. **Reçoit le message EnvXML**
2. **Récupération du certificat de Bob**



**Bob**

1. Bob reçoit EnvXML. Il récupère le nom de l’émetteur et veux sa clé publique pour déchiffrer la signature d’EnvXML.
2. Bob contacte l’Autorité de Certification(AC) afin de récupérer le certificat d’Alice.
3. Bob déchiffre la signature d’EnvXML à l’aide de la clé publique d’Alice et récupère ainsi un hash de MsgXML. Il hashe le MsgXML d’EnvXML et le compare au hash obtenu en déchiffrant la signature. Uniquement s’ils sont identiques, le processus continue. Bob déchiffre ensuite le jeton de MsgXML avec sa propre clé privée, il obtient ainsi la clé de session. A l’aide de cette dernière, il déchiffre le message chiffré et obtient ainsi le message que lui a envoyé Alice.

# Implémentation

Notre projet est composé de deux exécutables :

1. Un programme serveur pour l’autorité de certification
2. Un programme pour les utilisateurs.

Ces programmes sont développés en C++ et communiquent via la bibliothèque de sockets standards C.

Nous n’avons pas développés d’algorithmes cryptographiques. Nous avons utilisé les algorithmes de la bibliothèque C d’openssl (RSA pour chiffrer les clés de session, Blowfish pour les messages, et SHA-1 pour les hash).

# Améliorations proposées

Actuellement, le protocole fonctionne mais n’est pas utilisable dans la vie réelle pour plusieurs raisons :

* Les programmes ne mémorisent aucun certificat. En cas de redémarrage d’un client ou de l’autorité, tout doit être repris à zéro.
* Aucune gestion des conflits n’est faite : si Bob réclame un certificat alors que l’autorité connait déjà un Bob, cette authentification est refusée (d’où le problème si le client appelé Bob redémarre et veut se ré-authentifier).

La solution serait de stocker les certificats dans des fichiers et de les récupérer au démarrage de l’application. Il faudrait alors implémenter un système de date d’expiration des certificats pour ne pas bloquer indéfiniment un nom d’utilisateur.

Un deuxième problème est que les clients n’accusent pas réception de leur clé privée lors de l’authentification. Il est possible que l’autorité bloque un nom d’utilisateur alors que cet utilisateur n’a pas reçu sa clé privée, et veut donc se ré-authentifier. La solution serait ici de modifier le protocole d’authentification pour que le client confirme la bonne réception de sa clé privée.