Rapport de TL cryptographie SIS 2015

Introduction :

Pour réaliser ce TL, nous avons essayé de faire des choix de conception cohérents pour avoir à la fois une structure logique et organisée, une interface utilisateur agréable à utiliser. Nous avons eu quelques difficultés à comprendre pleinement les tenants et aboutissants du sujet mais nous sommes satisfaits de ce que nous avons réalisés au final. Nous allons ici vous présenter notre code source, les difficultés rencontrées et les conclusions tirées de ce TL.

1. **Code Source**
2. Arborescence

*ihm*

|\_ EquipmentPanel

|\_ PrincipalPanel

*security*

|\_ Certificat

|\_ CertificateAuthority

|\_ DerivateAuthority

|\_ Equipment

|\_ PaireCleRSA

*sockets*

|\_ Client

|\_ Server

1. IHM

Nous avons créé une interface homme machine en swing très simpliste avec une barre de menu sur la gauche de la fenêtre qui contient toutes les fonctions demandées dans le sujet, et une console qui nous affiche les résultats de ces dites fonctions.

1. Sockets

Le package sockets contient les deux classes serveur et client qui étaient proposées aussi dans le TL. Nous avons ajouté des méthodes pour envoyer des objets différents des String. Cela nous permet plus de flexibilité dans les protocoles d’insertion et de synchronisation. Les objets serveur et client peuvent être créés par des équipements.

1. Security objects

Nous avons créé ensuite les différents objets nécessaires au bon fonctionnement du logiciel. Les méthodes principales d’insertion et de synchronisation se trouvent dans la classe équipement.

* 1. Certificat et paire de clé RSA

Nous avons très peu modifié les classes certificat et clé RSA proposées dans le TL, nous avons juste rajouté un constructeur pour créer l’autocertificat en plus du constructeur de base pour faire un certificat à partir d’une clé en paramètre. Ces objets ont été rendus serializable pour pouvoir être envoyés via les méthodes crées dans les classes serveur et client.

* 1. Les CA et DA

Nous avons fait une classe pour formaliser les autorités certifiées et les autorités dérivées, avec la possibilité de convertir une autorité certifiée en autorité dérivée pour le transfert.

* 1. Equipement

Le constructeur réalise l’autocertification de l’équipement au moment de sa création.

Les méthodes d’affichage des informations de l’équipement sont aussi incluses dans cette classe. Les méthodes pour créer un serveur ou un client dans l’application courante sont aussi dans cette classe.

L’essentiel de la logique du programme se trouve dans les méthodes de synchronisation et d’insertion. Nous avons réalisé deux méthodes pour chacun de ces protocoles, une pour le serveur, l’autre pour le client. Elles contiennent quasiment le même code mais avec des différences au niveau de l’ordre d’envoi des données (en alternance, le client en premier puis le serveur répond).

1. **Problèmes rencontrés**

L’un des principal problème que nous avons rencontré est la synchronisation de notre travail. En effet, nous avons principalement travaillé depuis chez nous, sur deux PC différents et avons essayé pour cela de prendre en main l’outil web GitHub. Malgré de nombreux problèmes à cause du plugin d’eclipse eGit nous avons fini par trouver notre rythme et avons pu développer en parallèle sur le projet, ce qui s’est révélé assez efficace et agréable.

Nous avons rencontré quelques difficultés pour comprendre l’implémentation des communications entre serveurs et clients. Notre choix a été de permettre aux équipements d’être sur commande des serveurs ou des clients, pour l’insertion et/ou la synchronisation. Nous n’avons donc qu’un unique main pour toute notre application qui peuvent réaliser toutes les opérations.

Nous ne sommes toujours pas sûr que cette implémentation est exactement celle demandée, ou si il aurait fallu créer des équipements dédiés serveurs et d’autres dédiés clients. Notre choix est venu du fait que notre solution nous proposait plus de flexibilité lors des tests et qu’elle nous semblait plus complète.

Nous avons eu quelques difficultés aussi au moment où nous avons fait les cas de synchronisation où les équipements ne se faisaient pas confiance. Après relecture du sujet, nous avons compris que le cas où aucun des deux équipements ne se connaissait, la synchronisation n’était pas possible (et que ce cas était très peu probable en situation réelle).

Pour qu’un équipement A prouve à un équipement B auquel il fait confiance qu’il est lui-même digne de confiance, on suit le processus suivant :

* A envoie sa liste d’autorités certifiées sans les certificats
* B renvoie une autorité C qu’il a en commun avec lui
* A renvoie le certificat associé à C
* B vérifie ce certificat et le considère digne de confiance

Nous avons aussi repéré un problème dans notre programme, au moment de la phase de synchronisation. En effet, on peut mentir sur son identité en prenant le nom et la clé publique d’un des membres du réseau. Il appartient donc de vérifier que la personne ne nous ment pas sur son identité. Pour cela, nous demandons à l’équipement de s’authentifier via un certificat réalisé avec une clé publique temporaire.

Un dernier bug que nous avons trouvé était l’ajout de l’équipement lui-même dans ses autorités dérivées au moment de la synchronisation.

1. **Conclusions**

Pour conclure ce TL, nous avons apprécié d’allier le développement logiciel avec la sécurité informatique pour arriver à un produit fini, bien que loin d’être exploitable à notre sens. Cela nous a permis de revoir des outils tels que les sockets, swing ou encore les threads, ainsi que de prendre en main des outils de travail d’équipe ( gitHub ) qui nous resserviront de nombreuses fois à n’en pas douter.