ENSIIE

Rapport du SIPD – Contrôle d'Accès
Groupe 2

Réalisé par
Qixiang PENG
Kai HUANG
Yao SHEN
Nan LIU
Mai 2017

Contents

1 Introduction

Tient compte de l'importance des données personnelles, pour garantir la vie privée des individus et empêcher la violation de secret industriels, commerciaux et diplomatiques, l'equipe SMIS(UVSQ-INRIA-CNRS) a développé un Serveur Personnel de Données Sécurisé appelé PlugDB. PlugDB permet à l'individu d'exercer un contrôle sur ses données personnelles, tout en préservant leur disponibilité et la capacité de les partager. Ce serveur personnel est embarqué dans un token sécurisé qui combine la sécurité matérielle d'une carte à puce, la capacité de stockage d'une carte microSD, un lecteur d'empreintes digitales et des capacités de communication USB et Bluetooth. Le serveur embarqué structure toutes les données de l'utilisateur sous la forme d'une base de données relationnelle, indexe ces données et est capable de les interroger en SQL-light et d'y associer des droits d'accès.

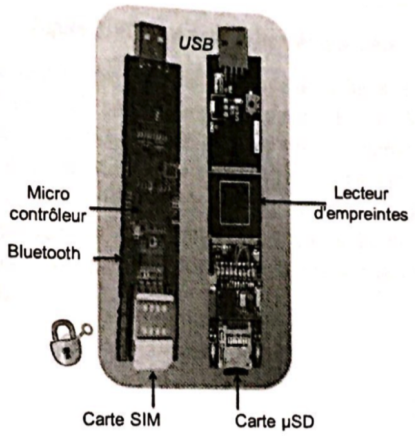

Figure 1: Token hébergeant un serveur personnel

L'objectif du projet est de développer une application Privacy-by-Design en exploitant la technologie PlugDB. Pour cette édition 2015 de projet, il est proposé de développer une application d'échanges de fichiers sécurisée, c'est à dire intégrant du contrôle d'accès et du contrôle d'usage.

Chaque individu dispose d'un espace numérique personnel organisé autour d'une cellule de confiance, appelé dans la suite du document TCell(Trusted Cell). Une Tcell est constituée d'un token hardware sécurisé embarquant le logiciel PlugDB et connecté au PC de l'utilisateur.

Le sujet de notre groupe est "Contrôle d'Accès", les missions principales du sujet sont, premièrement de créer une classe de filchier; deuxièmement de créer une classe de tag;

troisièmement de créer une classe d'utilisateur; quatrièmement de créer Association1 pour associer des tags aux fichiers dans la base de données; cinquième de créer Association2 pour associer des tags aux utilisateurs dans la base de données; finalement de définir ces règles, un utilisateur ne peux que visiter les fichiers qui ont la même tag de lui-même.

2 Schéma

La Figure2 est le schéma UML qui décrit les tableaux principals et les relations entre ces tableau.

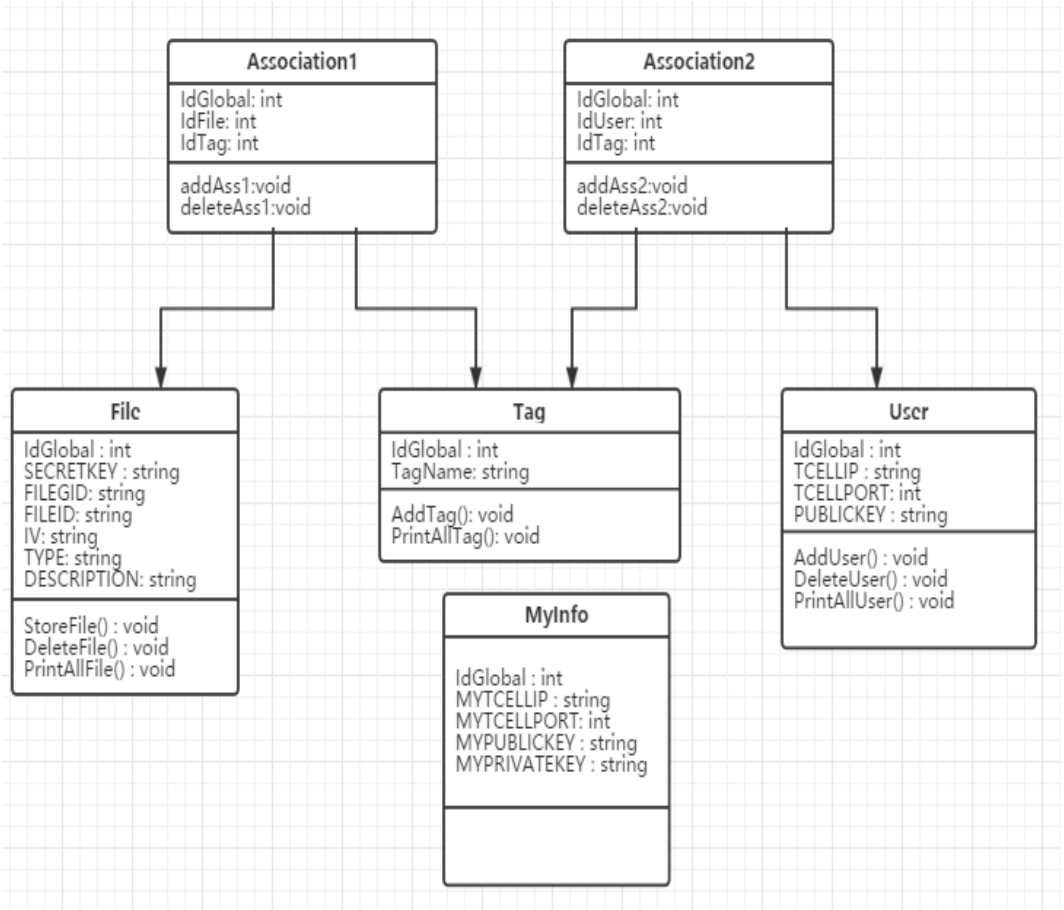

Figure 2: Le UML de notre base de données

Par le schéma UML, nous pouvons trouver facilement il y a 6 classes : Myinfo, User, File, Tag, Association1, Association2.

1. Myinfo

   Cette classe stocker l'information de notre token. Elle a 5 attributs: **IdGlobal**, **MYCELLIP**, **MYTCELLPORT**, **MYPUBLICKEY** et **MYPRIVATEKEY**. Elle n'a rien de association avec les autres tableaux.

2. User

   Cette classe stocke les informations des utilisateurs. Elle a 4 attributs: **IdGlobal**, **TCELLIP**, **TCELLPORT**, **PUBLICKEY** et 3 méthodes **AddUser()**, **DeleteUser()**, **PrintAllUser()**.

3. File

   Cette classe stocke les informations des fichiers ,elle a 7 attributs: **IdGlobal**, **SECRETKEY**, **FILEGID**, **FILEID**, **IV**, **TYPE**, **DESCRIPTION** et 3 méthodes : **StoreFile()**, **DeleteFile()**, **PrintAllFile()**.

4. Tag

   Cette classe stocker les tag des utilisateurs et des fichiers. Elle offre des possibilités pour créer une relation entre **User** et **File**. Elle a 2 attributs **IdGlobal**, **TagName** et 2 méthodes **AddTag()**, **PrintAllTag()**.

5. Association1

   Cette classe stocke toutes les associations entre **File** et **Tag**. Chaque tuple a **IdFile** et **IdTag** qui indique le tag cet fichier a. **IdGlobal** est la clé primaire.

6. Association2

   Cette classe stocke toutes les associations entre **User** et **Tag**. Chaque tuple a **IdUser** et **IdTag** qui indique le tag cet utilisateur a droits pour l'accès . **IdGlobal** est la clé primaire.

Nous combinons les 3 tableaux **User**, **Association2** et **Tag** pour créer la relation entre utilisateur et ses tags;
Nous combinons les 3 tableaux **File**, **Association1** et **Tag** pour créer la relation entre tag et ses fichiers.
Ici, le tableau **Tag** n'est pas référencé au même temps par les 2 tableaux. Ces 2 références sont successives.

3 Nos idées

Selon ceux décrits précédemment, par le UML que nous avon crée, comment realiser le contrôle d'accès d'utilisateur ? Nos idées sont dessous :

3.1 La base de données

Par le schéma que nous avons présenté, nous d'abord créons les tableaus et les associations entre les tableaus. Au début, nous créons 3 tableaus **File**, **Tag** et User. Et après le tableau **Association1** qui peut associe les **File** et **Tag**, le tableau **Association2** qui associe les **Tag** et **User**. A la demande, nous aussi créons le tableau **MyInfo** pour stocker les informations de notre Tcell.

3.2 Accès au fichier

Un utlisateur qui utilise le Tcell voudrait accéder à un fichier. Nous obtenons **IdGlobal** du tableau **User**, cherchons dans le tableau **Association2** par **IdUser** qui référence le tableau **User**. Après on obtient les tuples correspondants dans le tableau **Association2**. Par ces tuple [IdGlobal, IdUser, IdTag], nous pouvons trouver les **IdTag** qui représentent les fichiers ce utilisateur peut accéder.

Par ces **IdTag** qui correspond les **IdGlobal** dans le tableau **Tag**, nous cherchons les tuples dans le tableau **Association1 "WHERE IdTag = IdGlobal"**. Par ces tuples, nous obtient les **IdFile** qui référence **IdGlobal** dans le tableau **File**. A la fin, par ces Id du fichier, cet utilisatuer peut maintenant accéder aux les fichier qu'il a des droits.

3.3 Partage un fichier

Nous choisissons la mode **PUSH** pour partager un fichier avec un autre utilisateur. C'est pas un vrai partage. C'est réalisé par un envoie ce fichier à l'autre. Précisément, A partage un fichier avec B. A envoie une copie du fichier à B. Alors B peut avoir un nouveau tag qui l'aidera à accéder à cette copie désormais. Donc, c'est vraiment 2 utilisateurs lisent 2 fichiers. Les 2 fichiers sont mêmes mais avoir différents tags. Nous donnons un nouveau droit à B et il peut accéder à cette copie. Il n'y a pas de effet à A.

4 Réalisation

Pour réaliser toutes nos idées et toutes les fonctions du Tcell pour contrôler l'accès, nous utilisons les méthodes suivantes formant la structure du programme :

Il y a 4 projects : client, core, JDBS et tcell.

Dans project **client**, on met le test programme **AppMain** pour exécuter tout le programme.

Dans project **core**, on met les packages **api**, **beans**, **command**, **configuration**, **cryptoTools**, **message**, **tools**.

Project **JDBC** pour configurer les fichiers qui interagissent avec la base de données.

Dans project **tcell**, on mets des packages qui aident la base de données faire les différenes opérations selon les différents commandes.

4.1 Développement

Dans le fichier **ClientAPI**, Il y a beaucoup de APIs, Nous allons presenter le développement par un exemple – **addTag**. Quant à les détails du API, Nous dirons dans la deuxième partie.

Dans **AppMain.java**, quand nous appelons **ClientAPI.addTag()**, il va chercher la fonction **addTag** dans le fichier **ClientAPI**. Cette fonction appelle la fonction *addTag* de la classe AddTag dans le package **messages**. Le package **messages** est responsable de formuler les détails de communiquer avec la base de données.

Dans cette fonction *addTag*, il d'abord crée un **socket** pour communiquer et envoie le **AddTagCommand**. Ici, la classe AddTagCommand est définie dans le package **Command**, pour chaque command, il a un numéro de commande et le nom de tag.

Après, l'autre côté a reçu la commande, avec l'aide de **ClientConnectionManager** (dans le package **daemon** de **tcell**) , il peut identifier la type de la commande et faire les opérations correspondantes sur la base de données en appelant les fonctions dans le fichier TcellDAOToken.

A la fin, il va retourner un statut, et on va interpréter le statut en fonction de statut qu'il retourne.

De même, nous avons beaucoup d'APIs qui fonctionnent comme addTag ci-dessus. Par exemple, l'insertion d'un tuple dans le tableau Tag, le test d'existence d'un tag, l'insertion d'un tuple dans le tableau Association1, Association2, l'affichage de tous les tags, l'affichage de tous utilisateurs, l'affichage de tous les fichiers, etc.

4.2 API

Les APIs suivantes sont les APIs principales pour les clients :

4.2.1 StoreFile

Cette API permet aux utilisateurs de stocker les fichiers dans le Tcell et de rendre une classe **FileDesc** . Elle ajoute un nouveau tuple dans le tableau **File**.

4.2.2 AddTag

Cette API permet d'ajouter un nouveau tag dans le tableau **Tag**. Ce tag est le tag du fichier qu'on vient de stocker. Si il est déjà existe, on fait rien. Si il n'existe pas, on crée un nouveau tuple dans le tableau **Tag**.

4.2.3 AddAssociation1

Cette API permet de créer une association entre le fichier que l'on vient stocker dans le tableau **File** et son tag dans le tableau **Tag**. Elle crée un nouveau tuple dans le tableau **Association1**.

4.2.4 addFileWithTag

Cette API permet de stocker le fichier et au même temps créer les relations entre cet fichier et ses Tags. Cette fonction appelle 3 APIs : **StoreFile**, **AddTag** et **AddAssociation1**. C'est-à-dire il y a 3 étapes : stocker ce fichier dans la base de données ; ajouter son Tag; créer une association entre 2 tableaux.

4.2.5 AddUser

Cette API permet d'ajouter un nouveaux tuple dans le tableau. Elle retourne une classe **User**.

4.2.6 AddAssociation2

Cette API permet de créer une association entre le fichier que l'on vient d'ajouter dans le tableau **User** et son tag dans le tableau **Tag**. Elle crée un nouveaux tuple dans le tableau **Association2**.

4.2.7 addUserWithTag

Cette API permet d'ajouter un nouveaux utilisateur et au même temps créer les associations entre cet utilisateur et ses Tags. Cette fonction appelle 3 APIs : **AddUser**, **AddTag** et **AddAssociation2**. C'est-à-dire il y a 3 étapes : ajouter cet utilisateur dans la base de données ; ajouter son Tag; créer une association entre 2 tableaux.

4.2.8 GetAllFileDescOfOneUser

Cette API permet d'obtenir tous les FileDesc d'un certain utilisateur. Cette fonction appelle 3 APIs : **getTagIdByUserId**, **getFileIdListByTagId** et **getFileDescByFileId**.

4.2.9 getTagIdByUserId

Cette API permet de chercher tous le tag d'un utilisateur. Dans le tableau Association2, nous cherchons le IdTag qui ont un IdUser correspondant. Et il retourne un IdTag .

4.2.10 getFileIdListByTagId

Cette API permet de chercher tous les IdGlobals de fichier par le **IdGlobal de Tag** que l'on vient de trouver. En utilisant le tableau **Association1**, elle retourne une liste de **IdGlobal de fichier**. Alors, on a une liste de **IdGlobal** du **File**.

4.2.11 getFileDescByFileId

Cette API vise à chercher un FileDesc dont son **IDGlobal** est le IdGlobal de File dans la liste de IdGlobal de fichiers que l'on vient de trouver en appelant la fonction **getFileIdListByTagId**. Et elle retourne une classe **FileDesc**.

4.2.12 Les autres

Nous avons aussi ajouté autres APIs, comme la suppression d'un tag, la suppression d'un tag d'un utilisateur, la suppression d'un fichier, etc.

5 Difficultés et solutions

Au cours de la réalisation de ce projet, nous avons rencontré beaucoup de difficultés. Nous avons résolu une partie avec succès, mais il reste encore quelques problèmes.

1. C'était difficile pour nous de créer le schéma UML au début. Nous devons créer un schéma en forme d'arbre. Après discutions avec le professeur, à la fin, nous avons réussi à trouver un bon schéma.
2. C'est un système assez complexe, ce n'était pas facile pour comprendre et décider les séquences pour réaliser les fonctions. Nous avons pris trop de temps de comprendre chaque petit point.

3. La communication de **socket** – dans le package **beans** nous avons des classes qui stockent les informations dans la base de données. Nous utilisons writeobject() et readobject() pour transférer les instance des classes, mais il faut implémenter l'interface **Serializable**. Au début on a supprimé "implement Seralizable", il a conduit à ne pouvait pas faire le transfert avec succès. ça faisait longtemps pour trouver cet erreur.

4. Chaque fois que nous ajoutons une fonction, ça ne marche plus. Il faut toujours redémarrer l'eclipse même la machine virtuele.

6 Conclusion

Au cours de accomplir ce projet, nous avons rencontré beaucoup de difficultés mais nous avons aussi appris beaucoup de choses. Il est important de garantir la sécurité des informations personnelles. Token est une magique chose et nos travails sont pratiques et valeureux.