Rapport Projet de Structures Des Données

Nom Prénom N°Etu : Tahir Aans 28710067
Nom Prénom N°Etu : Ammour Wissem 28705652
Groupe 1

Blockchain appliquée à un processus électoral

Introduction

Dans le cadre du projet, nous nous intéressons aux élections par scrutin uninominal majoritaire à deux tours. Chaque citoyen, inscrit sur les listes électorales, va pouvoir voter pour 1 candidat, ce vote (appelé message dans notre programme) va devoir être crypté en utilisant la clé publique (pKey) du candidat choisi. Le candidat pourra quant à lui décrypter ce message en utilisant sa clé secrète (sKey). Le but de ce projet est donc de garantir l'intégrité, la sécurité et la transparence de l'élection.

Pour cela nous avons d'abords procédé à l'implémentation d'outils de cryptographie (grâce au protocole RSA), puis à la création d'un système de déclarations sécurisées par chiffrement asymétrique. Ensuite nous avons procédé à la manipulation d'une base centralisée de déclarations. Enfin nous avons Implémenté un mécanisme de consensus et manipulé une base décentralisée de déclarations.

Nous allons d'abord définir les différentes structures utilisées dans ce projet, puis nous expliquerons les fonctions principales implémentées. Nous allons ensuite passer à la description du code d'abord avec les commandes à utiliser pour la compilation et le lancement puis avec les différents fichiers que composent notre code. Enfin nous conclurons avec les limites de ce projet.

Définition des structures

Nous avons 9 structures afin de stocker et manipuler les différentes données.

***

Manipulations de structures sécurisées

```
typedef struct key{
    long k;
    long n;
}Key;
```

La structure Key permet de stocker un couple de nombre de type long qui formeront une clé (privée ou publique).

```
typedef struct signature{
    long *content; //tableau
    int size;
}Signature;
```

La structure Signature permet de stocker un tableau (content) de long (où chaque long représente un caractère (char) crypté), ainsi que sa taille (size). Permettant de vérifier l'authenticité des déclarations de vote.

```
typedef struct protected{
    Key* pKey;
    char* mess;
    Signature* sign;
}Protected;
```

La structure Protected permet de stocker la clé publique d'un citoyen, son vote (message) et la signature associée au vote. Elle crée des déclarations signées (données protégées).

***

Lecture et stockage des données dans des listes chaînées

```
typedef struct cellKey {
    Key * data ;
    struct cellKey* next ;
} CellKey ;
```

La structure CellKey permet de stocker une liste simplement chaînée de clés (Key). struct cellKey* next ; -> permet de passer au champs suivant de la liste.

```
typedef struct cellProtected{
    Protected* data ;
    struct cellProtected* next ;
} CellProtected ;
```

La structure CellProtected permet de stocker une liste simplement chaînée de déclarations signées (Protected).

***

Détermination du gagnant de l'élection

```c
typedef struct hashcell{
    Key* key;
    int val;
} HashCell;
```

La structure HashCell permet de stocker les clés (key) selon les valeurs (val) dans la table de Hachage (structure ci-dessous).

```c
typedef struct hashtable{
    HashCell** tab;
    int size;
} HashTable;
```

La structure HashTable permet de stocker les « cellules » (HashCell) sous forme de tableau (tab), ainsi que sa taille (size)

***

Structure d'un block et persistance

```c
typedef struct block{
    Key* author;
    CellProtected* votes;
    unsigned char* hash;
    unsigned char* previous_hash;
    int nonce;
}Block;
```

La structure Block permet de stocker plusieurs données différentes. Ainsi un block contiendra :

- la clé publique de son créateur (author).
- Une liste de déclarations (signées) de vote (votes).
- La valeur hachée du bloc (hash).
- La valeur hachée du bloc précédent (previous_hash).
- Une preuve de travail (nonce).

***

Structure arborescente

```
typedef struct block_tree_cell {
    Block * block ;
    struct block_tree_cell * father ;
    struct block_tree_cell * firstChild ;
    struct block_tree_cell * nextBro ;
    int height ;
} CellTree ;
```

Cette structure représente un arbre et permet de représenter des noeuds avec un nombre arbitraire de fils, sans avoir à créer de structure de liste chaînée.

Fonctions Principales

void generate_random_data(int nv, int nc): la fonction génère des clés publiques et privées nv fois. Puis enregistre ligne par ligne le couple de clés (publique et secrète) dans le fichier keys.txt. Parmi les clés publiques, on choisit aléatoirement nc fois différentes clés et on les enregistre ligne par ligne dans le fichier candidates.txt. On prend nv fois différentes clés et on crée un protected et une signature (représentant resp. une déclaration signée et la preuve de l'authenticité de la déclaration). On enregistre ligne par ligne les protected (declarations) dans le fichier declarations.txt.

void compute_proof_of_work(Block *B, int d): on commence avec l'attribut nonce égal à zéro, puis on incrémente l'attribut nonce jusqu'a ce que la valeur hachée du bloc commence par (d) zeros successifs.

Key* compute_winner(CellProtected* decl, CellKey* candidates, CellKey* voters, int sizeC, int sizeV): calcule le vainqueur de l'élection, étant donnés une liste de déclarations avec signatures valides (decl), une liste de candidats (candidates), et une liste de personnes autorisées à voter (voters). La fonction commence par créer deux tables de hachage :

— H_C : une table de hachage (de taille sizeC) pour la liste des candidats.

— H_V : une table de hachage (de taille sizeV) pour la liste des votants.

La fonction parcourt ensuite la liste des déclarations, et pour chaque déclaration, elle vérifie grâce aux deux tables de hachage que :

— la personne qui vote a le droit de voter et qu'elle n'a pas déjà voté,

— la personne sur qui porte le vote est bien un candidat de l'élection. Si toutes ces conditions sont vérifiées, le vote est comptabilisée dans la table H_C et la table H_V est mise à jour pour indiquer que ce votant vient de voter. Une fois que tous les votes ont été comptabilisés, la fonction détermine le gagnant en utilisant la table H_C.

Commandes
(MAKEFILE)

- compilation : make all1 (pour compiler le tout avec le test1 qui corresponds au test des exercices 1et2).

  make all2 (pour compiler le tout avec le test2 qui corresponds aux restes des exercices).

  make all (pour tout compiler).

- lancement : ./test1 (pour tester les exercices 1 et 2).

  ./test2 (pour tester les exercices restants)

Fichiers

Nous avons choisis de créer un « fichier.c » pour chaque exercice. Donc il y'a 9 fichiers pour les 9 exercices (par exemple exercice1.c pour l'exercice 1 etc).
De plus nous avons 2 autres « fichier.c » correspondant aux fichiers tests de notre projet.

Puis pour le header nous avons décider de mettre toutes les structures et le prototype de toutes les fonctions dans un seul et même fichier qui est « structures.h ». Nous avons commenté ce fichier afin que nous puissions savoir quelle structure ou quelle prototype de fonction appartient à quelle exercice, afin de faciliter la recherche de notre côté et la compréhension de notre code par d'autres personnes.

Conclusion

Il est intéressant d'utiliser la blockchain dans le cadre du processus de vote. La mise en œuvre assure la transparence du vote car tous les évaluateurs (assesseurs) ont accès à toutes les déclarations.

De plus, la structure de la blockchain peut empêcher la fraude dans une certaine mesure, il est donc nécessaire de créer en permanence des blocs contenants des fraudes (contrairement aux tables de hachages qui ne nous permettent pas de trouver les fraudes).

Il y’a cependant une certaine limite à ce projet. En effet ce projet n'est pas sans faille, la fraude est possible elle la limite seulement. de plus il n'y pas de revérification faite sur la liste des déclarations des votants après un bloc initialisé. Enfin nous n’avons pas la garanti d'avoir des clés différentes ce qui pose problème.