

# Rapport de projet

Réda Boudrouss

## Contents

<b>Introduction</b>	<b>1</b>
<b>Description du projet</b>	<b>1</b>
Organisation des dossiers . . . . .	1
Organisation du code . . . . .	2

## Introduction

Ce projet a pour but de faire un système de vote “electoral” de manière décentralisé avec de la blockchain. Toute notre cryptographie et sécurité est assurée par une approche naïve du chiffrement RSA.

## Description du projet

### Organisation des dossiers

Le projet se divise comme suit :

- **bin/** qui contient les fichiers compilés
- **blockchain/** qui contient les données stockées et les données aléatoires générés
- **data/** qui contient les données d’analyses des différents graphiques
- **img/** qui contient les graphiques stockés sous format image
- **report/** qui contient tout les fichiers nécessaires à la réalisation du présent rapport
- **src/** Ce dossier contient l’essentiel du code source du projet, nous expliqueront plus tard l’organisation du code.
- **include/** contient tout les headers du projet
- **tests/** contient tout les jeux de test unitaires par fonction. Tout est assurée sans memory-leak !

## Organisation du code

Chaque header dans le dossier **include/** a son code dans le fichier portant le même nom dans **src/** et a ses tests unitaires dans les fichiers portant le même noms suivi de **\_tests.c** dans le dossier **tests**. Excepté le fichier **files.h** et le fichier **main.c** qui n'a pas de header.

- **prime**

Ces fichiers contiennent toutes les fonctions de manipulation des nombres premiers et de l'aléatoire.

- **rsa**

Ces fichiers contiennent une approche naïve du RSA avec une manière de générer les clés publiques & privés et de décrypter/crypter des messages.

- **keys**

Ces fichiers reposent sur la structure suivante :

```
typedef struct _key
2 {
    long val;
4    long n;
} Key;
```

Cette structure permet de stocker et gérer les informations nécessaires aux clés publiques et privés RSA. Les fichiers contiennent donc les fonctions permettant d'initialiser les clés, de les libérer et de les convertir en chaîne de caractère et vice versa.

- **signs**

Ces fichiers reposent sur la structure suivante :

```
typedef struct _sign
2 {
    long *content;
4    int size;
} Signature;
```

Cette structure permet de stocker les informations nécessaire à une signature. Dans notre code cela représente à la cle publique du candidat (en chaîne de caractère) encrypté en RSA avec la clé privé du votant.

Les fichiers contiennent alors les différentes fonctions d'initialiser et de libérer une structure **Signature** et donc de signer une chaîne de caractère, sans oublier la conversion en chaîne de caractère et vice versa.

- **protec**

Ces fichiers reposent sur la structure suivantes :

```
typedef struct _protec
2 {
    Key *pKey;
```

```

4     char *mess;
      Signature *sgn;
6 } Protected;

```

Cette structure permet de stocker les informations d'une déclaration de vote. **pKey** étant la clé publique du votant, **mess** la chaîne de caractère de la clé publique du candidat et **sgn** la signature.

Les fichiers contiennent donc les différentes fonctions pour initialiser, vérifier et libérer une **Signature** et convertir une chaîne de caractère.

- **cells**

*J'aurais du découper ce fichier en plusieurs fichiers.*

Ces fichiers regroupent les fonctions dépendantes des 3 structures suivantes :

- **CellKey & CellProtected**

```

typedef struct cellKey
2 {
      Key *data;
4     struct cellKey *next;
} CellKey;
6
typedef struct cellProtected
8 {
      Protected *data;
10    struct cellProtected *next;
} CellProtected;

```

Cette structure est une liste chaînée de Key ou de Protected. Cela nous permettra de gérer une grande quantité de clé ou de déclaration de vote dont on ne connaît pas la taille en avance. Nous avons donc des fonctions pour les manipuler. (Ajouter un élément, libérer la liste, l'afficher et lire les données d'un fichier).

- **HashCell & HashTable**

```

typedef struct hashcell
2 {
      Key *key;
4     int val;
} HashCell;
6
typedef struct hashtable
8 {
      HashCell **tab;
10    int size;
} HashTable;

```

Cette structure est une table de hachage. Cela permet de récupérer et modifier les informations sur les votants et les candidats de manière rapide. Cela est nécessaire car notre programme doit être capable de gérer un très grand nombre de votant.

Nous avons donc aussi toutes les fonctions nécessaires pour cela. la fonction de hashing, et les fonctions permettant de créer libérer insérer une **HashCell**.

Nous avons aussi une fonction importante :

```
Key *compute_winner(CellProtected *decl, CellKey *candidates, CellKey  
    *voters, int sizeC, int sizeV);
```

Cette fonction détermine le gagnant de l'élection à partir de la liste chaînées de vote et des clés des votant et des candidats. Nous l'expliquerons plus en détail plus tard dans le rapport.