E-Voting using Homomorphic Encryption
Rapport de projet

Adrien Langou - Aurélien Rebourg - Pierre Blaess - Lucas Theze
Epita 2024 - SRS

9 mai 2023

Table des matières

1 Présentation du chiffrement homomorphe

Le chiffrement homomorphe est une forme de chiffrement permettant d'effectuer des opérations classiques telles que des additions ou des multiplications sur des données chiffrées sans avoir besoin de les déchiffrer. Il existe 3 différents type de chiffrement homomorphes :

- Le chiffrement homomorphe **partiel** ;
- Le chiffrement **quelque peu** homomorphe ;
- Le chiffrement homomorphe **complet**.

Le chiffrement homomorphe **partiel** effectue un seul type d'opération *(addition OU multiplication)* autant de fois que l'on souhaite. Le chiffrement **quelque peu** homomorphe effectue plusieurs types d'opérations, mais pour un nombre limité de fois. Et enfin, le chiffrement homomorphe **complet** effectue plusieurs types d'opérations autant de fois que l'on souhaite. Le chiffrement complet semble être la meilleure des trois options, mais ce type de chiffrement est très lent et nécessite un espace de stockage important. Plus globalement, ces 3 types de chiffrement ne sont pas encore très utilisés dans l'industrie pour des raisons de performances et de coût élevé.

Cet algorithme de chiffrement semble adapté pour un système de *e-voting*. Nous allons voir à travers ce rapport qu'il comporte des avantages et des inconvénients. Durant ce projet, nous avons choisi le chiffrement homomorphe partiel avec l'opération d'addition, car un système de vote ne nécessite pas de faire des multiplications.

2 Description détaillée du protocole du projet

Notre protocole fonctionne en 3 actions :

1. Authentification du client ;
2. Vote pour un des candidats ;
3. Récupération des résultats.

Remarque...

Les actions `1` et `2` sont appelées l'un après l'autre et vont de pair ensemble. Tandis que l'action 3, est appelée seule et ne dépend pas des autres. En effet, cette dernière ne nécessite aucun pré-requis du client, le serveur s'occupe de vérifier le droit de voir *(ou non)* les résultats.

2.1 Explication du protole de e-voting

Initialisation :
Au démarrage, le **serveur d'authentification** crée une paire de clés.
Le **serveur d'addition**, ce dernier interroge le **serveur d'authentification** pour récupérer la clé publique et la durée du vote.

Action 1 :
Le **serveur d'addition** demande la clé publique et la date de fin de vote.
Le client envoie la requête '`/pk?name={hashname}`' au **serveur d'authentification**. Ce dernier vérifie que le nom du votant est autorisé à voter et si ce dernier n'a jamais pas encore participé au vote. Si le votant à le droit de voter le **serveur d'authentification** crée un token aléatoire et le stock dans une base donnée commune entre les serveurs. Il renvoie, au client, ce token ainsi que sa clé publique.

Action 2 :
Le client envoie une requête au **serveur d'addition** avec en paramètre le token généré ainsi qu'un vecteur d'entier correspondant à son vote. Ce dernier est chiffré avec la clé publique du serveur.
Le **serveur d'addition** vérifie que le vote est encore en cours et que le token envoyé n'a jamais été utilisé. Il additionne le vote reçu avec le vecteur de résultat.

Action 3 :
Une fois le vote terminé, les utilisateurs peuvent faire une requête '`/results`' au **serveur d'authentification** pour récupérer les résultats chiffrés. Ce dernier va interroger le serveur **d'addition** pour récupérer le résultat final des votes. Il peut le déchiffrer avec sa clé privée pour renvoyer le résultat au client.

2.2 Schéma fonctionnel

`Ci-dessous le schéma fonctionnel de notre protocole de vote.`

Information,

Les couleurs (Rouge, Bleu et Verte) ci-dessus correspondent aux actions sur le schéma suivant.

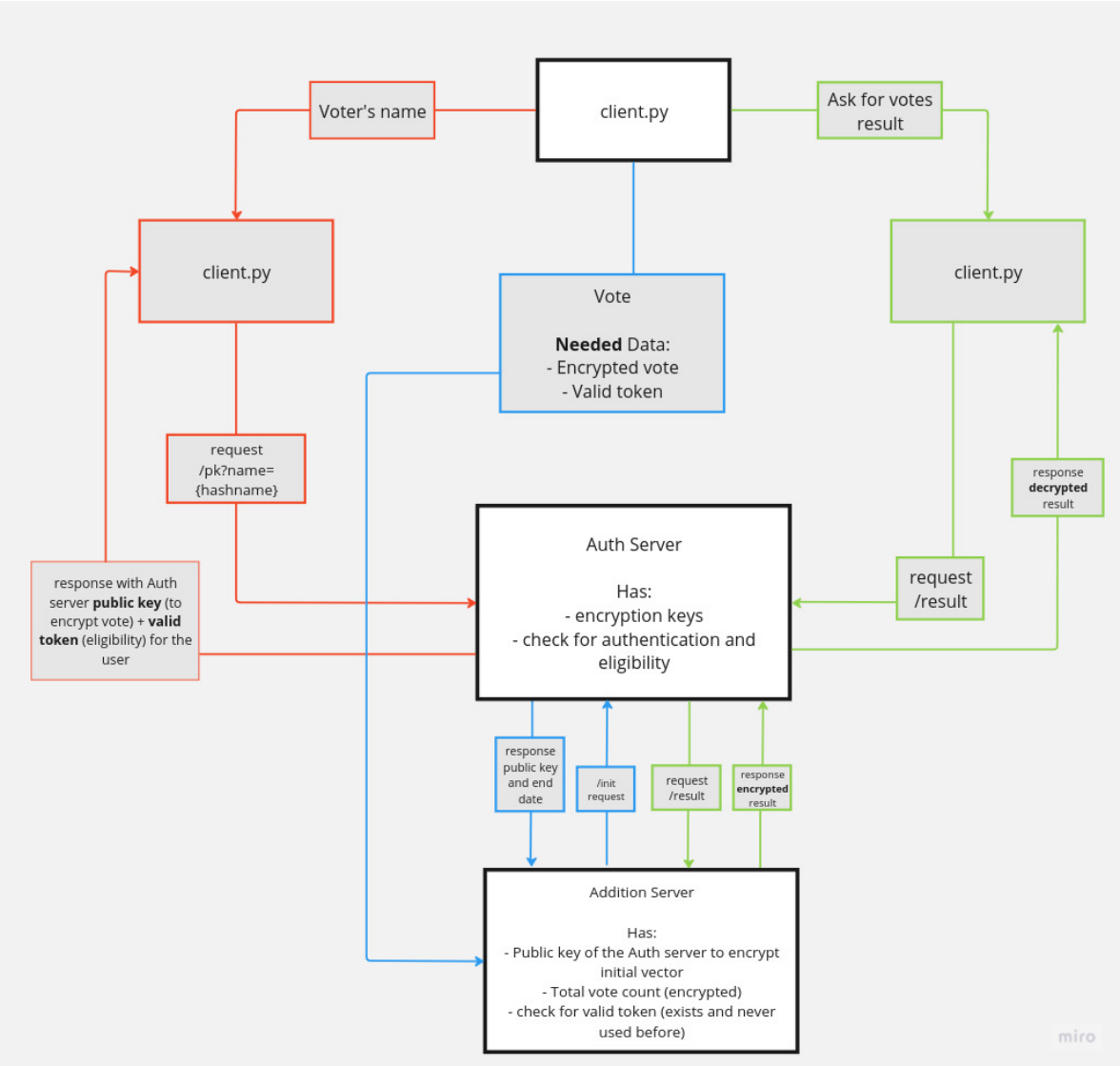

FIGURE 1 – Schéma fonctionnel du protocole de vote

3 Critères de sélection de la bibliothèque de chiffrement homomorphe

Pour notre système de vote, nous avons dû réfléchir à la manière dont nous voulions représenter la donnée d'un vote. Lorsqu'une personne doit voter, elle fait face à un certain nombre de candidats. Une liste de candidats pour être exact. Ainsi, un vote peut se matérialiser par un vecteur de 0 contenant un seul 1 à l'index représentant le candidat voté. Imaginons que nous avons une liste de candidats comme ceci :

[Jean, Michel, Alice, Bob]

Si une personne souhaite voter pour **Alice**, alors son vote sera représenté comme ceci :

[0,0,1,0]

Pour ajouter les votes de plusieurs personnes nous avons donc besoin de faire des additions sur des vecteurs. Prenons les votes suivants :

[0,0,1,0]
[0,1,0,0]
[0,0,1,0]

Le résultat de ce vote serait alors **[0,1,2,0]** et donc **Alice** serait la gagnante. Ainsi, nous avons dû trouver une bibliothèque pouvant effectuer des opérations d'addition sur des vecteurs.

Nous avons choisi de faire ce projet en Python. Après nos recherches, nous avons trouvé une bibliothèque intéressante portant le nom de **Pyfhel**. Cette bibliothèque peut effectuer des **additions**, soustractions, multiplications et des produits scalaires sur du chiffrement homomorphique. En plus d'avoir une documentation bien faite, nous avons trouvé des exemples de calculs sur des données chiffrées échangées à travers un serveur et un client. Ces exemples utilisent des **vecteurs d'entiers** de la bibliothèque **Numpy**. Exactement ce dont nous avions besoin. Après quelques tests pour comprendre le fonctionnement de la bibliothèque, nous avons donc fait le choix de l'utiliser pour notre projet.

4 Résultats des expériences

Dans cette section, nous allons détailler plusieurs expériences et leurs résultats sur le système de vote de ce projet. Nous allons partir sur une base de 4 candidats :

- *"Pierre Olivier Mercier"*
- *"Sebastien Bombal"*
- *"Constance Beguier"*
- *"Jean Lassalle"*

Les expériences qui ont été effectuées sont les suivantes :

1. Une personne qui vote ;
2. Une personne qui vote avec un nom déjà enregistré ;
3. Une personne qui essaye de voter alors que le vote est terminé ;
4. Une personne qui demande les résultats du vote ;
5. Une personne qui demande les résultats du vote sachant que le vote n'est pas terminé.

Pour la première expérience, voici ce que la personne voit lors de son vote :

```
dexiios@Ubuntu20:~/SRS_CRYPI/src$ python3 client.py
[E-Voting Client] Please enter your name :
> Bob
[E-Voting Client] Here are the candidates :
- 0 Pierre Olivier Mercier
- 1 Sebastien Bombal
- 2 Constance Beguier
- 3 Jean Lassalle
[E-Voting Client] Please enter a candidate key :
> 2
[E-Voting Client] Your vote has successfully been completed.
dexiios@Ubuntu20:~/SRS_CRYPI/src$
```

Les deux rectangles rouges correspondent aux entrées que la personne doit renseigner. Le premier correspond au nom de la personne et le second au numéro du candidat pour lequel elle vote. Un message s'affiche ensuite afin de confirmer que le vote a été pris en compte.

Lors de ce vote, un token associé au nom de la personne *(ici "Bob")* a été créé. Ce token est stocké dans une base de données en `json` et est associé à un booléen indiquant si le token a été utilisé pour voter. Voici le token de Bob après qu'il ait voté :

```
src > {} db.json > ...
1   {
2       "1c199b651cb5baaf189c7c3b6e7bd73b": true
3   }
```

Ce token correspond au hash du nom Bob. Le booléen à "true" indique que la personne associée à ce token a voté.

Regardons maintenant ce qu'il se passe si une personne essaye de voter **avec le même nom** *"Bob"* :

```
dexiios@Ubuntu20:~/SRS_CRYPI/src$ python3 client.py
[E-Voting Client] Please enter your name :
> Bob
[E-Voting Client] Here are the candidates :
- 0 Pierre Olivier Mercier
- 1 Sebastien Bombal
- 2 Constance Beguier
- 3 Jean Lassalle
[E-Voting Client] Please enter a candidate key :
> 2
[E-Voting Client] Error already voted
dexiios@Ubuntu20:~/SRS_CRYPI/src$
```

Une erreur s'affiche n'indiquant que ce nom à déjà voté.

Voyons maintenant ce qu'il se passe si une personne essaye de voter **alors que le vote est terminé** :

```
dexiios@Ubuntu20:~/SRS_CRYPI/src$ python3 client.py
[E-Voting Client] Please enter your name :
> Alice
[E-Voting Client] Here are the candidates :
- 0 Pierre Olivier Mercier
- 1 Sebastien Bombal
- 2 Constance Beguier
- 3 Jean Lassalle
[E-Voting Client] Please enter a candidate key :
> 2
[E-Voting Client] Error the vote is finished
dexiios@Ubuntu20:~/SRS_CRYPI/src$
```

Une erreur s'affiche indiquant que le vote est terminé.

Passons maintenant aux **résultats du vote**. Une personne, qu'elle ait votée ou non, peut demander à voir le résultat du vote :

```
dexiios@Ubuntu20:~/SRS_CRYPI/src$ python3 client.py -R
[E-Voting Client] Asking the server for vote results
[E-Voting Client] Winner is : Constance Beguier

[E-Voting Client] Pierre Olivier Mercier has 1 vote(s).
[E-Voting Client] Sebastien Bombal has 0 vote(s).
[E-Voting Client] Constance Beguier has 2 vote(s).
[E-Voting Client] Jean Lassalle has 0 vote(s).
dexiios@Ubuntu20:~/SRS_CRYPI/src$
```

Le vote étant terminé, la personne reçoit les résultats du vote.

Voyons maintenant ce qu'il se passe si une personne demande à voir les résultats d'un vote **alors que le vote n'est pas terminé** :

```
dexiios@Ubuntu20:~/SRS_CRYPI/src$ python3 client.py -R
[E-Voting Client] Asking the server for vote results
[E-Voting Client] Error the vote is not finished
dexiios@Ubuntu20:~/SRS_CRYPI/src$
```

Une erreur apparaît indiquant que le vote n'est pas fini. Ce mécanisme est **très important**, car si une personne vote et que n'importe qui a accès aux résultats au même moment, le vote ne sera plus **anonyme**.

5 Difficultés rencontrées durant le projet

Le challenge du e-voting est de pouvoir garder les votes **anonymes** et **intègres**. Pouvoir modifier les résultats serait catastrophique pour un vote à l'échelle nationale.

Dans notre projet, nous avons eu des difficultés à lier l'anonymisation et intégrité. L'intégrité est facile à résoudre avec ce type de chiffrement, mais l'anonymisation est compliquée à mettre en place, car le serveur additionnant les votes doit recevoir un vote sans pouvoir retracer la personne qui a voté. C'est donc après une longue réflexion que nous avons choisi d'ajouter un serveur intermédiaire vérifiant l'éligibilité et donnant accès à une personne de voter en lui fournissant un token irréversible *(hash du nom)*. Ainsi, le votant enverra sa requête au serveur comptabilisant les votes avec son hash, et donc ce serveur ne pourra pas retrouver le vote. En plus de cela, ce serveur ne peut pas déchiffrer les votes, car il ne dispose que de la **clé publique** pour additionner les votes.

6 Améliorations futures du projet

Ce projet nécessite certainement beaucoup d'améliorations en termes de gestion des requêtes et de sécurité afin d'éviter d'éventuelles interceptions. Une chose qui pourrait aussi être prise en compte est le vote blanc. Notre système actuel ne prend pas en compte le **vote blanc**. Si la personne votant entre un nombre plus grand que le nombre de candidats moins 1, alors nous effectuons un modulo le nombre de candidats afin de tomber sur un des candidats présents. Par exemple, si nous avons 4 candidats et que la personne vote pour 41, notre programme prend `41 modulo 4` soit 1. La personne aura alors voté le candidat associé au numéro 1.