Implémentation d'un Système de Vote Électronique Sécurisé et Vérifiable

Groupe H
Leo Gagey, Mael Éouzan, Neil Belhadj,
Nikolas Podevin, Noé Choplin

15 décembre 2025

Résumé

Ce rapport présente la conception et l'implémentation d'une architecture client-serveur permettant la tenue de votes électroniques anonymes. Le système repose sur des primitives cryptographiques avancées, notamment le chiffrement homomorphe de Paillier et les preuves à divulgation nulle de connaissance (Zero-Knowledge Proofs), pour garantir la confidentialité des bulletins tout en permettant le décompte public des voix par le serveur sans déchiffrement individuel.

Lien vers le dépôt GitLab :
https://code.up8.edu/nbelhadj/vote

Table des matières

1 Introduction

Le vote électronique pose des défis majeurs en termes de sécurité : il faut garantir que chaque votant est légitime, que son vote reste secret (anonymat), mais aussi que le résultat final correspond bien à la somme des votes exprimés (intégrité).

L'objectif de ce projet était de développer une application complète répondant à ces contraintes. Contrairement à une urne classique où l'anonymat est physique, notre système assure l'anonymat mathématique grâce au chiffrement homomorphe. Le serveur, qui centralise les votes, est capable d'additionner les bulletins chiffrés pour obtenir le résultat final chiffré, sans jamais avoir accès au contenu d'un vote individuel en clair.

2 Architecture et Choix Techniques

Nous avons opté pour une architecture hybride afin de tirer parti des forces de différents langages : la performance pour le serveur et la flexibilité pour le client.

2.1 Le Serveur (C++)

Le cœur cryptographique et la logique de gestion des votes sont implémentés en C++.

- **Bibliothèque GMP :** Nous utilisons GMP (GNU Multiple Precision Arithmetic Library) pour gérer les grands entiers nécessaires au chiffrement RSA et Paillier (clés de 2048 bits).
- **Performance :** Le C++ permet de traiter efficacement les opérations mathématiques lourdes (exponentiations modulaires) requises par le protocole.
- **Gestion des connexions :** Le serveur gère plusieurs clients simultanément via des sockets.

2.2 Le Client (Python)

La partie cliente est développée en Python pour sa rapidité de développement et ses bibliothèques riches.

- **Interface Graphique :** Utilisation de `Tkinter` pour offrir une interface utilisateur conviviale (Login, Liste des candidats, Boutons de vote).
- **Communication :** Implémentation de sockets bruts pour communiquer avec le serveur C++.

2.3 Protocole de Communication

Nous avons défini un protocole applicatif personnalisé. Les messages sont structurés avec un en-tête de taille fixe (32 bits) suivi du corps du message. Les échanges incluent des

codes de retour (ex : `00` pour succès, `E3` pour mot de passe invalide) facilitant le débogage et la gestion d'erreurs côté client.

3 Fonctionnalités et Implémentation Cryptographique

3.1 Authentification et Échange de Clés

La sécurité du canal est assurée par un échange de clés RSA.

1. Le serveur envoie sa clé publique.
2. Le client répond avec sa propre clé publique chiffrée.
3. Une phase de *Challenge* (envoi d'un nombre a, réponse attendue $a+1$) permet de valider la possession des clés privées respectives.
4. Une fois le canal sécurisé, l'utilisateur s'authentifie via un couple identifiant/mot de passe haché.

3.2 Le Vote via le Cryptosystème de Paillier

C'est la fonctionnalité centrale du projet. Le système de Paillier est homomorphe additif. Si l'on note $E(m)$ le chiffrement d'un message m, la propriété fondamentale est :

$$E(m_1) \cdot E(m_2) = E(m_1 + m_2) \pmod{n^2}$$

Cela permet au serveur d'agréger les votes (qui sont des 0 ou des 1 pour chaque candidat) en multipliant les chiffrés reçus. À la fin, le serveur déchiffre la somme totale sans jamais avoir vu les votes individuels.

Dans notre implémentation (fichier `paillier.py` et `paillier.hpp`), un vote pour un candidat est un chiffrement de 1, et un non-vote est un chiffrement de 0.

3.3 Preuve à Divulgation Nulle de Connaissance (ZKP)

Pour empêcher un utilisateur malveillant d'envoyer un vote corrompu (par exemple, chiffrer la valeur "100" pour donner 100 voix à un candidat), nous avons implémenté une preuve ZKP (Zero-Knowledge Proof) non-interactive.

Le client doit prouver que le texte chiffré c correspond soit à $m = 0$, soit à $m = 1$, sans révéler lequel. L'algorithme implémenté (visible dans `paillier.py`) fonctionne comme suit :

— Le client génère des engagements a_j et des défis partiels e_j.
— Il utilise l'aléa r utilisé lors du chiffrement pour construire une preuve z.
— Le serveur vérifie l'équation de validité :

$$z_j^n \equiv a_j \cdot (c \cdot g^{-m_j})^{e_j} \pmod{n^2}$$

Si la vérification échoue, le vote est rejeté (Code erreur `E5`).

4 Répartition des Tâches

Le développement a été collaboratif, avec une séparation des responsabilités selon les couches logicielles :

Membre	Contributions principales
Nikolas Podevin	**Développement de l'Interface Graphique (Client).** Conception des fenêtres Tkinter, gestion des événements utilisateurs, affichage dynamique des candidats et intégration visuelle.
Neil Belhadj	**Architecture Serveur et Gestion GitLab.** Mise en place du dépôt, structure du code C++, et intégration des modules cryptographiques.
Mael Éouzan	**Implémentation Cryptographique (C++).** Portage des algorithmes RSA et Paillier avec la bibliothèque GMP, gestion des grands nombres.
Leo Gagey	**Protocoles et Logique Client.** Développement du protocole de communication (sockets), logique d'authentification et gestion des commandes côté client.
Noé Choplin	**ZKP et Sécurité.** Recherche et implémentation des preuves Zero-Knowledge (Python/C++) pour la validation des bulletins.

Note : L'ensemble du groupe a participé à la conception de l'architecture globale et à la rédaction de la documentation technique.

5 Conclusion et Perspectives

5.1 Bilan

Nous avons réussi à implémenter un système de vote fonctionnel respectant les principes de confidentialité. L'utilisation conjointe de C++ et Python a permis d'obtenir un backend performant et un frontend accessible. L'intégration du ZKP est une réussite technique majeure qui renforce la robustesse du système face aux fraudes.

5.2 Limitations et Améliorations Futures

Certaines parties du projet pourraient être améliorées :

— **Portage RSA en C++ :** Comme indiqué dans notre suivi technique, le portage complet des fonctions RSA (notamment le padding complexe) est encore en cours de finalisation à 100%.

— **Gestion des pannes :** Un système de reconnexion (via cookie de session) a été conçu théoriquement mais reste à finaliser pour gérer les coupures réseau intempestives.
— **Interface Admin :** L'administration du vote se fait actuellement via des commandes consoles ; une interface graphique dédiée aux organisateurs serait un plus.

Ce projet nous a permis de mieux comprendre les enjeux concrets de la cryptographie appliquée et la complexité de sécuriser une architecture client-serveur critique.