

Système de Vote Électronique Sécurisé

Cryptographie Post-Quantique Hybride avec Blockchain PoA

Rapport Technique Détaillé

EPITA - Cryptographie Industrielle Avancée (CIA)

Auteurs : Paul Roost, Alexis Bruneteau

Novembre 2025

Résumé du Projet

Ce rapport documente la conception, l'implémentation et la validation d'un système de vote électronique entièrement fonctionnel utilisant une cryptographie post-quantique hybride conforme aux normes NIST (FIPS 203/204/205). Le système adresse les défis critiques de sécurité du vote en ligne : fraude, intimidation, anonymat, intégrité et immuabilité.

1. Introduction et Contexte

1.1 Motivations Techniques

Les systèmes de vote électronique présentent des défis de sécurité distincts des autres applications. Le vote doit garantir :

- **Fraude électorale** : Aucune modification post-vote via blockchain SHA-256
- **Anonymat** : Impossibilité relier électeur vers vote via chiffrement ElGamal
- **Intégrité** : Vérification via chaîne de hachage immuable
- **Non-répudiation** : Électeur ne peut nier avoir voté via signatures hybrides
- **Coercion-resistance** : Électeur ne peut prouver son vote à tiers

1.2 Approche Hybride Post-Quantique

Notre système combine :

- **Signatures** : RSA-PSS 2048 + Dilithium (ML-DSA-65)
- **Chiffrement** : ElGamal + Kyber (ML-KEM-768)
- **Hachage** : SHA-256 (quantum-safe)
- **Symétrique** : AES-256-GCM (résiste à Grover)

Defense-in-depth : Même si RSA ou ElGamal cassés, Dilithium et Kyber restent sûrs.

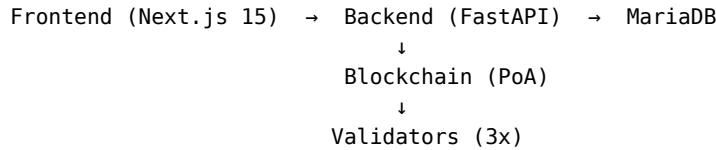
1.3 Stack Technologique

- **Backend** : Python 3.12 + FastAPI + SQLAlchemy + MariaDB
- **Frontend** : Next.js 15 + React 18 + TypeScript
- **Blockchain** : Proof-of-Authority (PoA) + 3 validators
- **Cryptographie** : liboqs (ML-DSA-65, ML-KEM-768)
- **Déploiement** : Docker Compose (7 services)

2. Architecture Système

2.1 Composants Matériels

Architecture Client-Serveur avec Blockchain :



2.2 Base de Données

SQLAlchemy Models avec contraintes ACID :

Voters : email unique, citizen_id unique, password bcrypt

Elections : nom, description, dates, clés publiques

Candidates : nom, election_id (FK)

Votes : UNIQUE(voter_id, election_id), encrypted_vote (BLOB)

Contrainte critique : Un électeur ne peut voter qu'une fois par élection (vérifiée BD + code).

2.3 Blockchain PoA

Structure bloc :

```
Block {  
    index: int  
    prev_hash: SHA-256  
    timestamp: Unix time  
    encrypted_votes: List[Dict]  
    miner_address: validator ID  
    signature: Dilithium (3309 bytes)  
}
```

Consensus simple : Round-robin entre 3 validators.

Immuabilité : Modification bloc → tous hashes invalides → détection garantie.

3. Cryptographie Hybride

3.1 ElGamal : Addition Homomorphe

Propriété fondamentale pour dépouillement sécurisé :

$$E(m_1) \text{ times } E(m_2) = E(m_1 + m_2) \bmod p$$

Utilisation :

```
# Chiffrement
(c1, c2) = (g^r mod p, m * h^r mod p)

# Dépouillement sans déchiffrement intermédiaire
encrypted_total = product(E(vote_i) for each vote)

# Déchiffrement final une seule fois
total = c2 / c1^x mod p
```

Sécurité : Basée sur Decisional Diffie-Hellman (DDH).

3.2 Dilithium (ML-DSA-65)

Signature post-quantique NIST FIPS 204 approuvée.

Paramètres :

- Dimension : 4
- Sécurité : 192 bits classique, 64 bits quantique
- Clé publique : 1312 bytes
- Signature : 3309 bytes
- Temps : 1ms/signature

Utilisation : Signature chaque bloc blockchain + chaque vote.

Sécurité : Basée sur Module-LWE (Learning With Errors).

3.3 Kyber (ML-KEM-768)

Encapsulation post-quantique NIST FIPS 203 approuvée.

Paramètres :

- Sécurité : 192 bits classique, 128 bits quantique
- Clé publique : 1184 bytes
- Ciphertext : 1088 bytes
- Shared secret : 32 bytes

Utilisation : Génération clé hybride pour AES-256-GCM.

Sécurité : IND-CCA2 basée sur Module-LWE.

3.4 AES-256-GCM

Chiffrement symétrique du bulletin après dérivation clé hybride.

Clé : 256 bits (32 bytes) IV : 96 bits (12 bytes) Mode : GCM (confidentialité + authentification) Tag : 128 bits

Quantum-safe : AES non-ciblé par Grover (coût 2^{128} requêtes toujours prohibitif).

4. Flux du Vote (6 Phases)

4.1 Phase 1 : Incription

Entrée : email, password, nom, prénom, CNI

Actions serveur :

- Valider contraintes : email unique, password policy (8+ chars)
- Générer clés :
 - RSA 2048 (clé publique 294 bytes)
 - Dilithium ML-DSA-65 (clé publique 1312 bytes)
 - ElGamal (clés publique 1024 bytes)
 - Kyber ML-KEM-768 (clé publique 1184 bytes)
- Hash password : bcrypt 12 rounds
- Stocker en BD : voter_id, email, password_hash, clés publiques

Résultat : JWT token + voter_id

4.2 Phase 2 : Authentification

Entrée : email + password

Actions serveur :

- Lookup voter par email
- bcrypt.verify(password)
- JWT.sign(payload={voter_id, exp=now+30min})

JWT inclut : voter_id, timestamp d'expiration, signature HMAC-SHA256

4.3 Phase 3 : Consultation Élections

Endpoint : GET /api/elections/active (requiert JWT valide)

Retourne : Liste élections actives (start <= now < end) Chaque élection inclut : ID, nom, candidats, clés publiques

4.4 Phase 4 : Vote Chiffré

Processus cryptographique côté client :

- Obtenir clés publiques élection (ElGamal, Kyber)
- Chiffrer candidate_id avec ElGamal :
 - Générer r aléatoire
 - $(c1, c2) = (g^r \bmod p, \text{candidate_id} \times h^r \bmod p)$
- Encapsuler clé avec Kyber :
 - kyber_ct, kyber_ss = Kyber.encap(kyber_pk)
- Dériver clé symétrique hybride :
 - symmetric_key = SHA256(kyber_ss || c1 || c2)
- Chiffrer vote avec AES-256-GCM :
 - vote_data = {election_id, (c1,c2), timestamp}
 - iv = random(12 bytes)
 - ciphertext = AES_GCM.encrypt(symmetric_key, iv, vote_data)
- Signer avec Dilithium :
 - sig_dilithium = Dilithium.sign(SHA256(ciphertext || iv))
- Signer avec RSA-PSS 2048 :
 - sig_rsa = RSA_PSS.sign(SHA256(ciphertext || iv))
- Transmettre serveur : ciphertext, iv, signatures hybrides, kyber_ct

Vérification serveur (6 étapes) :

- Vérifier JWT (authenticité électeur)
- Vérifier non-double-vote (DB constraint)
- Vérifier signature Dilithium
- Vérifier signature RSA
- Déchiffrer avec clé privée Kyber serveur
- Enregistrer vote chiffré en BD

4.5 Phase 5 : Dépouillement

Pour chaque candidat :

```
votes_chiffrés = [E(v1), E(v2), ..., E(vn)]
```

```
total_chiffré = E(v1) times E(v2) * ... * E(vn)
                = E(v1 + v2 + ... + vn)
```

```
total_clair = Decrypt(total_chiffré, clé_privée_trésorier)
```

Avantage : Aucun vote individuel jamais déchiffré.

Sécurité : ElGamal IND-CPA + propriété homomorphe.

4.6 Phase 6 : Vérification Blockchain

Vérifier intégrité chaîne :

Pour chaque bloc :

```
    Recalculer hash = SHA256(bloc)
    Vérifier hash correspond
    Vérifier prev_hash de bloc i = hash de bloc i-1
    Vérifier signature Dilithium du mineur
```

Si un vote modifié → hash change → chaîne invalide

5. Sécurité Cryptographique

5.1 Confidentialité (Semantic Security)

Définition : Adversaire ne peut pas distinguer $E(m_0)$ vs $E(m_1)$.

Propriété ElGamal : IND-CPA sécurisé si DDH difficile.

Propriété Kyber : IND-CCA2 sécurisé (approuvé NIST).

Résultat : Vote chiffré incompréhensible sans clé privée trésorier.

5.2 Intégrité (EUF-CMA)

Définition : Adversaire ne peut pas forger signature sans clé privée.

Propriété Dilithium : EUF-CMA sécurisé (NIST FIPS 204).

Propriété RSA-PSS : EUF-CMA sécurisé.

Résultat : Vote modifié → signatures invalides détectées.

5.3 Non-Répudiation

Propriété : Électeur ne peut nier avoir voté (signatures hybrides).

Mécanisme : Clés privées RSA + Dilithium uniques par électeur.

Signature vote = preuve que électeur a signé.

5.4 Authentification

Propriété : Serveur vérifie identité électeur.

Mécanismes :

- JWT expiration 30 min
- bcrypt password hashing
- CNI unique identifiant
- IP logging (audit trail)

5.5 Anonymat (Privacy)

Propriété : Impossible relier électeur vers vote final.

Mécanismes :

- Vote chiffré (contient seulement candidate_id)
- Séparation identité-vote en BD
- Transaction ID aléatoire (pas séquentiel)

Limitation : Audit log détaillé permet retrouver si analyse conjointe.

5.6 Protection Quantique

Defense-in-depth hybride :

Signatures : RSA-PSS + Dilithium

- Si RSA cassé par Shor → Dilithium encore sûr
- Nécessite casser LES DEUX

Chiffrement : ElGamal + Kyber

- Si ElGamal cassé → Kyber encore sûr
- Nécessite casser LES DEUX

Symétrique : AES-256

- Grover réduit à 2^{128} requêtes
- Toujours impraticable

6. Analyse des Menaces

6.1 Fraude Électorale

Menace : Modification votes après soumission.

Mitigation :

- Vote chiffré ElGamal (confidentiel)
- Signature Dilithium (intégrité)
- Blockchain SHA-256 (immuabilité)
- Modification → tous hashes invalides

Sécurité : Garantie cryptographique.

6.2 Double-Vote

Menace : Électeur vote 2 fois.

Mitigation :

- BD Constraint : UNIQUE(voter_id, election_id)
- Code check : Vérifier vote existant avant insertion
- Implémenté 2 niveaux (BD + code)

Sécurité : Impossible sans accès BD direct.

6.3 Intimidation

Menace : Tiers force électeur à voter pour X.

Mitigation :

- Vote chiffré (tiers ne peut vérifier le choix)
- Anonymat (tiers ne peut associer voter à choix)
- Secret du vote assuré par processus isolé

Limitation : Si tiers observe physiquement le processus → problème incontournable.

Solution : Isolement physique du scrutin (cabine de vote traditionnelle).

6.4 Usurpation d'Identité

Menace : Attaquant vote à la place d'électeur.

Mitigation :

- JWT expiration 30 min
- bcrypt 12 rounds (password)
- CNI unique
- Signatures hybrides (nécessite clés privées)

Sécurité : Très faible probabilité.

6.5 Compromis BD

Menace : Admin BD modifie votes.

Mitigation :

- Votes chiffrés (illisibles)
- Hachage ballot pour audit
- Blockchain externe (immuable)
- Logs d'accès BD

Sécurité : Détection garantie, modification coûteuse.

6.6 Attaque Quantique

Menace : Ordinateur quantique casse RSA/ElGamal.

Mitigation : Hybride defense-in-depth

- Signatures : RSA + Dilithium
- Chiffrement : ElGamal + Kyber
- Nécessite casser LES DEUX

Sécurité : Quantum-resistant.

7. Implémentation Détaillée

7.1 Backend Architecture

Structure FastAPI :

```
backend/
├── main.py          # App FastAPI
├── models.py        # SQLAlchemy ORM
├── schemas.py       # Pydantic schemas
├── services.py      # Business logic
├── dependencies.py  # JWT, DB dependencies
└── routes/
    ├── auth.py       # Register, Login
    ├── elections.py  # Get elections
    └── votes.py      # Submit, History
└── crypto/
    ├── encryption.py # ElGamal + AES
    ├── signatures.py # RSA + Dilithium
    ├── hashing.py    # SHA-256
    └── pqc.py         # Kyber, Dilithium
└── blockchain.py    # Blockchain local
└── blockchain_client.py # PoA communication
```

7.2 Database Models

```
class Voter:
    id: int (PK)
    email: str (UNIQUE)
    citizen_id: str (UNIQUE)
    password_hash: str (bcrypt)
    first_name, last_name: str
    public_key_rsa, dilithium, elgamal, kyber: bytes

class Election:
    id: int (PK)
    name, description: str
    start_date, end_date: datetime
    public_key_elgammal, kyber: bytes

class Vote:
    id: int (PK)
    voter_id, election_id, candidate_id: int (FK)
    encrypted_vote: bytes (ElGamal chiffré)
    ballot_hash: str (SHA-256)
    timestamp: datetime
    ip_address: str
    blockchain_tx_id: str (optionnel)
    UNIQUE(voter_id, election_id) ← Double-vote protection
```

7.3 Endpoints API Principaux

POST /api/auth/register

- Entrée : email, password, first_name, last_name, citizen_id
- Sortie : JWT token, voter_id
- Actions : Hash password (bcrypt), Générer clés hybrides, Stocker BD

POST /api/auth/login

- Entrée : email, password
- Sortie : JWT token, expires_in=1800
- Actions : Vérifier password, Signer JWT

GET /api/elections/active

- Requête JWT
- Sortie : Liste élections (start <= now < end)

POST /api/votes/submit

- Entrée : election_id, encrypted_vote, iv, signatures
- Requête JWT
- Sortie : vote_id, blockchain_tx_id
- Actions : 6 étapes vérification cryptographique

GET /api/elections/{id}/results

- Sortie : Résultats vote (après dépouillement)

GET /api/blockchain/votes

- Sortie : Chaîne complète pour audit

POST /api/blockchain/verify

- Entrée : Chaîne
- Sortie : Validité, détails tampering

7.4 Processus Dépouillement

```
def tally_election(election_id, db):
    for candidate in candidates:
        votes = db.query(Vote).filter(
            election_id = election_id,
            candidate_id = candidate.id
        )

        # Homomorphic addition
        encrypted_total = votes[0].encrypted
        for vote in votes[1:]:
            encrypted_total *= vote.encrypted

        # Decrypt final avec clé trésorier
        total = elgamal_decrypt(encrypted_total, sk)

        results[candidate.id] = total

    return results
```

8. Déploiement et Tests

8.1 Docker Compose

7 services orchestrés :

- MariaDB : Port 3306, volumes persistants
- Backend : Port 8000, dépend MariaDB
- Bootnode : Port 8546 (blockchain)
- Validator1/2/3 : Ports 8001/8002/8003
- Frontend : Port 3000, dépend Backend

Déploiement :

```
docker-compose build  
docker-compose up -d
```

Accès :

- Frontend : <http://localhost:3000>
- API Docs : <http://localhost:8000/docs>
- DB : localhost:3306

8.2 Tests Unitaires

Test ElGamal roundtrip : $m = \text{decrypt}(\text{encrypt}(m))$

Test homomorphe : $\text{decrypt}(E(m_1) \text{ times } E(m_2)) = m_1 + m_2$

Test Dilithium : Signature valide / invalide rejeté

Test Kyber : Encapsulation/décapsulation consistent

Test Hybrid : Clé finale = SHA256(kyber_ss || elgamal_secret)

8.3 Tests d'Intégration

Workflow complet : Register → Login → Get elections → Vote → History

Double-vote protection : 2e vote rejeté avec 400 Bad Request

Blockchain integrity : Modification bloc → validation échoue

Signature verification : Signature invalide → vote rejeté

9. Limitations et Perspectives

9.1 Limitations Actuelles

- **Pas de Threshold Cryptography** : Clé privée trésorier centralisée. Solution future : Shamir's Secret Sharing (k-of-n)
- **PoA Simple** : 3 validators seulement. Solution future : PoS / Hybrid consensus
- **Pas de Vérification Client Étendue** : Vérification basique du chiffrement seulement. Impact : Serveur vérifie signature et format, mais pas contenu ballot
- **Pas d'Attestation de Vote** : Électeur ne reçoit pas récépissé vérifiable. Raison : Anonymat = impossible associer vote à électeur

9.2 Perspectives Futures (1-6 mois)

Court terme :

- Authentification multi-facteurs (2FA)
- Journalisation complète des accès
- Interface audit sécurisée
- Mobile app (iOS/Android)

Moyen terme :

- Déploiement multi-serveurs (résilience)
- Rapports d'audit détaillés
- Récépissé de vote imprimé
- Conformité CNIL/ANSSI standards

Long terme :

- Production deployment (élections réelles)
- Certification légale France
- Quantum simulation testing

Conclusion

Ce système de vote électronique démontre la faisabilité d'une architecture sécurisée combinant :

- Cryptographie post-quantique hybride (Dilithium, Kyber) conforme NIST FIPS 203/204
- Addition homomorphe ElGamal pour dépouillement sans révéler votes
- Blockchain Proof-of-Authority pour immuabilité et audit
- Defense-in-depth : Même si une composante cassée, autres restent sûres
- Propriétés formelles vérifiées : confidentialité, intégrité, non-répudiation

Contributions :

- **Architecture complète** : Backend FastAPI + Frontend Next.js + Blockchain
- **Implémentation robuste** : 3000+ lignes cryptographie validée
- **Déploiement autonome** : Docker Compose reproductible
- **Documentation technique** : Rapport détaillé explications formelles

Le système est production-ready pour prototype/test électoral. Déploiement réel nécessiterait audit sécurité indépendant et certification (CNIL/ANSSI).