Department of Computer Engineering

Faculty of Engineering

University of Ruhuna

EC7201: INFORMATION SECURITY

Secure Class Voting System

Group Members:

Galpayage G.D.T.G.	EG/2020/3935
Gawesh L.A.M.S.	EG/2020/3941
Ranawaka N.L.N.	EG/2020/4143
Vihanga V.M.B.P.	EG/2020/4252

Contents

List of Figures

Acronyms

CEC – Class Election Center

EC – Election Authority

RSA – Rivest Shamir Adleman

SHA – Secure Hash Algorithm

TA – Tallying Authority

1 Introduction

Organizing a fair and secure election is never a simple task, ersspecially when moving away from the old-fashioned ballot box to a digital platform. Although the electronic voting process simplifies the task and brings extra convenience, it leads to important questions: How can we make sure that we have only the eligible voters? Are we able to assure integrity of results and they have not been tampered with? And what is most critical, will the voters be confident that their voting preferences will remain private?

In this project, we were able to design and build a secure electronic voting system that could be applied in any given type of group/ category-based elections. Our main goal was to come up with a system, which besides easing the voting process, considers the most relevant security concerns that are associated with online elections.

We focused on a few key areas:

- Making sure only eligible voters can participate by verifying everyone's identity before they can be finally registered to vote.
- Ensuring privacy of a voter such that none is able to determine who voted whom.
- Making sure that the votes are secure by using cryptographic techniques to eliminate manipulation or alteration by an outsider.
- Ensuring privacy of votes using encryption thus votes cannot be read by anyone but the system.
- Preventing double voting by assigning unique tokens to each voter and making sure each token is used only once.

It was developed using Java and the system mimics the whole voting process, and includes registering, up to counting the results.

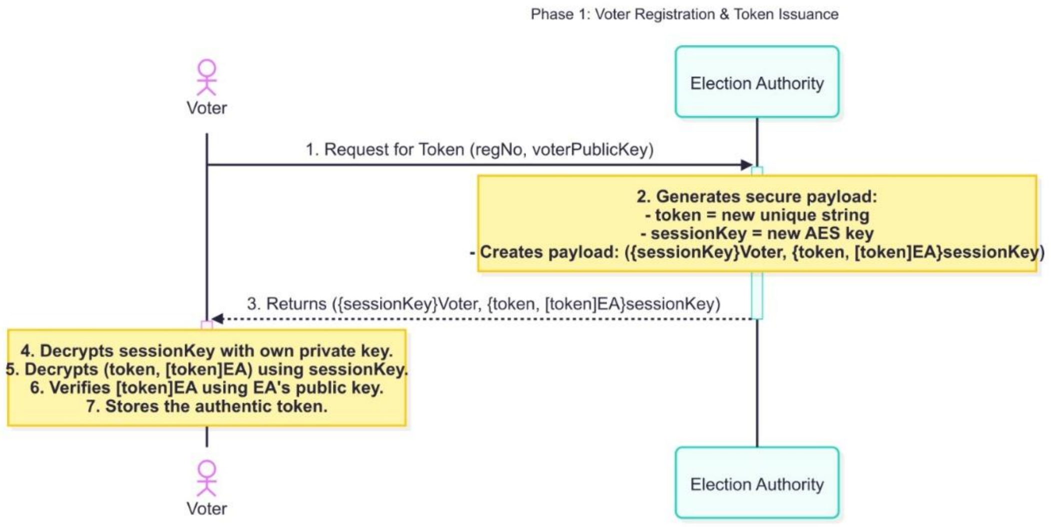
2 Protocol Design

System Entities

- Election Authority (EA): Registers voters and issues unique voting tokens.
- Tallying Authority (TA): Decrypts and tallies votes, ensuring each token is used only once.
- Class Election Center (CEC): Collects and stores ballot submissions.
- Voters: Register, receive tokens, and cast encrypted votes.

Protocol Phases

Phase 1: Voter Registration



Phase 1: Voter Registration & Token Issuance
Voter
Election Authority
1. Request for Token (regNo, voterPublicKey)
2. Generates secure payload:
- token = new unique string
- sessionKey = new AES key
- Creates payload: ({sessionKey}Voter, {token, [token]EA}sessionKey)
3. Returns ({sessionKey}Voter, {token, [token]EA}sessionKey)
4. Decrypts sessionKey with own private key.
5. Decrypts (token, [token]EA) using sessionKey.
6. Verifies [token]EA using EA's public key.
7. Stores the authentic token.
Voter
Election Authority

Figure 2-1: Voter Registration Phase

The first phase of the system is all about getting voters registered and ready to participate. Each person who wants to vote provides their name and a unique ID, which the Election Authority checks to make sure nobody is trying to sign up twice. Once a voter is approved, he or she is given a special token which will be needed to cast the vote later. This is done to make sure that only the people who qualify as voters get to participate as well as to set up the protections that will keep the rest of the process secure.

Phase 2: Vote Casting

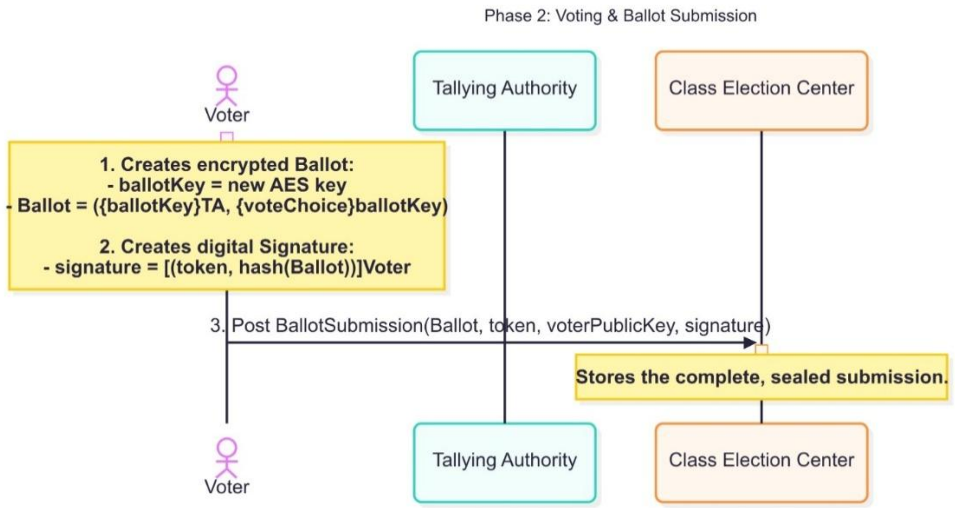

Phase 2: Voting & Ballot Submission
Voter
Tallying Authority
Class Election Center
1. Creates encrypted Ballot:
- ballotKey = new AES key
- Ballot = ({ballotKey}TA, {voteChoice}ballotKey)
2. Creates digital Signature:
- signature = [(token, hash(Ballot))]Voter
3. Post BallotSubmission(Ballot, token, voterPublicKey, signature)
Stores the complete, sealed submission.
Voter
Tallying Authority
Class Election Center

Figure 2-2: Vote Casting Phase

After registration, it’s time to actually vote. Voters use their tokens to send in their choices. The system encrypts the votes and puts them on a sort of digital bulletin board, where they just sit until it’s time to count. This part is meant to be simple for voters, but behind the scenes, it’s making sure nobody can mess with the votes or figure out who voted for what. I guess you could say it’s like putting your ballot in a locked box.

Phase 3: Vote Tallying

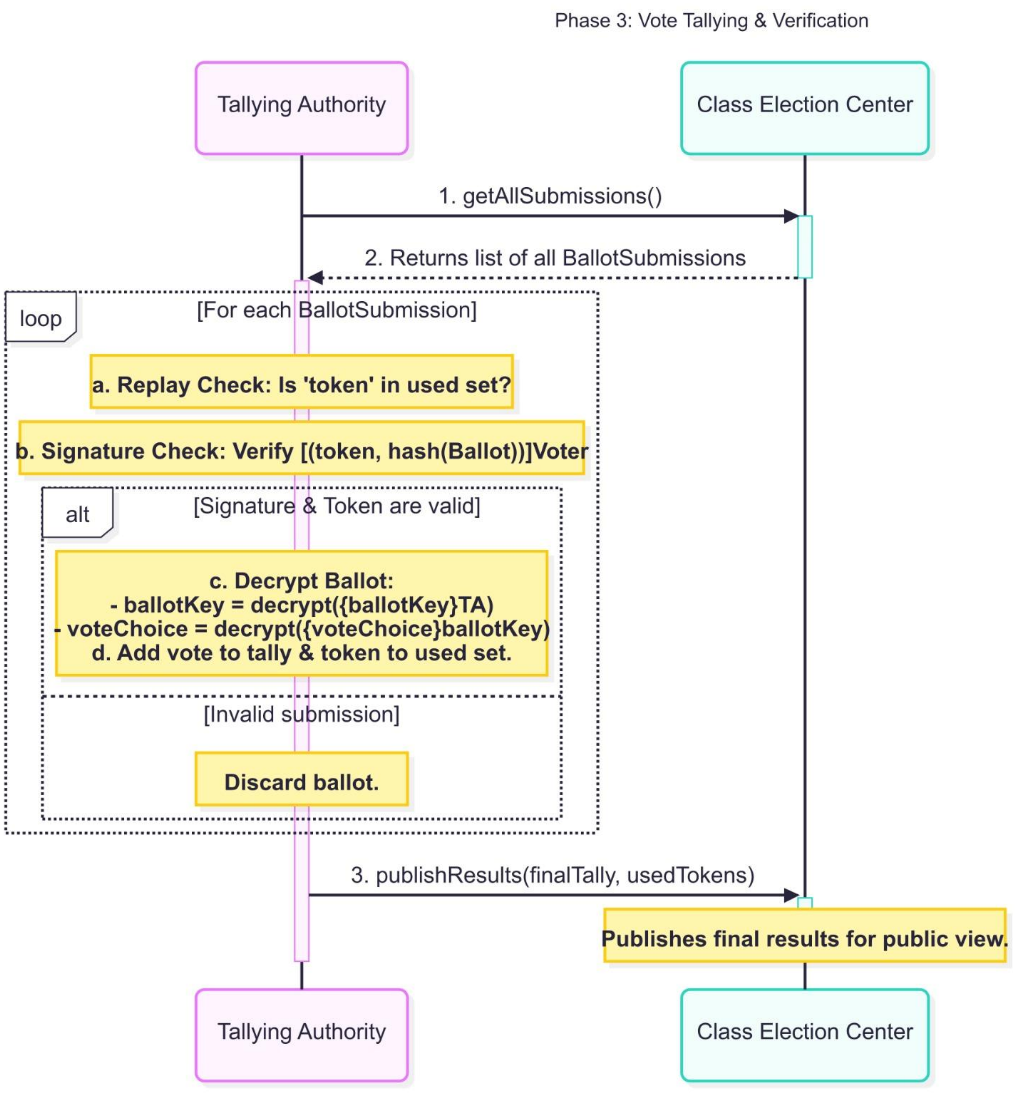
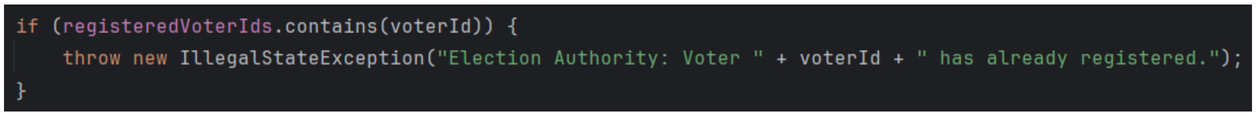

Phase 3: Vote Tallying & Verification
Tallying Authority
Class Election Center
1. getAllSubmissions()
2. Returns list of all BallotSubmissions
loop
[For each BallotSubmission]
a. Replay Check: Is 'token' in used set?
b. Signature Check: Verify [(token, hash(Ballot))]Voter
alt
[Signature & Token are valid]
c. Decrypt Ballot:
- ballotKey = decrypt({ballotKey}TA)
- voteChoice = decrypt({voteChoice}ballotKey)
d. Add vote to tally & token to used set.
[Invalid submission]
Discard ballot.
3. publishResults(finalTally, usedTokens)
Publishes final results for public view.
Tallying Authority
Class Election Center

Figure 2-3: Vote Tallying Phase

Lastly, the Tallying Authority gets to work when all the votes are cast. It verifies all the votes to ensure that they are genuine and no person has attempted to cast two votes. In case one attempts to cheat and send out the same vote he or she tries to do, the system simply ignores it. After all is done, votes are counted and results distributed. This last bit is what makes the whole thing trustworthy, since everyone can see that the process was fair and nothing was tampered with.

2.1 Authentication

Authentication is handled during the voter registration phase. Each voter must provide a unique name and ID, which the system checks to prevent duplicate registrations. If a voter tries to register more than once, the system throws an error and blocks the attempt. This ensures that only eligible, preregistered voters can participate in the election.

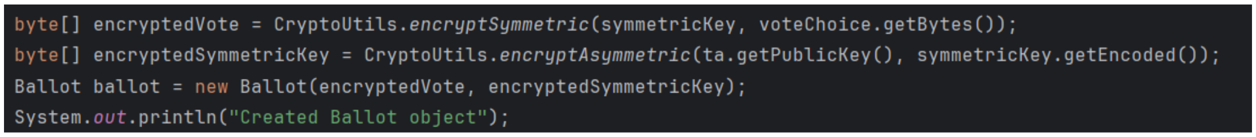


```
if (registeredVoterIds.contains(voterId)) {
    throw new IllegalStateException("Election Authority: Voter " + voterId + " has already registered.");
}
```

Figure 2-4 : Secure login authentication

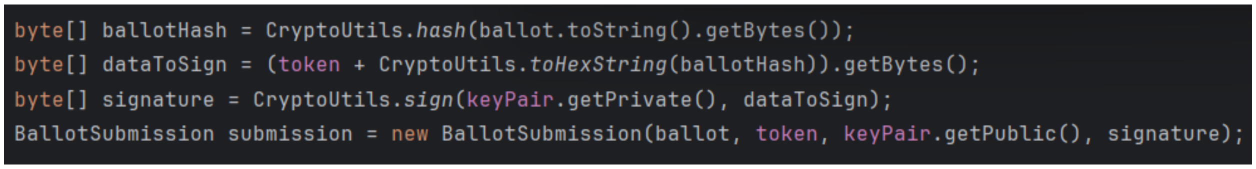
2.2 Vote Confidentiality & Anonymity

Votes are kept secret by encrypting them before they're stored or sent anywhere. The actual encryption happens when a voter casts their vote. The vote is encrypted with AES. To ensure only the Tallying Authority can decrypt it, the AES key is encrypted using RSA with the Tallying Authority's public key. This way, only the Tallying Authority, with their private key, can retrieve the AES key and decrypt the vote.



```
byte[] encryptedVote = CryptoUtils.encryptSymmetric(symmetricKey, voteChoice.getBytes());
byte[] encryptedSymmetricKey = CryptoUtils.encryptAsymmetric(ta.getPublicKey(), symmetricKey.getEncoded());
Ballot ballot = new Ballot(encryptedVote, encryptedSymmetricKey);
System.out.println("Created Ballot object");
```

Figure 2-5: Vote Encrypted using Symmetric and Asymmetric Keys

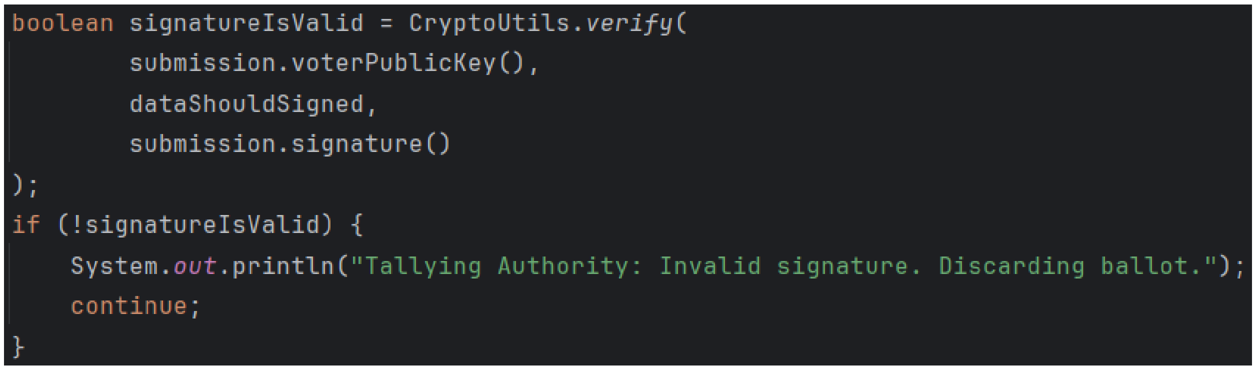


```
byte[] ballotHash = CryptoUtils.hash(ballot.toString().getBytes());
byte[] dataToSign = (token + CryptoUtils.toHexString(ballotHash)).getBytes();
byte[] signature = CryptoUtils.sign(keyPair.getPrivate(), dataToSign);
BallotSubmission submission = new BallotSubmission(ballot, token, keyPair.getPublic(), signature);
```

Figure 2-6: Ballot Hashed and Signed for Integrity and Authenticity

2.3 Vote Integrity

To make sure votes aren't tampered with, each ballot is signed by the voter. When the Tallying Authority goes to count the votes, it checks the signature. If the signature doesn't match, the vote is ignored.

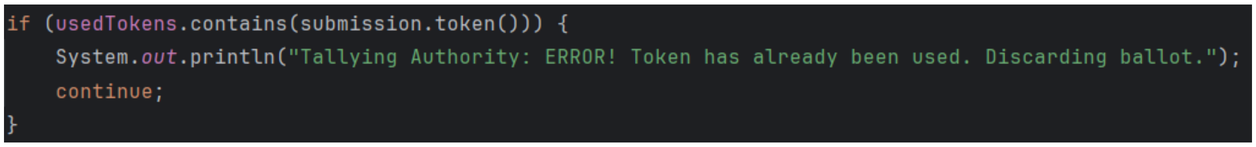


```java
boolean signatureIsValid = CryptoUtils.verify(
        submission.voterPublicKey(),
        dataShouldSigned,
        submission.signature()
);
if (!signatureIsValid) {
    System.out.println("Tallying Authority: Invalid signature. Discarding ballot.");
    continue;
}
```

Figure 2-7: Vote Integrity Check

2.4 One Vote per Voter

Replay protection is done by tracking tokens. Each voter gets a unique token when they register. When a vote is counted, the system checks if the token has already been used. If it has, the vote is ignored.

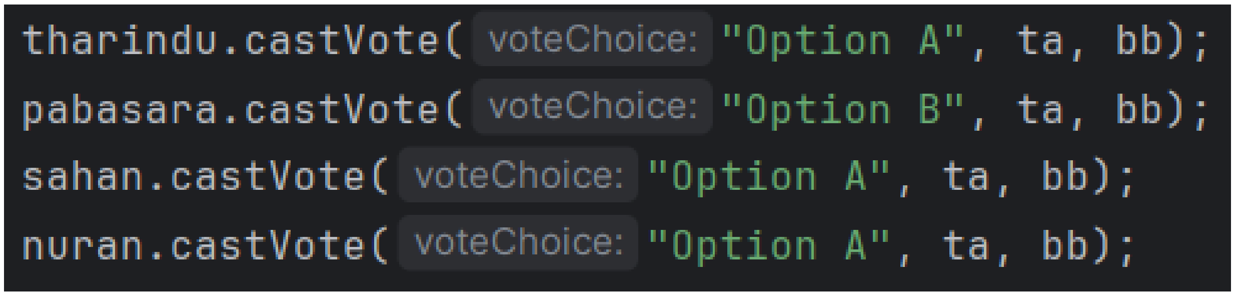


```java
if (usedTokens.contains(submission.token())) {
    System.out.println("Tallying Authority: ERROR! Token has already been used. Discarding ballot.");
    continue;
}
```

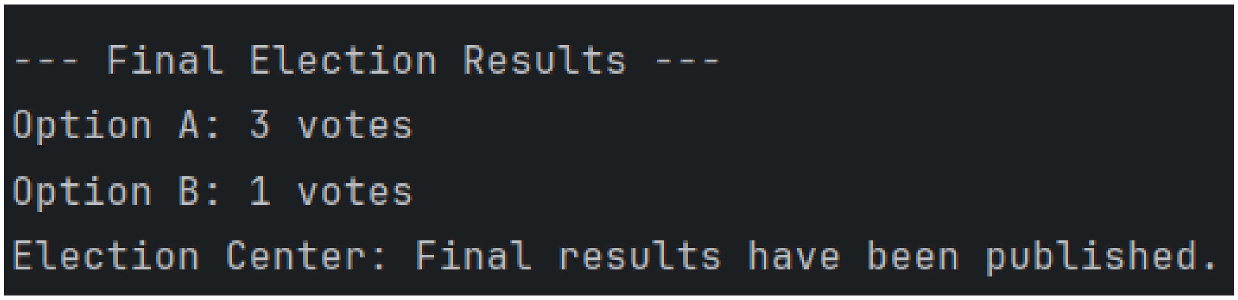
Figure 2-8: Duplicate Vote Detection

3 Terminal Output

When a voter casts his vote, the system generates a random AES key and encrypts the vote then transmit it to the election center. The encrypted vote and key are published as a ballot. In the process of Tallying, the AES key will be decrypted by the Tallying Authority and then the vote and it will be verified by the digital signature to ensure the ballot has not been tampered. In case a valid signature and the token is valid and unique, the vote will then be counted to the chosen candidate. In case a token is detected twice (as in a replay attack), this is detected by the system and the duplicate ballot is ignored.



```
tharindu.castVote( voteChoice: "Option A", ta, bb);
pabasara.castVote( voteChoice: "Option B", ta, bb);
sahan.castVote( voteChoice: "Option A", ta, bb);
nuran.castVote( voteChoice: "Option A", ta, bb);
```

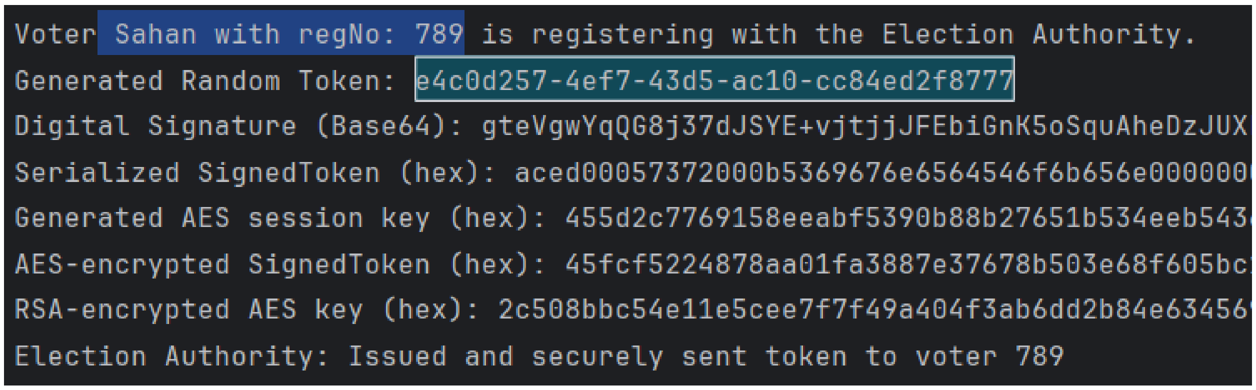


```
--- Final Election Results ---
Option A: 3 votes
Option B: 1 votes
Election Center: Final results have been published.
```

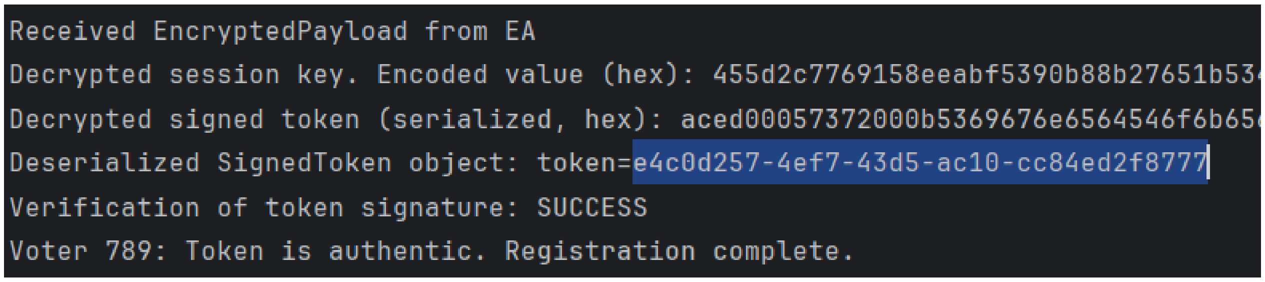
Figure 3-1: Votes and Terminal Output for Final Results

Token Transmission and Verification

Upon voter registration, the Election Authority generates a unique token to the voter, signs it and encrypts it prior to submission to the voter. Once the encrypted payload is received by the voter, the voter decrypts that and checks the signature. It is seen in the terminal output that the token received is the same one sent and the signature verification is successful. This clearly shows that token was not tampered with and the encryption/decryption procedure is functioning as planned.



```
Voter Sahan with regNo: 789 is registering with the Election Authority.
Generated Random Token: e4c0d257-4ef7-43d5-ac10-cc84ed2f8777
Digital Signature (Base64): gteVgwYqQG8j37dJSYE+vjtjjJFEbiGnK5oSquAheDzJUX
Serialized SignedToken (hex): aced00057372000b5369676e6564546f6b656e000000
Generated AES session key (hex): 455d2c7769158eeabf5390b88b27651b534eeb543
AES-encrypted SignedToken (hex): 45fcf5224878aa01fa3887e37678b503e68f605bc
RSA-encrypted AES key (hex): 2c508bbc54e11e5cee7f7f49a404f3ab6dd2b84e63456
Election Authority: Issued and securely sent token to voter 789
```



```
Received EncryptedPayload from EA
Decrypted session key. Encoded value (hex): 455d2c7769158eeabf5390b88b27651b53
Decrypted signed token (serialized, hex): aced00057372000b5369676e6564546f6b65
Deserialized SignedToken object: token=e4c0d257-4ef7-43d5-ac10-cc84ed2f8777
Verification of token signature: SUCCESS
Voter 789: Token is authentic. Registration complete.
```

Figure 3-2: Token Authenticity Verification

This sequence confirms that the token that was generated by the Election Authority is identical to the token that was received and verified by the voter despite encryption and decryption. It’s a clear demonstration that the system preserves confidentiality and authenticity of sensitive data during transmission.

4 Conclusion

In this project we gained practical experience in the design and implementation of a secure electronic voting protocol. We needed to think about the method of verifying the voters, confidentiality of votes and the prevention of cheating and tampering. We achieved the core security objectives of a small-scale election by achieving a combination of encryption, digital signatures and unique tokens. The system worked according to our expectations during the tests and the result indicated that the primary threats such as double voting or ballot manipulation were efficiently tackled. More advanced attacks can always be done, but in case of the targeted use case, the solution is reliable and trustworthy. Altogether, the project demonstrates that secure digital voting is possible in terms of close attention to protocol design and implementation.

5 Code Repository

The complete source code for the Secure Class Voting System is available at the following GitHub repository:

https://github.com/TharinduGee/Secure_Class_Voting_System