

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Лабораторна робота №2

з дисципліни

«Протоколи та алгоритми електронного голосування»

Виконав:

студент 4 курсу

групи ІІІ-02

Середюк Валентин

Київ – 2023

Лабораторна робота №2

Тема: Протокол Е-голосування зі сліпими підписами

Мета: Дослідити протокол Е-голосування зі сліпими підписами

Завдання

Змодельовати протокол Е-голосування зі сліпими підписами будь-якою мовою програмування та провести його дослідження. Для кодування повідомлень використовувати шифрування RSA, для реалізації ЕЦП використовувати алгоритм RSA.

Умови: В процесі голосування повинні приймати участь не менше 2 кандидатів та не менше 4 виборців. Повинні бути реалізовані сценарії поведінки на випадок порушення протоколу (виборець не проголосував, проголосував неправильно, виборець не має права голосувати, виборець хоче проголосувати повторно, виборець хоче проголосувати замість іншого виборця та інші).

На основі змодельованого протоколу провести його дослідження (Аналіз повинен бути розгорнутим та враховувати всі можливі сценарії подій під час роботи протоколу голосування):

1. Перевірити чи можуть голосувати ті, хто не має на це права.
2. Перевірити чи може виборець голосувати кілька разів.
3. Чи може хтось (інший виборець, ВК, стороння людина) дізнатися за кого проголосували інші виборці?
4. Перевірити чи може інший виборець чи стороння людина проголосувати замість іншого зареєстрованого виборця.
5. Чи може хтось (інший виборець, ВК, стороння людина) таємно змінити голос в бюлетені?
6. Чи може виборець перевірити, що його голос врахований при підведенні кінцевих результатів?

Хід роботи

Спочатку було реалізовано протокол електронного голосування, з сліпими підписами, який складається з наступних пунктів:

- Формування списку кандидатів та виборців.
- Кожен виборець створює 10 наборів із бюлетенями для кожного з кандидатів.
- Кожен виборець шифрує свої набори для передачі до ВК.
- ВК перевіряє 9 із 10 наборів для перевірки їх достовірності та підписує останній 10 набір та відправляє його виборцю.
- Виборець знімає маскування із бюлетенів та вибирає бюлетень для відправки із тим кандидатом, за якого хоче проголосувати.
- Кожен виборець шифрує свій бажаний бюлетень ключом ВК.
- Кожен виборець надсилає свій бюлетень до ВК.
- ЦВК розшифровує бюлетені, перевіряє підписи, підводить висновки та публікує результати голосування.

Для реалізації даного алгоритму було створено 4 класи, а саме клас для виборця, кандидата, бюлетня та виборчої комісії. Клас кандидата (код 1.1) містить інформацію про самого кандидата, його ім'я та кількість голосів за нього в проміжний момент часу.

```
Class Candidate {  
    private final String name;  
    private int votesCount;  
  
    public Candidate(String name) {  
        this.name = name;  
        votesCount = 0;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public int getVotesCount() {  
        return votesCount;  
    }  
}
```

```

    }

    public void votesInc() {
        votesCount++;
    }
}

```

Код 1.1 – Клас кандидата

Клас виборця (код 1.2) містить інформацію про виборця, його ключі, його голос та іншу другорядну інформацію. Також в ньому реалізоване шифрування RSA для інформації, яка буде передана виборчій комісії. Особливу увагу можна приділити методу, який генерує набори бюлетенів для успішної реалізації сліпого підпису. Детальний код у лістингу.

```

class Voter {
    private final boolean canVote;
    private boolean hasVoted;
    private boolean hasCounted;
    private int id;
    private final String name;
    private final KeyPair keyPair;
    private final byte[] key;
    private byte[] vote;
    private String encryptedVote;

    ...

    public void makeVote(int vote) throws
VoterHasAlreadyVotedException, VoteIsValidException
    {
        if (!hasVoted && signedBallots != null) {
            String[] data =
signedBallots.get(vote).getDecryptedData(keyPair.getPri
vate()).split(" ");
            if (Integer.parseInt(data[0]) == id &&
Integer.parseInt(data[1]) == vote) {
                this.vote = Integer.parseInt(data[1]);
                hasVoted = true;
            } else {
                throw new VoteIsValidException(name);
            }
        } else {
            throw new
VoterHasAlreadyVotedException(this.name);

```

```

    }
}

...

public void generateBallots(int examplesCount, int
candidatesCount) {
    ballotsExamples = new ArrayList<>();
    for (int i = 0; i < examplesCount; i++) {
        ArrayList<Ballot> temp = new ArrayList<>();
        for (int j = 0; j < candidatesCount; j++)
            temp.add(new Ballot(this, j));
        ballotsExamples.add(temp);
    }
}

...
}

```

Код 1.2 – Клас виборця

Клас бюлетеня про містить дані про сам бюлетень та дозволяє його шифрувати та розшифровувати (код 1.3).

```

import javax.crypto.Cipher;
import java.security.KeyPair;
import java.security.PrivateKey;
import java.util.Base64;

public class Ballot {
    private String data;
    private boolean isSigned = false;

    public Ballot(Voter voter, int vote) {
        data = encryptData(voter, voter.getId() + " " +
vote);
    }

    private String encryptData(Voter voter, String
data) {
        KeyPair keys = voter.getKeyPair();
        return Encryptor.encrypt(data,
keys.getPublic());
    }

    public String getDecryptedData(PrivateKey key) {

```

```

        return Encryptor.decrypt(data, key);
    }

    public boolean isSigned() {
        return isSigned;
    }

    public void makeSigned() {
        isSigned = true;
    }
}

```

Код 1.3 – Клас бюлетеня

Клас виборчої комісії (код 1.4) містить дані усіх виборців та кандидатів, методи для приймання голосів та підбивання підсумків, а також методи для шифрування самих бюлетенів за допомогою алгоритму RSA. Детальний код у лістингу.

```

class CentralElectionCommission {
    private Integer candidatesCount = 0;
    private Integer votersCount = 0;
    private final Map<Integer, Candidate> candidates;
    private final Map<Integer, Voter> voters;

    ...

    public void conductElection() {
        // Election phase: Voters cast their votes
        for (Voter voter : voters.values()) {
            if (voter.getVote() != -1) {
                System.out.println("Voter: " +
voter.getName() + " is counting...");
                String vote = "Vote for " +
voter.getVote(); // Assume all votes are for the first
candidate
                String encryptedVote =
Encryptor.encrypt(vote,
voter.getKeyPair().getPublic());
                voter.setEncryptedVote(encryptedVote);
            }
        }

        // Election phase: Central Election Commission
counts and announces results
    }
}

```

```

        int[] candidateVotes = new int[candidatesCount];
        for (int i = 0; i < candidatesCount; i++)
            candidateVotes[i] = 0;
        for (Voter voter : voters.values()) {
            String decryptedVote =
Encryptor.decrypt(voter.getEncryptedVote(),
voter.getKeyPair().getPrivate());
            if (decryptedVote != null) {
                String[] decryptedArray =
decryptedVote.split(" ");
                int index =
Integer.parseInt(decryptedArray[2]);
                candidateVotes[index]++;
                voter.makeCounted();
            } else if (voter.getVote() == -1 &&
voter.canVote()) {
                System.out.println("The voter " +
voter.getName() + " has not voted\n");
            } else {
                System.out.println("Invalid vote detected
for voter: " + voter.getName() + "\n");
            }
        }

        System.out.println("+-----+-----+
---+");
        System.out.println("|           ELECTION   RESULTS
|");
        System.out.println("+-----+-----+
---+");
        System.out.println("|      CANDIDATES      |      VOTES
|");
        System.out.println("+-----+-----+
---+");
        for (Candidate candidate : candidates.values()) {
            System.out.printf("| %16s | %12d |\n",
candidate.getName(), candidate.getVotesCount());
        }
        System.out.println("+-----+-----+
---+");
    }

    ...
}

```

Код 1.4 – Клас виборчої комісії

Оскільки під час використання даного протоколу можливі різні проблеми, наприклад виборець хоче проголосувати і т.д., то було додані нові типи виключень, які дозволяли перехоплювати моменти, коли виборці порушували умови. Увесь перелік нових виключень (рис. 1.1):

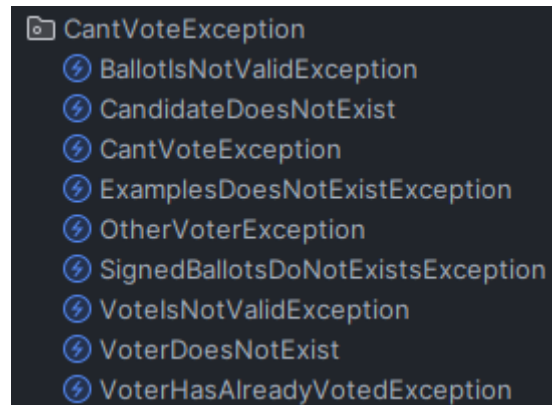


Рис. 1.1 – Список нових виключень

Демонстрація роботи алгоритму

Для демонстрації роботи було створено модель із 2 кандидатами та 6 виборцями, два з яких не мають права голосувати (код 2.1). Результат запуску програми (рис. 2.1):

```
CentralElectionCommission CEC = new
CentralElectionCommission(keyPairGenerator.generateKeyP
air());
CEC.addCandidate(candidate1);
CEC.addCandidate(candidate2);
CEC.addVoter(voter1);
CEC.addVoter(voter2);
CEC.addVoter(voter3);
CEC.addVoter(voter4);
CEC.addVoter(voter5);
CEC.addVoter(voter6);

voter1.generateBallots(countOfExamples,
CEC.getCandidatesCount());
voter1.setSignedBallots(CEC.getSignedBallot(voter1));
voter2.generateBallots(countOfExamples,
CEC.getCandidatesCount());
voter2.setSignedBallots(CEC.getSignedBallot(voter2));
voter3.generateBallots(countOfExamples,
CEC.getCandidatesCount());
voter3.setSignedBallots(CEC.getSignedBallot(voter3));
voter4.generateBallots(countOfExamples,
CEC.getCandidatesCount());
voter4.setSignedBallots(CEC.getSignedBallot(voter4));
voter5.generateBallots(countOfExamples,
CEC.getCandidatesCount());
voter5.setSignedBallots(CEC.getSignedBallot(voter5));
voter6.generateBallots(countOfExamples,
CEC.getCandidatesCount());
voter6.setSignedBallots(CEC.getSignedBallot(voter6));

CEC.makeVote(voter1, 0);
CEC.makeVote(voter2, 0);
CEC.makeVote(voter3, 0);
CEC.makeVote(voter3, 0);
CEC.makeVote(voter3, 0);
CEC.makeVote(voter3, 0);
CEC.makeVote(voter4, 1);
CEC.makeVote(voter5, 0);
```

```
CEC.makeVote(voter6, 1);  
  
// Conduct the election  
CEC.conductElection();
```

Код 2.1 – Методи для демонстрації роботи алгоритму

```
The voter Voter 2 can't vote  
  
The voter Voter 3 has already voted  
  
The voter Voter 3 has already voted  
  
The voter Voter 3 has already voted  
  
The voter Voter 6 can't vote  
  
+-----+-----+  
|           ELECTION RESULTS           |  
+-----+-----+  
|   CANDIDATES   |   VOTES   |  
+-----+-----+  
|   Candidate A   |     3     |  
|   Candidate B   |     1     |  
+-----+-----+
```

Рис. 2.1 – Результат запуску протоколу

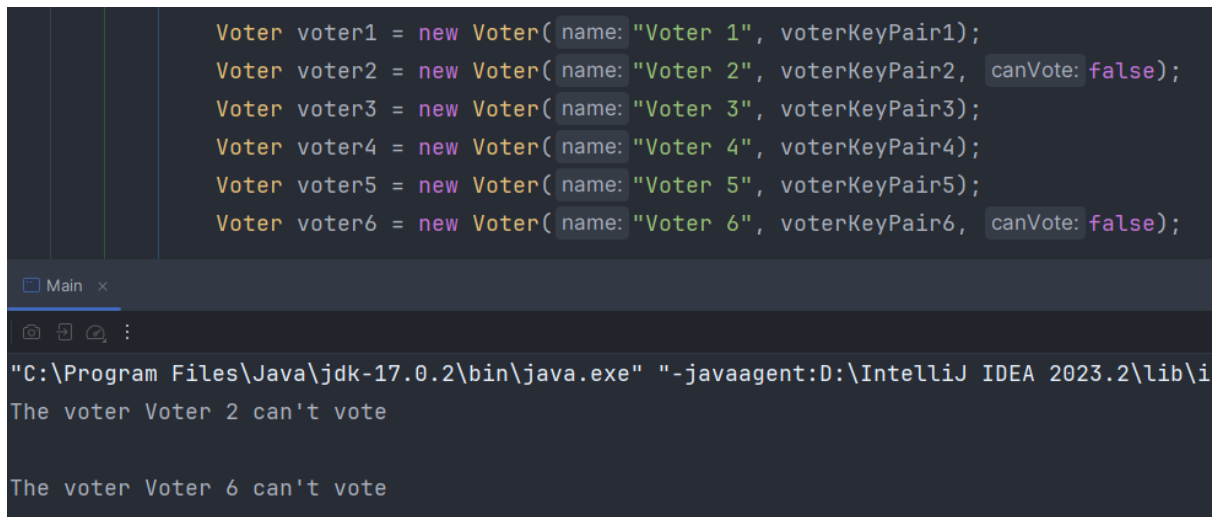
Як можна побачити із результату, виборець під номером 2 та виборець під номером 6 не був врахований, так як вони не мають права голосувати. Також можна побачити, що виборець з номером 3 намагався проголосувати чотири рази і йому не дало цього зробити, оскільки він уже голосував. Результат, який видала програма відповідає очікуваним результатам.

Дослідження протоколу

Для дослідження простого протоколу електронного голосування, скористаємось вимогами, які були надані у завданні до лабораторної роботи:

1. Перевірити чи можуть голосувати ті, хто не має на це права.
2. Перевірити чи може виборець голосувати кілька разів.
3. Чи може хтось (інший виборець, ЦВК, стороння людина) дізнатися за кого проголосували інші виборці?
4. Перевірити чи може інший виборець чи стороння людина проголосувати замість іншого зареєстрованого виборця.
5. Чи може хтось (інший виборець, ЦВК, стороння людина) таємно змінити голос в бюлетені?
6. Чи може виборець перевірити, що його голос врахований при підведенні кінцевих результатів?

Почнемо по-порядку, перша умова – Перевірити чи можуть голосувати ті, хто не має права (рис. 3.1).



```
Voter voter1 = new Voter( name: "Voter 1", voterKeyPair1);
Voter voter2 = new Voter( name: "Voter 2", voterKeyPair2, canVote: false);
Voter voter3 = new Voter( name: "Voter 3", voterKeyPair3);
Voter voter4 = new Voter( name: "Voter 4", voterKeyPair4);
Voter voter5 = new Voter( name: "Voter 5", voterKeyPair5);
Voter voter6 = new Voter( name: "Voter 6", voterKeyPair6, canVote: false);
```

Main x

"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaagent:D:\IntelliJ IDEA 2023.2\lib\i

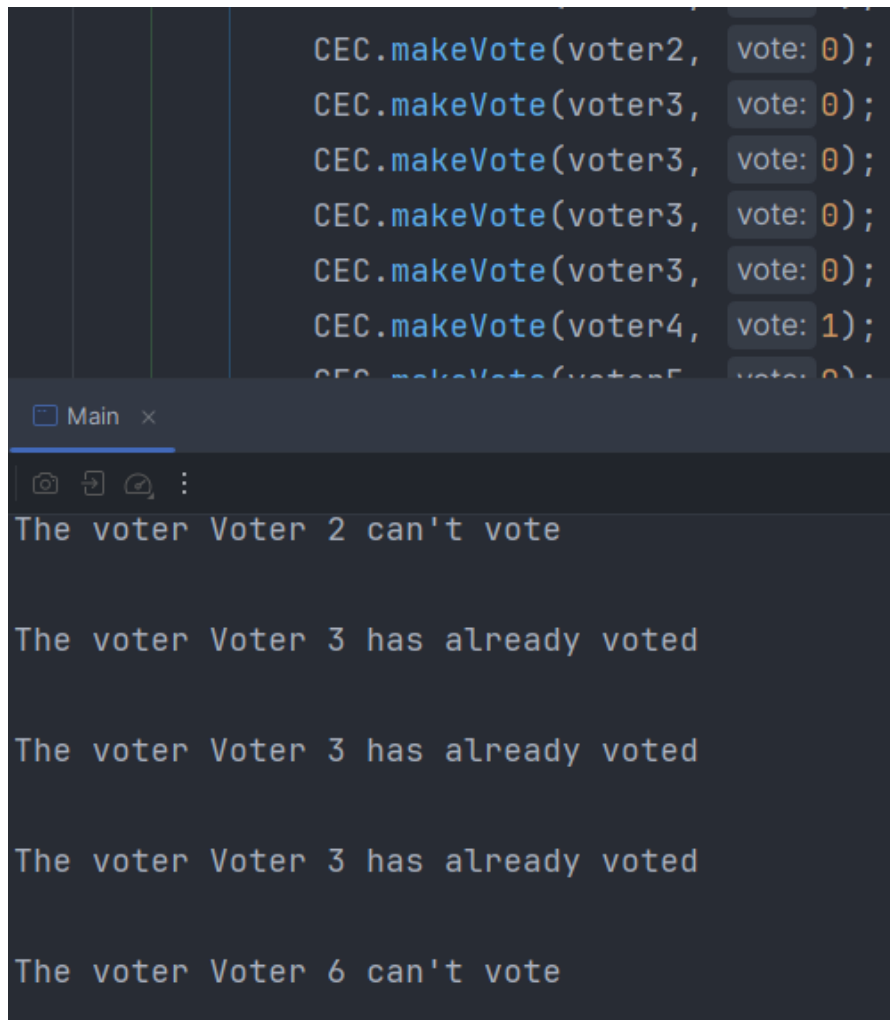
The voter Voter 2 can't vote

The voter Voter 6 can't vote

Рис. 3.1 – Результат запуску для виборців, які не можуть голосувати

Як можна побачити з результату, комісія не допустила голосувати тих, хто не може голосувати.

Другий пункт: Перевірити чи може виборець голосувати кілька разів (рис. 3.2).



```
CEC.makeVote(voter2, vote: 0);
CEC.makeVote(voter3, vote: 0);
CEC.makeVote(voter3, vote: 0);
CEC.makeVote(voter3, vote: 0);
CEC.makeVote(voter3, vote: 0);
CEC.makeVote(voter4, vote: 1);
CEC.makeVote(voter5, vote: 0);
```

Main x

The voter Voter 2 can't vote

The voter Voter 3 has already voted

The voter Voter 3 has already voted

The voter Voter 3 has already voted

The voter Voter 6 can't vote

Рис. 3.2 – Результат запуску для виборців, які голосують по декілька разів

Як можна побачити з результату, ті виборці, які голосували по декілька разів були враховані лише один раз, усі інші рази комісія не врахувала їх голос.

Третій пункт: Чи може хтось (інший виборець, ЦВК, стороння людина) дізнатися за кого проголосували інші виборці?

Алгоритм голосування з сліпим підписом не може гарантувати повної конфіденційності. ВК у будь який момент може розкрити дані, за кого проголосував виборець, що дає змогу ВК користуватись цією інформацією у корисних цілях. Наприклад спробуємо просто дізнатись хто і за кого проголосував під час підрахунку голосів (рис. 3.3):

```
String decryptedVote = Encryptor.decrypt(voter.getEncryptedVote(), voter.getKeyPair().getPrivate());
System.out.println(voter.getName() + "'s vote - " + decryptedVote);
```

Main x

Voter 1's vote - Vote for 0
Voter 2's vote - null
Voter 3's vote - Vote for 0
Voter 4's vote - Vote for 1
Voter 5's vote - Vote for 0
Voter 6's vote - null

Рис. 3.3 – Приклад доступу виборчої комісії до даних голосу виборця

Четвертий пункт: Перевірити чи може інший виборець чи стороння людина проголосувати замість іншого зареєстрованого виборця.

Так як, під час надсилання голосу, надсилає його сам виборець, а не хтось вказує, хто надсилає голос, то це не можливо. Є варіант, коли хтось інший скористається чужими вже підписаними бюлетенями. На такий випадок, шифрування не дозволить правильно розшифрувати голос, що призведе до виявлення порушення процесу голосування (рис. 3.4).

```
voter1.generateBallots(countOfExamples, CEC.getCandidatesCount());
voter1.setSignedBallots(CEC.getSignedBallot(voter1));
voter2.generateBallots(countOfExamples, CEC.getCandidatesCount());
voter2.setSignedBallots(CEC.getSignedBallot(voter2));
voter3.generateBallots(countOfExamples, CEC.getCandidatesCount());
voter3.setSignedBallots(CEC.getSignedBallot(voter1));
```

Main x

"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaagent:D:\IntelliJ IDEA 2023
The voter Voter 2 can't vote

Voter 3 voter is trying to vote for another voter

Voter 1's vote - Vote for 0
Voter 2's vote - null
Voter 3's vote - null

Рис. 3.4 – Визначення, чи голос відправлений від правильного виборця.

Як можна побачити, голос іншого виборця не зарахується.

П'ятий пункт: Чи може хтось (інший виборець, ЦВК, стороння людина) таємно змінити голос в бюлетені?

Голос надсилається виключно один раз, від виборця, надалі дані змінити не можна, тобто, ніхто не можна змінити голос, навіть сам виборець його змінити не може. Якщо взяти до уваги, що комісія буде просто зараховувати інший голос, замість наявного, то це відслідкувати неможливо. Але якщо потрібно саме замінити голос у бюлетені, то нічого не вийде, оскільки ВК не знає ключ шифрування, що замінити його (рис. 3.5).

```
if (decryptedVote != null) {  
    String[] decryptedArray = decryptedVote.split(regex: " ");  
    int index = Integer.parseInt(decryptedArray[2]);  
    candidateVotes[index]++;  
    voter.makeCounted();  
}
```

Рис. 3.5 – Фрагмент, де комісія може змінити дані про голос виборця

Останній пункт – Чи може виборець перевірити, що його голос врахований при підведенні кінцевих результатів?

Так, виборець може це зробити, як окремо, так і дізнатись результати від комісії, яка може надати список тих, проголосував (рис. 3.6).

```
System.out.println("Checking if " + voter2.getName() + " has been counted: " + voter2.checkIfCounted());  
System.out.println("Checking if " + voter3.getName() + " has been counted: " + voter3.checkIfCounted());  
System.out.println("Checking if " + voter4.getName() + " has been counted: " + voter4.checkIfCounted());
```

Main x

Checking if Voter 2 has been counted: false
Checking if Voter 3 has been counted: false
Checking if Voter 4 has been counted: true

Рис. 3.6 – Можливість перевірити, чи був голос врахованим

Висновок

Виконавши дану лабораторну роботу, було створено протокол електронного голосування із сліпими підписами. Для його реалізації було створено 4 основних сутності, такі як Виборець, Кандидат, Бюлетень та Виборча комісія. Також, для шифрування даних було використано RSA алгоритми (як для повідомлень, так і для реалізації ЕЦП), що дозволило забезпечити в певній мірі конфіденційність голосування.

Дослідження алгоритму показало, що даний алгоритм не є ідеальним, оскільки у ньому не виконуються певні умови. У даному алгоритмі, виборча комісія може дізнатись хто за кого проголосував, що є дуже суттєвим недоліком для алгоритму голосування. Також, комісія може вчинити самосуд та зараховувати голоси, які будуть відрізнятись від справжніх, що дає привід для недовіри такій системі голосування. Але в цілому, з іншими вимогами – даний алгоритм чудово справився.

Лістинг коду мовою Java

Main.java

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        final int countOfExamples = 10;
        try {
            // Generate key pairs for voters
            KeyPairGenerator keyPairGenerator =
KeyPairGenerator.getInstance("RSA");
            keyPairGenerator.initialize(2048);
            KeyPair voterKeyPair1 =
keyPairGenerator.generateKeyPair();
            KeyPair voterKeyPair2 =
keyPairGenerator.generateKeyPair();
            KeyPair voterKeyPair3 =
keyPairGenerator.generateKeyPair();
            KeyPair voterKeyPair4 =
keyPairGenerator.generateKeyPair();
            KeyPair voterKeyPair5 =
keyPairGenerator.generateKeyPair();
            KeyPair voterKeyPair6 =
keyPairGenerator.generateKeyPair();

            // Create candidates
            Candidate candidate1 = new
Candidate("Candidate A");
            Candidate candidate2 = new
Candidate("Candidate B");

            // Create voters
            Voter voter1 = new Voter("Voter 1",
voterKeyPair1);
            Voter voter2 = new Voter("Voter 2",
voterKeyPair2, false);
            Voter voter3 = new Voter("Voter 3",
voterKeyPair3);
            Voter voter4 = new Voter("Voter 4",
voterKeyPair4);
            Voter voter5 = new Voter("Voter 5",
voterKeyPair5);
```



```

        Voter voter6 = new Voter("Voter 6",
voterKeyPair6, false);

        // Create and configure Central Election
Commission
        CentralElectionCommission CEC = new
CentralElectionCommission(keyPairGenerator.generateKeyP
air());

        CEC.addCandidate(candidate1);
        CEC.addCandidate(candidate2);
        CEC.addVoter(voter1);
        CEC.addVoter(voter2);
        CEC.addVoter(voter3);
        CEC.addVoter(voter4);
        CEC.addVoter(voter5);
        CEC.addVoter(voter6);

        voter1.generateBallots(countOfExamples,
CEC.getCandidatesCount());

voter1.setSignedBallots(CEC.getSignedBallot(voter1));
        voter2.generateBallots(countOfExamples,
CEC.getCandidatesCount());

voter2.setSignedBallots(CEC.getSignedBallot(voter2));
        voter3.generateBallots(countOfExamples,
CEC.getCandidatesCount());

voter3.setSignedBallots(CEC.getSignedBallot(voter1));
        voter4.generateBallots(countOfExamples,
CEC.getCandidatesCount());

voter4.setSignedBallots(CEC.getSignedBallot(voter4));
        voter5.generateBallots(countOfExamples,
CEC.getCandidatesCount());

voter5.setSignedBallots(CEC.getSignedBallot(voter5));
        voter6.generateBallots(countOfExamples,
CEC.getCandidatesCount());

voter6.setSignedBallots(CEC.getSignedBallot(voter6));

        CEC.makeVote(voter1, 0);
        CEC.makeVote(voter2, 0);
        CEC.makeVote(voter3, 0);

```

```

        CEC.makeVote(voter3, 0);
        CEC.makeVote(voter3, 0);
        CEC.makeVote(voter3, 0);
        CEC.makeVote(voter4, 1);
        CEC.makeVote(voter5, 0);
        CEC.makeVote(voter6, 1);

        // Conduct the election
        CEC.conductElection();

        System.out.println("Checking if " +
voter2.getName() + " has been counted: " +
voter2.checkIfCounted());
        System.out.println("Checking if " +
voter3.getName() + " has been counted: " +
voter3.checkIfCounted());
        System.out.println("Checking if " +
voter4.getName() + " has been counted: " +
voter4.checkIfCounted());

        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

```

CentralElectionCommission.java

```

import java.security.*;
import java.util.*;

import CantVoteException.*;

class CentralElectionCommission {
    private Integer candidatesCount = 0;
    private Integer votersCount = 0;
    private KeyPair keys;
    private final Map<Integer, Candidate> candidates;
    private final Map<Integer, Voter> voters;

    public CentralElectionCommission(KeyPair keyPair) {
        candidates = new HashMap<>();
    }
}

```

```

        voters = new HashMap<>();
        this.keys = keyPair;
    }

    public void addCandidate(Candidate candidate) {
        candidates.put(candidatesCount, candidate);
        candidatesCount++;
    }

    public void addVoter(Voter voter) {
        voters.put(votersCount, voter);
        voter.setId(votersCount);
        votersCount++;
    }

    public void conductElection() {
        // Election phase: Voters cast their votes
        for (Voter voter : voters.values()) {
            if (voter.getVote() != -1) {
                // System.out.println("Voter: " +
                voter.getName() + " is counting...");
                String vote = "Vote for " +
                voter.getVote(); // Assume all votes are for the first
                candidate
                String encryptedVote =
                Encryptor.encrypt(vote,
                voter.getKeyPair().getPublic());
                voter.setEncryptedVote(encryptedVote);
            }
        }

        // Election phase: Central Election Commission
        counts and announces results
        int[] candidateVotes = new
        int[candidatesCount];
        for (int i = 0; i < candidatesCount; i++)
            candidateVotes[i] = 0;
        for (Voter voter : voters.values()) {
            String decryptedVote =
            Encryptor.decrypt(voter.getEncryptedVote(),
            voter.getKeyPair().getPrivate());
            System.out.println(voter.getName() + "'s
            vote - " + decryptedVote);
            if (decryptedVote != null) {
                String[] decryptedArray =

```

```

decryptedVote.split(" ");
        int index =
Integer.parseInt(decryptedArray[2]);
        candidateVotes[index]++;
        voter.makeCounted();
    } else if (voter.getVote() == -1 &&
voter.canVote()) {
//          System.out.println("The voter " +
voter.getName() + " has not voted\n");
        } else {
//          System.out.println("Invalid vote
detected for voter: " + voter.getName() + "\n");
        }
    }

    System.out.println("+-----+-----+
-----+");
    System.out.println("|          ELECTION  RESULTS
|");
    System.out.println("+-----+-----+
-----+");
    System.out.println("|      CANDIDATES      |
VOTES      |");
    System.out.println("+-----+-----+
-----+");
    for (Candidate candidate : candidates.values())
    {
        System.out.printf("| %16s | %12d |\n",
candidate.getName(), candidate.getVotesCount());
    }
    System.out.println("+-----+-----+
-----+");
}

    public void makeVote(Voter voter, int vote) {
        try {
            if (!voters.containsValue(voter)) {
                throw new
VoterDoesNotExist(voter.getName());
            }
            if (!candidates.containsKey(vote)) {
                throw new
CandidateDoesNotExist(candidates.get(vote).getName());
            }
            if (!voter.canVote()) {

```

```

        throw new
CantVoteException(voter.getName());
    }
    if
(!voters.get(voter.getId()).equals(voter)) {
        throw new
OtherVoterException(voter.getName());
    }
    if (!voter.hasSignedBallots()) {
        throw new
SignedBallotsDoNotExistsException(voter.getName());
    }
    voter.makeVote(vote);
    candidates.get(vote).votesInc();
} catch (Exception e) {
    System.out.println(e.getMessage() + "\n");
}
}

    public ArrayList<Ballot> getSignedBallot(Voter
voter) {
        ArrayList<ArrayList<Ballot>> voterExamples =
voter.getBallotsExamples();
        Random r = new Random();
        int randIndex =
r.nextInt(voterExamples.size());
        try {
            if (voterExamples.isEmpty())
                throw new
ExamplesDoesNotExistException(voter.getName());
            for (int i = 0; i < voterExamples.size();
i++) {
                if (i != randIndex) {
                    for (int j = 0; j <
voterExamples.get(i).size(); j++) {
                        String decryptedData =
voterExamples.get(i).get(j).getDecryptedData(voter.getKeyPair().getPrivate());
                        String[] data =
decryptedData.split(" ");
                        if (Integer.parseInt(data[0])
!= voter.getId() || Integer.parseInt(data[1]) != j) {
                            throw new
BallotIsNotValidException(decryptedData);
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    }
    }
    ArrayList<Ballot> signedBallots = new
ArrayList<>();
    for (int i = 0; i <
voterExamples.get(randIndex).size(); i++) {
        Ballot signedBallot =
voterExamples.get(randIndex).get(i);
        signedBallot.makeSigned();
        signedBallots.add(signedBallot);
    }
    return signedBallots;
} catch (BallotIsValidException |
ExamplesDoesNotExistException e) {
    System.out.println(e.getMessage());
    return null;
}
}

public int getCandidatesCount() {
    return candidatesCount;
}

public void printVotingStatus() {
    System.out.println("+-----+-----
-----+");
    System.out.println("|          CANDIDATES STATUS
|");
    System.out.println("+-----+-----
-----+");
    System.out.println("|      CANDIDATES      |
VOTES    |");
    System.out.println("+-----+-----
-----+");
    for (Candidate candidate : candidates.values())
{
        System.out.printf("| %16s | %12d |\n",
candidate.getName(), candidate.getVotesCount());
    }
    System.out.println("+-----+-----
-----+");
    System.out.println("|          VOTERS STATUS
|");
    System.out.println("+-----+-----

```

```

-----+");
        System.out.println("|          VOTERS          |
hasVOTE    |");
        System.out.println("+-----+-----+-----+
-----+");
        for (Voter voter : voters.values()) {
            System.out.printf("| %16s | %12b |\n",
voter.getName(), voter.hasVoted());
        }
        System.out.println("+-----+-----+-----+
-----+");
    }
}

    private String encryptVote(String vote, PublicKey
publicKey) {
        try {
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.ENCRYPT_MODE,
publicKey);

            byte[] encryptedBytes =
cipher.doFinal(vote.getBytes());
            return
Base64.getEncoder().encodeToString(encryptedBytes);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return null;
    }

    private String decryptVote(String encryptedVote,
PrivateKey privateKey) {
        try {
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.DECRYPT_MODE,
privateKey);

            byte[] encryptedBytes =
Base64.getDecoder().decode(encryptedVote);
            byte[] decryptedBytes =
cipher.doFinal(encryptedBytes);
            return new String(decryptedBytes);
        } catch (Exception ignored) { }
        return null;
    }
}

```

```

        public void printVotingStatus() {
            System.out.println("+-----+-----+
-----+");
            System.out.println("|          CANDIDATES STATUS
|");
            System.out.println("+-----+-----+
-----+");
            System.out.println("|      CANDIDATES      |
VOTES      |");
            System.out.println("+-----+-----+
-----+");
            for (Candidate candidate : candidates.values())
            {
                System.out.printf("| %16s | %12d |\n",
candidate.getName(), candidate.getVotesCount());
            }
            System.out.println("+-----+-----+
-----+");
            System.out.println("|          VOTERS STATUS
|");
            System.out.println("+-----+-----+
-----+");
            System.out.println("|      VOTERS      |
hasVOTE     |");
            System.out.println("+-----+-----+
-----+");
            for (Voter voter : voters.values()) {
                System.out.printf("| %16s | %12b |\n",
voter.getName(), voter.hasVoted());
            }
            System.out.println("+-----+-----+
-----+");
        }
    }
}

```

Candidate.java

```

class Candidate {
    private final String name;
    private int votesCount;

    public Candidate(String name) {
        this.name = name;
        votesCount = 0;
    }
}

```



```

    }

    public String getName() {
        return name;
    }

    public int getVotesCount() {
        return votesCount;
    }

    public void votesInc() {
        votesCount++;
    }
}

```

Voter.java

```

import CantVoteException.OtherVoterException;
import CantVoteException.VoteIsNotValidException;
import CantVoteException.VoterHasAlreadyVotedException;

import java.security.KeyPair;
import java.security.interfaces.RSAPublicKey;
import java.util.ArrayList;

class Voter {
    private final boolean canVote;
    private boolean hasVoted;
    private boolean hasCounted;
    private int id;
    private final String name;
    private final KeyPair keyPair;
    private int vote;
    private String encryptedVote;
    private ArrayList<ArrayList<Ballot>>
ballotsExamples;
    private ArrayList<Ballot> signedBallots;

    public Voter(String name, KeyPair keyPair) {
        this.name = name;
        this.keyPair = keyPair;
        canVote = true;
        hasVoted = false;
        hasCounted = false;
        vote = -1;
    }
}

```

```

    }

    public Voter(String name, KeyPair keyPair, boolean
canVote) {
        this.name = name;
        this.keyPair = keyPair;
        this.canVote = canVote;
        hasVoted = false;
        hasCounted = false;
        vote = -1;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public KeyPair getKeyPair() {
        return keyPair;
    }

    public int getVote() {
        return vote;
    }

    public String getEncryptedVote() {
        return encryptedVote;
    }

    public void setEncryptedVote(String encryptedVote)
{
        this.encryptedVote = encryptedVote;
    }

    public boolean hasVoted() {
        return hasVoted;
    }

```

```

    public boolean canVote() {
        return canVote;
    }

    public void makeVote(int vote) throws
VoterHasAlreadyVotedException, VoteIsNotValidException,
OtherVoterException {
        if (!hasVoted && signedBallots != null) {
            if
(signedBallots.get(vote).getDecryptedData(keyPair.getPr
ivate()) == null) {
                throw new OtherVoterException(name);
            }
            String[] data =
signedBallots.get(vote).getDecryptedData(keyPair.getPri
vate()).split(" ");
            if (Integer.parseInt(data[0]) == id &&
Integer.parseInt(data[1]) == vote) {
                this.vote = Integer.parseInt(data[1]);
                hasVoted = true;
            } else {
                throw new
VoteIsNotValidException(name);
            }
        } else {
            throw new
VoterHasAlreadyVotedException(this.name);
        }
    }

    public void makeCounted() {
        hasCounted = true;
    }

    public boolean checkIfCounted() {
        return hasCounted;
    }

    public void generateBallots(int examplesCount, int
candidatesCount) {
        ballotsExamples = new ArrayList<>();
        for (int i = 0; i < examplesCount; i++) {
            ArrayList<Ballot> temp = new ArrayList<>();
            for (int j = 0; j < candidatesCount; j++)
                temp.add(new Ballot(this, j));
        }
    }

```

```

        ballotsExamples.add(temp);
    }
}

public ArrayList<ArrayList<Ballot>>
getBallotsExamples() {
    return ballotsExamples;
}

public void setSignedBallots(ArrayList<Ballot>
signedBallots) {
    this.signedBallots = signedBallots;
}

public boolean hasSignedBallots() {
    if (signedBallots == null) return false;
    return !signedBallots.isEmpty();
}
}

```

Ballot.java

```

import javax.crypto.Cipher;
import java.security.KeyPair;
import java.security.PrivateKey;
import java.util.Base64;

public class Ballot {
    private String data;
    private boolean isSigned = false;

    public Ballot(Voter voter, int vote) {
        data = encryptData(voter, voter.getId() + " " +
vote);
    }

    private String encryptData(Voter voter, String
data) {
        KeyPair keys = voter.getKeyPair();
        return Encryptor.encrypt(data,
keys.getPublic());
    }

    public String getDecryptedData(PrivateKey key) {
        return Encryptor.decrypt(data, key);
    }
}

```

```

    }

    public boolean isSigned() {
        return isSigned;
    }

    public void makeSigned() {
        isSigned = true;
    }
}

```

Encryptor.java

```

import javax.crypto.Cipher;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.Base64;

public class Encryptor {
    public static String encrypt(String data, PublicKey
key) {
        try {
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.ENCRYPT_MODE, key);

            byte[] encryptedBytes =
cipher.doFinal(data.getBytes());
            return
Base64.getEncoder().encodeToString(encryptedBytes);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return null;
    }

    public static String decrypt(String data,
PrivateKey key) {
        try {
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.DECRYPT_MODE, key);

            byte[] encryptedBytes =
Base64.getDecoder().decode(data);
            byte[] decryptedBytes =
cipher.doFinal(encryptedBytes);

```

```

        return new String(decryptedBytes);
    } catch (Exception ignored) { }
    return null;
}
}

```

Custom Exceptions:

```

package CantVoteException;

public class VoterHasAlreadyVotedException extends
Exception{
    public VoterHasAlreadyVotedException(String name) {
        super("The voter " + name + " has already
voted");
    }
}

public class OtherVoterException extends Exception {
    public OtherVoterException(String name) {
        super(name + " voter is trying to vote for
another voter");
    }
}

public class VoterDoesNotExist extends Exception {
    public VoterDoesNotExist(String name) {
        super("The voter " + name + " doesn't exist in
CEC list");
    }
}

public class CantVoteException extends Exception {
    public CantVoteException(String name) {
        super("The voter " + name + " can't vote");
    }
}

public class CandidateDoesNotExist extends Exception {
    public CandidateDoesNotExist(String name) {
        super("The candidate " + name + " doesn't exist
in CEC list");
    }
}

```

```
public class VoteIsNotValidException extends Exception
{
    public VoteIsNotValidException(String name) {
        super("The " + name + "'s vote is not valid");
    }
}

public class SignedBallotsDoNotExistsException extends
Exception {
    public SignedBallotsDoNotExistsException(String
name) {
        super("Signed ballots for " + name + " do not
exists");
    }
}

public class ExamplesDoesNotExistException extends
Exception {
    public ExamplesDoesNotExistException(String name) {
        super("The ballots examples for " + name + "
doesn't exists");
    }
}

public class BallotIsNotValidException extends
Exception {
    public BallotIsNotValidException(String data) {
        super("The Ballot with data " + data + " is not
valid");
    }
}
```