

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

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

**Лабораторна робота №1**

з дисципліни

«Протоколи та алгоритми електронного голосування»

Виконав:

студент 4 курсу

групи ІІІ-02

Середюк Валентин

Київ – 2023

## **Лабораторна робота №1**

**Тема:** Простий протокол Е-голосування

**Мета:** Дослідити протокол простого Е-голосування

### **Завдання**

Змодельовати простий протокол Е-голосування будь-якою мовою програмування та провести його дослідження. Для кодування повідомлень використовувати метод гамування, 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, 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?

## Хід роботи

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

- Формування списку кандидатів та виборців.
- Кожен виборець підписує свій бюлетень своїм ключем.
- Кожен виборець шифрує свій бюлетень ключом ЦВК.
- Кожен виборець надсилає свій бюлетень до ЦВК.
- ЦВК розшифровує бюлетені, перевіряє підписи, підводить висновки та публікує результати голосування.

Для реалізації даного алгоритму було створено 3 класи, а саме клас для виборця, кандидата 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 гамма шифрування для інформації, яка буде передана виборчій комісії. 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 {
        if (!hasVoted) {
            String voteString =
Integer.toBinaryString(vote);
            this.vote = voteString.getBytes();
            for (int i = 0; i < this.vote.length; i++)
{
                this.vote[i] -= 48;
                this.vote[i] = (byte) (this.vote[i] ^
key[i % key.length]);
            }
            hasVoted = true;
        } else {
            throw new
VoterHasAlreadyVotedException(this.name);
        }
    }

    ...

    public int getDecryptedGammaVote() {
        if (vote == null)
            return -1;
        if (vote.length == 0)
            return -1;
```

```

        byte[] temp = new byte[vote.length];
        for (int i = 0; i < vote.length; i++) {
            temp[i] = (byte) (vote[i] ^ key[i %
key.length]);
            temp[i] += 48;
        }
        String res = new String(temp);
        return Integer.parseInt(res);
    }

    ...
}

```

Код 1.2 – Клас виборця

Клас виборчої комісії (код 1.3) 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.getDecryptedGammaVote() != -1) {
                System.out.println("Voter: " +
voter.getName() + " is counting...");
                String vote = "Vote for " +
voter.getDecryptedGammaVote(); // Assume all votes are
for the first candidate
                String encryptedVote =
encryptVote(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 =
decryptVote(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.getDecryptedGammaVote() ==
-1 && voter.canVote()) {
                System.out.println("The voter " +
voter.getName() + " has not voted");
            } else {
                System.out.println("Invalid vote
detected for voter: " + voter.getName());
            }
        }

        System.out.println("\n\nElection Results:");
        for (int i = 0; i < candidatesCount; i++) {
            System.out.println("Candidate: " +
candidates.get(i).getName() + " -> votes: " +
candidateVotes[i]);
        }
    }
    ...
}

```

Код 1.3 – 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):

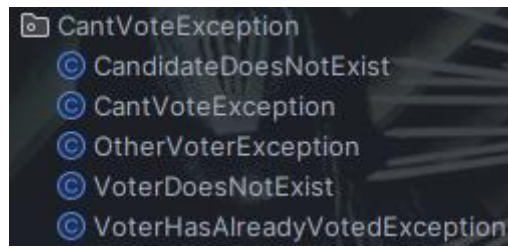


Рис. 1.1 – Список нових виключень

### Демонстрація роботи алгоритму

Для демонстрації роботи було створено модель із 2 кандидатами та 5 виборцями, один з яких не має права голосувати (рис. 2.1). Результат запуску програми (рис. 2.2):

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

CEC.makeVote(voter1, vote: 0);
CEC.makeVote(voter1, vote: 1);
CEC.makeVote(voter2, vote: 0);
CEC.makeVote(voter3, vote: 1);
CEC.printVotingStatus();
CEC.makeVote(voter4, vote: 1);
CEC.makeVote(voter5, vote: 0);

// Conduct the election
CEC.printVotingStatus();
CEC.conductElection();
```

Рис. 2.1 – Методи для демонстрації роботи алгоритму



```
The voter Voter 1 has already voted
The voter Voter 2 can't vote
Voter: Voter 1 is counting...
Voter: Voter 3 is counting...
Voter: Voter 4 is counting...
Voter: Voter 5 is counting...
Invalid vote detected for voter: Voter 2

Election Results:
Candidate: Candidate A -> votes: 2
Candidate: Candidate B -> votes: 2
```

Рис. 2.2 – Результат запуску протоколу

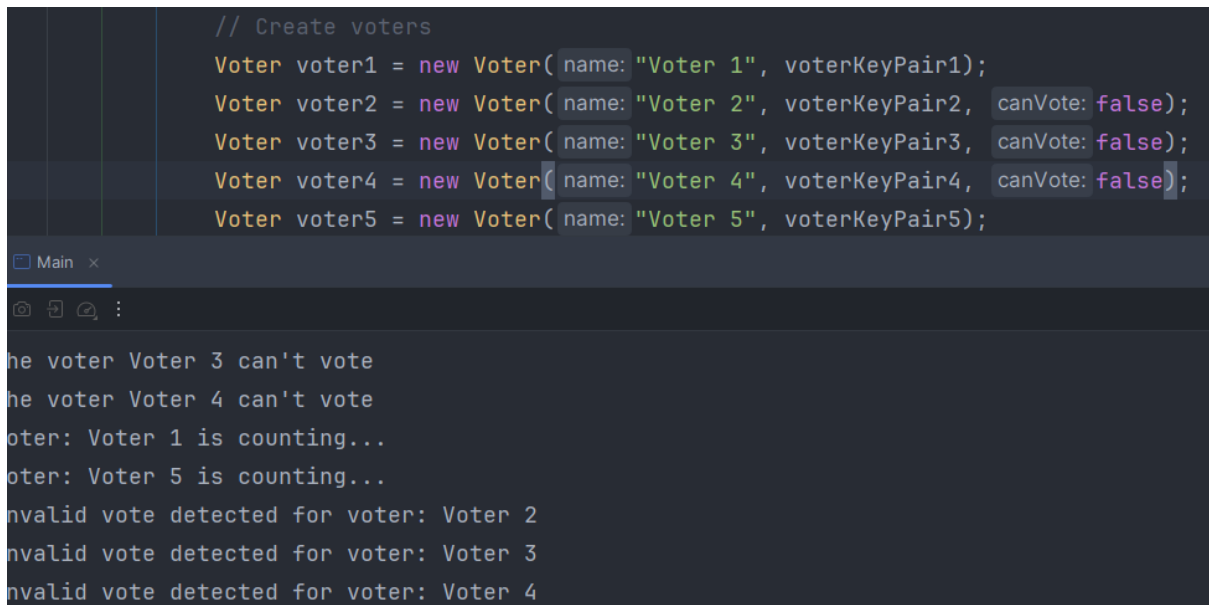
Як можна побачити із результату, виборець під номером 2 не був врахований, так як він не має права голосувати. Також можна побачити, що виборець з номером 1 намагався проголосувати двічі і йому не дало цього зробити, оскільки він уже голосував. Результат, який видала програма відповідає очікуваним результатам.

## 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).



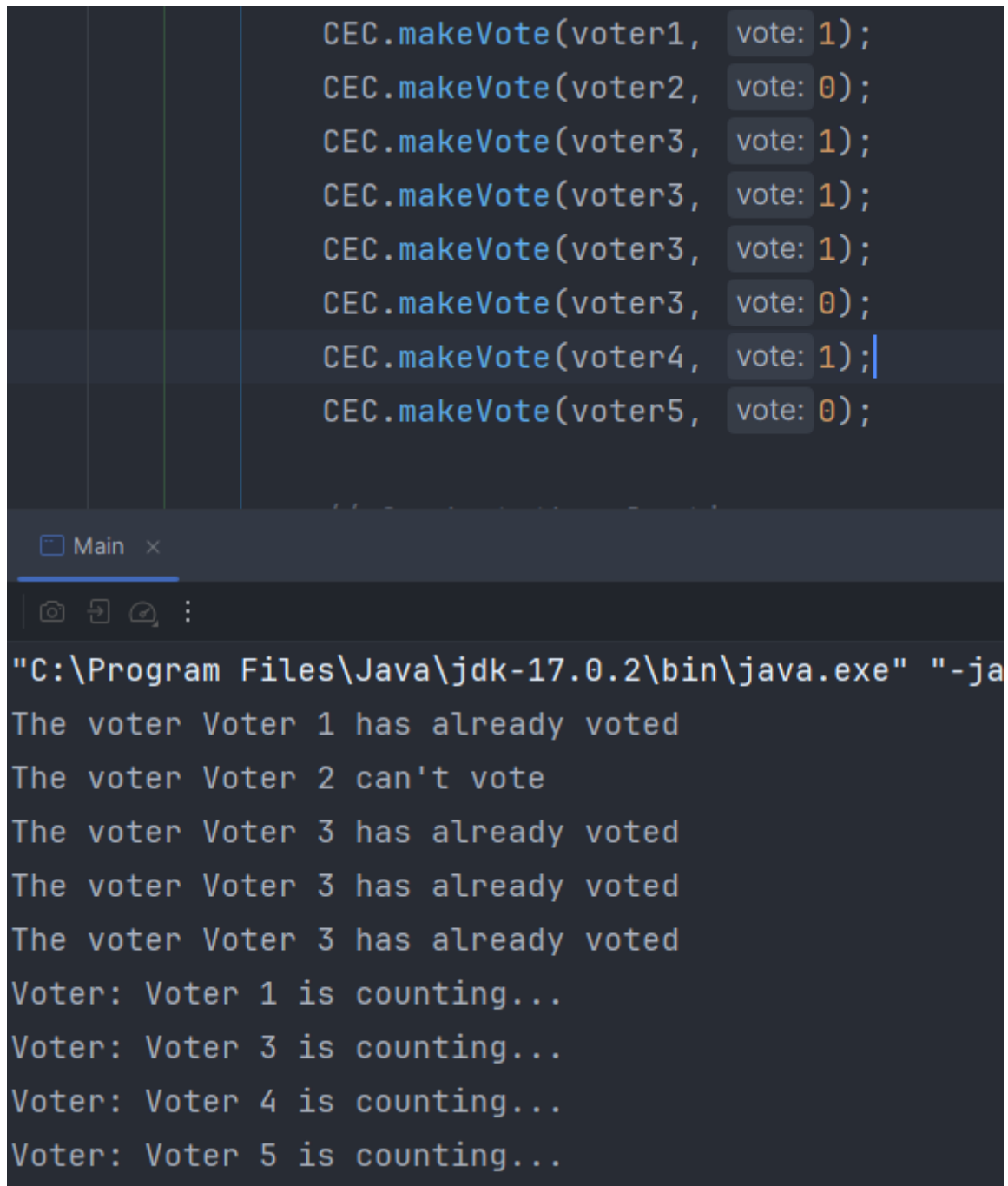
```
// Create voters
Voter voter1 = new Voter( name: "Voter 1", voterKeyPair1);
Voter voter2 = new Voter( name: "Voter 2", voterKeyPair2, canVote: false);
Voter voter3 = new Voter( name: "Voter 3", voterKeyPair3, canVote: false);
Voter voter4 = new Voter( name: "Voter 4", voterKeyPair4, canVote: false);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

Main x
he voter Voter 3 can't vote
he voter Voter 4 can't vote
oter: Voter 1 is counting...
oter: Voter 5 is counting...
nvalid vote detected for voter: Voter 2
nvalid vote detected for voter: Voter 3
nvalid vote detected for voter: Voter 4
```

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).



The screenshot shows a Java IDE with a dark theme. The top part displays a code editor with the following Java code:

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

Below the code editor, the IDE's interface shows a tab labeled "Main" and a toolbar with icons for running, debugging, and other actions. The bottom part of the screenshot shows the console output of the program:

```
"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-ja
The voter Voter 1 has already voted
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
Voter: Voter 1 is counting...
Voter: Voter 3 is counting...
Voter: Voter 4 is counting...
Voter: Voter 5 is counting...
```

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?

Простий алгоритм має з цим проблему, виборча комісія, наприклад, має прямий доступ до голосу виборця та хто проголосував, змінити його вона не може, але це впливає на конфіденційність даних виборців. Наприклад виборча комісія розшифровує бюлетень виборця без використання його ключів, що робить дані голосу виборця не захищеними (рис. 3.3):

```
for (Voter voter : voters.values()) {  
    String decryptedVote = decryptVote(voter.getEncryptedVote(), vot  
    if (decryptedVote != null) {  
        String[] decryptedArray = decryptedVote.split(" ");
```

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, надсилаю його сам виборець, а не хтось вказує, хто надсилає голос, то це не можливо. До того ж, під час валідації перевіряється відповідність особи, яка голосує до тієї, що є в списках комісії, тобто ніхто інший за виборця не проголосує (рис. 3.4).

```
if (!voters.get(voter.getId()).equals(voter)) {  
    throw new OtherVoterException(voter.getName());  
}
```

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. Тим більше, комісія отримує його лише у

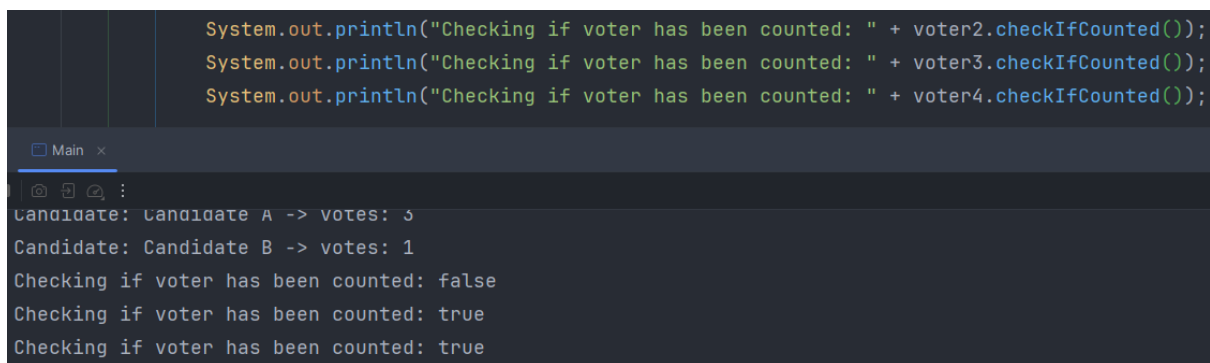
зашифрованому вигляді, тобто вона не буде знати як зашифровано, щоб змінити голос та накласти новий шифр (рис. 3.5).

```
String decryptedVote = decryptVote(voter.getEncryptedVote(), voter.getKeyPair())
if (decryptedVote != null) {
```

Рис. 3.5 – Фрагмент, де комісія отримує 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 voter has been counted: " + voter2.checkIfCounted());
System.out.println("Checking if voter has been counted: " + voter3.checkIfCounted());
System.out.println("Checking if voter has been counted: " + voter4.checkIfCounted());
```

Main x

⌕ ↻ 🔍 :

Candidate: Candidate A -> votes: 3

Candidate: Candidate B -> votes: 1

Checking if voter has been counted: false

Checking if voter has been counted: true

Checking if voter has been counted: true

Рис. 3.6 – Можливість перевірити, чи був голос врахованим

## **Висновок**

Виконавши дану лабораторну роботу, було створено простий протокол електронного голосування. Для його реалізації було створено 3 основних сутності, такі як Виборець, Кандидат та Виборча комісія. Також, для шифрування даних було використано гамма та RSA алгоритми, 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. Але в цілому, з іншими вимогами – даний алгоритм чудово справився.

## Лістинг коду мовою Java

### Main.java

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

public class Main {
    public static void main(String[] args) {
        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();

            // Create candidates
            Candidate candidatel = 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);

            // Create and configure Central Election
Commission
            CentralElectionCommission CEC = new
```

```

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

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

    // Conduct the election
    CEC.conductElection();

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

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

```

## CentralElectionCommission.java

```

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

import CantVoteException.*;

import javax.crypto.*;

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

```



```

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

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.getDecryptedGammaVote() != -1) {
            System.out.println("Voter: " +
voter.getName() + " is counting...");
            String vote = "Vote for " +
voter.getDecryptedGammaVote(); // Assume all votes are
for the first candidate
            String encryptedVote =
encryptVote(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 =
decryptVote(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.getDecryptedGammaVote() ==
-1 && voter.canVote()) {
        System.out.println("The voter " +
voter.getName() + " has not voted");
    } else {
        System.out.println("Invalid vote
detected for voter: " + voter.getName());
    }
}

    System.out.println("\n\nElection Results:");
    for (int i = 0; i < candidatesCount; i++) {
        System.out.println("Candidate: " +
candidates.get(i).getName() + " -> votes: " +
candidateVotes[i]);
    }
}

    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());
            }
            voter.makeVote(vote);
            candidates.get(vote).votesInc();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

```

```

    }
}

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.VoterHasAlreadyVotedException;  
  
import java.security.KeyPair;  
import java.security.interfaces.RSAPublicKey;  
  
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 Voter(String name, KeyPair keyPair) {  
        this.name = name;  
        this.keyPair = keyPair;  
        canVote = true;  
        hasVoted = false;  
        hasCounted = false;  
        key = getKeyFromRSAKey();  
    }  
  
    public Voter(String name, KeyPair keyPair, boolean  
canVote) {  
        this.name = name;  
        this.keyPair = keyPair;  
        this.canVote = canVote;  
        hasVoted = false;  
        hasCounted = false;  
        key = getKeyFromRSAKey();  
    }  
  
    public int getId() {  
        return id;  
    }  
}
```

```

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

    public String getName() {
        return name;
    }

    public KeyPair getKeyPair() {
        return keyPair;
    }

    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 {
        if (!hasVoted) {
            String voteString =
Integer.toBinaryString(vote);
            this.vote = voteString.getBytes();
            for (int i = 0; i < this.vote.length; i++)
            {
                this.vote[i] -= 48;
                this.vote[i] = (byte) (this.vote[i] ^
key[i % key.length]);
            }
            hasVoted = true;
        } else {
            throw new

```

```

VoterHasAlreadyVotedException(this.name);
    }
}

private byte[] getKeyFromRSAKey() {
    RSAPublicKey publicKey = (RSAPublicKey)
keyPair.getPublic();
    byte[] res =
publicKey.getModulus().toByteArray();
    for (int i = 0; i < res.length; i++) {
        res[i] %= 2;
    }
    return res;
}

public int getDecryptedGammaVote() {
    if (vote == null)
        return -1;
    if (vote.length == 0)
        return -1;
    byte[] temp = new byte[vote.length];
    for (int i = 0; i < vote.length; i++) {
        temp[i] = (byte) (vote[i] ^ key[i %
key.length]);
        temp[i] += 48;
    }
    String res = new String(temp);
    return Integer.parseInt(res);
}

public void makeCounted() {
    hasCounted = true;
}

public boolean checkIfCounted() {
    return hasCounted;
}
}

```

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");
    }
}
```