

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, бюлетеня 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 та методи для генерації та отримання бюлетенів. 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;

    ...

    public void generateBallots(int examplesCount, int candidatesCount,
    PublicKey key) {
        ballotsExamples = new ArrayList<>();
        r = Encryptor.findR(key);
        for (int i = 0; i < examplesCount; i++) {
            ArrayList<Ballot> temp = new ArrayList<>();
            for (int j = 0; j < candidatesCount; j++) {
                Ballot tempBallot = new Ballot(this, j);
                BigInteger m_ =
                Encryptor.getM_(Integer.parseInt(tempBallot.getData()), key, r);
                tempBallot.setData(String.valueOf(m_));
                temp.add(tempBallot);
            }
            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).

```

public class Ballot {
    private String data;

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

    public Ballot(String data) {
        this.data = data;
    }

    public void encrypt(PublicKey key) {
        data = Encryptor.encrypt(data, key);
    }
}

```

```

    }

    public void decrypt(PrivateKey key) {
        data = Encryptor.decrypt(data, key);
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}

```

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 для отримання набору із сліпим підписом RSA. Детальний код у лістингу.

```

class CentralElectionCommission {
    ...
    private final KeyPair keys;
    private final Map<Integer, Candidate> candidates;
    private final Map<Integer, Voter> voters;
    private final ArrayList<Boolean> isVoterChecked;
    private final Map<Integer, Ballot> ballots;

    ...

    public void conductElection() {
        for (Integer voterId : ballots.keySet()) {
            String signedData = ballots.get(voterId).getData();
            BigInteger s = Encryptor.getS(new BigInteger(signedData),
voters.get(voterId).getR(), keys.getPublic());
            BigInteger m = Encryptor.getM(s, keys.getPublic());
            int vote = m.mod(BigInteger.TEN).intValue();
//            if (voterId == 3)
//                candidates.get(0).votesInc();
//            else
                candidates.get(vote).votesInc();
                voters.get(voterId).makeCounted();
//            System.out.println(voters.get(voterId).getName() + "'s vote -
" + vote);
        }

        printVotingResults();
    }

    public void sendBallot(Voter voter, Ballot ballot) {
        try {
            if (ballot != null) {
                Ballot decryptedBallot = new
Ballot(Encryptor.decrypt(ballot.getData(), keys.getPrivate()));
                String signedData = decryptedBallot.getData();
                BigInteger s = Encryptor.getS(new BigInteger(signedData),
voter.getR(), keys.getPublic());
                BigInteger m = Encryptor.getM(s, keys.getPublic());
                int voterId = m.divide(BigInteger.TEN).intValue();

```

```

        int vote = m.mod(BigInteger.TEN).intValue();
        if (!voters.containsKey(voterId))
            throw new Exception("Incorrect ballot");
        if (voter.getId() != voterId)
            throw new OtherVoterException(voter.getName());
        if (!isVoterChecked.get(voterId))
            throw new
SignedBallotsDoNotExistsException(voter.getName());
        if (checkBallotData(voter, vote)) {
            ballots.put(voterId, decryptedBallot);
            voter.vote();
        }
    } else {
        throw new SignedBallotsDoNotExistsException("null");
    }
} catch (Exception e) {
    System.out.println(e.getMessage());
}
}

public boolean checkBallotData(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());
        if (voter.hasVoted())
            throw new VoterHasAlreadyVotedException(voter.getName());
        return true;
    } catch (Exception e) {
        System.out.println(e.getMessage() + "\n");
    }
    return false;
}

public ArrayList<Ballot> getSignedBallot(ArrayList<ArrayList<Ballot>>
ballotsList, BigInteger r) {
    Random rnd = new Random();
    int randIndex = rnd.nextInt(ballotsList.size());
    try {
        if (ballotsList.isEmpty())
            throw new ExamplesDoesNotExistException();
        int voterId = -1;
        int prevVoterId = -1;
        for (int i = 0; i < ballotsList.size(); i++) {
            if (i != randIndex) {
                for (int j = 0; j < ballotsList.get(i).size(); j++) {
                    String ballotData =
ballotsList.get(i).get(j).getData();
                    BigInteger s_ = Encryptor.getS_(new
BigInteger(ballotData), keys.getPrivate());
                    BigInteger s = Encryptor.getS(s_, r,
keys.getPublic());

```

```

        BigInteger m = Encryptor.getM(s, keys.getPublic());
        voterId = m.divide(BigInteger.TEN).intValue();
        int vote = m.mod(BigInteger.TEN).intValue();
        if (voterId != prevVoterId && prevVoterId != -1)
            throw new IncorrectBallotsList();
        if (vote != j)
            throw new
BallotIsNotValidException(voters.get(voterId).getName());
        if (isVoterChecked.get(voterId))
            throw new VoterAlreadyHasSignedBallots();
        prevVoterId = voterId;
    }
}

ArrayList<Ballot> signedBallots = new ArrayList<>();
for (int i = 0; i < ballotsList.get(randIndex).size(); i++) {
    Ballot signedBallot = ballotsList.get(randIndex).get(i);
    String signedData = signedBallot.getData();
    BigInteger s_ = Encryptor.getS_(new BigInteger(signedData),
keys.getPrivate());
    signedBallot.setData(String.valueOf(s_));
    signedBallots.add(signedBallot);
}
isVoterChecked.set(voterId, true);
return signedBallots;
} catch (Exception e) {
    System.out.println(e.getMessage());
    return null;
}
}
...
}

```

Код 1.4 – Клас виборчої комісії

Оскільки, у даній лабораторній роботі використовується лише один алгоритм шифрування, то було розроблено клас, який дозволяв виконувати шифрування та дешифрування за допомогою алгоритму RSA, а також накладати маску сліпого підпису RSA (код 1.5).

```

public class Encryptor {
    public static String encrypt(String data, PublicKey key) {
        try {
            BigInteger m = new BigInteger(data);
            BigInteger e = ((RSAPublicKey) key).getPublicExponent();
            BigInteger n = ((RSAPublicKey) key).getModulus();
            return String.valueOf(m.modPow(e, n));
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return null;
    }

    public static String decrypt(String data, PrivateKey key) {
        try {
            BigInteger m = new BigInteger(data);
            BigInteger d = ((RSAPrivateKey) key).getPrivateExponent();

```

```

        BigInteger n = ((RSAPrivateKey) key).getModulus();
        return String.valueOf(m.modPow(d, n));
    } catch (Exception ignored) { }
    return null;
}

public static BigInteger findR(PublicKey key) {
    BigInteger n = ((RSAPublicKey) key).getModulus();
    BigInteger r = BigInteger.probablePrime(8, new Random());
    while (!r.gcd(n).equals(BigInteger.ONE)) r =
BigInteger.probablePrime(8, new Random());
    return r;
}

public static BigInteger getM_(int data, PublicKey key, BigInteger r) {
    BigInteger m_ = new BigInteger(String.valueOf(data));
    BigInteger e = ((RSAPublicKey) key).getPublicExponent();
    BigInteger n = ((RSAPublicKey) key).getModulus();
    return m_.multiply(r.pow(e.intValue())).mod(n);
}

public static BigInteger getS_(BigInteger m_, PrivateKey key) {
    BigInteger d = ((RSAPrivateKey) key).getPrivateExponent();
    BigInteger n = ((RSAPrivateKey) key).getModulus();
    return m_.modPow(d, n);
}

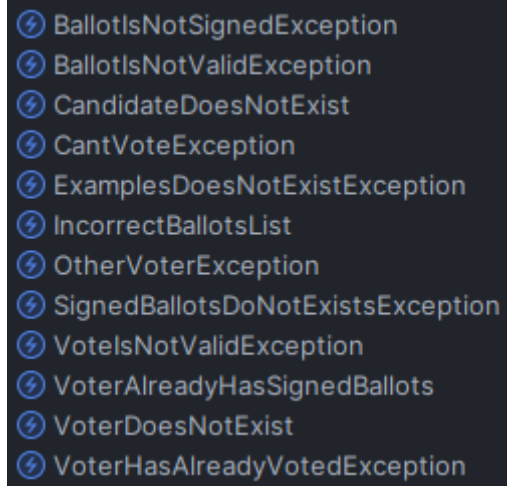
public static BigInteger getS(BigInteger s_, BigInteger r, PublicKey
key) {
    BigInteger n = ((RSAPublicKey) key).getModulus();
    return s_.multiply(r.modInverse(n)).mod(n);
}

public static BigInteger getM(BigInteger s, PublicKey key) {
    BigInteger e = ((RSAPublicKey) key).getPublicExponent();
    BigInteger n = ((RSAPublicKey) key).getModulus();
    return s.modPow(e, n);
}
}

```

Код 1.5 – Клас шифрувальник методом RSA

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):



A screenshot of a code editor showing a list of new exceptions. Each item in the list is preceded by a blue circular icon containing a white lightning bolt. The list is as follows:

- ⚡ BallotIsNotSignedException
- ⚡ BallotIsNotValidException
- ⚡ CandidateDoesNotExist
- ⚡ CantVoteException
- ⚡ ExamplesDoesNotExistException
- ⚡ IncorrectBallotsList
- ⚡ OtherVoterException
- ⚡ SignedBallotsDoNotExistException
- ⚡ VotesNotValidException
- ⚡ VoterAlreadyHasSignedBallots
- ⚡ VoterDoesNotExist
- ⚡ VoterHasAlreadyVotedException

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(CECKeyPair);

    for (Candidate candidate : candidates)
        CEC.addCandidate(candidate);

    for (Voter voter : voters)
        CEC.addVoter(voter);

    for (Voter voter : voters) {
        voter.generateBallots(countOfExamples,
candidates.size(), CECKeyPair.getPublic());
voter.setSignedBallots(CEC.getSignedBallot(voter.getBallotsExamples(),
voter.getR()));
    }

    Ballot voter1Ballot =
voter1.chooseSignedBallotWithCandidate(CECKeyPair.getPublic(), 0);
voter1Ballot.encrypt(CECKeyPair.getPublic());
CEC.sendBallot(voter1, voter1Ballot);

    Ballot voter2Ballot =
voter2.chooseSignedBallotWithCandidate(CECKeyPair.getPublic(), 0);
voter2Ballot.encrypt(CECKeyPair.getPublic());
CEC.sendBallot(voter2, voter2Ballot);

    Ballot voter3Ballot =
voter3.chooseSignedBallotWithCandidate(CECKeyPair.getPublic(), 0);
voter3Ballot.encrypt(CECKeyPair.getPublic());
CEC.sendBallot(voter3, voter3Ballot);
CEC.sendBallot(voter3, voter3Ballot);
CEC.sendBallot(voter3, voter3Ballot);

    Ballot voter4Ballot =
voter4.chooseSignedBallotWithCandidate(CECKeyPair.getPublic(), 1);
voter4Ballot.encrypt(CECKeyPair.getPublic());
CEC.sendBallot(voter4, voter4Ballot);

    Ballot voter5Ballot =
voter5.chooseSignedBallotWithCandidate(CECKeyPair.getPublic(), 0);
voter5Ballot.encrypt(CECKeyPair.getPublic());
CEC.sendBallot(voter5, voter5Ballot);

    Ballot voter6Ballot =
voter6.chooseSignedBallotWithCandidate(CECKeyPair.getPublic(), 0);
voter6Ballot.encrypt(CECKeyPair.getPublic());
CEC.sendBallot(voter6, voter6Ballot);

    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 6 can't vote

+-----+-----+
|      ELECTION RESULTS      |
+-----+-----+
| CANDIDATES | VOTES |
+-----+-----+
| Candidate A |      3 |
| Candidate B |      1 |
+-----+-----+

```

Рис. 2.1 – Результат запуску протоколу

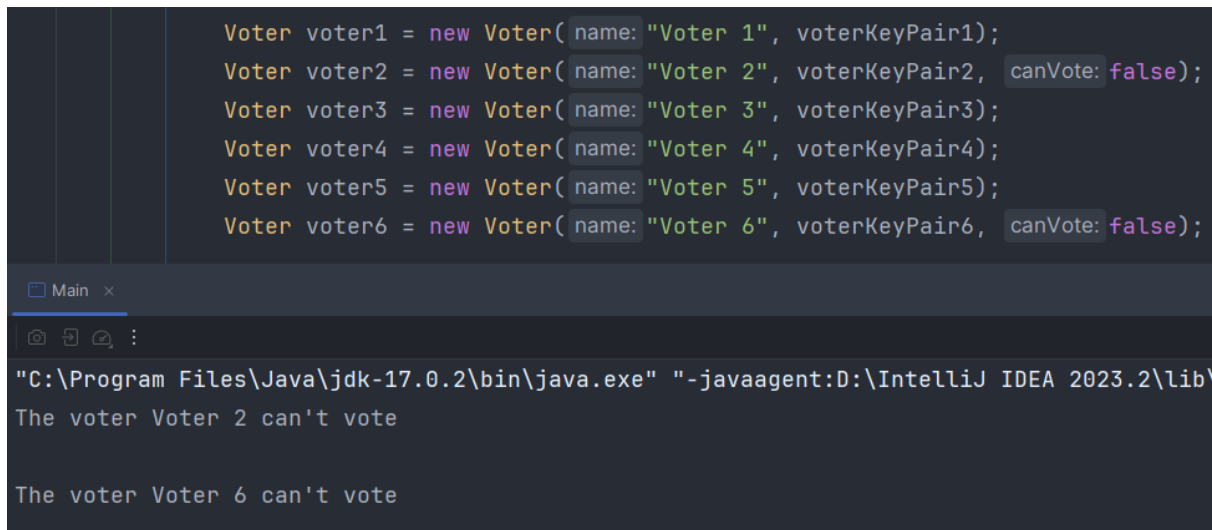
Як можна побачити із результату, виборець під номером 2 та виборець під номером 6 не був врахований, так як вони не мають права голосувати. Також можна побачити, що виборець з номером 3 намагався проголосувати три рази і йому не дало цього зробити, оскільки він уже голосував. Результат, який видала програма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

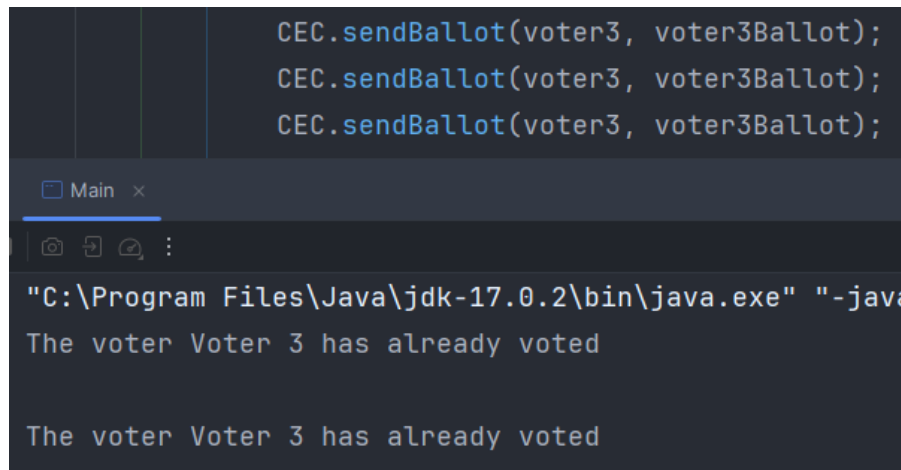
The voter Voter 2 can't vote

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.sendBallot(voter3, voter3Ballot);
CEC.sendBallot(voter3, voter3Ballot);
CEC.sendBallot(voter3, voter3Ballot);
```

Main x

"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-java
The voter Voter 3 has already voted
The voter Voter 3 has already voted

Рис. 3.2 – Результат запуску для виборця, який голосує 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):

```
7 public void conductElection() {
8     for (Integer voterId : ballots.keySet()) {
9         String signedData = ballots.get(voterId).getData();
10        BigInteger s = Encryptor.getS(new BigInteger(signedData), voters.get(vote
11        BigInteger m = Encryptor.getM(s, keys.getPublic());
12        int vote = m.mod(BigInteger.TEN).intValue();
13        candidates.get(vote).votesInc();
14        voters.get(voterId).makeCounted();
15        System.out.println(voters.get(voterId).getName() + "'s vote - " + vote);
16    }
17 }
```

Main x

Voter 1's vote - 0
Voter 3's vote - 0
Voter 4's vote - 1
Voter 5's vote - 0

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).

```
voter1Ballot.encrypt(CECKeyPair.getPublic());
CEC.sendBallot(voter1, voter1Ballot);
CEC.sendBallot(voter2, voter1Ballot);
```

Main x

"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaagent:D
Voter 2 voter is trying to vote for another voter

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. Але, якщо ж враховувати, що комісія буде просто враховувати голос, який вона захоче, то таким чином, можна буде змінити голос (рис. 3.5).

```

1 usage  ▸ Valentyn Serediuk *
public void conductElection() {
    for (Integer voterId : ballots.keySet()) {
        String signedData = ballots.get(voterId).getData();
        BigInteger s = Encryptor.getS(new BigInteger(signedData), voters.get(voterId).getR(), keys.getPublic());
        BigInteger m = Encryptor.getM(s, keys.getPublic());
        int vote = m.mod(BigInteger.TEN).intValue();
        if (voterId == 3)
            candidates.get(0).votesInc();
        else
            candidates.get(vote).votesInc();
        voters.get(voterId).makeCounted();
    }
}

```

ELECTION RESULTS	
CANDIDATES	VOTES
Candidate A	4
Candidate B	0

Рис. 3.5 – Фрагмент, де комісія може змінити зарахований 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 і власне бюлетені виборців (рис. 3.6).

```

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

```

```

Checking if Voter 2 has been counted: false
There is not ballot for the voter Voter 2
Checking if Voter 3 has been counted: true
1924751662063165945325854696616524277954186847382182534297756488553436751286120253180366869342322741862836299948130169944
Checking if Voter 4 has been counted: true
1009879832464977526301932205393221617279433080666349019013573292520788702955818198965819074382839455530934643501386665555

```

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 сліпого 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 static java.util.Arrays.asList;

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();

            KeyPair CECKeyPair = 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);

            ArrayList<Candidate> candidates = new
                ArrayList<>(asList(candidate1, candidate2));
            ArrayList<Voter> voters = new ArrayList<>(asList(voter1,
                voter2, voter3, voter4, voter5, voter6));

            // Create and configure Central Election Commission
            CentralElectionCommission CEC = new
                CentralElectionCommission(CECKeyPair);

            for (Candidate candidate : candidates)
                CEC.addCandidate(candidate);

            for (Voter voter : voters)
                CEC.addVoter(voter);

            for (Voter voter : voters) {
                if (voter.getId() != 3) {
                    voter.generateBallots(countOfExamples,
                        candidates.size(), CECKeyPair.getPublic());
                    voter.setSignedBallots(CEC.getSignedBallot(voter.getBallotsExamples(),
                        voter.getR()));
                }
                voter4.generateFakeBallots(countOfExamples, candidates.size(),
                    CECKeyPair.getPublic());
            }
        }
    }
}
```

```

voter4.setSignedBallots(CEC.getSignedBallot(voter4.getBallotsExamples(),
voter4.getR()));

        Ballot voter1Ballot =
voter1.chooseSignedBallotWithCandidate(CECKeypair.getPublic(), 0);
voter1Ballot.encrypt(CECKeypair.getPublic());
CEC.sendBallot(voter1, voter1Ballot);
CEC.sendBallot(voter2, voter1Ballot);

        Ballot voter2Ballot =
voter2.chooseSignedBallotWithCandidate(CECKeypair.getPublic(), 0);
voter2Ballot.encrypt(CECKeypair.getPublic());
CEC.sendBallot(voter2, voter2Ballot);

        Ballot voter3Ballot =
voter3.chooseSignedBallotWithCandidate(CECKeypair.getPublic(), 0);
voter3Ballot.encrypt(CECKeypair.getPublic());
CEC.sendBallot(voter3, voter3Ballot);
CEC.sendBallot(voter3, voter3Ballot);
CEC.sendBallot(voter3, voter3Ballot);

        Ballot voter4Ballot =
voter4.chooseSignedBallotWithCandidate(CECKeypair.getPublic(), 1);
voter4Ballot.encrypt(CECKeypair.getPublic());
CEC.sendBallot(voter4, voter4Ballot);

        Ballot voter5Ballot =
voter5.chooseSignedBallotWithCandidate(CECKeypair.getPublic(), 0);
voter5Ballot.encrypt(CECKeypair.getPublic());
CEC.sendBallot(voter5, voter5Ballot);

        Ballot voter6Ballot =
voter6.chooseSignedBallotWithCandidate(CECKeypair.getPublic(), 0);
voter6Ballot.encrypt(CECKeypair.getPublic());
CEC.sendBallot(voter6, voter6Ballot);

        // Conduct the election
        CEC.conductElection();

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

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

```

CentralElectionCommission.java

```

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

    public CentralElectionCommission(KeyPair keyPair) {
        candidates = new TreeMap<>();
        voters = new TreeMap<>();
        isVoterChecked = new ArrayList<>();
        ballots = new TreeMap<>();
        this.keys = keyPair;
    }

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

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

    public void conductElection() {
        for (Integer voterId : ballots.keySet()) {
            String signedData = ballots.get(voterId).getData();
            BigInteger s = Encryptor.getS(new BigInteger(signedData),
voters.get(voterId).getR(), keys.getPublic());
            BigInteger m = Encryptor.getM(s, keys.getPublic());
            int vote = m.mod(BigInteger.TEN).intValue();
//            if (voterId == 3)
//                candidates.get(0).votesInc();
//            else
                candidates.get(vote).votesInc();
            voters.get(voterId).makeCounted();
//            System.out.println(voters.get(voterId).getName() + "'s vote -
" + vote);
        }

        printVotingResults();
    }

    public void sendBallot(Voter voter, Ballot ballot) {
        try {
            if (ballot != null && ballot.getData() != null) {
                Ballot decryptedBallot = new
Ballot(Encryptor.decrypt(ballot.getData(), keys.getPrivate()));
                String signedData = decryptedBallot.getData();
                BigInteger s = Encryptor.getS(new BigInteger(signedData),
voter.getR(), keys.getPublic());
                BigInteger m = Encryptor.getM(s, keys.getPublic());
                int voterId = m.divide(BigInteger.TEN).intValue();
                int vote = m.mod(BigInteger.TEN).intValue();
                if (!voters.containsKey(voterId))
                    throw new Exception("Incorrect ballot");
            }
        }
    }
}

```

```

        if (voter.getId() != voterId)
            throw new OtherVoterException(voter.getName());
        if (!isVoterChecked.get(voterId))
            throw new
SignedBallotsDoNotExistsException(voter.getName());
        if (checkBallotData(voter, vote)) {
            ballots.put(voterId, decryptedBallot);
            voter.vote();
        }
    } else {
        throw new SignedBallotsDoNotExistsException("null");
    }
} catch (Exception e) {
    System.out.println(e.getMessage());
}
}

public boolean checkBallotData(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());
        if (voter.hasVoted())
            throw new VoterHasAlreadyVotedException(voter.getName());
        return true;
    } catch (Exception e) {
        System.out.println(e.getMessage() + "\n");
    }
    return false;
}

public ArrayList<Ballot> getSignedBallot(ArrayList<ArrayList<Ballot>>
ballotsList, BigInteger r) {
    Random rnd = new Random();
    int randIndex = rnd.nextInt(ballotsList.size());
    try {
        if (ballotsList.isEmpty())
            throw new ExamplesDoesNotExistException();
        int voterId = -1;
        int prevVoterId = -1;
        for (int i = 0; i < ballotsList.size(); i++) {
            if (i != randIndex) {
                for (int j = 0; j < ballotsList.get(i).size(); j++) {
                    String ballotData =
ballotsList.get(i).get(j).getData();
                    BigInteger s_ = Encryptor.getS_(new
BigInteger(ballotData), keys.getPrivate());
                    BigInteger s = Encryptor.getS(s_, r,
keys.getPublic());
                    BigInteger m = Encryptor.getM(s, keys.getPublic());
                    voterId = m.divide(BigInteger.TEN).intValue();
                    int vote = m.mod(BigInteger.TEN).intValue();

```

```

        if (voterId != prevVoterId && prevVoterId != -1)
            throw new IncorrectBallotsList();
        if (vote != j)
            throw new
BallotIsNotValidException(voters.get(voterId).getName());
        if (isVoterChecked.get(voterId))
            throw new VoterAlreadyHasSignedBallots();
        prevVoterId = voterId;
    }
}

ArrayList<Ballot> signedBallots = new ArrayList<>();
for (int i = 0; i < ballotsList.get(randIndex).size(); i++) {
    Ballot signedBallot = ballotsList.get(randIndex).get(i);
    String signedData = signedBallot.getData();
    BigInteger s_ = Encryptor.getS_(new BigInteger(signedData),
keys.getPrivate());
    signedBallot.setData(String.valueOf(s_));
    signedBallots.add(signedBallot);
}
isVoterChecked.set(voterId, true);
return signedBallots;
} catch (Exception e) {
    System.out.println(e.getMessage());
    return null;
}
}

public Ballot getBallotFor(Voter voter) {
    if (ballots.containsKey(voter.getId()))
        return ballots.get(voter.getId());
    else
        return new Ballot("There is not ballot for the voter " +
voter.getName());
}

public void printVotingResults() {
    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 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

```

class Voter {
    private final boolean canVote;
    private boolean hasVoted;
    private boolean hasCounted;
    private int id;
    private final String name;
    private final KeyPair keyPair;
    private BigInteger r;
    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;
    }

    public Voter(String name, KeyPair keyPair, boolean canVote) {
        this.name = name;
        this.keyPair = keyPair;
    }
}

```

```

        this.canVote = canVote;
        hasVoted = false;
        hasCounted = false;
    }

    public int getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public KeyPair getKeyPair() {
        return keyPair;
    }

    public void vote() {
        hasVoted = true;
    }

    public boolean hasVoted() {
        return hasVoted;
    }

    public boolean canVote() {
        return canVote;
    }

    public void makeCounted() {
        hasCounted = true;
    }

    public boolean checkIfCounted() {
        return hasCounted;
    }

    public BigInteger getR() {
        return r;
    }

    public void generateBallots(int examplesCount, int candidatesCount,
        PublicKey key) {
        ballotsExamples = new ArrayList<>();
        r = Encryptor.findR(key);
        for (int i = 0; i < examplesCount; i++) {
            ArrayList<Ballot> temp = new ArrayList<>();
            for (int j = 0; j < candidatesCount; j++) {
                Ballot tempBallot = new Ballot(this, j);
                BigInteger m_ =
Encryptor.getM_(Integer.parseInt(tempBallot.getData()), key, r);
                tempBallot.setData(String.valueOf(m_));
                temp.add(tempBallot);
            }
            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();
    }

    public Ballot chooseSignedBallotWithCandidate(PublicKey key, int
candidate) {
        try {
            if (signedBallots == null)
                throw new SignedBallotsDoNotExistsException(name);
            for (Ballot ballot : signedBallots) {
                String signedData = ballot.getData();
                BigInteger s = Encryptor.getS(new BigInteger(signedData),
r, key);

                BigInteger m = Encryptor.getM(s, key);
                int voterId = m.divide(BigInteger.TEN).intValue();
                int vote = m.mod(BigInteger.TEN).intValue();
                if (voterId != id)
                    throw new SignedBallotsDoNotExistsException(name);
                if (vote == candidate)
                    return ballot;
            }
        } catch (SignedBallotsDoNotExistsException e) {
            System.out.println(e.getMessage());
        }
        return new Ballot(this, -1);
    }

    public void generateFakeBallots(int examplesCount, int candidatesCount,
PublicKey key) {
        ballotsExamples = new ArrayList<>();
        r = Encryptor.findR(key);
        for (int i = 0; i < examplesCount; i++) {
            ArrayList<Ballot> temp = new ArrayList<>();
            for (int j = 0; j < candidatesCount; j++) {
                Ballot tempBallot = new Ballot(this, 0);
                BigInteger m_ =
Encryptor.getM(Integer.parseInt(tempBallot.getData()), key, r);
                tempBallot.setData(String.valueOf(m_));
                temp.add(tempBallot);
            }
            ballotsExamples.add(temp);
        }
    }
}

```

Ballot.java

```

public class Ballot {
    private String data;

```



```

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

public Ballot(String data) {
    this.data = data;
}

public void encrypt(PublicKey key) {
    data = Encryptor.encrypt(data, key);
}

public void decrypt(PrivateKey key) {
    data = Encryptor.decrypt(data, key);
}

public String getData() {
    return data;
}

public void setData(String data) {
    this.data = data;
}
}

```

Encryptor.java

```

public class Encryptor {
    public static String encrypt(String data, PublicKey key) {
        try {
            BigInteger m = new BigInteger(data);
            BigInteger e = ((RSAPublicKey) key).getPublicExponent();
            BigInteger n = ((RSAPublicKey) key).getModulus();
            return String.valueOf(m.modPow(e, n));
        } catch (Exception ignore) {}
        return null;
    }

    public static String decrypt(String data, PrivateKey key) {
        BigInteger m = new BigInteger(data);
        BigInteger d = ((RSAPrivateKey) key).getPrivateExponent();
        BigInteger n = ((RSAPrivateKey) key).getModulus();
        return String.valueOf(m.modPow(d, n));
    }

    public static BigInteger findR(PublicKey key) {
        BigInteger n = ((RSAPublicKey) key).getModulus();
        BigInteger r = BigInteger.probablePrime(8, new Random());
        while (!r.gcd(n).equals(BigInteger.ONE)) r =
        BigInteger.probablePrime(8, new Random());
        return r;
    }

    public static BigInteger getM_(int data, PublicKey key, BigInteger r) {
        BigInteger m_ = new BigInteger(String.valueOf(data));
        BigInteger e = ((RSAPublicKey) key).getPublicExponent();
        BigInteger n = ((RSAPublicKey) key).getModulus();
        return m_.multiply(r.pow(e.intValue())) .mod(n);
    }
}

```

```

    public static BigInteger getS_(BigInteger m_, PrivateKey key) {
        BigInteger d = ((RSAPrivateKey) key).getPrivateExponent();
        BigInteger n = ((RSAPrivateKey) key).getModulus();
        return m_.modPow(d, n);
    }

    public static BigInteger getS(BigInteger s_, BigInteger r, PublicKey
key) {
        BigInteger n = ((RSAPublicKey) key).getModulus();
        return s_.multiply(r.modInverse(n)).mod(n);
    }

    public static BigInteger getM(BigInteger s, PublicKey key) {
        BigInteger e = ((RSAPublicKey) key).getPublicExponent();
        BigInteger n = ((RSAPublicKey) key).getModulus();
        return s.modPow(e, n);
    }
}

```

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");
    }
}

public class VoterAlreadyHasSignedBallots extends Exception {
    public VoterAlreadyHasSignedBallots() {
        super("The voter has already signed ballots");
    }
}

public class IncorrectBallotsList extends Exception {
    public IncorrectBallotsList() {
        super("Incorrect ballots list");
    }
}

package CantVoteException;

public class BallotIsNotSignedException extends Exception {
    public BallotIsNotSignedException() {
        super("The ballot is not signed");
    }
}
```