

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

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Лабораторна робота №3

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 з двома виборчими комісіями

Мета: Дослідити протокол Е-голосування з двома виборчими комісіями

Завдання

Змодельовати протокол Е-голосування з двома виборчими комісіями будь-якою мовою програмування та провести його дослідження. Для кодування повідомлень використовувати метод Ель-Гамала, для реалізації ЕЦП використовувати алгоритм DSA.

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 двома виборчими комісіями, де одна з них є бюро реєстрації (БР), а інша є самою ж виборчою комісією (ВК), який складається з наступних пунктів:

- Кожен виборець відправляє повідомлення до БР, у якому він запитує реєстраційний номер.
- БР повертає виборцю випадковий реєстраційний номер. БР зберігає список даних реєстраційних номерів разом зі списком їх отримувачів на випадок, якщо хтось спробує проголосувати двічі.
- БР відправляє список реєстраційних номерів до ВК.
- Кожен виборець обирає для себе випадковий ідентифікаційний номер (на власний вибір). Після чого він створює повідомлення, в якому міститься даний ідентифікаційний номер, реєстраційний номер від БР та свій е-бюлетень. Він надсилає це повідомлення до ВК.
- ВК перевіряє реєстраційний номер по списку, який їй надало БР. Якщо номер є в списку, то ВК вилучає його зі списку, щоб уникнути повторного голосування. ВК добавляє ідентифікаційний номер до списку тих, хто проголосував та одразу зараховує отриманий голос.
- Після закінчення виборчого процесу, ВК публікує результати разом зі списком, в якому містяться ідентифікаційні номери виборців та відповідні їм бюлетені.

Для реалізації даного алгоритму було декілька класів для реалізації усього процесу голосування. Розпочнемо з класу бюро реєстрації (код 1.1) який містить інформацію про уже зареєстрованих виборців, а також метод для реєстрації нових виборців і присвоювання їм унікальних номерів.

```

public class RegistrationOffice {
    private final ElectionCommission electionCommission;
    private final Map<Voter, BigInteger> registrationList;

    public RegistrationOffice(ElectionCommission electionCommission) {
        this.electionCommission = electionCommission;
        registrationList = new HashMap<>();
    }

    public BigInteger registerVoter(Voter voter) {
        try {
            if (registrationList.containsKey(voter)) {
                throw new
VoterHasAlreadyRegisteredException(voter.getName());
            }
            if (!voter.canVote()) {
                throw new CantVoteException(voter.getName());
            }
            BigInteger registrationNumber = generateRegistrationNumber();
            registrationList.put(voter, registrationNumber);

electionCommission.sendRegistrationList(registrationList.values().stream().
toList());
            return registrationNumber;
        } catch (VoterHasAlreadyRegisteredException | CantVoteException
exception) {
            System.out.println(exception.getMessage());
        }
        return null;
    }

    private BigInteger generateRegistrationNumber() {
        SecureRandom random = new SecureRandom();
        BigInteger res;
        do {
            res = new BigInteger(64, random);
        } while (registrationList.containsValue(res));
        return res;
    }
}

```

Код 1.1 – Клас бюро реєстрації

Клас виборчої комісії (код 1.2) містить інформацію про кандидатів, реєстраційні номери, які вже проголосували, а також список реєстраційних номер отриманих від БР. Містить методи для отримання повідомлень від виборця, перевірки та обробки їх. Також використовується для оголошення результатів голосування.

```

public class ElectionCommission {
    private List<Candidate> candidates;
    private List<BigInteger> registrationList;
    private final Map<BigInteger, Ballot> registeredVotes;

    ...

    public void sendMessage(SignedEncryptedMessage signedEncryptedMessage)

```

```

{
    try {
        BigInteger decryptedMessage =
signedEncryptedMessage.elGamal.decrypt(signedEncryptedMessage.message);
        BigInteger id =
decryptedMessage.divide(BigInteger.TEN.pow(24));
        BigInteger regId =
decryptedMessage.mod(BigInteger.TEN.pow(24)).divide(BigInteger.TEN.pow(2));
        BigInteger ballot =
decryptedMessage.mod(BigInteger.TEN.pow(2));
        VoteMessage message = new VoteMessage(
            id,
            regId,
            new Ballot(ballot.toString())
        );
        if (signedEncryptedMessage.dsa.verify(decryptedMessage,
signedEncryptedMessage.signature)) {
            if (checkVoteMessage(message)) {
                int vote = Integer.parseInt(message.ballot.getData());
                for (Candidate candidate : candidates) {
                    if (candidate.getId() == vote) {
                        registeredVotes.put(message.regId,
message.ballot);

                        candidate.incrementVotes();
                        return;
                    }
                }
                throw new CandidateDoesNotExist("candidate" + vote);
            } else {
                throw new SignatureVerificationException();
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    private boolean checkVoteMessage(VoteMessage message) throws
        VoterIsNotRegisteredException,
        VoterHasAlreadyVotedException,
        CandidateDoesNotExist {
        if (!registrationList.contains(message.regId)) {
            throw new VoterIsNotRegisteredException();
        }
        if (registeredVotes.containsKey(message.regId)) {
            throw new VoterHasAlreadyVotedException();
        }
        if (!isCandidateInList(Integer.parseInt(message.ballot.getData())))
        {
            throw new CandidateDoesNotExist("with id " +
message.ballot.getData());
        }
        return true;
    }

    ...
}

```

Код 1.2 – Клас виборчої комісії

Для повідомлень був створений клас `VoteMessage`, який містить інформацію про ід виборця, його реєстраційний номер, а також бюлетень, який він надсилає (код 1.3).

```
public class VoteMessage {
    public final BigInteger id;
    public final BigInteger regId;
    public final Ballot ballot;

    public VoteMessage(BigInteger id, BigInteger regId, Ballot ballot) {
        this.id = id;
        this.regId = regId;
        this.ballot = ballot;
    }

    @Override
    public String toString() {
        BigInteger id_ = id.multiply(BigInteger.TEN.pow(24));
        BigInteger regId_ = regId.multiply(BigInteger.TEN.pow(2));
        return (id_.add(regId_).add(new
        BigInteger(ballot.getData()))).toString();
    }
}
```

Код 1.3 – Клас для повідомлень

Так як повідомлення повинно бути підписаним та зашифрованим, то для таких повідомлень був створений окремий клас (код 1.4), який приймав звичайне повідомлення та підписував і шифрував його за допомогою DSA та ElGamal відповідно.

```
public class SignedEncryptedMessage {
    public BigInteger[] message;
    public ElGamal elGamal;
    public DSA dsa;
    public byte[] signature;

    public SignedEncryptedMessage(VoteMessage voteMessage, ElGamal elGamal,
    DSA dsa) {
        this.elGamal = elGamal;
        this.dsa = dsa;
        BigInteger msg = new BigInteger(voteMessage.toString());
        signature = dsa.sign(msg);
        message = elGamal.encrypt(msg);
    }
}
```

Код 1.4 – Клас для зашифрованого та підписаного повідомлення

У даній лабораторній роботі використовується лише один алгоритм для шифрування та один для реалізації ЕЦП. Для шифрування було розроблено клас `ElGamal`, який дозволяв виконувати шифрування та дешифрування за допомогою алгоритму ElGamal (код 1.5).

```

public class ElGamal {

    private BigInteger p;
    private BigInteger g;
    private BigInteger x;
    private BigInteger y;

    public ElGamal() {
        generateKeyPair();
    }

    private void generateKeyPair() {
        p = BigInteger.probablePrime(512, new SecureRandom());
        g = new BigInteger("2");

        x = new BigInteger(512, new SecureRandom());

        y = g.modPow(x, p);
    }

    public BigInteger[] encrypt(BigInteger plaintext) {
        BigInteger k = new BigInteger(512, new SecureRandom());

        BigInteger a = g.modPow(k, p);

        BigInteger b = plaintext.multiply(y.modPow(k, p)).mod(p);

        return new BigInteger[]{a, b};
    }

    public BigInteger decrypt(BigInteger[] ciphertext) {
        BigInteger a = ciphertext[0];
        BigInteger b = ciphertext[1];

        BigInteger s = a.modPow(x, p);
        BigInteger sInverse = s.modInverse(p);

        return b.multiply(sInverse).mod(p);
    }
}

```

Код 1.5 – Клас шифрувальник методом ElGamal

Для підпису повідомлень було створено клас DSA для накладання та верифікації ЕЦП (код 1.6).

```

public class DSA {

    private KeyPair keyPair;

    public DSA() {
        generateKeyPair();
    }

    private void generateKeyPair() {
        try {
            KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
            keyGen.initialize(2048);
            keyPair = keyGen.generateKeyPair();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }
}

```

```

    }
}

public byte[] sign(BigInteger msg) {
    byte[] message = String.valueOf(msg).getBytes();
    try {
        Signature dsa = Signature.getInstance("SHA256withDSA");
        dsa.initSign(keyPair.getPrivate());
        dsa.update(message);
        return dsa.sign();
    } catch (NoSuchAlgorithmException | InvalidKeyException |
SignatureException e) {
        e.printStackTrace();
        return null;
    }
}

public boolean verify(BigInteger msg, byte[] signature) {
    byte[] message = String.valueOf(msg).getBytes();
    try {
        Signature verifyDsa = Signature.getInstance("SHA256withDSA");
        verifyDsa.initVerify(keyPair.getPublic());
        verifyDsa.update(message);
        return verifyDsa.verify(signature);
    } catch (NoSuchAlgorithmException | InvalidKeyException |
SignatureException e) {
        e.printStackTrace();
        return false;
    }
}
}

```

Код 1.6 – Клас для підпису методом DSA

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):

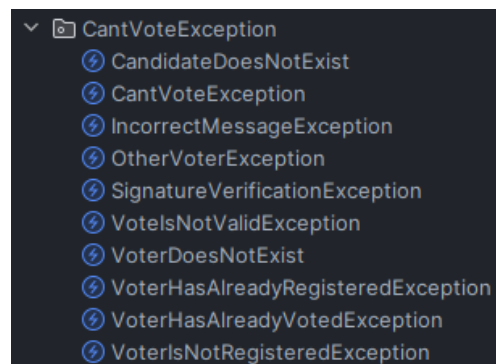


Рис. 1.1 – Список нових виключень

Демонстрація роботи алгоритму

Для демонстрації роботи було створено модель із 3 кандидатами та 10 виборцями (код 2.1). Результат запуску програми (рис. 2.1):

```
public class Main {
    public static void main(String[] args) {
        List<Integer> testVotes = new ArrayList<>(asList(0, 1, 2, 0, 2, 1,
2, 2, 0, 1));
        int count = 0;

        List<Candidate> candidates =
CandidateFactory.generateCandidates(3);
        List<Voter> voters = VoterFactory.generateVoters(10);

        ElectionCommission electionCommission = new ElectionCommission();
        RegistrationOffice registrationOffice = new
RegistrationOffice(electionCommission);

        electionCommission.setCandidates(candidates);

        for (Voter voter : voters) {
            voter.setRegId(registrationOffice.registerVoter(voter));
            voter.setId(IDGenerator.generateID());
        }

        for (Voter voter : voters) {
            voter.createKeys();
            VoteMessage message =
voter.getVoteMessage(testVotes.get(count));
            SignedEncryptedMessage signedEncryptedMessage = new
SignedEncryptedMessage(message, voter.getElGamal(), voter.getDsa());
            electionCommission.sendMessage(signedEncryptedMessage);
            count++;
        }

        electionCommission.conductElection();
        electionCommission.printVotesList();
    }
}
```

Код 2.1 – Методи для демонстрації роботи алгоритму

ELECTION RESULTS			
CANDIDATES		VOTES	
candidate0		3	
candidate1		3	
candidate2		4	

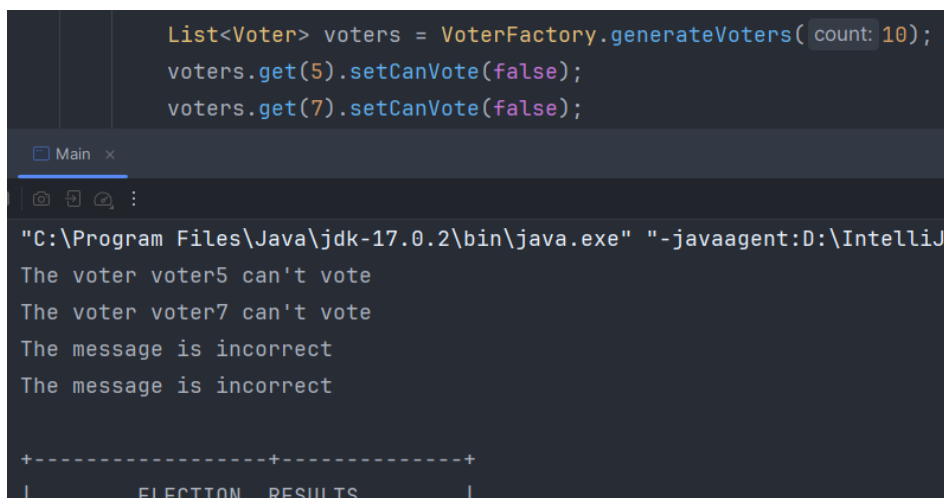
Рис. 2.1 – Результат запуску протоколу

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. Для цього заберемо у двох виборців право на голосування (рис. 3.1).



```
List<Voter> voters = VoterFactory.generateVoters(count: 10);
voters.get(5).setCanVote(false);
voters.get(7).setCanVote(false);
```

Main x

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

The voter voter5 can't vote

The voter voter7 can't vote

The message is incorrect

The message is incorrect

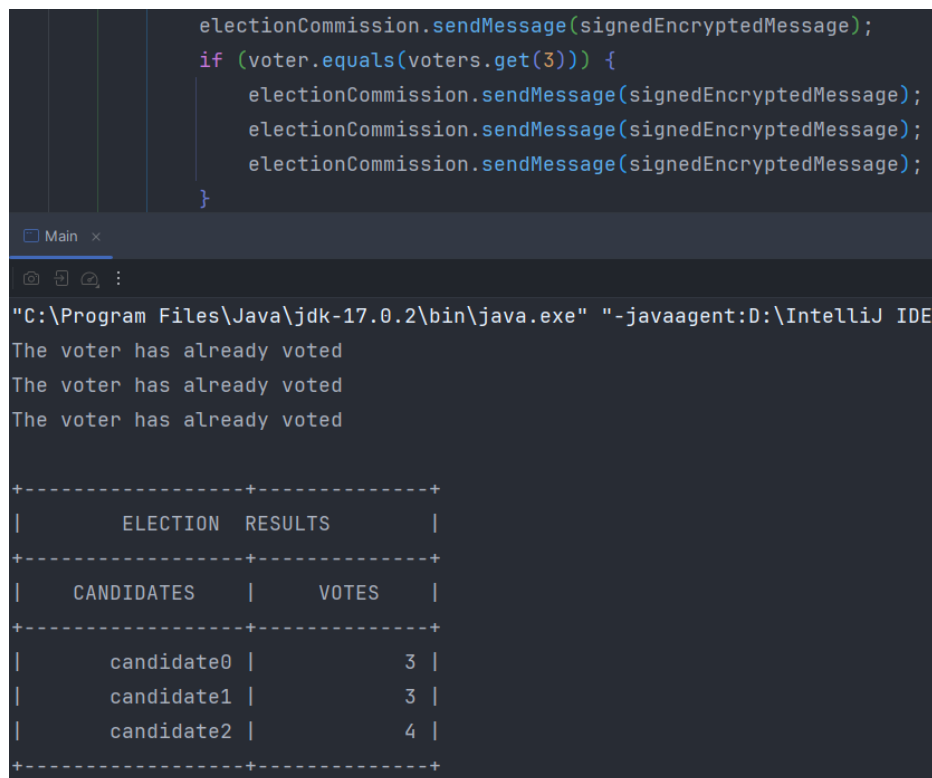
+-----+-----+

| ELECTION RESULTS |

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, а також повідомлення від них виявились неправильним, оскільки їм не видали реєстраційні номери.

Другий пункт: Перевірити чи може виборець голосувати кілька разів. Додамо в код, що якщо виборець це наприклад 4 виборець (індекс 3 у списку), то він проголосує ще 3 рази (рис. 3.2).



```
electionCommission.sendMessage(signedEncryptedMessage);
if (voter.equals(voters.get(3))) {
    electionCommission.sendMessage(signedEncryptedMessage);
    electionCommission.sendMessage(signedEncryptedMessage);
    electionCommission.sendMessage(signedEncryptedMessage);
}
```

Main x

"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaagent:D:\IntelliJ IDE\..."

The voter has already voted
The voter has already voted
The voter has already voted

ELECTION RESULTS	
CANDIDATES	VOTES
candidate0	3
candidate1	3
candidate2	4

Рис. 3.2 – Результат запуску для виборця, який голосує по декілька разів

Як можна побачити з результату, виборець, який голосує по декілька раз, був врахований лише один раз, усі інші рази комісія не врахувала його голос.

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

Алгоритм голосування з двома виборчими комісіями розроблений таким чином, що тільки той, хто знає конкретний реєстраційний номер може дізнатись за кого було віддано голос. ВК не може дізнатись хто проголосував за кого, оскільки не знає кому належить кожен із номерів. Але

наприклад БР має список усіх, хто зареєструвався та номери, які були їм присвоєні, тому вони мають повний доступ результатів голосування кожного з виборців (рис. 3.3):

COUNTED BALLOTS	
REGISTRATION ID	BALLOT
2750217013285777210	2
3266407631397156713	2
3454251921658107089	0
3507900932428520817	0
3599679881272115918	1
6667713361774992625	0
10773346000024321022	2
15096716822142059221	1
15665818142842307215	1
17868893165812637447	2

Рис. 3.3 – Список врахованих голосів, який БР може зрівняти зі своїм списком та отримати результати голосування усіх виборців.

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

Так як, для надсилання голосу потрібно мати унікальний номер, реєстраційний номер та сам бюлетень із голосом, то для надсилання голосу замість іншого виборця потрібно дізнатись лише його реєстраційний номер, а інші дані вже дописати власноруч. Такий варіант можливий якщо хтось здогадається реєстраційний номер іншого виборця, наприклад вгадавши його. Так як номери постійно генеруються випадково, то шанс того, що ми вгадаємо є занадто малим (рис. 3.4).

```
Voter chanceVoter = new Voter( name: "chance", canVote: true);
chanceVoter.setRegId(new BigInteger( numBits: 64, new SecureRandom()));
chanceVoter.setId(IDGenerator.generateID());
chanceVoter.createKeys();
VoteMessage msg = chanceVoter.getVoteMessage( candidateld: 2);
SignedEncryptedMessage signedEncryptedMsg = new SignedEncryptedMessage(msg,
electionCommission.sendMessage(signedEncryptedMsg);
```

Main x

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

The voter is not registered

ELECTION RESULTS	
CANDIDATES	VOTES
candidate0	3
candidate1	3
candidate2	4

Рис. 3.4 – Спроба вгадати чужий реєстраційний номер

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 може 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. Але дані зміни зможе побачити виборець у кінцевому списку та виявить підміну голосу для свого реєстраційного номеру

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

Так, виборець може це зробити, оскільки після завершення голосування, ВК публікує список усіх врахованих голосів, який складається із реєстраційного номеру та власне голосу, який був врахований (рис. 3.5).

COUNTED BALLOTS	
REGISTRATION ID	BALLOT
333578527083621065	1
2271707572946468420	0
3155172996132681912	2
5069379910210322072	1
5488687001463565468	2
6567906665743578528	1
9530758370324863385	2
14039857299865783443	0
15134568392595892701	0
17575400495044385311	2

Рис. 3.5 – Таблиця із кінцевими результатами голосування

Висновок

Виконавши дану лабораторну роботу, було створено протокол електронного голосування з двома виборчими комісіями. Для його реалізації було розподілено повноваження між двома частинами, а саме бюро реєстрацій та виборчою комісією. Бюро реєстрації було наділено правами реєстрації виборців та надання їм особливим реєстраційних номер. Виборча комісія, в свою чергу, була наділена правами для отримання голосів цих виборців, їх верифікації та підрахунку голосів. При чому, ВК приймає голоси тільки від тих, хто є в списку зареєстрованих від БР. Також, для шифрування та дешифрування повідомлень було використано алгоритм Ель-Гамалю, а для реалізації ЕЦП – алгоритм DSA.

Дослідження алгоритму показало, що даний алгоритм не є ідеальним, оскільки у ньому не виконуються певні умови. Даний алгоритм надає БР можливість дізнатись голоси виборців за результатами голосування, а також при спільній домовленості БР і ВК, вони можуть підробляти голоси. Але в цілому, з іншими вимогами – даний алгоритм справився.

Лістинг коду мовою Java

Main.java

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.List;

import static java.util.Arrays.asList;

public class Main {
    public static void main(String[] args) {
        List<Integer> testVotes = new ArrayList<>(asList(0, 1, 2, 0, 2, 1,
2, 2, 0, 1));
        int count = 0;

        List<Candidate> candidates =
CandidateFactory.generateCandidates(3);
        List<Voter> voters = VoterFactory.generateVoters(10);
        //      voters.get(5).setCanVote(false);
        //      voters.get(7).setCanVote(false);

        ElectionCommission electionCommission = new ElectionCommission();
        RegistrationOffice registrationOffice = new
RegistrationOffice(electionCommission);

        electionCommission.setCandidates(candidates);

        for (Voter voter : voters) {
            voter.setRegId(registrationOffice.registerVoter(voter));
            voter.setId(IDGenerator.generateID());
        }
        //
        voters.get(3).setRegId(registrationOffice.registerVoter(voters.get(3)));

        //      Voter chanceVoter = new Voter("chance", true);
        //      chanceVoter.setRegId(new BigInteger(64, new SecureRandom()));
        //      chanceVoter.setId(IDGenerator.generateID());
        //      chanceVoter.createKeys();
        //      VoteMessage msg = chanceVoter.getVoteMessage(2);
        //      SignedEncryptedMessage signedEncryptedMsg = new
SignedEncryptedMessage(msg, chanceVoter.getElGamal(),
chanceVoter.getDsa());
        //      electionCommission.sendMessage(signedEncryptedMsg);

        for (Voter voter : voters) {
            voter.createKeys();
            VoteMessage message =
voter.getVoteMessage(testVotes.get(count));
            SignedEncryptedMessage signedEncryptedMessage = new
SignedEncryptedMessage(message, voter.getElGamal(), voter.getDsa());
            electionCommission.sendMessage(signedEncryptedMessage);
            //      if (voter.equals(voters.get(3))) {
            //          electionCommission.sendMessage(signedEncryptedMessage);
            //          electionCommission.sendMessage(signedEncryptedMessage);
            //          electionCommission.sendMessage(signedEncryptedMessage);
            //      }
            count++;
        }
    }
}
```



```

        electionCommission.conductElection();
        electionCommission.printVotesList();
    }
}

```

ElectionCommission.java

```

import CantVoteException.*;

import java.math.BigInteger;
import java.util.*;

public class ElectionCommission {
    private List<Candidate> candidates;
    private List<BigInteger> registrationList;
    private final Map<BigInteger, Ballot> registeredVotes;

    public ElectionCommission() {
        registeredVotes = new TreeMap<>();
    }

    public void setCandidates(List<Candidate> candidates) {
        this.candidates = candidates;
    }

    public void sendRegistrationList(List<BigInteger> registrationList) {
        this.registrationList = new ArrayList<>(registrationList);
    }

    public void sendMessage(SignedEncryptedMessage signedEncryptedMessage)
    {
        try {
            if (signedEncryptedMessage.message == null) {
                throw new IncorrectMessageException();
            }
            BigInteger decryptedMessage =
signedEncryptedMessage.elGamal.decrypt(signedEncryptedMessage.message);
            BigInteger id =
decryptedMessage.divide(BigInteger.TEN.pow(24));
            BigInteger regId =
decryptedMessage.mod(BigInteger.TEN.pow(24)).divide(BigInteger.TEN.pow(2));
            BigInteger ballot =
decryptedMessage.mod(BigInteger.TEN.pow(2));
            VoteMessage message = new VoteMessage(
                id,
                regId,
                new Ballot(ballot.toString())
            );
            if (signedEncryptedMessage.dsa.verify(decryptedMessage,
signedEncryptedMessage.signature)) {
                if (checkVoteMessage(message)) {
                    int vote = Integer.parseInt(message.ballot.getData());
                    for (Candidate candidate : candidates) {
                        if (candidate.getId() == vote) {
                            registeredVotes.put(message.regId,
message.ballot);

                            candidate.incrementVotes();
                            return;
                        }
                    }
                }
            }
        }
    }
}

```

```

        throw new CandidateDoesNotExist("candidate" + vote);
    }
    } else {
        throw new SignatureVerificationException();
    }
} catch (Exception e) {
    System.out.println(e.getMessage());
}
}

private boolean checkVoteMessage(VoteMessage message) throws
    VoterIsNotRegisteredException,
    VoterHasAlreadyVotedException,
    CandidateDoesNotExist {
    if (!registrationList.contains(message.regId)) {
        throw new VoterIsNotRegisteredException();
    }
    if (registeredVotes.containsKey(message.regId)) {
        throw new VoterHasAlreadyVotedException();
    }
    if (!isCandidateInList(Integer.parseInt(message.ballot.getData())))
{
        throw new CandidateDoesNotExist("with id " +
message.ballot.getData());
    }
    return true;
}

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

public void printVotesList() {
    System.out.println("\n+-----+-----+");
    System.out.println("|          COUNTED   BALLOTS          |");
    System.out.println("+-----+-----+");
    System.out.println("|    REGISTRATION ID    |    BALLOT    |");
    System.out.println("+-----+-----+");
    for (BigInteger regId : registeredVotes.keySet()) {
        System.out.printf("| %21s | %12s |\n", regId,
registeredVotes.get(regId).getData());
    }
    System.out.println("+-----+-----+\n");
}

private boolean isCandidateInList(int candidateId) {
    for (Candidate candidate : candidates) {
        if (candidate.getId() == candidateId) {
            return true;
        }
    }
    return false;
}

```

```
}  
}
```

RegistrationOffice.java

```
import CantVoteException.*;  
  
import java.math.BigInteger;  
import java.security.SecureRandom;  
import java.util.HashMap;  
import java.util.Map;  
  
public class RegistrationOffice {  
    private final ElectionCommission electionCommission;  
    private final Map<Voter, BigInteger> registrationList;  
  
    public RegistrationOffice(ElectionCommission electionCommission) {  
        this.electionCommission = electionCommission;  
        registrationList = new HashMap<>();  
    }  
  
    public BigInteger registerVoter(Voter voter) {  
        try {  
            if (registrationList.containsKey(voter)) {  
                throw new  
VoterHasAlreadyRegisteredException(voter.getName());  
            }  
            if (!voter.canVote()) {  
                throw new CantVoteException(voter.getName());  
            }  
            BigInteger registrationNumber = generateRegistrationNumber();  
            registrationList.put(voter, registrationNumber);  
  
electionCommission.sendRegistrationList(registrationList.values().stream().  
toList());  
            return registrationNumber;  
        } catch (VoterHasAlreadyRegisteredException | CantVoteException  
exception) {  
            System.out.println(exception.getMessage());  
        }  
        return null;  
    }  
  
    private BigInteger generateRegistrationNumber() {  
        SecureRandom random = new SecureRandom();  
        BigInteger res;  
        do {  
            res = new BigInteger(64, random);  
        } while (registrationList.containsValue(res));  
        return res;  
    }  
}
```

Candidate.java

```
public class Candidate {  
    private static int examples = 0;  
    private final int id;  
    private final String name;
```

```

private int votesCount = 0;

public Candidate(String name) {
    this.name = name;
    id = examples;
    examples++;
}

public int getId() {
    return id;
}

public String getName() {
    return name;
}

public int getVotesCount() {
    return votesCount;
}

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

```

Voter.java

```

import java.math.BigInteger;

public class Voter {
    private boolean canVote;
    private BigInteger id;
    private BigInteger regId;
    private final String name;
    private ElGamal elGamal;
    private DSA dsa;

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

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

    public void setRegId(BigInteger regId) {
        if (regId != null)
            this.regId = regId;
    }

    public String getName() {
        return name;
    }

    public BigInteger getId() {
        return id;
    }

    public boolean canVote() {

```

```

        return canVote;
    }

    public void setCanVote(boolean canVote) {
        this.canVote = canVote;
    }

    public VoteMessage getVoteMessage(int candidateId) {
        return new VoteMessage(id, regId, new
Ballot(String.valueOf(candidateId)));
    }

    public void createKeys() {
        elGamal = new ElGamal();
        dsa = new DSA();
    }

    public ElGamal getElGamal() {
        return elGamal;
    }

    public DSA getDsa() {
        return dsa;
    }
}

```

Ballot.java

```

public class Ballot {
    private String data;

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

    public String getData() {
        return data;
    }

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

```

CandidateFactory.java

```

import java.util.ArrayList;
import java.util.List;

public class CandidateFactory {
    public static List<Candidate> generateCandidates(int count) {
        List<Candidate> candidates = new ArrayList<>();
        for (int i = 0; i < count; i++) {
            candidates.add(CandidateFactory.createCandidate("candidate" +
i));
        }
        return candidates;
    }
}

```

```
public static Candidate createCandidate(String name) {  
    return new Candidate(name);  
}  
}
```

VoterFactory.java

```
import java.util.ArrayList;  
import java.util.List;  
  
public class VoterFactory {  
    public static List<Voter> generateVoters(int count) {  
        List<Voter> voters = new ArrayList<>();  
        for (int i = 0; i < count; i++) {  
            voters.add(VoterFactory.createVoter("voter" + i, true));  
        }  
        return voters;  
    }  
  
    public static Voter createVoter(String name, boolean canVote) {  
        return new Voter(name, canVote);  
    }  
}
```

Ballot.java

```
public class Ballot {  
    private String data;  
  
    public Ballot(String data) {  
        this.data = data;  
    }  
  
    public String getData() {  
        return data;  
    }  
  
    public void setData(String data) {  
        this.data = data;  
    }  
}
```

VoteMessage.java

```
import java.math.BigInteger;  
  
public class VoteMessage {  
    public final BigInteger id;  
    public final BigInteger regId;  
    public final Ballot ballot;  
  
    public VoteMessage(BigInteger id, BigInteger regId, Ballot ballot) {  
        this.id = id;  
        this.regId = regId;  
        this.ballot = ballot;  
    }  
}
```

```

@Override
public String toString() {
    try {
        BigInteger id_ = id.multiply(BigInteger.TEN.pow(24));
        BigInteger regId_ = regId.multiply(BigInteger.TEN.pow(2));
        return (id_.add(regId_).add(new
BigInteger(ballot.getData()))).toString();
    } catch (Exception ignored) {}
    return "Invalid";
}
}

```

SignedEncryptedMessage.java

```

import java.math.BigInteger;

import CantVoteException.IncorrectMessageException;

public class SignedEncryptedMessage {
    public BigInteger[] message;
    public ElGamal elGamal;
    public DSA dsa;
    public byte[] signature;

    public SignedEncryptedMessage(VoteMessage voteMessage, ElGamal elGamal,
DSA dsa) {
        this.elGamal = elGamal;
        this.dsa = dsa;
        String msgData = voteMessage.toString();
        if (!msgData.equals("Invalid")) {
            BigInteger msg = new BigInteger(msgData);
            signature = dsa.sign(msg);
            message = elGamal.encrypt(msg);
        }
    }
}

```

ElGamal.java

```

import java.math.BigInteger;
import java.security.SecureRandom;

public class ElGamal {

    private BigInteger p;
    private BigInteger g;
    private BigInteger x;
    private BigInteger y;

    public ElGamal() {
        generateKeyPair();
    }

    private void generateKeyPair() {
        p = BigInteger.probablePrime(512, new SecureRandom());
        g = new BigInteger("2");

        x = new BigInteger(512, new SecureRandom());
    }
}

```

```

        y = g.modPow(x, p);
    }

    public BigInteger[] encrypt(BigInteger plaintext) {
        BigInteger k = new BigInteger(512, new SecureRandom());

        BigInteger a = g.modPow(k, p);

        BigInteger b = plaintext.multiply(y.modPow(k, p)).mod(p);

        return new BigInteger[]{a, b};
    }

    public BigInteger decrypt(BigInteger[] ciphertext) {
        BigInteger a = ciphertext[0];
        BigInteger b = ciphertext[1];

        BigInteger s = a.modPow(x, p);
        BigInteger sInverse = s.modInverse(p);

        return b.multiply(sInverse).mod(p);
    }
}

```

DSA.java

```

import java.math.BigInteger;
import java.security.*;
import java.security.SignatureException;

public class DSA {

    private KeyPair keyPair;

    public DSA() {
        generateKeyPair();
    }

    private void generateKeyPair() {
        try {
            KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
            keyGen.initialize(2048);
            keyPair = keyGen.generateKeyPair();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }

    public byte[] sign(BigInteger msg) {
        byte[] message = String.valueOf(msg).getBytes();
        try {
            Signature dsa = Signature.getInstance("SHA256withDSA");
            dsa.initSign(keyPair.getPrivate());
            dsa.update(message);
            return dsa.sign();
        } catch (NoSuchAlgorithmException | InvalidKeyException |
SignatureException e) {
            e.printStackTrace();
            return null;
        }
    }
}

```



```
    }

    public boolean verify(BigInteger msg, byte[] signature) {
        byte[] message = String.valueOf(msg).getBytes();
        try {
            Signature verifyDsa = Signature.getInstance("SHA256withDSA");
            verifyDsa.initVerify(keyPair.getPublic());
            verifyDsa.update(message);
            return verifyDsa.verify(signature);
        } catch (NoSuchAlgorithmException | InvalidKeyException |
SignatureException e) {
            e.printStackTrace();
            return false;
        }
    }
}
```