

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

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Лабораторна робота №6

з дисципліни

«Протоколи та алгоритми електронного голосування»

Виконав:

студент 4 курсу

групи ІІІ-02

Середюк Валентин

Київ – 2023

Лабораторна робота №6

Тема: Протокол Е-голосування без підтвердження

Мета: Дослідити протокол Е-голосування без підтвердження

Завдання

Змодельовати протокол Е-голосування без підтвердження будь-якою мовою програмування та провести його дослідження. Для кодування повідомлень використовувати метод Ель-Гамала, для кодування Е-бюлетеня використовувати метод BBS.

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 без підтвердження.

Для реалізації даного алгоритму було поетапно виконано усі пункти із протоколу голосування без підтвердження. Розглянемо кожен етап окремо.

Спочатку було створено певну кількість номерів, які потім були вислані до виборчої комісії (код 1.1).

```
public void generateId(int amount) {  
    Random random = new Random();  
    for (int i = 0; i < amount; i++) {  
        voterIds.add(random.nextInt(10000));  
    }  
    tokens = electionCommission.getTokens(voterIds);  
}
```

Код 1.1 – Створення номерів для виборців

Далі ВК створює ключі закриті та відкриті, при чому ключі для Ель-Гамала однакові для усіх, а ось ключі для BBS у кожного індивідуальні. ВК запам'ятовує приватні ключі кожного з виборців та генерує токени, які містять номер та відкриті ключі (код 1.2).

```
public ElectionCommission() {  
    voterPrivateKeys = new HashMap<>();  
    ballots = new ArrayList<>();  
    elGamalKeyPair = KeyPairGenerator.generateElGamalKeyPair();  
}  
  
public List<Token> getTokens(List<Integer> voterIds) {  
    BBSKeyPair bbsKeyPair = KeyPairGenerator.generateBBSKeyPair();  
    for (Integer id : voterIds)  
        voterPrivateKeys.put(id, bbsKeyPair.privateKey);  
  
    List<Token> tokens = new ArrayList<>();  
    for (int i = 0; i < voterIds.size(); i++) {  
        Token token = new Token(voterIds.get(i), elGamalKeyPair.publicKey,  
                                bbsKeyPair.publicKey);  
        tokens.add(token);  
    }  
  
    return tokens;  
}
```

Код 1.2 – Створення ключів та токенів

Процес створення ключів наведений на коді 1.3.

```
public class KeyPairGenerator {  
    public static BBSKeyPair generateBBSKeyPair() {
```

```

        BigInteger[] pq = generatePrime();
        BigInteger n = pq[0].multiply(pq[1]);
        return new BBSKeyPair(new BBSPrivateKey(pq[0], pq[1]), new
BBSPublicKey(n));
    }

    public static ElGamalKeyPair generateElGamalKeyPair() {
        BigInteger[] elGamal = generateElGamal();
        return new ElGamalKeyPair(new ElGamalPrivateKey(elGamal[3],
elGamal[0]), new ElGamalPublicKey(elGamal[0], elGamal[1], elGamal[2]));
    }

    private static BigInteger[] generatePrime() {
        var random = new Random();

        var p = BigInteger.probablePrime(64, random);
        var q = BigInteger.probablePrime(64, random);

        while (p.compareTo(q) == 0) {
            p = BigInteger.probablePrime(64, random);
            q = BigInteger.probablePrime(64, random);
        }

        return new BigInteger[]{p, q};
    }

    private static BigInteger[] generateElGamal() {
        BigInteger p = BigInteger.probablePrime(512, new SecureRandom());
        BigInteger g = new BigInteger("2");

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

        BigInteger y = g.modPow(x, p);
        return new BigInteger[]{p, g, y, x};
    }
}

```

Код 1.3 – Створення пар ключів двох типів

Далі виборець робить запит бо бюро реєстрація задля отримання даних для входу у програму та токєну (код 1.4).

```

public void register(Voter voter) {
    if (!voter.canVote) {
        System.out.println("The voter can't vote");
        return;
    }
    if (voterData.containsKey(voter)) {
        System.out.println("The voter has already registered");
        return;
    }
    Random random = new Random();
    String login = String.valueOf(random.nextInt());
    String password = String.valueOf(random.nextInt());
    Credentials credentials = new Credentials(login, password);
    voterData.put(voter, new VoterData(credentials,
tokens.get(tokenIndex).voterId));
    voter.setCredentialsAndToken(credentials, tokens.get(tokenIndex));
    tokenIndex++;
}

```

Код 1.4 – Процес реєстрації виборця та отримання ним даних для входу

Отримавши дані, виборець встановлює застосунок та входить по даним, які отримав у БР. Також він встановлює токен, який отримав, якщо авторизація пройшла успішно (код 1.5 та 1.6)

```
public void install() {
    if (credentials == null) {
        System.out.println("The voter is not registered");
        return;
    }
    application = new Application(credentials, token);
}
```

Код 1.5 – Процес встановлення застосунку

```
public Application(Credentials credentials, Token token) {
    if (registrationOffice.checkCredentials(credentials)) {
        this.token = token;
    } else {
        System.out.println("Incorrect credentials");
    }
}
```

Код 1.6 – Процес авторизації

При успішній авторизації, виборець вибирає кандидата, за якого хоче проголосувати та створює Е-бюлетень, який потім шифрує методом BBS. Потім створюється повідомлення із даними шифрування, яке шифрується вже алгоритмом Ель-Гамаль та надсилається зашифроване повідомлення до ВК (код 1.7).

```
public void vote(Candidate candidate) {
    BBSResult encryptResult =
    BBS.encrypt(String.valueOf(candidate.getId()), token.bbsPublicKey);
    VoteMessage voteMessage = new VoteMessage(encryptResult,
    token.voterId);
    BigInteger messageToEncrypt = voteMessage.getMessage();
    BigInteger[] encrypted = ElGamal.encrypt(messageToEncrypt,
    token.elGamalPublicKey);
    electionCommission.sendEncryptedBallot(encrypted);
}
```

Код 1.7 – Процес шифрування бюлетеня та надсилання його

При отриманні зашифрованого повідомлення, ВК починає розшифровувати його, використовуючи приватний ключ Ель-Гамаль. При успішному розшифруванні та перевірки бюлетеня, голос враховується за певного кандидата (код 1.8).

```
public void sendEncryptedBallot(BigInteger[] encrypted) {
    VoteMessage voteMessage = getVoteMessageFromEncrypted(encrypted);
    String ballotData = BBS.decrypt(voteMessage.encryptedMessage,
```

```

voteMessage.x0, voterPrivateKeys.get(voteMessage.id));
    Candidate candidate = candidates.stream().filter(c -> c.getId() ==
Integer.parseInt(ballotData)).findFirst().orElse(null);
    if (candidate != null) {
        if (!isVoterVotedBefore(voteMessage.id)) {
            ballots.add(new Ballot(voteMessage.encryptedMessage,
voteMessage.id));
            candidate.incrementVotes();
        } else {
            System.out.printf("The voter with id %d has already voted\n",
voteMessage.id);
        }
    } else {
        System.out.println("The candidate doesn't exist");
    }
}

private VoteMessage getVoteMessageFromEncrypted(BigInteger[] encrypted) {
    BigInteger decryptedInteger = ElGamal.decrypt(encrypted,
elGamalKeyPair.privateKey);
    BigInteger id = decryptedInteger.mod(BigInteger.TEN.pow(10));
    decryptedInteger = decryptedInteger.divide(BigInteger.TEN.pow(10));
    BigInteger x0 = decryptedInteger.mod(BigInteger.TEN.pow(42));
    decryptedInteger = decryptedInteger.divide(BigInteger.TEN.pow(42));
    int length = (int) Math.ceil((double)
String.valueOf(decryptedInteger).length() / 2);
    BigInteger[] encryptedMessage = new BigInteger[length];
    for (int i = length - 1; i >= 0; i--) {
        encryptedMessage[i] = decryptedInteger.mod(new BigInteger("100"));
        decryptedInteger = decryptedInteger.divide(new BigInteger("100"));
    }
    return new VoteMessage(new BBSResult(encryptedMessage, x0),
id.intValue());
}

```

Код 1.8 – Процес розшифрування повідомлення та зарахування голосу
виборця

Надалі залишається лише підвести підсумки та вивести результат
голосування.

Демонстрація роботи алгоритму

Для демонстрації роботи було створено модель із 2 кандидатами та 4 виборцями (код 2.1). Результат запуску програми (рис. 2.1):

```
public class Main {
    public static void main(String[] args) {
        List<Candidate> candidates = DataFactory.getCandidates(4);
        List<Voter> voters = DataFactory.getVoters(10);

        ElectionCommission electionCommission = new ElectionCommission();
        electionCommission.setCandidates(candidates);
        RegistrationOffice registrationOffice = new
RegistrationOffice(electionCommission);
        registrationOffice.generateId(voters.size());
        voters.forEach(registrationOffice::register);

        Application.electionCommission = electionCommission;
        Application.registrationOffice = registrationOffice;

        for (Voter voter : voters) {
            voter.install();
            voter.vote(candidates.get((int) (Math.random() *
candidates.size())));
        }

        electionCommission.printResult();
    }
}
```

Код 2.1 – Методи для демонстрації роботи алгоритму

ELECTION RESULTS		
CANDIDATES	VOTES	
Candidate 0	1	
Candidate 1	3	
Candidate 2	2	
Candidate 3	4	

Рис. 2.1 – Результат запуску протоколу

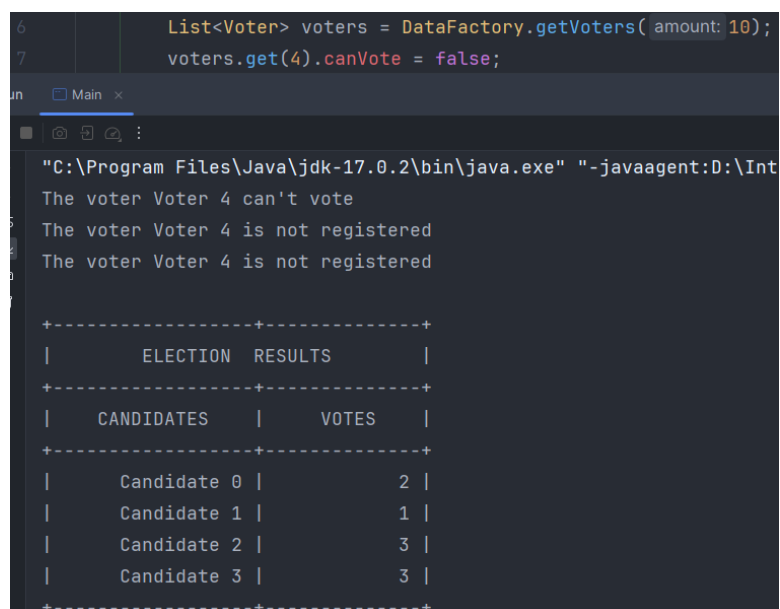
Дослідження протоколу

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.**

Спробуємо замінити одного виборця на виборця без можливості голосувати (рис. 3.1).



```
6 List<Voter> voters = DataFactory.getVoters( amount: 10);
7 voters.get(4).canVote = false;
```

Terminal Output:

```
"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaagent:D:\Int
The voter Voter 4 can't vote
The voter Voter 4 is not registered
The voter Voter 4 is not registered
```

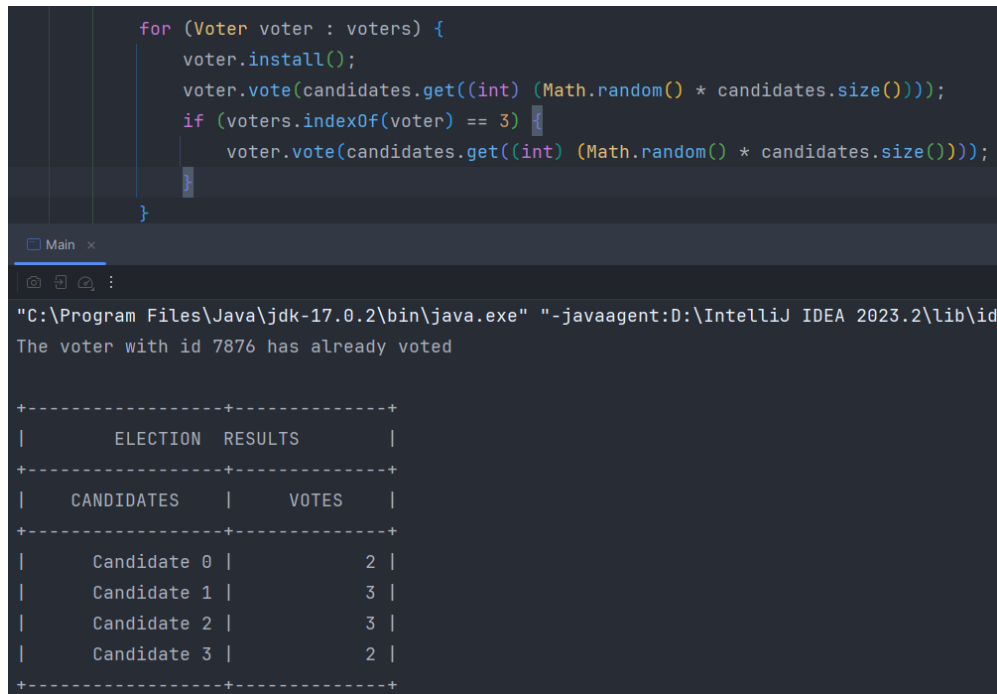
ELECTION RESULTS	
CANDIDATES	VOTES
Candidate 0	2
Candidate 1	1
Candidate 2	3
Candidate 3	3

Рисунок 3.1 – Голосування виборця, який не має права

Як можна побачити із результату, виборець не був врахований у результат голосування.

Другий пункт: **Перевірити чи може виборець голосувати кілька разів.**

Спробуємо відіслати від виборця ще раз його голос, але за іншого кандидата (рис. 3.2).



```
for (Voter voter : voters) {  
    voter.install();  
    voter.vote(candidates.get((int) (Math.random() * candidates.size())));  
    if (voters.indexOf(voter) == 3) {  
        voter.vote(candidates.get((int) (Math.random() * candidates.size())));  
    }  
}
```

Main x

"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaagent:D:\IntelliJ IDEA 2023.2\lib\id
The voter with id 7876 has already voted

ELECTION RESULTS	
CANDIDATES	VOTES
Candidate 0	2
Candidate 1	3
Candidate 2	3
Candidate 3	2

Рисунок 3.2 – Спроба проголосувати двічі

Як можна побачити, голос не був зарахований, ВК відхилила спробу надіслати ще раз повідомлення із бюлетенем.

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

Протокол голосування без підтвердження розроблений так, що ВК отримує тільки номер виборця, а ось кому він привласнений вона не знає, тому і дізнатись хто за кого проголосував не зможе. Ніхто інший знати не може, хто за когось проголосував, оскільки ця інформація більше ніде не виставляється.

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

Так як, для голосування, потрібно вводити свої логін та пароль, а до них прив'язаний певний токен із номером та ключами, то для надсилання голосу іншою особою, їй потрібно дізнатись логін та пароль від застосунку іншої людини, а також її токен. Якщо ж людина нікому не буде повідомляти дану інформацію, то ніхто не зможе проголосувати за неї.

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

У даному протоколі усі виборці повністю довіряються ВК, яка не виставляє ніяких доказів на відповідність бюлетенів виборців. Якщо ВК захтітиме змінити голоси, то може зробити це без проблем.

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

Ні, виборець не може це зробити, оскільки після завершення голосування, ВК не публікує список ніяких бюлетенів, лише сам результат голосування із голосів для кожного кандидата.

Висновок

Виконавши дану лабораторну роботу, було створено протокол електронного голосування без підтвердження.

Для його реалізації, було розроблено систему, яка дозволяла створювати бюлетені після отримання реєстраційних даних та токенів від БР, які використовуються для шифрування бюлетенів, а також проводити процес розшифрування однією ВК без втручання інших.

Дослідження алгоритму показало, що даний алгоритм гарно справляється із першими 4 пунктами дослідження. Але недоліки даного алгоритму є занадто важливими для чесного голосування. У даному випадку ВК має усі права, та може змінювати бюлетені як їй заманеться. Тому потрібно лише надіятись, що ВК виявиться доброчесною.

Лістинг коду мовою Java

Main.java

```
public class Main {
    public static void main(String[] args) {
        List<Candidate> candidates = DataFactory.getCandidates(4);
        List<Voter> voters = DataFactory.getVoters(10);
        // voters.get(4).canVote = false;

        ElectionCommission electionCommission = new ElectionCommission();
        electionCommission.setCandidates(candidates);
        RegistrationOffice registrationOffice = new
RegistrationOffice(electionCommission);
        registrationOffice.generateId(voters.size());
        voters.forEach(registrationOffice::register);

        Application.electionCommission = electionCommission;
        Application.registrationOffice = registrationOffice;

        for (Voter voter : voters) {
            voter.install();
            voter.vote(candidates.get((int) (Math.random() *
candidates.size())));
            if (voters.indexOf(voter) == 3) {
                voter.vote(candidates.get((int) (Math.random() *
candidates.size())));
            }
        }

        electionCommission.printResult();
    }
}
```

RegistrationOffice.java

```
public class RegistrationOffice {
    private ElectionCommission electionCommission;
    private List<Integer> voterIds;
    private List<Token> tokens;
    private Map<Voter, VoterData> voterData;
    private static int tokenIndex = 0;

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

    public void generateId(int amount) {
        Random random = new Random();
        for (int i = 0; i < amount; i++) {
            voterIds.add(random.nextInt(10000));
        }
        tokens = electionCommission.getTokens(voterIds);
    }

    public void register(Voter voter) {
        if (!voter.canVote) {
            System.out.printf("The voter %s can't vote\n",

```

```

voter.getName());
    return;
}
if (voterData.containsKey(voter)) {
    System.out.printf("The voter %s has already registered\n",
voter.getName());
    return;
}
Random random = new Random();
String login = String.valueOf(random.nextInt());
String password = String.valueOf(random.nextInt());
Credentials credentials = new Credentials(login, password);
voterData.put(voter, new VoterData(credentials,
tokens.get(tokenIndex).voterId));
voter.setCredentialsAndToken(credentials, tokens.get(tokenIndex));
tokenIndex++;
}

public boolean checkCredentials(Credentials credentials) {
    for (VoterData vd : voterData.values()) {
        if (vd.checkCredentials(credentials)) {
            return true;
        }
    }
    return false;
}
}

```

ElectionCommission.java

```

public class ElectionCommission {
    private List<Candidate> candidates;
    private Map<Integer, BBSPrivateKey> voterPrivateKeys;
    private List<Ballot> ballots;
    private ElGamalKeyPair elGamalKeyPair;

    public ElectionCommission() {
        voterPrivateKeys = new HashMap<>();
        ballots = new ArrayList<>();
        elGamalKeyPair = KeyPairGenerator.generateElGamalKeyPair();
    }

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

    public List<Token> getTokens(List<Integer> voterIds) {
        BBSKeyPair bbsKeyPair = KeyPairGenerator.generateBBSKeyPair();
        for (Integer id : voterIds)
            voterPrivateKeys.put(id, bbsKeyPair.privateKey);

        List<Token> tokens = new ArrayList<>();
        for (int i = 0; i < voterIds.size(); i++) {
            Token token = new Token(voterIds.get(i),
elGamalKeyPair.publicKey, bbsKeyPair.publicKey);
            tokens.add(token);
        }

        return tokens;
    }
}

```

```

    public void sendEncryptedBallot(BigInteger[] encrypted) {
        VoteMessage voteMessage = getVoteMessageFromEncrypted(encrypted);
        String ballotData = BBS.decrypt(voteMessage.encryptedMessage,
voteMessage.x0, voterPrivateKeys.get(voteMessage.id));
        Candidate candidate = candidates.stream().filter(c -> c.getId() ==
Integer.parseInt(ballotData)).findFirst().orElse(null);
        if (candidate != null) {
            if (!isVoterVotedBefore(voteMessage.id)) {
                ballots.add(new Ballot(voteMessage.encryptedMessage,
voteMessage.id));
                candidate.incrementVotes();
            } else {
                System.out.printf("The voter with id %d has already
voted\n", voteMessage.id);
            }
        } else {
            System.out.println("The candidate doesn't exist");
        }
    }

    private VoteMessage getVoteMessageFromEncrypted(BigInteger[] encrypted)
    {
        BigInteger decryptedInteger = ElGamal.decrypt(encrypted,
elGamalKeyPair.privateKey);
        BigInteger id = decryptedInteger.mod(BigInteger.TEN.pow(10));
        decryptedInteger = decryptedInteger.divide(BigInteger.TEN.pow(10));
        BigInteger x0 = decryptedInteger.mod(BigInteger.TEN.pow(42));
        decryptedInteger = decryptedInteger.divide(BigInteger.TEN.pow(42));
        int length = (int) Math.ceil((double)
String.valueOf(decryptedInteger).length() / 2);
        BigInteger[] encryptedMessage = new BigInteger[length];
        for (int i = length - 1; i >= 0; i--) {
            encryptedMessage[i] = decryptedInteger.mod(new
BigInteger("100"));
            decryptedInteger = decryptedInteger.divide(new
BigInteger("100"));
        }
        return new VoteMessage(new BBSResult(encryptedMessage, x0),
id.intValue());
    }

    private boolean isVoterVotedBefore(int voterId) {
        for (Ballot ballot : ballots) {
            if (ballot.voterId == voterId)
                return true;
        }
        return false;
    }

    public void printResult() {
        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.getVotes());
        }
        System.out.println("+-----+-----+\n");
    }

```

```
}  
}
```

Candidate.java

```
public class Candidate {  
    private static int examples = 0;  
    private int id;  
    private String name;  
    private int votes = 0;  
  
    public Candidate(String name) {  
        this.name = name;  
        id = examples;  
        examples++;  
    }  
  
    public int getId() {  
        return id;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void incrementVotes() {  
        votes++;  
    }  
  
    public int getVotes() {  
        return votes;  
    }  
  
    @Override  
    public String toString() {  
        return name;  
    }  
}
```

Voter.java

```
public class Voter {  
    private String name;  
    private Credentials credentials;  
    private Token token;  
    private Application application;  
  
    public boolean canVote = true;  
  
    public Voter(String name) {  
        this.name = name;  
    }  
  
    public void setCredentialsAndToken(Credentials credentials, Token  
token) {  
        this.credentials = credentials;  
        this.token = token;  
    }  
}
```

```

public void install() {
    if (credentials == null) {
        System.out.printf("The voter %s is not registered\n", name);
        return;
    }
    application = new Application(credentials, token);
}

public void vote(Candidate candidate) {
    if (credentials == null) {
        System.out.printf("The voter %s is not registered\n", name);
        return;
    }
    if (application == null) {
        System.out.printf("The voter %s is not authorized\n", name);
        return;
    }
    application.vote(candidate);
}

public String getName() {
    return name;
}
}

```

DataFactory.java

```

public class DataFactory {
    public static List<Candidate> getCandidates(int amount) {
        List<Candidate> candidates = new ArrayList<>();
        for (int i = 0; i < amount; i++) {
            candidates.add(new Candidate("Candidate " + i));
        }
        return candidates;
    }

    public static List<Voter> getVoters(int amount) {
        List<Voter> voters = new ArrayList<>();
        for (int i = 0; i < amount; i++) {
            voters.add(new Voter("Voter " + i));
        }
        return voters;
    }
}

```

Token.java

```

public class Token {
    public final int voterId;
    public final ElGamalPublicKey elGamalPublicKey;
    public final BBSPublicKey bbsPublicKey;

    public Token(int voterId, ElGamalPublicKey elGamalPublic, BBSPublicKey
bbsPublic) {
        this.voterId = voterId;
        this.elGamalPublicKey = elGamalPublic;
        this.bbsPublicKey = bbsPublic;
    }
}

```



```

@Override
public String toString() {
    return "Token{" +
        "voterId=" + voterId +
        ", elGamalPublicKey=" + elGamalPublicKey +
        ", bbsPublicKey=" + bbsPublicKey +
        "}\n";
}
}

```

VoterData.java

```

public class VoterData {
    public Credentials credentials;
    public int tokenId;

    public VoterData(Credentials credentials, int tokenId) {
        this.credentials = credentials;
        this.tokenId = tokenId;
    }

    public boolean checkCredentials(Credentials credentials) {
        if (credentials == null)
            return false;
        return credentials.login.equals(this.credentials.login)
            && credentials.password.equals(this.credentials.password);
    }
}

```

VoteMessage.java

```

public class VoteMessage {
    public BigInteger[] encryptedMessage;
    public BigInteger x0;
    public int id;

    public VoteMessage(BBSResult bbsResult, int id) {
        this.encryptedMessage = bbsResult.encryptedMessage;
        this.x0 = bbsResult.x0;
        this.id = id;
    }

    public BigInteger getMessage() {
        BigInteger res = new BigInteger("0");
        for (BigInteger bigInteger : encryptedMessage) {
            res = res.add(bigInteger);
            res = res.multiply(new BigInteger("100"));
        }
        res = res.multiply(BigInteger.TEN.pow(40));
        res = res.add(x0);
        res = res.multiply(BigInteger.TEN.pow(10));
        res = res.add(BigInteger.valueOf(id));
        return res;
    }
}

```

Ballot.java

```

public class Ballot {
    public BigInteger[] encryptedMessage;
    public int voterId;

    public Ballot(BigInteger[] encryptedMessage, int voterId) {
        this.encryptedMessage = encryptedMessage;
        this.voterId = voterId;
    }
}

```

Application.java

```

public class Application {
    public static ElectionCommission electionCommission;
    public static RegistrationOffice registrationOffice;
    private Token token;

    public Application(Credentials credentials, Token token) {
        if (registrationOffice.checkCredentials(credentials)) {
            this.token = token;
        } else {
            System.out.println("Incorrect credentials");
        }
    }

    public void vote(Candidate candidate) {
        BBSResult encryptResult =
BBS.encrypt(String.valueOf(candidate.getId()), token.bbsPublicKey);
        VoteMessage voteMessage = new VoteMessage(encryptResult,
token.voterId);
        BigInteger messageToEncrypt = voteMessage.getMessage();
        BigInteger[] encrypted = ElGamal.encrypt(messageToEncrypt,
token.elGamalPublicKey);
        electionCommission.sendEncryptedBallot(encrypted);
    }
}

```

KeyPairGenerator.java

```

public class KeyPairGenerator {
    public static BBSKeyPair generateBBSKeyPair() {
        BigInteger[] pq = generatePrime();
        BigInteger n = pq[0].multiply(pq[1]);
        return new BBSKeyPair(new BBSPrivateKey(pq[0], pq[1]), new
BBSPublicKey(n));
    }

    public static ElGamalKeyPair generateElGamalKeyPair() {
        BigInteger[] elGamal = generateElGamal();
        return new ElGamalKeyPair(new ElGamalPrivateKey(elGamal[3],
elGamal[0]), new ElGamalPublicKey(elGamal[0], elGamal[1], elGamal[2]));
    }

    private static BigInteger[] generatePrime() {
        var random = new Random();

        var p = BigInteger.probablePrime(64, random);
        var q = BigInteger.probablePrime(64, random);
    }
}

```

```

        while (p.compareTo(q) == 0) {
            p = BigInteger.probablePrime(64, random);
            q = BigInteger.probablePrime(64, random);
        }

        return new BigInteger[]{p, q};
    }

    private static BigInteger[] generateElGamal() {
        BigInteger p = BigInteger.probablePrime(512, new SecureRandom());
        BigInteger g = new BigInteger("2");

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

        BigInteger y = g.modPow(x, p);
        return new BigInteger[]{p, g, y, x};
    }
}

```

BBS.java

```

public class BBS {
    public static BigInteger generateX(BigInteger n) {
        BigInteger x;
        do {
            x = getRandomCoprime(n);
        } while (gcd(x, n).compareTo(BigInteger.ZERO) == 0);
        return x;
    }

    public static BigInteger getRandomCoprime(BigInteger n) {
        BigInteger rand;
        do {
            rand = new BigInteger(n.bitLength(), new SecureRandom());
        } while (gcd(rand, n).compareTo(BigInteger.ONE) != 0 ||
rand.compareTo(n) > 0);
        return rand;
    }

    public static BigInteger gcd(BigInteger a, BigInteger b) {
        if (b.compareTo(BigInteger.ZERO) == 0) {
            return a;
        } else {
            return gcd(b, a.mod(b));
        }
    }

    public static BigInteger squareModN(BigInteger a, BigInteger n) {
        return a.modPow(BigInteger.TWO, n);
    }

    public static BigInteger getBit(BigInteger a) {
        return a.testBit(0) ? BigInteger.ONE : BigInteger.ZERO;
    }

    public static BigInteger[] stringToBits(String str) {
        BigInteger[] bits = new BigInteger[str.length()];
        for (int i = 0; i < str.length(); i++) {
            bits[i] = BigInteger.valueOf(str.charAt(i));
        }
    }
}

```

```

        return bits;
    }

    public static BBSResult encrypt(String str, BBSPublicKey publicKey) {
        BigInteger n = publicKey.n;
        BigInteger x = generateX(n);
        BigInteger x0 = squareModN(x, n);
        BigInteger[] bits = stringToBits(str);
        BigInteger xi = x0;
        BigInteger[] encryptedBits = new BigInteger[bits.length];
        for (int i = 0; i < bits.length; i++) {
            BigInteger mi = bits[i];
            xi = squareModN(xi, n);
            BigInteger bi = getBit(xi);
            encryptedBits[i] = mi.xor(bi);
        }
        return new BBSResult(encryptedBits, x0);
    }

    public static String decrypt(BigInteger[] bits, BigInteger x0,
BBSPublicKey privateKey) {
        BigInteger p = privateKey.p;
        BigInteger q = privateKey.q;
        BigInteger n = p.multiply(q);
        List<BigInteger> result = new ArrayList<BigInteger>();
        BigInteger b0 = getBit(x0);
        BigInteger xi = x0;
        for (BigInteger mi : bits) {
            xi = squareModN(xi, n);
            BigInteger bi = getBit(xi);
            result.add(mi.xor(bi));
        }
        StringBuilder sb = new StringBuilder();
        for (BigInteger bigInteger : result) {
            sb.append((char) bigInteger.intValue());
        }
        return sb.toString();
    }
}

```

BBSKeyPair.java

```

public class BBSKeyPair {
    public final BBSPublicKey privateKey;
    public final BBSPublicKey publicKey;

    public BBSKeyPair(BBSPublicKey privateKey, BBSPublicKey publicKey) {
        this.privateKey = privateKey;
        this.publicKey = publicKey;
    }
}

```

BBSPublicKey.java

```

public class BBSPublicKey {
    public BigInteger n;

    public BBSPublicKey(BigInteger n) {
        this.n = n;
    }
}

```

```
}  
}
```

BBSPrivateKey.java

```
public class BBSPrivateKey {  
    public BigInteger p;  
    public BigInteger q;  
  
    public BBSPrivateKey(BigInteger p, BigInteger q) {  
        this.p = p;  
        this.q = q;  
    }  
}
```

ElGamal.java

```
public class ElGamal {  
    public static BigInteger[] encrypt(BigInteger plaintext,  
    ElGamalPublicKey publicKey) {  
        BigInteger g = publicKey.g;  
        BigInteger p = publicKey.p;  
        BigInteger y = publicKey.y;  
  
        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 static BigInteger decrypt(BigInteger[] ciphertext,  
    ElGamalPrivateKey privateKey) {  
        BigInteger p = privateKey.p;  
        BigInteger x = privateKey.x;  
  
        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);  
    }  
}
```

ElGamalKeyPair.java

```
public class ElGamalKeyPair {  
    public final ElGamalPrivateKey privateKey;  
    public final ElGamalPublicKey publicKey;  
  
    public ElGamalKeyPair(ElGamalPrivateKey privateKey, ElGamalPublicKey  
    publicKey) {  
        this.privateKey = privateKey;  
    }  
}
```

```
        this.publicKey = publicKey;
    }
}
```

ElGamalPublicKey.java

```
public class ElGamalPublicKey {
    public BigInteger p;
    public BigInteger g;
    public BigInteger y;

    public ElGamalPublicKey(BigInteger p, BigInteger g, BigInteger y) {
        this.p = p;
        this.g = g;
        this.y = y;
    }
}
```

ElGamalPrivateKey.java

```
public class ElGamalPrivateKey {
    public BigInteger x;
    public BigInteger p;

    public ElGamalPrivateKey(BigInteger x, BigInteger p) {
        this.x = x;
        this.p = p;
    }
}
```

BBSResult.java

```
public class BBSResult {
    public BigInteger[] encryptedMessage;
    public BigInteger x0;

    public BBSResult(BigInteger[] encryptedMessage, BigInteger x0) {
        this.encryptedMessage = encryptedMessage;
        this.x0 = x0;
    }
}
```

Credentials.java

```
public class Credentials {
    public String login;
    public String password;

    public Credentials(String login, String password) {
        this.login = login;
        this.password = password;
    }
}
```