НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Лабораторна робота №5

з дисципліни

«Протоколи та алгоритми електронного голосування»

Виконав:
студент 4 курсу
групи ІП-02
Середюк Валентин

Київ – 2023

Лабораторна робота №5

Тема: Протокол Е-голосування з розділенням комісії на незалежні частини

Мета: Дослідити протокол Е-голосування з розділенням комісії на незалежні частини

Завдання

Змоделювати протокол Е-голосування з розділенням комісії на незалежні частини будь-якою мовою програмування та провести його дослідження. Для кодування повідомлень використовувати метод RSA, 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 розділенням комісії на незалежні частини.

Для реалізації даного алгоритму було поетапно виконано усі пункти із протоколу голосування з розділенням комісії на незалежні частини. Розглянемо кожен етап окремо.

ЦВК формує список виборців та кандидатів та присвоює їм унікальні номери (код 1.1).

```
public void setCandidates(List<Candidate> candidates) {
    this.candidates = new ArrayList<>(candidates);
    candidates.forEach(candidate ->
candidate.setId(Generator.generateNonPrimeId()));
}

public void setVoters(List<Voter> voters) {
    this.voters = new ArrayList<>(voters);
    voters.forEach(voter -> voter.setId(Generator.generateNonPrimeId()));
}
```

Код 1.1 – Додавання виборців та кандидатів та присвоювання їм номерів

Виборець обирає свого кандидата для голосування та створює два бюлетеня із множників номера кандидата (код 1.2).

```
public void createBallots(int candidateId) {
    int[] multipliers = Util.getMultipliers(candidateId);
    for (int i = 0; i < ballots.length; i++) {
        ballots[i] = new Ballot(String.valueOf(multipliers[i]));
    }
}
```

Код 1.2 – Створення бюлетенів із множників

Далі виборець шифрує обидва бюлетеня відкритим ключем ЦВК за допомогою RSA. Додає свій номер та підписує повідомлення за допомогою DSA (код 1.3 та 1.4).

```
public void encryptBallots(PublicKey publicKey) {
    for (Ballot ballot : ballots) {
        ballot.setData(RSA.encrypt(ballot.getData(), publicKey));
    }
}

public void createVoteMessages() {
    for (int i = 0; i < ballots.length; i++) {
        voteMessages[i] = new VoteMessage(id, ballots[i]);
    }
}

public void signMessages() {
```

```
    for (VoteMessage message : voteMessages) {
        message.addSignature(keyPair);
    }
}
```

Код 1.3 – Шифрування, створення повідомлень та підпис

```
public void addSignature(KeyPair keyPair) {
    signature = DSA.sign(new BigInteger(toString()), keyPair.getPrivate());
    DSAPublicKey = keyPair.getPublic();
}
```

Код 1.4 – Метод створення підпису

Далі виборець надсилає підписані повідомлення першій та другій ВК по одній частині.

Обидві ВК перевіряють підпис та додають бюлетені з повідомлення, якщо ж раніше не було надіслані повідомлення від цього виборця (код 1.5).

```
public void sendVoteMessage(VoteMessage voteMessage) {
    if (checkSignature(voteMessage)) {
        if (containsId(voteMessage)) {
            System.out.println("The vote message has been already sent");
        } else {
            votes.add(voteMessage);
        }
    }
}

private boolean checkSignature(VoteMessage voteMessage) {
    if (voteMessage.signature != null) {
        return DSA.verify(new BigInteger(voteMessage.toString()),
voteMessage.signature, voteMessage.DSAPublicKey);
    }
    return false;
}

private boolean containsId(VoteMessage voteMessage) {
    for (VoteMessage vote : votes) {
        if (vote.voterId == voteMessage.voterId) {
            return true;
        }
    }
    return false;
}
```

Код 1.5 – Процес перевірки повідомлень

Після перевірки усіх повідомлень ВК публікують усі зашифровані бюлетені та номери, які їм надійшли.

Потім ЦВК об’єднує відповідні бюлетені із двох списків ВК по номеру виборця. Після чого розшифровує вже об’єднаний шифротекст своїм ключем та зараховує голоси. По закінченню підрахунку виводяться результати голосування (код 1.6)

```
public void sendVotesListFromElectionCommissions(List<VoteMessage>
firstPart, List<VoteMessage> secondPart) {
    for (VoteMessage message : firstPart) {
        VoteMessage secondMessage = findMessageById(message.voterId,
secondPart);
        if (secondMessage != null) {
            BigInteger firstPartId = new
BigInteger(message.ballot.getData());
            BigInteger secondPartId = new
BigInteger(secondMessage.ballot.getData());
            BigInteger data = firstPartId.multiply(secondPartId);
            String dataToDecrypt = String.valueOf(data);
            String decryptedData = RSA.decrypt(dataToDecrypt,
keyPair.getPrivate());
            Candidate candidate = findCandidate(decryptedData);
            if (candidate != null) {
                votes.put(message.voterId, dataToDecrypt);
                candidate.incrementVotes();
            } else {
                System.out.println("Candidate not found");
            }
        } else {
            System.out.println("Voter send only one part");
        }
    }
}

private VoteMessage findMessageById(int id, List<VoteMessage> messages) {
    for (VoteMessage message : messages) {
        if (message.voterId == id) {
            return message;
        }
    }
    return null;
}

private Candidate findCandidate(String data) {
    for (Candidate candidate : candidates) {
        if (candidate.getId() == Integer.parseInt(data)) {
            return candidate;
        }
    }
    return null;
}
```

Код 1.6 – Процес об’єднання та розшифрування бюлетенів, а також підрахунку голосів

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) {
        int[] votes = new int[] { 1, 0, 2, 3, 2, 1, 2, 0, 2, 3 };

        List<Candidate> candidates = DataFactory.getCandidates(4);
        List<Voter> voters = DataFactory.getVoters(10);
        CentralElectionCommission centralElectionCommission = new
CentralElectionCommission();
        centralElectionCommission.setCandidates(candidates);
        centralElectionCommission.setVoters(voters);
        ElectionCommission electionCommissionA = new ElectionCommission();
        ElectionCommission electionCommissionB = new ElectionCommission();

        for (int i = 0; i < votes.length; i++) {
            voters.get(i).createBallots(candidates.get(votes[i]).getId());

voters.get(i).encryptBallots(centralElectionCommission.getPublicKey());
            voters.get(i).createVoteMessages();
            voters.get(i).signMessages();
            VoteMessage[] voteMessages = voters.get(i).getVoteMessages();
            electionCommissionA.sendVoteMessage(voteMessages[0]);
            electionCommissionB.sendVoteMessage(voteMessages[1]);
        }

        centralElectionCommission.sendVotesListFromElectionCommissions(
                electionCommissionA.getVotes(),
                electionCommissionB.getVotes()
        );

        centralElectionCommission.printResult();
    }
}
```

Код 2.1 – Методи для демонстрації роботи алгоритму

```
+------------------+--------------+
|        ELECTION  RESULTS        |
+------------------+--------------+
|    CANDIDATES    |     VOTES    |
+------------------+--------------+
|      Candidate 0 |            2 |
|      Candidate 1 |            2 |
|      Candidate 2 |            4 |
|      Candidate 3 |            2 |
+------------------+--------------+
```

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).

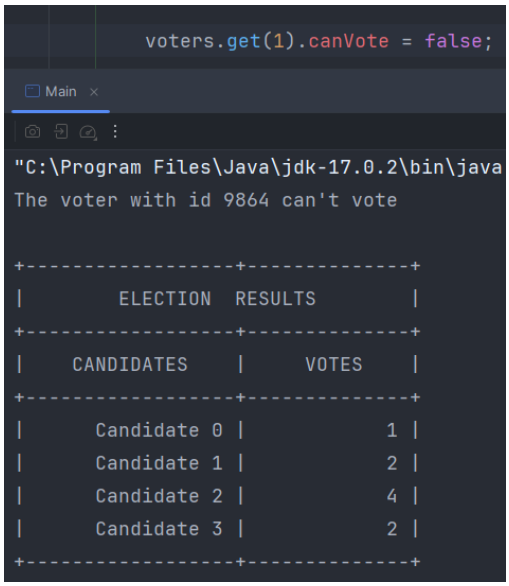

Рисунок 3.1 – Голосування виборця, який не має права

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).

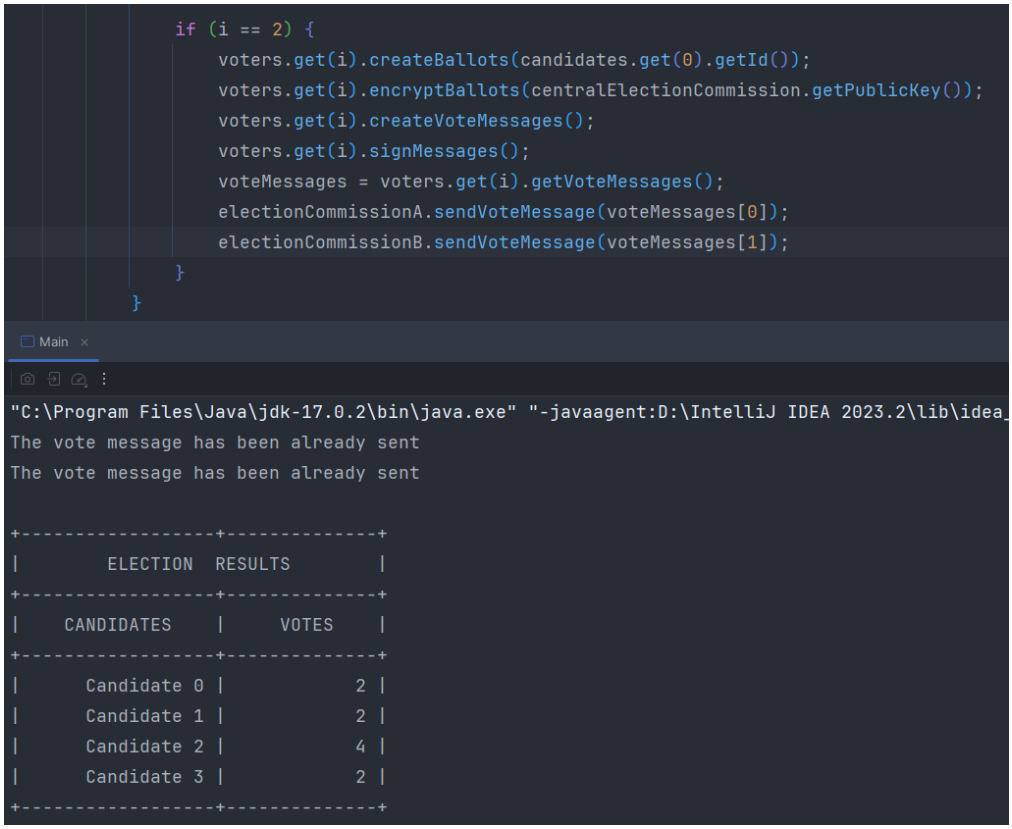

Рисунок 3.2 – Спроба проголосувати двічі

Як можна побачити, голос не був зарахований, а дві ВК відхили спробу надіслати ще раз повідомлення із бюлетенями.

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

Протокол голосування з розділенням на незалежні виборчі комісії розроблений так, що окрема виборча комісія не може дізнатись, хто за кого проголосував, якщо правильно розділити числа на множники. ЦВК у свою чергу ж при отримані повідомлень знає, хто голосував та при розшифровуванні дізнається результати голосів виборців.

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

Якщо хтось дізнається номер іншого виборця, то він може надіслати повідомлення з його номером, але при перевірці підпису, виявиться, що підписане повідомлення було іншим виборцем, що підтвердить підробку голосу.

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

Якщо ВК будуть заодно, то вони можуть зговоритись та одночасно підмінити правильно зашифровані бюлетені. Але при оголошені результату ЦВК, виборець помітить, що його голос не збігається із кінцевим. Також сама ЦВК може підмінити голос, але тоді ВК помітить, що ЦВК врахувала не правильні частини бюлетенів.

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

Так, виборець може це зробити, оскільки після завершення голосування, ЦВК публікує список номерів виборців та відповідні зашифровані білети, які може перевірити виборець на правильність.

Висновок

Виконавши дану лабораторну роботу, було створено протокол електронного голосування з розділенням комісії на незалежні частини.

Для його реалізації, було розроблено систему, яка дозволяла ділити бюлетені на дві частини та окремо надсилати різним ВК. Після чого ЦВК збирала результати, які були надіслані до ВК та об’єднувала частини бюлетенів, з яких вже й підраховувала голоси.

Дослідження алгоритму показало, що даний алгоритм відповідає усім вимогам. Із його недоліків можна виділити те, що при зговорі ВК, бюлетені виборців можна підробити, що порушує протокол проведення голосування. Також виборець зможе це помітити при публікації результатів від ВК та оскаржити голосування.

Лістинг коду мовою Java

Main.java

```
public class Main {
    public static void main(String[] args) {
        int[] votes = new int[] { 1, 0, 2, 3, 2, 1, 2, 0, 2, 3 };

        List<Candidate> candidates = DataFactory.getCandidates(4);
        List<Voter> voters = DataFactory.getVoters(10);
        CentralElectionCommission centralElectionCommission = new
CentralElectionCommission();
        centralElectionCommission.setCandidates(candidates);
        centralElectionCommission.setVoters(voters);
        ElectionCommission electionCommissionA = new ElectionCommission();
        ElectionCommission electionCommissionB = new ElectionCommission();

        for (int i = 0; i < votes.length; i++) {
            voters.get(i).createBallots(candidates.get(votes[i]).getId());

voters.get(i).encryptBallots(centralElectionCommission.getPublicKey());
            voters.get(i).createVoteMessages();
            voters.get(i).signMessages();
            VoteMessage[] voteMessages = voters.get(i).getVoteMessages();
            electionCommissionA.sendVoteMessage(voteMessages[0]);
            electionCommissionB.sendVoteMessage(voteMessages[1]);

        }

        centralElectionCommission.sendVotesListFromElectionCommissions(
                electionCommissionA.getVotes(),
                electionCommissionB.getVotes()
        );

        centralElectionCommission.printResult();
    }
}
```

CentralElectionCommission.java

```
public class CentralElectionCommission {
    private List<Candidate> candidates;
    private List<Voter> voters;
    private KeyPair keyPair;
    private Map<Integer, String> votes;

    public CentralElectionCommission() {
        try {
            KeyPairGenerator keyPairGenerator =
KeyPairGenerator.getInstance("RSA");
            keyPairGenerator.initialize(2048);
            keyPair = keyPairGenerator.genKeyPair();
            votes = new HashMap<>();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

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

```
        candidates.forEach(candidate ->
candidate.setId(Generator.generateNonPrimeId()));
    }

    public void setVoters(List<Voter> voters) {
        this.voters = new ArrayList<>(voters);
        voters.forEach(voter ->
voter.setId(Generator.generateNonPrimeId()));
    }

    public PublicKey getPublicKey() {
        return keyPair.getPublic();
    }

    public void sendVotesListFromElectionCommissions(List<VoteMessage>
firstPart, List<VoteMessage> secondPart) {
        for (VoteMessage message : firstPart) {
            VoteMessage secondMessage = findMessageById(message.voterId,
secondPart);
            if (secondMessage != null) {
                BigInteger firstPartId = new
BigInteger(message.ballot.getData());
                BigInteger secondPartId = new
BigInteger(secondMessage.ballot.getData());
                BigInteger data = firstPartId.multiply(secondPartId);
                String dataToDecrypt = String.valueOf(data);
                String decryptedData = RSA.decrypt(dataToDecrypt,
keyPair.getPrivate());
                Candidate candidate = findCandidate(decryptedData);
                if (candidate != null) {
                    if
(Objects.requireNonNull(findVoter(message.voterId)).canVote) {
                        votes.put(message.voterId, dataToDecrypt);
                        candidate.incrementVotes();
                    }
                    else {
                        System.out.printf("The voter with id %d can't
vote\n", message.voterId);
                    }
                } else {
                    System.out.printf("Candidate %s not found\n",
decryptedData);
                }
            } else {
                System.out.println("Voter send only one part");
            }
        }
    }

    private VoteMessage findMessageById(int id, List<VoteMessage> messages)
{
        for (VoteMessage message : messages) {
            if (message.voterId == id) {
                return message;
            }
        }
        return null;
    }

    private Candidate findCandidate(String data) {
        for (Candidate candidate : candidates) {
```

```
            if (candidate.getId() == Integer.parseInt(data)) {
                return candidate;
            }
        }
        return null;
    }

    private Voter findVoter(int id) {
        for (Voter voter : voters) {
            if (voter.getId() == id) {
                return voter;
            }
        }
        return null;
    }

    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");
    }
}
```

ElectionCommission.java

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

public class ElectionCommission {
    List<VoteMessage> votes = new ArrayList<>();

    public void sendVoteMessage(VoteMessage voteMessage) {
        if (checkSignature(voteMessage)) {
            if (containsId(voteMessage)) {
                System.out.println("The vote message has been already
sent");
            } else {
                votes.add(voteMessage);
            }
        }
    }

    private boolean checkSignature(VoteMessage voteMessage) {
        if (voteMessage.signature != null) {
            return DSA.verify(new BigInteger(voteMessage.toString()),
voteMessage.signature, voteMessage.DSAPublicKey);
        }
        return false;
    }

    private boolean containsId(VoteMessage voteMessage) {
        for (VoteMessage vote : votes) {
```

```
            if (vote.voterId == voteMessage.voterId) {
                return true;
            }
        }
        return false;
    }

    public List<VoteMessage> getVotes() {
        return votes;
    }
}
```

Candidate.java

```
public class Candidate {
    private int id;
    private String name;
    private int votes = 0;

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

    public int getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public int getVotes() {
        return votes;
    }

    public void incrementVotes() {
        votes++;
    }
}
```

Voter.java

```
public class Voter {
    private int id;
    private String name;
    private Ballot[] ballots;
    private VoteMessage[] voteMessages;
    private KeyPair keyPair;

    public boolean canVote = true;

    public Voter(String name) {
        this.name = name;
        ballots = new Ballot[2];
        voteMessages = new VoteMessage[2];
```

```
        keyPair = Generator.generateDSAKeyPair();
    }

    public int getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void createBallots(int candidateId) {
        int[] multipliers = Util.getMultipliers(candidateId);
        for (int i = 0; i < ballots.length; i++) {
            ballots[i] = new Ballot(String.valueOf(multipliers[i]));
        }
    }

    public void encryptBallots(PublicKey publicKey) {
        for (Ballot ballot : ballots) {
            ballot.setData(RSA.encrypt(ballot.getData(), publicKey));
        }
    }

    public void createVoteMessages() {
        for (int i = 0; i < ballots.length; i++) {
            voteMessages[i] = new VoteMessage(id, ballots[i]);
        }
    }

    public void signMessages() {
        for (VoteMessage message : voteMessages) {
            message.addSignature(keyPair);
        }
    }

    public VoteMessage[] getVoteMessages() {
        return voteMessages;
    }
}
```

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

```
public class VoteMessage {
    public int voterId;
    public Ballot ballot;
    public byte[] signature;
    public PublicKey DSAPublicKey;

    public VoteMessage(int voterId, Ballot ballot) {
        this.voterId = voterId;
        this.ballot = ballot;
    }

    public void addSignature(KeyPair keyPair) {
        signature = DSA.sign(new BigInteger(toString()),
keyPair.getPrivate());
        DSAPublicKey = keyPair.getPublic();
    }

    @Override
    public String toString() {
        return voterId + ballot.getData();
    }
}
```

RSA.java

```
public class RSA {
    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));
    }
}
```

DSA.java

```
public class DSA {
    public static byte[] sign(BigInteger msg, PrivateKey privateKey) {
        byte[] message = String.valueOf(msg).getBytes();
        try {
            Signature dsa = Signature.getInstance("SHA256withDSA");
            dsa.initSign(privateKey);
```

```
            dsa.update(message);
            return dsa.sign();
        } catch (NoSuchAlgorithmException | InvalidKeyException |
SignatureException e) {
            System.out.println(e.getMessage());
            return null;
        }
    }

    public static boolean verify(BigInteger msg, byte[] signature,
PublicKey publicKey) {
        byte[] message = String.valueOf(msg).getBytes();
        try {
            Signature verifyDsa = Signature.getInstance("SHA256withDSA");
            verifyDsa.initVerify(publicKey);
            verifyDsa.update(message);
            return verifyDsa.verify(signature);
        } catch (NoSuchAlgorithmException | InvalidKeyException |
SignatureException e) {
            System.out.println(e.getMessage());
            return false;
        }
    }
}
```

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;
    }
}
```

Generator.java

```
public class Generator {
    public static int generateNonPrimeId() {
        Random rand = new Random();
        int primeId;
        do {
            primeId = rand.nextInt(10000);
        } while (Util.isPrime(primeId));
        return primeId;
    }

    public static KeyPair generateDSAKeyPair() {
        try {
```

```
            KeyPairGenerator keyPairGenerator =
KeyPairGenerator.getInstance("DSA");
            keyPairGenerator.initialize(2048);
            return keyPairGenerator.generateKeyPair();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return null;
    }
}
```

Util.java

```
public class Util {
    public static boolean isPrime(int number) {
        for (int i = 2; i <= Math.sqrt(number); i++) {
            if (number % i == 0) {
                return false;
            }
        }
        return true;
    }

    public static int[] getMultipliers(int value) {
        for (int i = (int) Math.sqrt(value); i > 1; i--) {
            if (value % i == 0) {
                return new int[] {i, value / i};
            }
        }
        return new int[] {value, 1};
    }
}
```