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

Лабораторна робота №4

з дисципліни

«Протоколи та алгоритми електронного голосування»

Виконав:
студент 4 курсу
групи ІП-02
Середюк Валентин

Київ – 2023

Лабораторна робота №4

Тема: Протокол Е-голосування з перемішуванням

Мета: Дослідити протокол Е-голосування з перемішуванням

Завдання

Змоделювати протокол Е-голосування з перемішуванням будь-якою мовою програмування та провести його дослідження. Для кодування повідомлень використовувати метод RSA, для реалізації ЕЦП використовувати алгоритм Ель-Гамаля.

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 перемішуванням.

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

Спочатку формуються список з усіх виборців, які будуть приймати участь у даному голосуванні. Далі кожен виборець створює бюлетень із кандидатом, за якого він хоче проголосувати та шифрує його методом RSA (код 1.1).

```
public void createBallot(int candidateId) {
    ballot = new Ballot(id, candidateId);
}

public void encryptBallot(List<Voter> voters) {
    String randomString = getRandomString();
    randomStrings.add(randomString);
    String ballotString = ballot.getData() + randomString;
    messages.add(ballotString);
    ballots.add(ballot);
    byte[] firstStageOfEncrypting = encryptFirstStage(ballotString);
    byte[] secondStageOfEncrypting = encryptSecondStage(new
String(firstStageOfEncrypting, StandardCharsets.UTF_8));
    encodedMessages.add(secondStageOfEncrypting);
}

private String getRandomString() {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < 4; i++)
        sb.append(Util.getRandomChar());
    return sb.toString();
}
```

Код 1.1 – Створення та шифрування бюлетеня

Процес шифрування ділиться на два етапи. Перший етап це додавання випадкового рядка до бюлетеня, а потім по черзі, шифрується відкритим ключем кожного виборця, починаючи з останнього у списку (код 1.2).

```
private byte[] encryptFirstStage(String data) {
    byte[] res = data.getBytes(StandardCharsets.UTF_8);
    List<PublicKey> keys = PublicKeys.getPublicKeys();
    for (int i = 0; i < keys.size(); i++) {
        res = RSA.encrypt(new String(res, StandardCharsets.UTF_8),
keys.get(keys.size() - i - 1));
        encodedBytes.add(res);
    }
```

```
    return res;
}
```

Код 1.2 – Перший етап шифрування

На другому етапі шифрування, процедура повторюється, але при кожному шифруванні ключем виборців, додається один випадковий рядок (код 1.3).

```
private byte[] encryptSecondStage(String data) {
    byte[] res = data.getBytes(StandardCharsets.UTF_8);
    List<PublicKey> keys = PublicKeys.getPublicKeys();
    for (int i = 0; i < keys.size(); i++)
        res = addStringAndEncrypt(new String(res, StandardCharsets.UTF_8),
keys.get(keys.size() - i - 1));
    return res;
}

private byte[] addStringAndEncrypt(String data, PublicKey publicKey) {
    String randomString = getRandomString();
    randomStrings.add(randomString);
    byte[] randomStringBytes =
randomString.getBytes(StandardCharsets.UTF_8);
    byte[] dataBytes = data.getBytes(StandardCharsets.UTF_8);
    byte[] res = new byte[dataBytes.length + randomStringBytes.length];
    System.arraycopy(dataBytes, 0, res, 0, dataBytes.length);
    System.arraycopy(randomStringBytes, 0, res, dataBytes.length,
randomStringBytes.length);
    return RSA.encrypt(new String(res, StandardCharsets.UTF_8), publicKey);
}
```

Код 1.3 – Другий етап шифрування

Після успішного шифрування, усі виборці надсилають свої бюлетені першому виборцю.

Перший виборець розшифровує усі бюлетені своїм приватним ключем. Він має впевнитись, що його бюлетень наявний у списку усіх бюлетені по випадковому рядку, який він додавав до свого бюлетеня. Далі він прибирає усі додаткові рядки та перемішує бюлетені. Отримані бюлетені він надсилає наступному по списку виборцю (код 1.4).

```
public void decryptBallots() {
    for (int i = 0; i < encodedMessages.size(); i++) {
        String decryptedMessage = RSA.decrypt(encodedMessages.get(i),
keyPair.getPrivate());
        String removed = decryptedMessage.substring(0,
decryptedMessage.length() - 4);
        byte[] encrypted = RSA.encrypt(removed, keyPair.getPublic());
        encodedMessages.set(i, encrypted);
        if (randomStrings.size() > examples + 1)
            randomStrings.remove(randomStrings.size() - 1);
    }
}
```

```
public void shuffleBallots() {
    Collections.shuffle(encodedMessages);
}
```

Код 1.4 – Процес розшифрування та видалення випадкових рядків

Цей процес продовжується допоки усі виборці не пройдуть повне коло. Після чого останній у списку виборець надсилає свій результат першому виборцю. Схожа процедура повторюється, але тепер кожен виборець лише розшифровує бюлетені, перевіряє лише чи його бюлетень на місці та підписує за допомогою ЕЦП (методом Ель-Гамаля), якщо ж ніяких проблем не виявлено (код 1.5).

```
public void decryptBallotsWithOutDeleting() {
    for (int i = 0; i < encodedMessages.size(); i++) {
        String decryptedMessage = RSA.decrypt(encodedMessages.get(i),
keyPair.getPrivate());
        byte[] encrypted = RSA.encrypt(decryptedMessage,
keyPair.getPublic());
        encodedMessages.set(i, encrypted);
    }
}

public void shuffleBallots() {
    Collections.shuffle(encodedMessages);
}

public void sign() {
    for (int i = 0; i < encodedMessages.size(); i++) {
        BigInteger[] signature = elGamal.sign(new
String(encodedMessages.get(i), StandardCharsets.UTF_8));
        signatures.set(i, signature);
    }
}
```

Код 1.5 – Процес розшифрування та підпису бюлетенів

Отримані бюлетені знову надсилаються наступному по списку виборцю. Він зобов’язаний перевірити підпис попереднього виборця та зняти його, якщо все гаразд (код 1.6). Надалі дії повторюються до останнього виборця.

```
public void verifySign() {
    for (Ballot ballot : ballots) {
        if (!elGamal.verifySign(ballot.getData(), ballot.getSignatures()))
{
            System.out.println("Ballot " + ballot.getData() + " is not
verified!");
        }
        ballot.removeSign();
    }
}
```

Код 1.6 – Процес перевірки підпису

Після підписання бюлетенів останнім виборцем, від надсилає їх усім учасникам голосування, які мають впевнитись у правильності підписання останнім.

Якщо усі етапи пройшли згідно протоколу, то з бюлетенів прибираються випадкові рядки та оголошуються результати голосування (код 1.7).

```
public void deleteFirstRandomString() {
    randomStrings.clear();
    for (int i = 0; i < encodedMessages.size(); i++) {
        String msg = encodedMessages.get(i).substring(0,
messages.get(i).length() - 4);
        messages.set(i, msg);
    }
}

public static String getResult(List<Candidate> candidates) {
    for (String message : messages) {
        int vote = Integer.parseInt(message.substring(message.length() -
1));
        for (Candidate candidate : candidates) {
            if (candidate.getId() == vote) {
                candidate.incrementVotes();
            }
        }
    }
    StringBuilder result = new StringBuilder();
    for (Candidate candidate : candidates) {
        result.append(candidate.getName()).append(":
").append(candidate.getVotesCount()).append("\n");
    }
    return result.toString();
}
```

Код 1.7 – Процес видалення останнього випадкового рядка та підведення підсумків

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();
        List<Voter> voters = DataFactory.getVoters();

        Voter A = voters.get(0);
        Voter B = voters.get(1);
        Voter C = voters.get(2);
        Voter D = voters.get(3);

        for (Voter voter : voters) {
            PublicKeys.addPublicKey(voter.getPublicKey());
        }

        for (Voter voter : voters) {
            voter.createBallot(new Random().nextInt(2));
        }

        for (Voter voter : voters) {
            voter.encryptBallot(voters);
        }

        for (int i = 1; i < voters.size(); i++) {
            voters.get(i).sendBallotTo(A);
        }

        for (Voter voter : voters) {
            voter.decryptBallots();
            voter.shuffleBallots();
        }

        A.decryptBallotsWithOutDeleting();
        A.sign();
        A.shuffleBallots();
        A.sendBallots(B);
        A.sendBallots(C);
        A.sendBallots(D);

        B.verifySign();
        B.decryptBallotsWithOutDeleting();
        B.sign();
        B.shuffleBallots();
        B.sendBallots(A);
        B.sendBallots(C);
        B.sendBallots(D);

        C.verifySign();
        C.decryptBallotsWithOutDeleting();
        C.sign();
        C.shuffleBallots();
        C.sendBallots(A);
        C.sendBallots(B);
        C.sendBallots(D);

        D.verifySign();
```

```
        D.decryptBallotsWithOutDeleting();
        D.sign();
        D.shuffleBallots();
        D.sendBallots(A);
        D.sendBallots(B);
        D.sendBallots(C);

        A.verifySign();
        A.deleteFirstRandomString();

        System.out.println("ELECTION RESULT");
        System.out.println(Voter.getResult(candidates));
    }
}
```

Код 2.1 – Методи для демонстрації роботи алгоритму

```
ELECTION RESULT
candidate1: 3
candidate2: 1
```

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.**

Так як усі знають, хто має брати участь у голосуванні, то усі запідозрять, що з’явився зайвий бюлетень і припинять проводити голосування. Також, кожен виборець знає кому він має надсилати свої повідомлення з бюлетенями, тому вони не зможуть потрапити до тих, хто не має на це права.

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

Сам по собі протокол побудований так, що від кожного виборця повинен бути один і тільки один голос. Тому декілька голосів від одного виборця буде свідчати, що хтось порушив правила проведення голосування.

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

Алгоритм голосування з перемішуванням розроблений так, що на кожному етапі голосування, бюлетені перемішуються, що не дає змоги зрозуміти де чий голос. Але останній виборець буде завжди мати перевагу, оскільки він буде знати результати голосування раніше інших.

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

Якщо ж хтось захоче підмінити чужий голос, то це буде викрито під час розшифрування бюлетенів. Кожен виборець зберігає свою історію проміжних етапів, яку він порівнює із теперішнім. Якщо ж буде якась відміна від історії його бюлетеня, до виборець зможе зрозуміти, що відбулась підміна його бюлетеня.

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

Для заміни, комусь потрібно дізнатись усі випадкові рядки, які використовував виборець. Також виборець зможе помітити підміну бюлетеня під час того, як він буде його розшифровувати.

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

Так, виборець може це зробити, оскільки після завершення голосування, виборці можуть переглянути результати ще з прикріпленим останнім випадковим рядком та дізнатись, чи їх голос відповідає тому, за якого вони голосували.

Висновок

Виконавши дану лабораторну роботу, було створено протокол електронного голосування з перемішуванням. Особливість даного протоколу полягає в тому, що у ньому відсутня третя довірена сторона (наприклад ВК), а сам процес проводять самі ж виборці.

Для його реалізації, було розроблено систему, яка дозволяла шифрувати бюлетені у два етапи згідно протоколу, а потім почергово їх розшифровувати кожним із виборців. Також, для шифрування та дешифрування повідомлень було використано алгоритм RSA, а для реалізації ЕЦП – алгоритм Ель-Гамаля.

Дослідження алгоритму показало, що даний алгоритм відповідає усім вимогам. Із його недоліків можна виділити те, що останній виборець по списку завжди має перевагу над іншими, якщо результатом виборів якось можна скористатись.

Лістинг коду мовою Java

Main.java

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

        Voter A = voters.get(0);
        Voter B = voters.get(1);
        Voter C = voters.get(2);
        Voter D = voters.get(3);

        for (Voter voter : voters) {
            PublicKeys.addPublicKey(voter.getPublicKey());
        }

        for (Voter voter : voters) {
            voter.createBallot(new Random().nextInt(2));
        }

        for (Voter voter : voters) {
            voter.encryptBallot(voters);
        }

        for (int i = 1; i < voters.size(); i++) {
            voters.get(i).sendBallotTo(A);
        }

        for (Voter voter : voters) {
            voter.decryptBallots();
            voter.shuffleBallots();
        }

        A.decryptBallotsWithOutDeleting();
        A.sign();
        A.shuffleBallots();
        A.sendBallots(B);
        A.sendBallots(C);
        A.sendBallots(D);

        B.verifySign();
        B.decryptBallotsWithOutDeleting();
        B.sign();
        B.shuffleBallots();
        B.sendBallots(A);
        B.sendBallots(C);
        B.sendBallots(D);

        C.verifySign();
        C.decryptBallotsWithOutDeleting();
        C.sign();
        C.shuffleBallots();
        C.sendBallots(A);
        C.sendBallots(B);
        C.sendBallots(D);

        D.verifySign();
        D.decryptBallotsWithOutDeleting();
        D.sign();
```

```
        D.shuffleBallots();
        D.sendBallots(A);
        D.sendBallots(B);
        D.sendBallots(C);

        A.verifySign();
        A.deleteFirstRandomString();

        System.out.println("ELECTION RESULT");
        System.out.println(Voter.getResult(candidates));
    }
}
```

Voter.java

```
public class Voter {
    private static int examples = 0;
    private int id;
    private String name;
    private KeyPair keyPair;
    private ElGamal elGamal;
    private Ballot ballot;
    private static List<String> randomStrings = new ArrayList<>();
    private static List<String> messages = new ArrayList<>();
    private static List<byte[]> encodedMessages = new ArrayList<>();
    private List<byte[]> encodedBytes = new ArrayList<>();
    private static List<Ballot> ballots = new ArrayList<>();
    private List<Ballot> resBallots = new ArrayList<>();
    private static List<BigInteger[]> signatures = new ArrayList<>();

    public Voter(String name) {
        this.name = name;
        id = examples;
        examples++;
        generateKeys();
        elGamal = new ElGamal();
        signatures.addAll(Collections.nCopies(4, new BigInteger[2]));
    }

    private void generateKeys() {
        try {
            KeyPairGenerator keyPairGenerator =
KeyPairGenerator.getInstance("RSA");
            keyPairGenerator.initialize(1024);
            keyPair = keyPairGenerator.generateKeyPair();
        } catch (Exception e) {
            System.out.println("Failed to generate keys:\n" +
e.getMessage());
        }
    }

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

    public void createBallot(int candidateId) {
        ballot = new Ballot(id, candidateId);
    }

    public void encryptBallot(List<Voter> voters) {
```

```
        String randomString = getRandomString();
        randomStrings.add(randomString);
        String ballotString = ballot.getData() + randomString;
        messages.add(ballotString);
        ballots.add(ballot);
        byte[] firstStageOfEncrypting = encryptFirstStage(ballotString);
        byte[] secondStageOfEncrypting = encryptSecondStage(new
String(firstStageOfEncrypting, StandardCharsets.UTF_8));
        encodedMessages.add(secondStageOfEncrypting);
    }

    private String getRandomString() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 4; i++)
            sb.append(Util.getRandomChar());
        return sb.toString();
    }

    private byte[] encryptFirstStage(String data) {
        byte[] res = data.getBytes(StandardCharsets.UTF_8);
        List<PublicKey> keys = PublicKeys.getPublicKeys();
        for (int i = 0; i < keys.size(); i++) {
            res = RSA.encrypt(new String(res, StandardCharsets.UTF_8),
keys.get(keys.size() - i - 1));
            encodedBytes.add(res);
        }
        return res;
    }

    private byte[] encryptSecondStage(String data) {
        byte[] res = data.getBytes(StandardCharsets.UTF_8);
        List<PublicKey> keys = PublicKeys.getPublicKeys();
        for (int i = 0; i < keys.size(); i++)
            res = addStringAndEncrypt(new String(res,
StandardCharsets.UTF_8), keys.get(keys.size() - i - 1));
        return res;
    }

    private byte[] addStringAndEncrypt(String data, PublicKey publicKey) {
        String randomString = getRandomString();
        randomStrings.add(randomString);
        byte[] randomStringBytes =
randomString.getBytes(StandardCharsets.UTF_8);
        byte[] dataBytes = data.getBytes(StandardCharsets.UTF_8);
        byte[] res = new byte[dataBytes.length + randomStringBytes.length];
        System.arraycopy(dataBytes, 0, res, 0, dataBytes.length);
        System.arraycopy(randomStringBytes, 0, res, dataBytes.length,
randomStringBytes.length);
        return RSA.encrypt(new String(res, StandardCharsets.UTF_8),
publicKey);
    }

    public void sendBallotTo(Voter voter) {
        if (messages.size() > 4)
            System.out.println("Message sending error");
    }

    public void decryptBallots() {
        for (int i = 0; i < encodedMessages.size(); i++) {
            String decryptedMessage = RSA.decrypt(encodedMessages.get(i),
keyPair.getPrivate());
```

```
            String removed = decryptedMessage.substring(0,
decryptedMessage.length() - 4);
            byte[] encrypted = RSA.encrypt(removed, keyPair.getPublic());
            encodedMessages.set(i, encrypted);
            if (randomStrings.size() > examples + 1)
                randomStrings.remove(randomStrings.size() - 1);
        }
    }

    public void decryptBallotsWithOutDeleting() {
        for (int i = 0; i < encodedMessages.size(); i++) {
            String decryptedMessage = RSA.decrypt(encodedMessages.get(i),
keyPair.getPrivate());
            byte[] encrypted = RSA.encrypt(decryptedMessage,
keyPair.getPublic());
            encodedMessages.set(i, encrypted);
        }
    }

    public void shuffleBallots() {
        Collections.shuffle(encodedMessages);
    }

    public void sign() {
        for (int i = 0; i < encodedMessages.size(); i++) {
            BigInteger[] signature = elGamal.sign(new
String(encodedMessages.get(i), StandardCharsets.UTF_8));
            signatures.set(i, signature);
        }
    }

    public void sendBallots(Voter voter) {
        voter.setResBallots(ballots);
    }

    private void setResBallots(List<Ballot> tempBallots) {
        this.resBallots = new ArrayList<>(tempBallots);
    }

    public void verifySign() {
        for (Ballot ballot : ballots) {
            if (!elGamal.verifySign(ballot.getData(),
ballot.getSignatures())) {
                System.out.println("Ballot " + ballot.getData() + " is not
verified!");
            }
            ballot.removeSign();
        }
    }

    public void deleteFirstRandomString() {
        randomStrings.clear();
        for (int i = 0; i < encodedMessages.size(); i++) {
            String msg = messages.get(i).substring(0,
messages.get(i).length() - 4);
            messages.set(i, msg);
        }
    }

    public static String getResult(List<Candidate> candidates) {
        for (String message : messages) {
```

```
            int vote = Integer.parseInt(message.substring(message.length()
- 1));
            for (Candidate candidate : candidates) {
                if (candidate.getId() == vote) {
                    candidate.incrementVotes();
                }
            }
        }
        StringBuilder result = new StringBuilder();
        for (Candidate candidate : candidates) {
            result.append(candidate.getName()).append(":
").append(candidate.getVotesCount()).append("\n");
        }
        return result.toString();
    }
}
```

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++;
    }
}
```

DataFactory.java

```
public class DataFactory {
    public static List<Candidate> getCandidates() {
        List<Candidate> candidates = new ArrayList<Candidate>();
        candidates.add(new Candidate("candidate1"));
        candidates.add(new Candidate("candidate2"));
        return candidates;
    }

    public static List<Voter> getVoters() {
```

```
        List<Voter> voters = new ArrayList<Voter>();
        voters.add(new Voter("A"));
        voters.add(new Voter("B"));
        voters.add(new Voter("C"));
        voters.add(new Voter("D"));
        return voters;
    }
}
```

PublicKeys.java

```
public class PublicKeys {
    private static List<PublicKey> publicKeys = new ArrayList<>();

    public static void addPublicKey(PublicKey publicKey) {
        publicKeys.add(publicKey);
    }

    public static List<PublicKey> getPublicKeys() {
        return publicKeys;
    }
}
```

Ballot.java

```
public class Ballot {
    private int voterId;
    private int candidateId;
    private BigInteger[] signature;

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

    public String getData() {
        return voterId + " votes for " + candidateId;
    }

    public void addSignature(BigInteger[] signature) {
        this.signature = signature;
    }

    public BigInteger[] getSignatures() {
        return signature;
    }

    public void removeSign() {
        signature = null;
    }
}
```

ElGamal.java

```
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[] sign(String message) {
        BigInteger m = getHashedText(message);

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

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

        BigInteger s = m.multiply(y.modPow(k, p)).mod(p);

        return new BigInteger[]{r, s};
    }

    public boolean verifySign(String message, BigInteger[] ciphertext) {
        BigInteger m = getHashedText(message);

        BigInteger r = ciphertext[0];
        BigInteger s = ciphertext[1];

        if (
                r.compareTo(BigInteger.ZERO) < 0
                || r.compareTo(p) > 0
                || s.compareTo(BigInteger.ZERO) < 0
                || s.compareTo(p.subtract(BigInteger.ONE)) > 0)
            return false;

        BigInteger a = r.modPow(x, p);
        BigInteger b = a.modInverse(p);

        return s.multiply(b).mod(p).compareTo(m) == 0;
    }

    private BigInteger getHashedText(String message) {
        byte[] bytes = message.getBytes(StandardCharsets.UTF_8);
        int sum = getSum(bytes);
        return new BigInteger(String.valueOf(sum));
    }

    private int getSum(byte[] bytes) {
        int sum = 0;
        for (byte aByte : bytes) sum += (int) aByte;
        return sum;
    }
}
```

RSA.java

```
public class RSA {
    public static byte[] encrypt(String plainText, Key publicKey) throws
Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        return cipher.doFinal(plainText.getBytes());
    }

    public static String decrypt(byte[] cipherText, Key privateKey) throws
Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decryptedBytes = cipher.doFinal(cipherText);
        return new String(decryptedBytes);
    }
}
```

Util.java

```
public class Util {
    private static final String chars =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

    public static char getRandomChar() {
        return chars.charAt(new Random().nextInt(chars.length()));
    }
}
```