МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Протоколы анонимности

ОТЧЁТ
ПО ДИСЦИПЛИНЕ
«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»

студента 5 курса 531 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Гельфанова Даниила Руслановича

Преподаватель
аспирант _____ Р. А. Фарахутдинов
подпись, дата

Саратов 2023

1 Постановка задачи

Необходимо реализовать протокол двух агентств Фудзиока-Окамото-Ота.

2 Теоретические сведения

В криптографии протоколы тайного голосования — протоколы обмена данными для реализации безопасного тайного электронного голосования через интернет при помощи компьютеров, телефонов или других специальных вычислительных машин. Это направление криптографии всё ещё развивается, но уже применяется на практике.

Основная идея протокола двух агентств состоит в том, чтобы заменить одно избирательное агентство двумя, чтобы они контролировали друг друга. Пусть здесь и далее V — регистратор, в обязанности которого входит подготовка списков, а также допуск или недопуск участника до голосования. C -счетчик, это агентство отвечает за получение и подсчет голосов.

Схема Фудзиока-Окамото-Ота, разработанная в 1992 году, основывается на протоколе двух агентств и криптографической подписи вслепу. Несильно усложняя протокол, эта схема частично решает проблему сговора двух агентств. Для работы протокола необходим заранее выбранный способ маскирующего шифрования, под которым избиратель присылает регистратору бюллетень. Ослепляющее (маскирующее) шифрование — особый вид шифрования, позволяющее удостовериться в том, что документ подлинный и подписан авторизованным пользователем, но не даёт узнать содержащиеся в нём данные. Маскирующее шифрование должно быть коммутативным с электронной подписью.

Алгоритм схемы Фудзиока-Окамото-Ота:

Вход: число избирателей и число избираемых.

Выход: результаты голосования

Шаг 1. Голосующий генерирует ключи для асимметричного шифрования: $E\_pubKey, E\_privKey$.

Шаг 2. Счётчик генерирует ключи: C_pubKey и $C\_privKey$. Публичный ключ рассказывает всем.

Шаг 3. Регистратор составляет списки:

- Регистратор генерирует ключи: V_pubKey и $V\_privKey$. Публичный ключ рассказывается всем.
- Регистратор выкладывает списки потенциальных голосующих, тех кому можно голосовать.
- Голосующие сообщают регистратору о желании голосовать. Приходят к регистратору и подтверждают личность. Там же регистрируют свой E_pubKey.
- После этого Регистратор выкладывает списки тех, кто голосует.

Шаг 4. Голосующий:

- Генерирует ключ для симметричного шифрования: key.
- Делает выбор в бюллетене B.
- Шифрует бюллетень: $B\_en = encrypt(B, key)$.
- Генерируется число r, и с его помощь скрывается содержимое бюллетеня $B\_en\_bl = blind(B\_en, r, V\_pubKey)$.
- Подписывает зашифрованный скрытый бюллетень: $B\_en\_bl\_sigE = sign(B\_en\_bl, E\_privKey)$.
- Отправляем регистратору $B\_en\_bl\_sigE$ и ещё заодно E_pubKey, чтобы регистратору было проще понять от кого сообщение.

Шаг 5. Регистратор

Принимает сообщение, проверяет, что оно подписано легитимный голосующим (напомним, что голосующие указывали регистратору свои публичные ключи при регистрации): $B\_en\_bl = sign(B\_en\_bl\_sigE, E\_pubKey)$.

После того как убедился, что голосующий легальный, подписывает вслепую: $B\_en\_bl\_sigV = sign(B\_en\_bl, V\_privKey)$.

Отправляет $B\_en\_bl\_sigV$ голосующему.

Шаг 6. Голосующий раскрывает бюллетень с помощью числа r: $B\_en\_sigV = unblind(B\_en\_bl\_sigV, r)$. Вот так регистратор подписал наш бюллетень, не заглядывая туда. Теперь проверить подпись регистратора может любой, в том числе и счётчик. Осталось только ему это отослать.

Шаг 7. Счётчик:

- Проверяет, что B_en_sigV действительно подписан регистратором: $B\_en = unsign(B\_en\_sigV, V\_pubKey)$.
- Помещает B_en в специальный список в открытом доступе после оговоренного времени.

Шаг 8. Голосующий, как увидят, что списки опубликованы, действуют дальше. Находят в этом списке номер их B_en, и отсылают счётчику этот номер и key. Счётчик расшифровывает и подсчитывает результаты.

3 Результаты работы

3.1 Сведения о программе

Программа была реализована на языке программирования Java. В ней есть 9 классов: FOO, $FOOService$, $Administrator$, $Voter$, $Counter$, $AESService$, $RSAService$, SHA, и $Signature$.

В классе FOO происходит считывание входных параметров: количество голосующих и количество избираемых на выборах.

Класс $OSSService$– класс реализации самого протокола. Для инициализации передаются количество голосующих и количество избираемых на выборах. Каждому шагу алгоритма выше соответствует собственный метод.

Класс $Administrator$ – класс Регистратора. При инициализации объекта происходит генерация ключей RSA и модуля RSA – числа n. В классе описаны методы для регистрации голосующих, получения бюллетени от голосующего, подписи вслепую, проверки подписи голосующего.

Класс $Counter$ – класс Счетчика. При инициализации объекта происходит генерация ключей RSA и модуля RSA – числа n. В классе описаны

методы для получения зашифрованных подписанных бюллетеней, проверки подписи Регистратора, расшифрования бюллетеней по ключу AES голосующего и подсчета с последующей публикацией результатов выборов.

Класс $Voter$ – класс Голосующего. При инициализации объекта происходит генерация ключей RSA и присвоение идентификатора голосующего. В классе описаны методы для генерации ключа AES, выбора голоса и его шифрования, маскировки бюллетени, подписи замаскированной бюллетени по схеме Онга-Шнорра-Шамира, снятия маскировки бюллетени, поиска себя в специальном списке проголосовавших, который публикуется после выборов.

Класс $AESService$ – класс шифрования AES. В нем описаны методы для генерации ключа, идентификационного вектора, шифрования, расшифрования.

Класс $RSAService$ – класс шифрования RSA. При инициализации объекта этого класса происходит генерация открытого и закрытого ключей.

Класс SHA – класс для получения хэш функции по алгоритму SHA.

Класс $Signature$ – класс для подписи, в нем содержатся 3 переменные S_1, S_2 и открытый ключ h.

3.2 Тестирование программы

В качестве модульного тестирования был написан класс $FOOServiceTest$, в котором проверяются случаи, когда:

- зарегистрироваться на выборы пытается человек, у которого нет на это право;

- регистратор проверяет подпись голосовавшего на бюллетене, и проверка не проходит;

- счетчик получает бюллетень, которая не подписана регистратором.

Проверки представлены на рисунках 1-3.

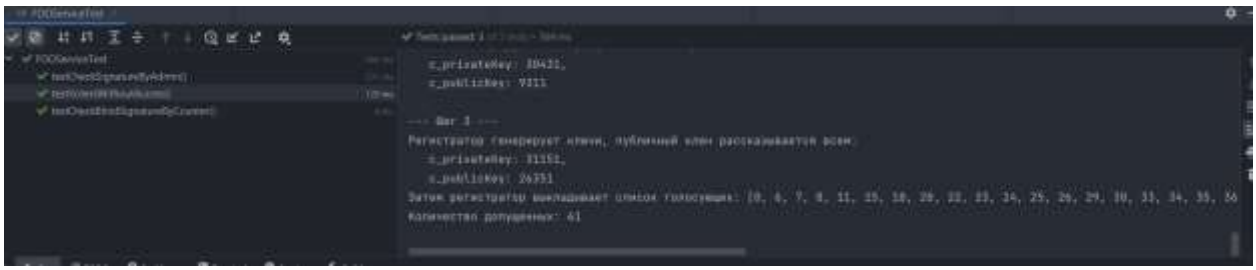

Рисунок 1 – Из 100 зарегистрированных допущены до выборов не все

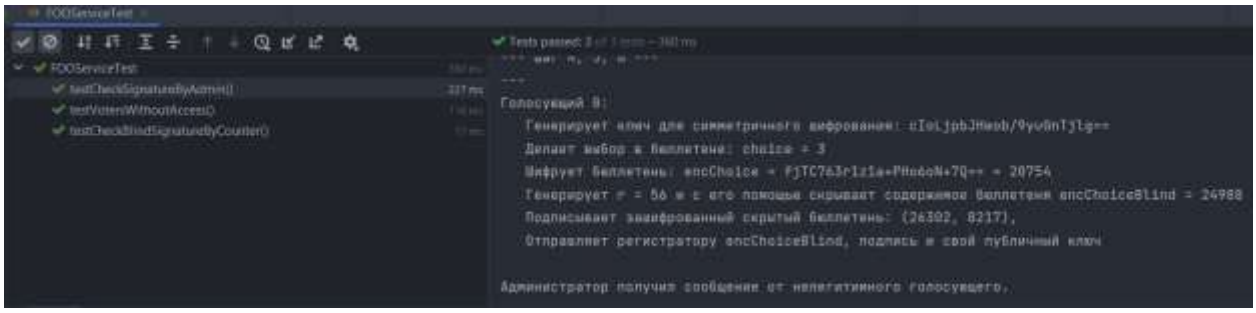

Рисунок 2 – Проверка подписи на бюллетене не прошла проверку

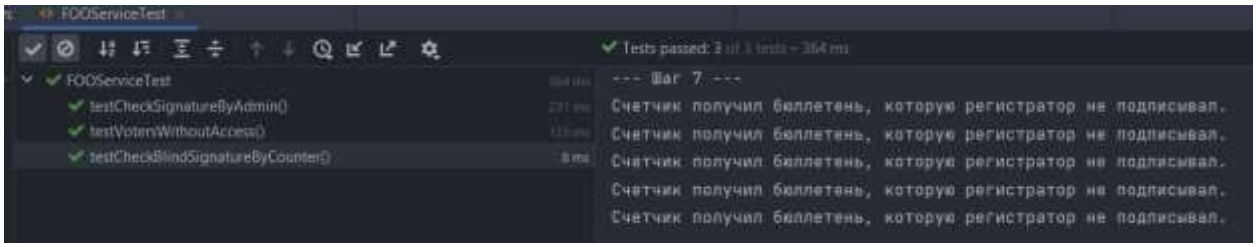

Рисунок 3 – 5 человек отправило счетчику бюллетень, которую регистратор не подписывал

На рисунках ниже представлен вывод программы для 10 голосующих и 3 избираемых.

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 8. FOO\out\artifacts\FOO_jar> java -jar
 FOO.jar 10 3
Количество голосующих: 10
Количество избираемых: 3

--- Шаг 1 ---
Голосующие сгенерировали ключи для асимметричного шифрования:
Голосующий 0:
    privateKey = 20827
    publicKey = 21763
Голосующий 1:
    privateKey = 5879
    publicKey = 7939
Голосующий 2:
    privateKey = 27337
    publicKey = 3273
Голосующий 3:
    privateKey = 22673
    publicKey = 5297
Голосующий 4:
    privateKey = 1103
    publicKey = 30767
Голосующий 5:
    privateKey = 29717
    publicKey = 53
Голосующий 6:
    privateKey = 31667
    publicKey = 1403
Голосующий 7:
    privateKey = 25471
    publicKey = 42031
Голосующий 8:
    privateKey = 11947
    publicKey = 2311
Голосующий 9:
    privateKey = 15329
    publicKey = 28769

--- Шаг 2 ---
```

Рисунок 4 – Шаг 1

```
--- Шаг 2 ---
Счетчик генерирует ключи, публичный ключ рассказывается всем:
   c_privateKey: 19897,
   c_publicKey: 3577

--- Шаг 3 ---
Регистратор генерирует ключи, публичный ключ рассказывается всем:
   c_privateKey: 18989,
   c_publicKey: 28409
Затем регистратор выкладывает список голосующих: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Рисунок 5 – Шаг 2, 3

```
--- Шаг 4, 5, 6 ---
---
Голосующий 0:
   Генерирует ключ для симметричного шифрования: Grt4ZTo4evkGuxvZFo4Fmw==
   Делает выбор в бюллетене: choice = 1
   Шифрует бюллетень: encChoice = oJj9bHVfJLtTwOyxTidbxw== = 9681
   Генерирует r = 16149 и с его помощью скрывает содержимое бюллетеня encChoiceBlind = 27470
   Подписывает зашифрованный скрытый бюллетень: (2556, 1302),
   Отправляет регистратору encChoiceBlind, подпись и свой публичный ключ

Регистратор:
   Принимает сообщение и проверяет, что оно подписано легитимным голосующим
   Подписывает вслепую сообщение: encChoiceBlindSigned = 1278
   Отправляет encChoiceBlindSigned голосующему

Голосующий раскрывает бюллетень с помощью числа r:
   encChoiceSigned = 8104
---
---
Голосующий 1:
   Генерирует ключ для симметричного шифрования: XfkVoRRwt1g+f29N9ookew==
   Делает выбор в бюллетене: choice = 2
   Шифрует бюллетень: encChoice = AHPdnb8T526qsu5h4N6QTw== = 28689
   Генерирует r = 28538 и с его помощью скрывает содержимое бюллетеня encChoiceBlind = 32682
   Подписывает зашифрованный скрытый бюллетень: (27727, 4083),
   Отправляет регистратору encChoiceBlind, подпись и свой публичный ключ

Регистратор:
   Принимает сообщение и проверяет, что оно подписано легитимным голосующим
   Подписывает вслепую сообщение: encChoiceBlindSigned = 16302
   Отправляет encChoiceBlindSigned голосующему

Голосующий раскрывает бюллетень с помощью числа r:
   encChoiceSigned = 12669
---
```

Рисунок 6 – Шаг 4, 5, 6 (вывод только для первых 2х голосовавших)

```
--- Шаг 7 ---
---
Голосующий 0 отправляет счетчику:
   encChoice = 9681
   encChoiceSigned = 8104

Счетчик:
   Проверяет, что encChoiceSigned действительно подписал регистратор
   Помещает encChoice в специальный список в открытом доступе после оговоренного времени
---
---
Голосующий 1 отправляет счетчику:
   encChoice = 28689
   encChoiceSigned = 12669

Счетчик:
   Проверяет, что encChoiceSigned действительно подписал регистратор
   Помещает encChoice в специальный список в открытом доступе после оговоренного времени
---
```

Рисунок 7 – Шаг 7 (вывод только для первых 2х голосовавших)

```
Голосованиие окончено. Счетчик опубликовал список: [JR8EldTbtQNY7kfy4S6j/g==, VNpn25MC20bUpbx
QwFzBiw==, FP85GOJZdkfSLagm3tlzGQ==, QNviPH51vYozAtnUfLeRFg==, sKwX/Z4q0F6sVD7SdJYDNg==, lu7A
lkFpxpT75odFh4GD+Q==, +CAU3dxgePvFqDx5FPUJlA==, 9J+VHI8CeKfMv532IA+fbw==, /w26OJ5t/5b+BoDD9Vw
n5Q==, Teofwf8TW7fnE9wvDZ+P8Q==]

--- Шаг 8 ---
Результаты голосования:
Избираемый #1: 3
Избираемый #2: 4
Избираемый #3: 3

Результаты голосования в процентах:
Избираемый #1: 30.00%
Избираемый #2: 40.00%
Избираемый #3: 30.00%

Победитель выборов: избираемый #2
```

Рисунок 8 – Результаты выборов

ПРИЛОЖЕНИЕ А

Листинг программы

```
import java.security.InvalidParameterException;

public class FOO {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Входные параметры отсутствуют");
            return;
        }
        if (args[0].equals("/help") || args[0].equals("h")) {
            System.out.println("""
                    Программе должны передаваться следующие параметры:
                    \t- число участников голосования
                    \t- число избираемых""");
            return;
        }
        if (args.length < 2) {
            System.out.println("Передано некорректное число
параметров.");
            return;
        }
        int voterCount, electedCount;
        try {
            voterCount = Integer.parseInt(args[0]);
            electedCount = Integer.parseInt(args[1]);
            if (voterCount < 1 || electedCount < 1) {
                throw new InvalidParameterException("Количество
голосующих и избираемых должно быть положительным числом.");
            }
        } catch (IndexOutOfBoundsException e) {
            System.out.println("Выход за пределы массива.");
            return;
        } catch (NumberFormatException e) {
            System.out.println("Ошибка при чтении входных
параметров.");
            return;
        } catch (InvalidParameterException e){
            System.out.println(e.getMessage());
            return;
        }
        FOOService service = new FOOService(voterCount, electedCount);
        service.step1();
    }
}

import java.awt.image.AreaAveragingScaleFilter;
import java.math.BigInteger;
import java.security.PublicKey;
import java.util.ArrayList;
import java.util.HashMap;

public class FOOService {
    private final int electedCount;
    private ArrayList<Voter> voters = new ArrayList<>();
```

```
    private ArrayList<Integer> elected = new ArrayList<>();
    private Counter counter;

    private Administrator admin;

    public FOOService(int voterCount, int electedCount) {
        for (int i = 0; i < voterCount; ++i) {
            voters.add(new Voter(i));
        }
        this.electedCount = electedCount;
        for (int i = 1; i <= electedCount; ++i) {
            elected.add(i);
        }
        System.out.printf("Количество    голосующих:    %d\nКоличество
избираемых: %d\n", this.voters.size(), this.electedCount);
    }

    public void step1() {
        System.out.println("\n--- Шаг 1 ---\nГолосующие сгенерировали
ключи для асимметричного шифрования:");
        for (int i = 0; i < voters.size(); ++i) {
            System.out.printf("""
                    Голосующий %d:
                        privateKey = %s
                        publicKey = %s
                    """,
//                                                                  i,
RSAService.convertPrivateKeyToString(voters.get(i).getPrivateKey()).su
bstring(0,                 40)                 +                 "...",
RSAService.convertPublicKeyToString(voters.get(i).getPublicKey()).subs
tring(0, 40) + "...");
                    i,                  voters.get(i).getPrivateKey(),
voters.get(i).getPublicKey());
        }
        this.step2();
    }

    private void step2() {
        this.counter = new Counter(electedCount);
        System.out.printf("""
                \n--- Шаг 2 ---
                Счетчик генерирует ключи, публичный ключ рассказывается
всем:
                   c_privateKey: %s,
                   c_publicKey: %s
                   """,                          counter.getPrivateKey(),
counter.getPublicKey());
        for (Voter voter : voters) {
            voter.setCounterKey(counter.getPublicKey());
        }
        this.step3();
    }

    private void step3() {
        this.admin = new Administrator();
        this.setVotersModule();
        admin.setCounterKey(counter.getPublicKey());
        counter.setAdminKey(admin.getPublicKey());
```

```
        for (Voter voter : voters) {
            voter.setAdminKey(admin.getPublicKey());
            admin.registrateVoter(voter);
        }
        System.out.printf("""
                          \n--- Шаг 3 ---
                          Регистратор генерирует ключи, публичный ключ
рассказывается всем:
                             c_privateKey: %s,
                             c_publicKey: %s
                          Затем    регистратор    выкладывает    список
голосующих: %s
                          """,
                  admin.getPrivateKey(),            admin.getPublicKey(),
admin.printListOfVoters());
        this.step4and5and6();
    }

    private void step4and5and6() {
        System.out.println("\n--- Шаг 4, 5, 6 ---");
        for (Voter voter : voters) {
            voter.generateSymmetricKey();
            voter.makeChoiceAndEncrypt(elected);
            voter.blindAndSign();
            System.out.printf("""
                              ---
                              Голосующий %d:
                                 Генерирует   ключ   для   симметричного
шифрования: %s
                                 Делает выбор в бюллетене: choice = %d
                                 Шифрует бюллетень: encChoice = %s = %d
                                 Генерирует  r  =  %d  и  с  его  помощью
скрывает содержимое бюллетеня encChoiceBlind = %d
                                 Подписывает    зашифрованный    скрытый
бюллетень: %s,
                                 Отправляет регистратору encChoiceBlind,
подпись и свой публичный ключ

                              """,
                      voter.getId(),

AESService.convertSecretKeyToString(voter.getAesKey()),
                      voter.getChoice(),
                      voter.getEncChoice(),
                      voter.getEncChoiceSHA(),
                      voter.getR(),
                      voter.getEncChoiceBlind(),
                      voter.getSignature());
            BigInteger              encChoiceBlindSigned              =
admin.sendBallot(voter.getEncChoiceBlind(),       voter.getSignature(),
voter.getPublicKey());
            if (encChoiceBlindSigned == null) {
                System.exit(1);
            }
            System.out.printf("""
                              Регистратор:
                                 Принимает сообщение и проверяет, что оно
подписано легитимным голосующим
```

```
                                Подписывает      вслепую      сообщение:
encChoiceBlindSigned = %d
                                Отправляет          encChoiceBlindSigned
голосующему

                            """,
                    encChoiceBlindSigned);
            voter.setEncChoiceBlindSigned(encChoiceBlindSigned);
            voter.unblind();
            System.out.printf("""
                            Голосующий раскрывает бюллетень с помощью
числа r:
                               encChoiceSigned = %d
                            ---
                            """,
                    voter.getEncChoiceSigned());
        }
        this.step7();
    }

    private void step7() {
        System.out.println("\n--- Шаг 7 ---");
        for (Voter voter : voters) {
//                  System.out.println("\n  i  =  "  +  voter.getId()  +
"\ngetEncChoice = " + voter.getEncChoice() + "\ngetEncChoiceSHA = " +
voter.getEncChoiceSHA()
//                                      +  "\ngetEncChoiceSigned  =  "  +
voter.getEncChoiceSigned() + "\ngetN = " +  voter.getN());
            counter.takeEncChoiceSigned(voter.getEncChoice(),
voter.getEncChoiceSHA(), voter.getEncChoiceSigned(), voter.getN());
            System.out.printf("""
                            ---
                            Голосующий %d отправляет счетчику:
                               encChoice = %d
                               encChoiceSigned = %d

                            Счетчик:
                               Проверяет,      что      encChoiceSigned
действительно подписал регистратор
                               Помещает encChoice в специальный список
в открытом доступе после оговоренного времени
                            ---
                            """,
                    voter.getId(),
                    voter.getEncChoiceSHA(),
                    voter.getEncChoiceSigned());
        }
        System.out.println("\nГолосованиие      окончено.      Счетчик
опубликовал список: " + counter.getEncChoices());
        this.step8();
    }

    private void step8() {
        System.out.println("\n--- Шаг 8 ---");
        for (Voter voter : voters) {

counter.decryptAndUpdateResults(voter.findInList(counter.getEncChoices
()), voter.getAesKey(), voter.getIv());
```

```
        }
        counter.publishResults();
    }

    private void setVotersModule() {
        for (Voter voter : voters) {
            voter.setN(admin.getN());
        }
    }
}

import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import java.math.BigInteger;
import java.security.*;
import java.util.ArrayList;
import java.util.Random;

public class Voter {
    private final BigInteger privateKey;
    private final BigInteger publicKey;
    private BigInteger counterKey;
    private BigInteger adminKey;
    private SecretKey aesKey;
    private IvParameterSpec iv;
    private BigInteger n;
    private BigInteger r;
    private final int id;
    private int choice;
    private String encChoice;
    private       BigInteger       encChoiceSHA,       encChoiceBlind,
encChoiceBlindSigned, encChoiceSigned;
    private Signature s;
    private BigInteger k, h; // для подписи

    public Voter(int id) {
        RSAService service = new RSAService();
        this.privateKey = service.getPrivateKey();
        this.publicKey = service.getPublicKey();
        this.id = id;
//        SecureRandom rnd = new SecureRandom();
//        this.n = BigInteger.probablePrime(30, rnd);
    }

    public BigInteger getPrivateKey() {
        return privateKey;
    }

    public BigInteger getPublicKey() {
        return publicKey;
    }

    public int getId() {
        return id;
    }
```

```
    public BigInteger getEncChoiceSHA() {
        return encChoiceSHA;
    }

    public String getEncChoice() {
        return encChoice;
    }

    public Signature getSignature() {
        return s;
    }

    public BigInteger getEncChoiceBlind() {
        return encChoiceBlind;
    }

    public BigInteger getEncChoiceBlindSigned() {
        return encChoiceBlindSigned;
    }

    public BigInteger getEncChoiceSigned() {
        return encChoiceSigned;
    }

    public BigInteger getN() {
        return n;
    }

    public SecretKey getAesKey() {
        return aesKey;
    }

    public IvParameterSpec getIv() {
        return iv;
    }

    public int getChoice() {
        return choice;
    }

    public BigInteger getR() {
        return r;
    }

    public void setCounterKey(BigInteger counterKey) {
        this.counterKey = counterKey;
    }

    public void setAdminKey(BigInteger adminKey) {
        this.adminKey = adminKey;
    }

    public           void           setEncChoiceBlindSigned(BigInteger
encChoiceBlindSigned) {
        this.encChoiceBlindSigned = encChoiceBlindSigned;
    }
```

```
    public void setN(BigInteger n) {
        this.n = n;
    }

    public void generateSymmetricKey() {
        try {
            AESService service = new AESService();
            this.aesKey = service.getPublicKey();
            this.iv = service.getIv();
        } catch (NoSuchAlgorithmException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public void makeChoiceAndEncrypt(ArrayList<Integer> elected) {
        Random rnd = new Random();
        this.choice = elected.get(rnd.nextInt(elected.size()));
        try {
            this.encChoice = AESService.encrypt(String.valueOf(choice),
aesKey, iv);
        } catch (NoSuchAlgorithmException | BadPaddingException |
IllegalBlockSizeException |
                InvalidAlgorithmParameterException                    |
InvalidKeyException | NoSuchPaddingException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
        this.encChoiceSHA = SHA.sha(encChoice).mod(n);
    }


    public void blindAndSign() {
        SecureRandom rnd = new SecureRandom();
        this.r = new BigInteger(n.bitLength(), rnd).mod(n);
        while (!r.gcd(n).equals(BigInteger.ONE)) {
            this.r = new BigInteger(n.bitLength(), rnd).mod(n);
        }
        encChoiceBlind    =    encChoiceSHA.multiply(r.modPow(adminKey,
n)).mod(n);
        this.signMessage();
    }

    public void signMessage() {
        if (encChoiceBlind == null) {
            System.out.println("Попытка подписать сообщение, которого
не существует.");
            System.exit(1);
        }
        SecureRandom rnd = new SecureRandom();
        this.k = new BigInteger(n.bitLength(), rnd).mod(n);
        while (!k.gcd(n).equals(BigInteger.ONE)) {
            this.k = new BigInteger(n.bitLength(), rnd).mod(n);
        }
        BigInteger invK = k.modInverse(n);
        this.h = invK.multiply(invK).negate().mod(n);
        BigInteger invTwo = BigInteger.TWO.modInverse(n);
        //BigInteger r = new BigInteger(n.bitLength(), rnd).mod(n);
```

```
        BigInteger invR = r.modInverse(n);
        this.s = new Signature(

invTwo.multiply(encChoiceBlind.multiply(invR).add(r)).mod(n),

k.multiply(invTwo).multiply(encChoiceBlind.multiply(invR).subtract(r))
.mod(n),
                h, n);
    }

    public void unblind() {
        this.encChoiceSigned                                                =
this.encChoiceBlindSigned.multiply(r.modInverse(n)).mod(n);
    }

    public int findInList(ArrayList<String> encChoices) {
        for (int i = 0; i < encChoices.size(); ++i) {
            if (encChoices.get(i).equals(encChoice)){
                return i;
            }
        }
        return -1;
    }
}

import java.math.BigInteger;

public class Signature {
    private final BigInteger S1, S2;
    private final BigInteger h, n;

    public  Signature(BigInteger  S1,  BigInteger  S2,  BigInteger  h,
BigInteger n) {
        this.S1 = S1;
        this.S2 = S2;
        this.h = h;
        this.n = n;
    }

    public BigInteger getS1() {
        return S1;
    }

    public BigInteger getS2() {
        return S2;
    }

    public BigInteger getH() {
        return h;
    }

    public BigInteger getN() {
        return n;
    }

    @Override
    public String toString() {
        return "(" + S1 + ", " + S2 + ")";
```

```
    }
}

import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class SHA {
    public static BigInteger sha(String message) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-1");
            byte[] messageDigest = md.digest(message.getBytes());
            return new BigInteger(1, messageDigest);
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }
}

import java.math.BigInteger;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.ArrayList;

public class Administrator {
    private final BigInteger privateKey, publicKey, n;
//    private final BigInteger privateKeyInt, publicKeyInt;
    private BigInteger counterKey;

    // private HashMap<PublicKey, Integer> voters = new HashMap<>();
    private ArrayList<VoterInfo> voters = new ArrayList<>();

    private static class VoterInfo {
        public BigInteger key;
        public int id;

        public VoterInfo(BigInteger key, int id) {
            this.key = key;
            this.id = id;
        }
    }

    public Administrator() {
        RSAService service = new RSAService();
        this.privateKey = service.getPrivateKey();
        this.publicKey = service.getPublicKey();
        this.n = service.getN();
//                                                this.privateKeyInt      =
SHA.sha(RSAService.convertPrivateKeyToString(privateKey));
//                                                 this.publicKeyInt      =
SHA.sha(RSAService.convertPublicKeyToString(publicKey));
    }

    public BigInteger getPrivateKey() {
        return privateKey;
    }

    public BigInteger getPublicKey() {
```

```
        return publicKey;
    }

    public BigInteger getN() {
        return n;
    }

    //    public BigInteger getPublicKeyInt() {
//        return publicKeyInt;
//    }

    //    public HashMap<PublicKey, Integer> getVoters() {
//        return voters;
//    }

    public void setCounterKey(BigInteger counterKey) {
        this.counterKey = counterKey;
    }

    public void registrateVoter(Voter voter) {
        voters.add(new VoterInfo(voter.getPublicKey(), voter.getId()));
    }

    public String printListOfVoters() {
        String res = "[";
        for (int i = 0; i < voters.size(); ++i) {
            if (i != voters.size() - 1) {
                res += voters.get(i).id + ", ";
            } else {
                res += voters.get(i).id + "]";
            }
        }
        return res;
    }

    public BigInteger sendBallot(BigInteger encChoiceBlind, Signature
signature, BigInteger publicKey) {
        if   (!findVoter(publicKey)   ||   !checkMessage(encChoiceBlind,
signature)) {
            System.out.println("Администратор получил сообщение от
нелегитимного голосующего.");
            return null;
        }
        return signBlind(encChoiceBlind, signature.getN());
    }

    private BigInteger signBlind(BigInteger encChoiceBlind, BigInteger
n) {
        return encChoiceBlind.modPow(this.privateKey, n);
    }

    private boolean checkMessage(BigInteger encChoiceBlind, Signature s)
{
        BigInteger s1 = s.getS1();
        BigInteger s2 = s.getS2();
        BigInteger                          check                          =
s1.multiply(s1).add(s.getH().multiply(s2.multiply(s2))).mod(s.getN());
        //System.out.println("check: " + check);
```

```
        return check.equals(encChoiceBlind);
    }

    private boolean findVoter(BigInteger publicKey) {
        for (VoterInfo vi : voters) {
            if (vi.key.equals(publicKey)) {
                return true;
            }
        }
        //System.out.println("Не нашлось голосующего.");
        return false;
    }
}

import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import java.math.BigInteger;
import java.security.*;
import java.util.ArrayList;

public class Counter {
    private final BigInteger privateKey;
    private final BigInteger publicKey;

    private ArrayList<String> encChoices = new ArrayList<>();
    private ArrayList<Integer> results = new ArrayList<>();

    private BigInteger adminKey;

    public Counter(int electedCount) {
        RSAService service = new RSAService();
        this.privateKey = service.getPrivateKey();
        this.publicKey = service.getPublicKey();
        for (int i = 0; i < electedCount; ++i) {
            results.add(0);
        }
    }

    public BigInteger getPrivateKey() {
        return privateKey;
    }

    public BigInteger getPublicKey() {
        return publicKey;
    }

    public ArrayList<String> getEncChoices() {
        return encChoices;
    }

    public void setAdminKey(BigInteger adminKey) {
        this.adminKey = adminKey;
    }
```

```
    public void takeEncChoiceSigned(String encChoice, BigInteger
encChoiceSHA, BigInteger encChoiceSigned, BigInteger n) {
        if (!checkMessage(encChoiceSHA, encChoiceSigned, n)) {
            System.out.println("Счетчик получил бюллетень, которую
регистратор не подписывал.");
            return;
        }
        encChoices.add(encChoice);
    }

    private boolean checkMessage(BigInteger encChoice, BigInteger
encChoiceSigned, BigInteger n) {
//       System.out.println("check: " + encChoiceSigned.modPow(adminKey,
n));
        return encChoice.equals(encChoiceSigned.modPow(adminKey, n));
    }

    public void decryptAndUpdateResults(int vIndex, SecretKey aesKey,
IvParameterSpec iv) {
        if (vIndex == -1) {
            System.out.println("Голосующий не смог найти себя в
списке.");
            return;
        }
        Integer choice = null;
        try {
            choice                                                  =
Integer.parseInt(AESService.decrypt(encChoices.get(vIndex),     aesKey,
iv)) - 1;
        } catch (NoSuchAlgorithmException | NoSuchPaddingException |
InvalidAlgorithmParameterException     |     InvalidKeyException     |
BadPaddingException | IllegalBlockSizeException e) {
            System.out.println("Ошибка в расшифровании одного из
голосов: " + e.getMessage());
        } catch (NumberFormatException e) {
            System.out.println("В качестве голоса получено не число.\n"
+ e.getMessage());
        }
        if (choice == null) {
            return;
        }
        results.set(choice, results.get(choice) + 1);
    }

    public void publishResults() {
        int winner = findWinner();
        System.out.println("Результаты голосования:");
        for (int i = 0; i < results.size(); ++i) {
            System.out.println("Избираемый #" + (i + 1) + ": " +
results.get(i));
        }
        System.out.println("\nРезультаты голосования в процентах:");
        for (int i = 0; i < results.size(); ++i) {
            System.out.println("Избираемый #" + (i + 1) + ": " +
String.format("%.2f", (double)results.get(i) / encChoices.size() * 100,
2) + "%");
        }
        if (drawn(results.get(winner - 1))){
```

```
            System.out.println("По итогом выборов победитель не
определен. Необходимо провести повторное голосавание.");
        } else {
            System.out.println("\nПобедитель выборов: избираемый #" +
(winner));
        }
    }

    private boolean drawn(Integer max) {
        int cnt = 0;
        for(int res : results) {
            if (res == max) {
                ++cnt;
            }
        }
        return cnt != 1;
    }

    private int findWinner() {
        int max = 0;
        int winner = -1;
        for(int i = 0; i < results.size(); ++i){
            if (results.get(i) > max) {
                max = results.get(i);
                winner = i + 1;
            }
        }
        return winner;
    }
}

import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import java.math.BigInteger;
import java.security.*;
import java.util.ArrayList;

public class Counter {
    private final BigInteger privateKey;
    private final BigInteger publicKey;

    private ArrayList<String> encChoices = new ArrayList<>();
    private ArrayList<Integer> results = new ArrayList<>();

    private BigInteger adminKey;

    public Counter(int electedCount) {
        RSAService service = new RSAService();
        this.privateKey = service.getPrivateKey();
        this.publicKey = service.getPublicKey();
        for (int i = 0; i < electedCount; ++i) {
            results.add(0);
        }
    }
```

```
    public BigInteger getPrivateKey() {
        return privateKey;
    }

    public BigInteger getPublicKey() {
        return publicKey;
    }

    public ArrayList<String> getEncChoices() {
        return encChoices;
    }

    public void setAdminKey(BigInteger adminKey) {
        this.adminKey = adminKey;
    }

    public void takeEncChoiceSigned(String encChoice, BigInteger
encChoiceSHA, BigInteger encChoiceSigned, BigInteger n) {
        if (!checkMessage(encChoiceSHA, encChoiceSigned, n)) {
            System.out.println("Счетчик получил бюллетень, которую
регистратор не подписывал.");
            return;
        }
        encChoices.add(encChoice);
    }

    private boolean checkMessage(BigInteger encChoice, BigInteger
encChoiceSigned, BigInteger n) {
//       System.out.println("check: " + encChoiceSigned.modPow(adminKey,
n));
        return encChoice.equals(encChoiceSigned.modPow(adminKey, n));
    }

    public void decryptAndUpdateResults(int vIndex, SecretKey aesKey,
IvParameterSpec iv) {
        if (vIndex == -1) {
            System.out.println("Голосующий не смог найти себя в
списке.");
            return;
        }
        Integer choice = null;
        try {
            choice                                                  =
Integer.parseInt(AESService.decrypt(encChoices.get(vIndex),     aesKey,
iv)) - 1;
        } catch (NoSuchAlgorithmException | NoSuchPaddingException |
InvalidAlgorithmParameterException     |     InvalidKeyException     |
BadPaddingException | IllegalBlockSizeException e) {
            System.out.println("Ошибка в расшифровании одного из
голосов: " + e.getMessage());
        } catch (NumberFormatException e) {
            System.out.println("В качестве голоса получено не число.\n"
+ e.getMessage());
        }
        if (choice == null) {
            return;
        }
        results.set(choice, results.get(choice) + 1);
```

```
    }

    public void publishResults() {
        int winner = findWinner();
        System.out.println("Результаты голосования:");
        for (int i = 0; i < results.size(); ++i) {
            System.out.println("Избираемый #" + (i + 1) + ": " +
results.get(i));
        }
        System.out.println("\nРезультаты голосования в процентах:");
        for (int i = 0; i < results.size(); ++i) {
            System.out.println("Избираемый #" + (i + 1) + ": " +
String.format("%.2f", (double)results.get(i) / encChoices.size() * 100,
2) + "%");
        }
        if (drawn(results.get(winner - 1))){
            System.out.println("По итогом выборов победитель не
определен. Необходимо провести повторное голосавание.");
        } else {
            System.out.println("\nПобедитель выборов: избираемый #" +
(winner));
        }
    }

    private boolean drawn(Integer max) {
        int cnt = 0;
        for(int res : results) {
            if (res == max) {
                ++cnt;
            }
        }
        return cnt != 1;
    }

    private int findWinner() {
        int max = 0;
        int winner = -1;
        for(int i = 0; i < results.size(); ++i){
            if (results.get(i) > max) {
                max = results.get(i);
                winner = i + 1;
            }
        }
        return winner;
    }
}

import javax.crypto.*;
import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.*;
import java.security.spec.EncodedKeySpec;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;

public class RSAService {
```

```
//    private final PublicKey publicKey;
//    private final PrivateKey privateKey;
    private final int keySize = 16;
    private BigInteger p, q, n, privateKey, publicKey;

    private static final String algorithm = "RSA";

    public BigInteger getPublicKey() {
        return publicKey;
    }

    public BigInteger getPrivateKey() {
        return privateKey;
    }

    public BigInteger getN() {
        return n;
    }

    public RSAService() {
        SecureRandom rnd = new SecureRandom();
        int size = keySize / 2;
        this.p = BigInteger.probablePrime(size, rnd);
        this.q = BigInteger.probablePrime(size, rnd);
        while (p.equals(q)){
            q = BigInteger.probablePrime(size, rnd);
        }
        this.n = p.multiply(q);
        BigInteger                          phi                          =
(p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));
        do {
            this.publicKey    =    new    BigInteger(phi.bitLength(),
rnd).mod(phi);
        } while(!this.publicKey.gcd(phi).equals(BigInteger.ONE));
        this.privateKey = publicKey.modInverse(phi);
//        KeyPairGenerator generator = null;
//        try {
//            generator = KeyPairGenerator.getInstance("RSA");
//        } catch (NoSuchAlgorithmException e) {
//            System.out.println(e.getMessage());
//            System.exit(1);
//        }
//        generator.initialize(keySize);
//        KeyPair pair = generator.generateKeyPair();
//        this.publicKey = pair.getPublic();
//        this.privateKey = pair.getPrivate();
    }

    public static String encrypt(String input, PublicKey publicKey) {
        try {
            Cipher cipher = Cipher.getInstance(algorithm);
            cipher.init(Cipher.ENCRYPT_MODE, publicKey);
            byte[]                      cipherText                      =
cipher.doFinal(input.getBytes(StandardCharsets.UTF_8));
            return Base64.getEncoder().encodeToString(cipherText);
        } catch (NoSuchPaddingException e) {
            System.out.println(e.getMessage());
```

```
        } catch (IllegalBlockSizeException | InvalidKeyException |
BadPaddingException | NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
        return null;
    }

    public static String decrypt(String cipherText, PrivateKey
privateKey) {
        try {
            Cipher cipher = Cipher.getInstance(algorithm);
            cipher.init(Cipher.DECRYPT_MODE, privateKey);
            byte[] plainText =
cipher.doFinal(Base64.getDecoder().decode(cipherText));
            return new String(plainText, StandardCharsets.UTF_8);
        } catch (NoSuchPaddingException e) {
            System.out.println(e.getMessage());
        } catch (IllegalBlockSizeException | NoSuchAlgorithmException |
BadPaddingException | InvalidKeyException e) {
            throw new RuntimeException(e);
        }
        return null;
    }

    public static String convertPublicKeyToString(PublicKey secretKey)
{
        byte[] rawData = secretKey.getEncoded();
        return Base64.getEncoder().encodeToString(rawData);
    }

    public static String convertPrivateKeyToString(PrivateKey
secretKey) {
        byte[] rawData = secretKey.getEncoded();
        return Base64.getEncoder().encodeToString(rawData);
    }

    public static PrivateKey convertStringToPrivateKey(String
encodedKey) {
        try {
            byte[] decodedKey = Base64.getDecoder().decode(encodedKey);
            KeyFactory keyFactory = KeyFactory.getInstance("RSA");
            EncodedKeySpec publicKeySpec = new
PKCS8EncodedKeySpec(decodedKey);
            return keyFactory.generatePrivate(publicKeySpec);
        } catch (NoSuchAlgorithmException e) {
            System.out.println(e.getMessage());
        } catch (InvalidKeySpecException e) {
            throw new RuntimeException(e);
        }
        return null;
    }

    public static PublicKey convertStringToPublicKey(String encodedKey)
{
        try {
            byte[] decodedKey = Base64.getDecoder().decode(encodedKey);
            KeyFactory keyFactory = KeyFactory.getInstance("RSA");
```

```
            EncodedKeySpec publicKeySpec = new
X509EncodedKeySpec(decodedKey);
            return keyFactory.generatePublic(publicKeySpec);
        } catch (NoSuchAlgorithmException e) {
            System.out.println(e.getMessage());
        } catch (InvalidKeySpecException e) {
            throw new RuntimeException(e);
        }
        return null;
    }
}

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class FOOServiceTest {
    @Test
    void testVotersWithoutAccess() {
        FOOService service = new FOOService(100, 3);
        service.setWithoutAccessTest(true);
        service.step1();
        int withAccess =
service.getAdmin().getVotersWithAccess().size();
        System.out.println("Количество допущенных: " + withAccess);
        assertTrue(withAccess > 0 && withAccess ==
service.getAdmin().getVoters().size());
    }

    @Test
    void testCheckSignatureByAdmin() {
        FOOService service = new FOOService(5, 3);
        service.setStep5Test();
        service.step1();
        assertTrue(service.checkErrorSignature());
    }

    @Test
    void testCheckBlindSignatureByCounter() {
        FOOService service = new FOOService(5, 3);
        service.setStep7Test();
        service.step1();
        assertTrue(service.getCounter().checkErrorSign());
    }
}
```