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 обмена ключами

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 обмена ключами Нидхэма – Шрёдера.

2 Теоретические сведения

В 1978 году Роджер Нидхэм и Майкл Шредер разработали протокол, который позволяет безопасно обмениваться данными в децентрализованной сети.

Протокол состоит из 5 шагов:

1) участник A отправляет запрос доверительному центру C, в котором он говорит, что нужно установить сеанс связи с другим участником B. Запрос содержит следующие параметры: имена участников A и B и случайное число R_A, которое является уникальным для данного сеанса;

2) C в ответ отправляет сообщение, которые было зашифровано на секретном ключе K_{AC}. K_{AC} является общим ключом для участника A и доверительного центра C. Сообщение содержит сеансовый ключ k, а также его копию, которая зашифрована на общем секретном ключе K_{BC} участника B и центра C;

3) участник A отправляет B эту копию, после чего второй проводит расшифрование и получает ключ k;

4) участник B отправляет A зашифрованное на ключе s сообщение со случайным числом R_B. Это нужно для того, чтобы удостовериться, что A владеет ключом k и показать, что сам B знает его;

5) участник A отправляет B сообщение с числом $R_B - 1$, которое зашифровано на k. В результате два участника получают общий сеансовый ключ.

Алгоритм работы протокола Нидхэма – Шрёдера.

Вход: длина ключа n, которая может быть равна 128, 192, 256.

Выход: генерация сеансового ключа k и обмен ключом между пользователями A и B.

Шаг 1. $A \to C$: A, B, R_A.

Шаг 2. C $\to A$: $E_{K_{AC}}(R_A, B, k, E_{K_{BC}}(k, A))$.

Шаг 3. $A \to B$: $E_{K_{BC}}(k, A)$.

Шаг 4. $B \to A$: $E_k(R_B)$.

Шаг 5. $A \to B$: $E_k(R_B - 1)$. Если A передал корректное $R_B - 1$ пользователю B, то алгоритм закончен, оба участника получили сеансовый ключ.

Достоинством этого протокола является то, что он гарантирует взаимную аутентификацию двух участников A и B, а также то, что он владеет свойством подтверждения ключа, то есть таким свойством, при помощи которого один из участников протокола удостоверяется, что другой участник точно владеет секретными ключами, которые были получены в протоколе.

Недостатком протокола является то, что существует возможность использования старых сеансовых ключей. Если злоумышленник M получит доступ к старому ключу, то он сможет предпринять успешное вскрытие. Ему нужно только записать сообщение Алисы Бобу на шаге 3. Тогда, имея ключ, он может выдать себя за Алису.

Рассмотрим пример работы протокола. Есть 2 пользователя: Антон и Богдан. Сначала Антон случайно выбирает $N_A = 121$ и отправляет доверительному центру свое имя, имя второго пользователя и 121. Пусть $K_{AC} = a123$, $K_{BC} = b321$. Доверительный центр генерирует сеансовый ключ $s = qwerty$. Затем он отправляет Антону сообщение «$E_{a123}(121, \text{Богдан}, qwerty, E_{b321}(qwerty, \text{Антон}))$». Антон отправляет Богдану «$E_{b321}(qwerty, \text{Антон})$». Богдан расшифровывает сообщение и получает сеансовый ключ $s = qwerty$. После этого он случайно выбирает N_B. Пусть N_B=1024. Богдан отправляет Антону сообщение «$E_{qwerty}(1024)$», а второй, если действительно знает s, отправляет «$E_{qwerty}(1023)$». Установлен сеансовый ключ $s = qwerty$.

3 Результаты работы

3.1 Сведения о программе

Программа была реализована на языке программирования Java. В ней есть 4 класса: $Main$, $Participant$, $Server$ и $NeedhamSchroederService$.

В классе $Main$ происходит инициализация входного параметра – длины ключа, который будет сгенерирован для сеанса связи между двумя участниками, и запуск самого протокола.

Класс $Participant$ – класс участника протокола. Для инициализации объекта $Participant$ указывается имя участника и личный ключ K_A. В классе реализованы следующие методы:

- `private void generateRandomInt()` – вычисление случайного числа участника сеанса связи;

Класс $Server$ – класс доверительного центра. Для инициализации объекта данного класса указывается длина ключа $keyLength$. В классе описаны следующие методы:

- `private void setKeys()` – установка личных ключей для двух пользователей, сеансового ключа и вектора инициализации для шифрования.
- `public static SecretKey generateKey(int n)` – генерация ключа длины n в шифровании AES.
- `public static IvParameterSpec generateIv()` – генерация инициализационного вектора.
- `public static String encrypt(String algorithm, String input, SecretKey key, IvParameterSpec iv)` – шифрование строки шифром AES.
- `public static String decrypt(String algorithm, String cipherText, SecretKey key, IvParameterSpec iv)` – расшифрование строки шифром AES.
- `public static String convertSecretKeyToString(SecretKey secretKey)` – конвертации ключа в строку.

- `public static SecretKey convertStringToSecretKeyto(String encodedKey)` – конвертация строки в ключ.

Класс *NeedhamSchroederService* – реализация работы протокола. При создании объекта этого типа происходит создание доверительного центра (объекта *Server*) и 2 участников протокола – Алисы и Боба. После чего происходит запуск обмена ключами. В классе описаны следующие методы

- `private void runProtocol()` – имплементация самого алгоритма.
- `private String getEncrypted(String info, boolean isAlice)` – шифрование данных для Алисы или Боба.
- `private String getDecrypted(String encInfo, boolean isAlice)` – расшифрование данных для Алисы или Боба.

3.2 Тестирование программы

Для реализации модульного тестирования были написаны тестовые классы *ServerTest* и *NeedhamSchroederServiceTest*.

В классе *ServerTest* содержится 2 метода для тестирования шифрования, расшифрования данных, а также конвертации ключа в строку и обратно. Результат отработки теста представлен на рисунке 1.

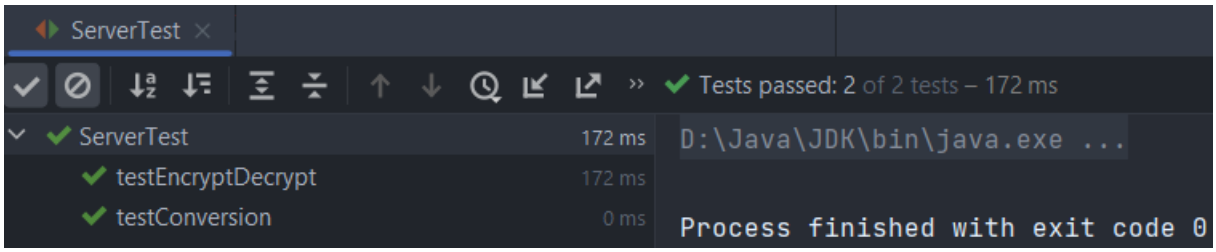

Рисунок 1 – Запуск *ServerTest*

В классе *NeedhamSchroederServiceTest* содержится 2 тестовых метода. Первый – для проверки вызова исключения *InvalidParameterException*, если передана некорректная длина ключа. Второй – проверка работы протокола обмена ключами, где смотрится, что после работы протокола у двоих пользователей совпадает сеансовый ключ. Результат отработки тестов представлен на рисунке 2.

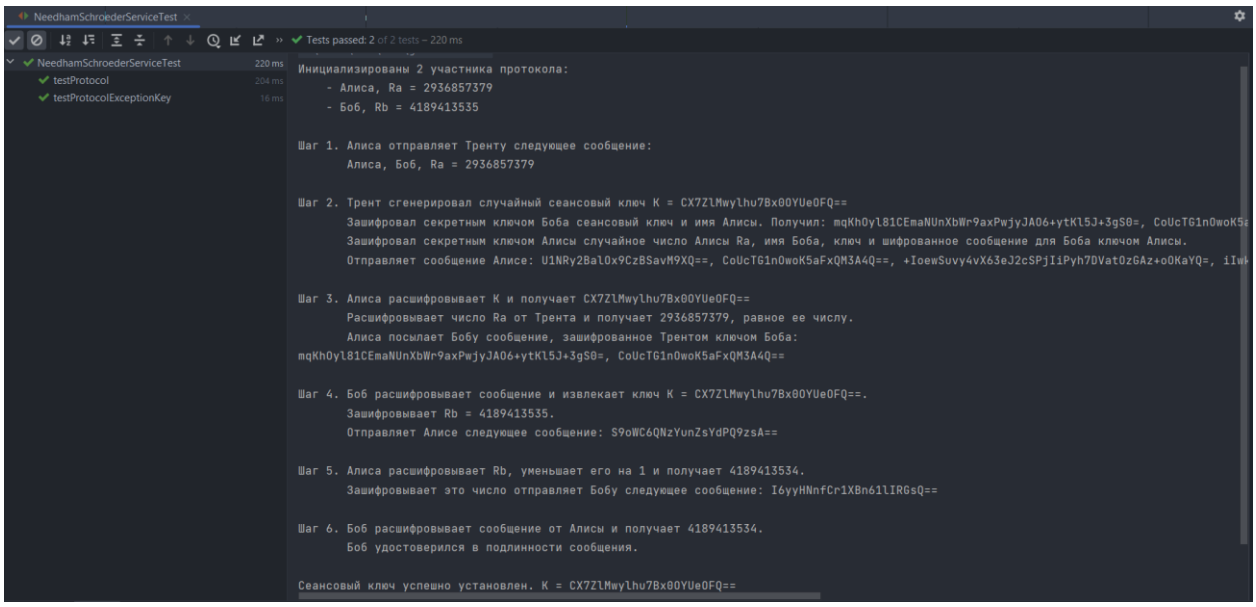

Рисунок 2 – Запуск *NeedhamSchroederServiceTest*

На рисунках 3-7 представлено тестирование программы.

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 2. Needham-Schroeder\out
\artifacts\Needham_Schroeder_jar> java -jar '.\Task 2. Needham-Schroeder.jar'
Входные параметры отсутсвуют
```

Рисунок 3 – Негативное тестирование (входной параметр не указан)

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 2. Needham-Schroeder\out\artifacts\Needham_
Schroeder_jar> java -jar '.\Task 2. Needham-Schroeder.jar' asd
Некорректное значение длины ключа, должно быть передано число!
```

Рисунок 4 – Негативное тестирование (входной параметр не является числом)

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 2. Needham-Schroeder\out\artifacts\Nee
dham_Schroeder_jar> java -jar '.\Task 2. Needham-Schroeder.jar' 100
Exception in thread "main" java.lang.IllegalArgumentException: Некорректный ввод. Допустимая
 длина ключа: 128, 192, 256
        at Main.main(Main.java:28)
```

Рисунок 5 – Негативное тестирование (передана некорректная длина ключа)

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 2. Needham-Schroeder\out\artifacts\Needham_Schroeder_java -jar '.\Task 2. Needham-Schroeder.jar' 128
Инициализированы 2 участника протокола:
        - Алиса, Ra = 3466003182
        - Боб, Rb = 2959440387

Шаг 1. Алиса отправляет Тренту следующее сообщение:
       Алиса, Боб, Ra = 3466003182

Шаг 2. Трент сгенерировал случайный сеансовый ключ K = uMP+DXT7kyy0Gkevv4HS1w==
       Зашифровал секретным ключом Боба сеансовый ключ и имя Алисы. Получил: ptctUgpiOFzqX22P7ZxxIHjj54OSebTme9YNmbr353o=, kbjiRGSa3QXBSYGCroGFYQ==
       Зашифровал секретным ключом Алисы случайное число Алисы Ra, имя Боба, ключ и шифрованное сообщение для Боба ключом Алисы.
       Отправляет сообщение Алисе: nVRquvm9DqXrp6mo53GfpA==, kbjiRGSa3QXBSYGCroGFYQ==, CwHIrf0CGoT6gMMG2vZgQMDgybbPueFeY3rmZ8NqEoE=, KQ93cNHK7C1o9qyU7FY3ezu
5s1IsXKBgJGA9O5TgSqCLmuk6IjYoafy3v/XnORwd, EDuBQ7e5RjD/iS39PBaqzjxm807fFFmfGzOKygF6/40=

Шаг 3. Алиса расшифровывает K и получает uMP+DXT7kyy0Gkevv4HS1w==
       Расшифровывает число Ra от Трента и получает 3466003182, равное ее числу.
       Алиса посылает Бобу сообщение, зашифрованное Трентом ключом Боба:
ptctUgpiOFzqX22P7ZxxIHjj54OSebTme9YNmbr353o=, kbjiRGSa3QXBSYGCroGFYQ==

Шаг 4. Боб расшифровывает сообщение и извлекает ключ K = uMP+DXT7kyy0Gkevv4HS1w==.
       Зашифровывает Rb = 2959440387.
       Отправляет Алисе следующее сообщение: 7ydmLqW21LgV2IKlh8EAqA==

Шаг 5. Алиса расшифровывает Rb, уменьшает его на 1 и получает 2959440386.
       Зашифровывает это число отправляет Бобу следующее сообщение: 34v4meBjgIXR67DPc5wP+w==
Шаг 6. Боб расшифровывает сообщение от Алисы и получает 2959440386.
       Боб удостоверился в подлинности сообщения.

Сеансовый ключ успешно установлен. K = uMP+DXT7kyy0Gkevv4HS1w==
```

Рисунок 6 – Положительное тестирование (входной параметр 128)

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 2. Needham-Schroeder\out\artifacts\Needham_Schroeder_jar> java -jar '.\Task 2. Needham-Schroeder.jar'
256
Инициализированы 2 участника протокола:
        - Алиса, Ra = 2156232069
        - Боб, Rb = 3054547845

Шаг 1. Алиса отправляет Тренту следующее сообщение:
       Алиса, Боб, Ra = 2156232069

Шаг 2. Трент сгенерировал случайный сеансовый ключ K = WSsrrbjt88R0IcDuRvQhnY/X6Q4n+hlYhxAEsv5vVtQ=
       Зашифровал секретным ключом Боба сеансовый ключ и имя Алисы. Получил: A/oAjO2MCKKxX4+6lEYFDunFXW7SWx9NzurJeDd7CnOUKR/IFl1O5jQ0DSm8zs68, KxSkZyHf2V2ks
1HDQ/tcmw==
       Зашифровал секретным ключом Алисы случайное число Алисы Ra, имя Боба, ключ и шифрованное сообщение для Боба ключом Алисы.
       Отправляет сообщение Алисе: gSDz7XspUzCqrGc6cl1LXg==, KxSkZyHf2V2ks1HDQ/tcmw==, uP0Cpaju2XEKyl1Vtsg+PSUAOgBHJmPnfB2K5fxNqIUNsqg3ONRNBMFHvJoCWlfr, s9Q
ohyqAlWu74VKo4Qaezpj2iF8NoubFPsxRXKOf2QV08SS8aYwHPriQw0H+f/8Czltjn5JT537Czh2Ea03YVSMr+zFc9Hj33WoqRO4G6/Q=, jRdQYWwoWWb9pIDOdMXGWQor1qZ7C1hhdwzUJTK+yHI=

Шаг 3. Алиса расшифровывает K и получает WSsrrbjt88R0IcDuRvQhnY/X6Q4n+hlYhxAEsv5vVtQ=
       Расшифровывает число Ra от Трента и получает 2156232069, равное ее числу.
       Алиса посылает Бобу сообщение, зашифрованное Трентом ключом Боба:
A/oAjO2MCKKxX4+6lEYFDunFXW7SWx9NzurJeDd7CnOUKR/IFl1O5jQ0DSm8zs68, KxSkZyHf2V2ks1HDQ/tcmw==

Шаг 4. Боб расшифровывает сообщение и извлекает ключ K = WSsrrbjt88R0IcDuRvQhnY/X6Q4n+hlYhxAEsv5vVtQ=.
       Зашифровывает Rb = 3054547845.
       Отправляет Алисе следующее сообщение: dCypse3WsjNdwMlXhxeK7w==

Шаг 5. Алиса расшифровывает Rb, уменьшает его на 1 и получает 3054547844.
       Зашифровывает это число отправляет Бобу следующее сообщение: D/bqwmun0OdKYReoQ5mmzw==

Шаг 6. Боб расшифровывает сообщение от Алисы и получает 3054547844.
       Боб удостоверился в подлинности сообщения.

Сеансовый ключ успешно установлен. K = WSsrrbjt88R0IcDuRvQhnY/X6Q4n+hlYhxAEsv5vVtQ=
```

Рисунок 7 – Положительное тестирование (входной параметр 256)

ПРИЛОЖЕНИЕ А

Листинг программы

```
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

public class Main {
    public static void main(String[] args) throws
InvalidAlgorithmParameterException, NoSuchPaddingException,
IllegalBlockSizeException, NoSuchAlgorithmException,
BadPaddingException, InvalidKeyException {
        if (args.length == 0) {
            System.out.println("Входные параметры отсутсвуют");
            return;
        }
        if (args[0].equals("/help")) {
            System.out.println("""
                    Программе должен передаваться 1 параметр:
                    \t- длина ключа в битах (128, 192, 256)""");
            return;
        }
        int keyLength;
        try {
            keyLength = Integer.parseInt(args[0]);
        } catch (NumberFormatException e) {
            System.out.println("Некорректное значение длины ключа,
должно быть передано число!");
            return;
        }
        if (keyLength != 128 && keyLength != 192 && keyLength != 256) {
            throw new IllegalArgumentException("Некорректный ввод.
Допустимая длина ключа: 128, 192, 256");
        }
        NeedhamSchroederService service = new
NeedhamSchroederService(keyLength);
    }
}

import javax.crypto.SecretKey;
import java.math.BigInteger;
import java.util.Random;

public class Participant {
    private final String name;
    private BigInteger randomInt;
    private SecretKey key, sessionKey;

    public String getName() {
        return name;
    }

    public BigInteger getRandomInt() {
```

```
        return randomInt;
    }

    public SecretKey getSessionKey() {
        return sessionKey;
    }

    public SecretKey getKey() {
        return key;
    }

    public void setSessionKey(SecretKey sessionKey) {
        this.sessionKey = sessionKey;
    }

    public Participant(String name, SecretKey key) {
        this.name = name;
        this.key = key;
        this.generateRandomInt();
    }

    private void generateRandomInt() {
        Random rand = new Random();
        this.randomInt = new BigInteger(32, rand);
    }
}

import javax.crypto.*;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.Base64;

public class Server {
    private SecretKey sessionKey, keyA, keyB;
    private IvParameterSpec iv;
    private final int keyLength;

    public SecretKey getSessionKey() {
        return sessionKey;
    }

    public int getKeyLength() {
        return keyLength;
    }

    public SecretKey getKeyA() {
        return keyA;
    }

    public SecretKey getKeyB() {
        return keyB;
    }

    public IvParameterSpec getIv() {
```

```
        return iv;
    }

    public Server(int keyLength) throws NoSuchAlgorithmException {
        this.keyLength = keyLength;
        this.setKeys();
    }

    private void setKeys() throws NoSuchAlgorithmException {
        this.sessionKey = generateKey(this.keyLength);
        this.keyA = generateKey(this.keyLength);
        this.keyB = generateKey(this.keyLength);
        this.iv = generateIv();
    }

    public static SecretKey generateKey(int n) throws
NoSuchAlgorithmException {
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init(n);
        return keyGenerator.generateKey();
    }

    public static IvParameterSpec generateIv() {
        byte[] iv = new byte[16];
        new SecureRandom().nextBytes(iv);
        return new IvParameterSpec(iv);
    }

    public static String encrypt(String algorithm, String input,
SecretKey key, IvParameterSpec iv) throws NoSuchPaddingException,
NoSuchAlgorithmException, InvalidAlgorithmParameterException,
InvalidKeyException, BadPaddingException, IllegalBlockSizeException {
        Cipher cipher = Cipher.getInstance(algorithm);
        cipher.init(Cipher.ENCRYPT_MODE, key, iv);
        byte[] cipherText = cipher.doFinal(input.getBytes());
        return Base64.getEncoder().encodeToString(cipherText);
    }

    public static String decrypt(String algorithm, String cipherText,
SecretKey key, IvParameterSpec iv) throws NoSuchPaddingException,
NoSuchAlgorithmException, InvalidAlgorithmParameterException,
InvalidKeyException, BadPaddingException, IllegalBlockSizeException {
        Cipher cipher = Cipher.getInstance(algorithm);
        cipher.init(Cipher.DECRYPT_MODE, key, iv);
        byte[] plainText =
cipher.doFinal(Base64.getDecoder().decode(cipherText));
        return new String(plainText);
    }

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

    public static SecretKey convertStringToSecretKeyto(String
encodedKey) {
        byte[] decodedKey = Base64.getDecoder().decode(encodedKey);
```

```
        return new SecretKeySpec(decodedKey, 0, decodedKey.length,
"AES");
    }
}


import javax.crypto.*;
import java.math.BigInteger;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

public class NeedhamSchroederService {
    private final Participant alice, bob;
    private final Server server;

    public Participant getAlice() {
        return alice;
    }

    public Participant getBob() {
        return bob;
    }

    public NeedhamSchroederService(int keyLength) throws
NoSuchAlgorithmException, InvalidAlgorithmParameterException,
NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException,
InvalidKeyException {
        this.server = new Server(keyLength);
        this.alice = new Participant("Алиса", this.server.getKeyA());
        this.bob = new Participant("Боб", this.server.getKeyB());
        System.out.println("Инициализированы 2 участника протокола: \n"
                + "\t- Алиса, Ra = " + this.alice.getRandomInt() + "\n"
                + "\t- Боб, Rb = " + this.bob.getRandomInt() + "\n");
        this.runProtocol();
    }

    private void runProtocol() throws
InvalidAlgorithmParameterException, NoSuchPaddingException,
IllegalBlockSizeException, NoSuchAlgorithmException,
BadPaddingException, InvalidKeyException {
        System.out.println("Шаг 1. Алиса отправляет Тренту следующее
сообщение: \n" + "       "
                + this.alice.getName() + ", "
                + this.bob.getName() +
                ", Ra = " + this.alice.getRandomInt() + "\n");

        String aliceRanomIntEncForA =
getEncrypted(this.alice.getRandomInt().toString(), true);
        String bobNameEncForA = getEncrypted(this.bob.getName(), true);
        String sessionKeyEncForA =
getEncrypted(Server.convertSecretKeyToString(this.server.getSessionKey
()), true);
        String sessionKeyEncForB =
getEncrypted(Server.convertSecretKeyToString(this.server.getSessionKey
()), false);
        String aliceNameEncForB = getEncrypted(this.bob.getName(),
true);
```

```
        String                encForASessionKeyEncForB                =
getEncrypted(sessionKeyEncForB, true);
        String encForAAliceNameEncForB = getEncrypted(aliceNameEncForB,
true);
        System.out.println("Шаг   2.   Трент   сгенерировал   случайный
сеансовый ключ K = "
                +
Server.convertSecretKeyToString(this.server.getSessionKey()) + "\n" + "
"
                + "Зашифровал секретным ключом Боба сеансовый ключ и имя
Алисы. Получил: " + sessionKeyEncForB + ", " + aliceNameEncForB + "\n"
+ "       "
                + "Зашифровал секретным ключом Алисы случайное число
Алисы Ra, имя Боба, ключ и шифрованное сообщение для Боба ключом Алисы."
+ "\n" + "        "
                + "Отправляет сообщение Алисе: "
                + aliceRanomIntEncForA + ", "
                + bobNameEncForA + ", "
                + sessionKeyEncForA + ", "
                + encForASessionKeyEncForB + ", "
                + encForAAliceNameEncForB + "\n");

        String decryptedKForA = getDecrypted(sessionKeyEncForA, true);

this.alice.setSessionKey(Server.convertStringToSecretKeyto(decryptedKF
orA));
        String                  decryptedRandomIntForA                  =
getDecrypted(aliceRanomIntEncForA, true);
        if                        (!this.alice.getRandomInt().equals(new
BigInteger(decryptedRandomIntForA))) {
            System.out.println("Протокол завершен на шаге 3, так как
Алиса получила некорректный Ra от Трента." +
                    "Изначальный Ra = " + this.alice.getRandomInt() +
", Ra от Трента = " + decryptedRandomIntForA);
            return;
        }
        System.out.print("Шаг 3. Алиса расшифровывает K и получает " +
decryptedKForA + "\n" + "       " +
                "Расшифровывает число Ra от Трента и получает " +
decryptedRandomIntForA + ", равное ее числу." + "\n" + "       ");
        sessionKeyEncForB   =   getDecrypted(encForASessionKeyEncForB,
true);
        aliceNameEncForB = getDecrypted(encForAAliceNameEncForB, true);
        System.out.println("Алиса     посылает     Бобу     сообщение,
зашифрованное Трентом ключом Боба:\n"
                + sessionKeyEncForB + ", " + aliceNameEncForB + "\n");

        String decryptedKForB = getDecrypted(sessionKeyEncForB, false);

this.bob.setSessionKey(Server.convertStringToSecretKeyto(decryptedKFor
B));
        String                    bobRandomIntEncForA                   =
Server.encrypt("AES/CBC/PKCS5Padding",
this.bob.getRandomInt().toString(),          this.bob.getSessionKey(),
this.server.getIv());
        System.out.println("Шаг  4.  Боб  расшифровывает  сообщение  и
извлекает ключ K = " + decryptedKForB + ".\n" + "       "
```

```
                + "Зашифровывает Rb = " + this.bob.getRandomInt() +
".\n" + "       "
                + "Отправляет Алисе следующее сообщение: " +
bobRandomIntEncForA + "\n");

        BigInteger decryptedBobInt = new
BigInteger(Server.decrypt("AES/CBC/PKCS5Padding", bobRandomIntEncForA,
this.alice.getSessionKey(), this.server.getIv()));
        decryptedBobInt = decryptedBobInt.subtract(BigInteger.ONE);
        String encNewBobInt = Server.encrypt("AES/CBC/PKCS5Padding",
decryptedBobInt.toString(), this.alice.getSessionKey(),
this.server.getIv());
        System.out.println("Шаг 5. Алиса расшифровывает Rb, уменьшает
его на 1 и получает " + decryptedBobInt + ".\n" + "       "
                + "Зашифровывает это число отправляет Бобу следующее
сообщение: " + encNewBobInt + "\n");

        BigInteger newBobInt = new
BigInteger(Server.decrypt("AES/CBC/PKCS5Padding", encNewBobInt,
this.bob.getSessionKey(), this.server.getIv()));
        if
(this.bob.getRandomInt().subtract(BigInteger.ONE).equals(newBobInt)) {
            System.out.println("Шаг 6. Боб расшифровывает сообщение от
Алисы и получает " + newBobInt + ".\n" + "       "
                    + "Боб удостоверился в подлинности сообщения.\n");
            if (!decryptedKForA.equals(decryptedKForB)) {
                System.out.println("Произошла ошибка в работе
протоколе, сеансовый ключ у Алисы и Боба не совпадает.");
                return;
            }
            System.out.println("Сеансовый ключ успешно установлен. K =
" + Server.convertSecretKeyToString(this.alice.getSessionKey()));
        }
    }

    private String getEncrypted(String info, boolean isAlice) throws
InvalidAlgorithmParameterException, NoSuchPaddingException,
IllegalBlockSizeException, NoSuchAlgorithmException,
BadPaddingException, InvalidKeyException {
        if (isAlice) {
            return Server.encrypt("AES/CBC/PKCS5Padding", info,
this.alice.getKey(), this.server.getIv());
        } else {
            return Server.encrypt("AES/CBC/PKCS5Padding", info,
this.bob.getKey(), this.server.getIv());
        }
    }

    private String getDecrypted(String encInfo, boolean isAlice) throws
InvalidAlgorithmParameterException, NoSuchPaddingException,
IllegalBlockSizeException, NoSuchAlgorithmException,
BadPaddingException, InvalidKeyException {
        if (isAlice) {
            return Server.decrypt("AES/CBC/PKCS5Padding", encInfo,
this.alice.getKey(), this.server.getIv());
        } else {
            return Server.decrypt("AES/CBC/PKCS5Padding", encInfo,
this.bob.getKey(), this.server.getIv());
```

```
        }
    }
}

import junit.framework.TestCase;
import org.junit.Assert;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

public class ServerTest extends TestCase {
    public void testEncryptDecrypt() throws NoSuchAlgorithmException,
InvalidAlgorithmParameterException,            NoSuchPaddingException,
IllegalBlockSizeException, BadPaddingException, InvalidKeyException {
        String input = "baeldung";
        SecretKey key = Server.generateKey(128);
        IvParameterSpec ivParameterSpec = Server.generateIv();
        String algorithm = "AES/CBC/PKCS5Padding";
        String  cipherText  =  Server.encrypt(algorithm,  input,  key,
ivParameterSpec);
        String plainText = Server.decrypt(algorithm, cipherText, key,
ivParameterSpec);
        Assert.assertEquals(input, plainText);
    }

    public void testConversion() throws NoSuchAlgorithmException {
        SecretKey encodedKey = Server.generateKey(128);
        String                      encodedString                      =
Server.convertSecretKeyToString(encodedKey);
        SecretKey                      decodeKey                      =
Server.convertStringToSecretKeyto(encodedString);
        Assert.assertEquals(encodedKey, decodeKey);
    }
}

import junit.framework.TestCase;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

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

public class NeedhamSchroederServiceTest extends TestCase {
    public void testProtocolExceptionKey() {
        java.security.InvalidParameterException thrown = assertThrows(
                java.security.InvalidParameterException.class,
                () -> new NeedhamSchroederService(100),
                "Ожидалось          исключение          в          new
NeedhamSchroederService(100), но его не было."
        );
```

```
        assertTrue(thrown.getMessage().contains("Wrong keysize"));
    }

    public void testProtocol() throws
InvalidAlgorithmParameterException, NoSuchPaddingException,
IllegalBlockSizeException, NoSuchAlgorithmException,
BadPaddingException, InvalidKeyException {
        NeedhamSchroederService service = new
NeedhamSchroederService(128);
        assertEquals(service.getAlice().getSessionKey(),
service.getBob().getSessionKey());
    }
}
```