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 передачи секретного ключа по открытому каналу**

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 протоколы передачи секретного ключа по открытому каналу Encrypted Key Exchange (EKE) на базе алгоритма Эль-Гамаля.

**2 Теоретические сведения**

Безопасность схемы Эль-Гамаля основана на трудоемкости вычисления дискретных логарифмов в конечном поле. Для генерации пары ключе сначала выбирается простое число и 2 случайных числа, и , где – первообразный корень по модулю , а . Затем вычисляется . Открытым ключом являются . И , и можно сделать общими для группы пользователей. Закрытым ключом является .

Протокол обмена зашифрованными ключами (Encrypted Key Exchange, EKE) был разработан Стивом Белловином и Майклом Мерритом. Он обеспечивает безопасность и проверку подлинности в компьютерных сетях, новым образом используя и симметричную криптографию, и криптографию с открытым ключом: общий секретный ключ используется для шифрования генерированного случайным образом открытого ключа. В протоколе Алиса и Боб имеют общий пароль .

Алгоритм работы протокола EKE на базе Эль-Гамаля.

Вход: простое число и первообразный корень .

Выход: установление сеансового ключа .

Шаг 1. Алиса выбирает свой закрытый ключ и считает открытый ключ . После чего посылает Бобу незашифрованное сообщение:

Шаг 2. Боб генерирует случайный сеансовый ключ .Затем он выбирает случайное число (закрытый ключ) (для алгоритма Эль-Гамаля, независимо от других случайных чисел, выбираемых для ), и сообщение, которое он посылает Алисе, выглядит так:

Шаг 3. Алиса расшифровывает сообщение и получает ключ как . Она генерирует случайную строку , шифрует с помощью ключа и посылает результат:

Шаг 4. Боб расшифровывает сообщение, получая . Он генерирует другую случайную строку, , шифрует обе строки ключом и посылает Алисе результат:

Шаг 5. Алиса расшифровывает сообщение, получая . Если строка , полученная от Боба, это та самая строка, которую она послала Бобу на шаге 3, она, используя , шифрует строку и посылает Бобу:

Шаг 6. Боб расшифровывает сообщение, получая . Если строка , полученная от Алисы, это та самая строка, которую он послал ей на шаге 4, то протокол завершен. Теперь обе стороны могут обмениваться информацией, используя в качестве сеансового ключа.

Ева, сидя между Алисой и Бобом, знает только и несколько сообщений, зашифрованных ключом . В других протоколах Ева могла бы попробовать угадать и затем проверить свои предположения. В рассматриваемом протоколе Ева не может проверять свои предположения, не вскрыв при этого и алгоритм с открытым ключом ( и ). Тогда, если и выбираются случайным образом, то эта проблема будет непреодолимой.

Ответная часть протокола, соответствующая шагам 3-6, обеспечивает подтверждение. Шаги 3-5 доказывают Алисе, что Боб знает ключ, этапы 4-6 доказывают Бобу, что Алиса знает ключ. В протоколе для решения этой же задачи используется обмен метками времени.

**3 Результаты работы**

**3.1 Сведения о программе**

Программа была реализована на языке программирования Java. В ней есть 4 класса: , , и .

В классе происходит инициализация входного параметра – длины простого числа , который будет сгенерирован для схемы Эль-Гамаля. Также этот параметр будет являться ограничением для длины сеансового ключа.

Класс – класс участника протокола. Для инициализации объекта указывается имя участника, простое число , первообразный корень и объект класcа . В классе реализованы следующие методы:

- public void generateSessionKey(BigInteger p) – генерация сеансового ключа;

- public void setSessionKey(BigInteger sessionKeyInt) – установка сеансового ключа.

- private void setRandomPrivateKey() – установка закрытого ключа для схемы Эль-Гамаля.

- public String getEncrypted(String info, SecretKey key) – шифрование данных по ключу.

- public String getDecrypted(String encInfo, SecretKey key) – расшифрование данных по ключу.

- public void generateString() – генерация случайной строки.

Класс – класс с описанием методом для шифрования и расшифрования. При инициализации объекта данного класса генерируется общий ключ и инициализационный вектор. В классе описаны следующие методы:

* public static SecretKey generateKey(int keyLength) – генерация ключа длины в шифровании .
* public static IvParameterSpec generateIv() – генерация инициализационного вектора.
* public static String encrypt(String algorithm, String input, SecretKey key, IvParameterSpec iv) – шифрование строки шифром .
* public static String decrypt(String algorithm, String cipherText, SecretKey key, IvParameterSpec iv) – расшифрование строки шифром .
* public static String convertSecretKeyToString(SecretKey secretKey) – конвертации ключа в строку.
* public static SecretKey convertStringToSecretKeyto(String encodedKey) – конвертация строки в ключ.

Класс – реализация работы протокола. Для инициализации объекта передается простое число . При инициализации случайным образом вычисляется первообразный корень и создаются объект класса и 2 участника протокола – Алиса и Боб. После чего происходит запуск работы протокола. Каждому шагу алгоритма выше соответствует собственный метод. Помимо этих методов в классе описаны следующие методы:

* public static BigInteger generatePrimitiveRoot(BigInteger n) – генерация примитива по модулю . В данном методы выбирается случайное число до . И начиная от него в цикле происходит поиск примитивного корня.
* public static boolean isPrimitive(BigInteger prime, BigInteger n, ArrayList<BigInteger> orders) – проверка, является число примитивом по утверждению, описанному в пункте 2.
* public static ArrayList<BigInteger> getDivisors(BigInteger num) – метод для получения списка делителей числа.

**3.2 Тестирование программы**

Для реализации модульного тестирования были написаны тестовые классы и .

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

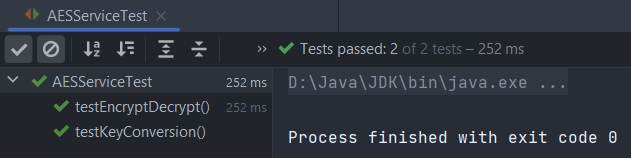


Рисунок 1 – Запуск

В классе содержится 2 тестовых метода. Первый – для проверки вызова исключения , если для создания объекта передано составное число. Второй – проверка работы протокола , где смотрится, что после работы протокола у двоих пользователей совпадает сеансовый ключ. Результат отработки тестов представлен на рисунке 2.

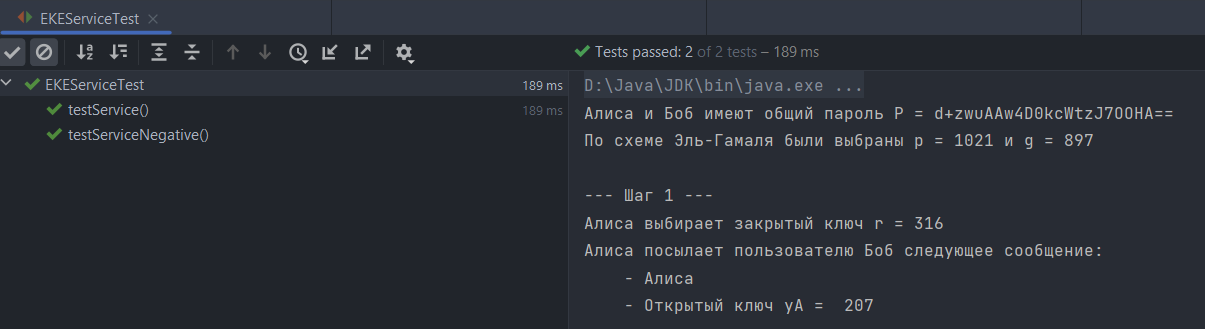


Рисунок 2 – Запуск

На рисунках 3-7 представлено тестирование программы.



Рисунок 3 – Негативное тестирование (входной параметр не указан)

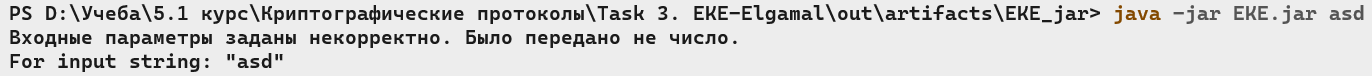


Рисунок 4 – Негативное тестирование (входной параметр не является числом)



Рисунок 5 – Положительное тестирование (входной параметр 25), шаги 1-3

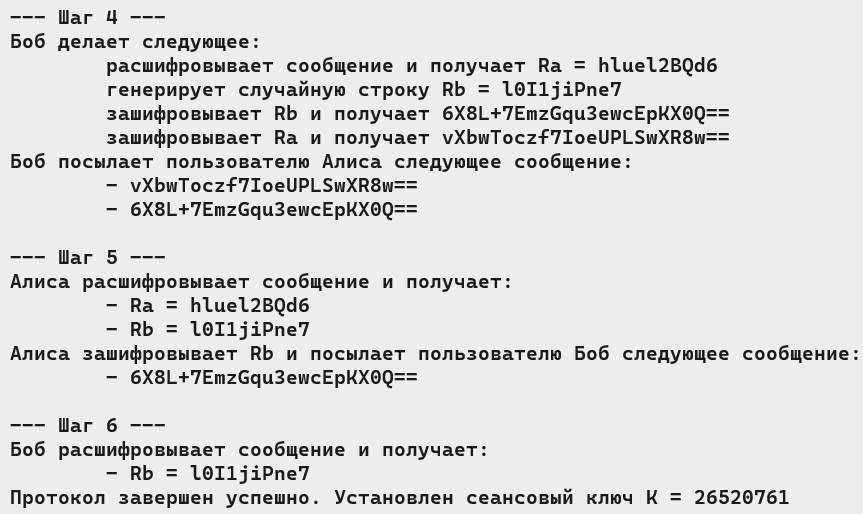


Рисунок 6 – – Положительное тестирование (входной параметр 25), шаги 4-6

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

import javax.crypto.BadPaddingException;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.NoSuchPaddingException;

import java.math.BigInteger;

import java.security.InvalidAlgorithmParameterException;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

public class EKE {

public static void main(String[] args) throws NoSuchAlgorithmException, InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException, InvalidKeyException {

if (args.length == 0) {

System.out.println("Входные параметры отсутсвуют");

return;

}

if (args[0].equals("/help")) {

System.out.println("""

Программе должен передаваться следующий параметр:

\t- длина простого числа в битах""");

return;

}

BigInteger p;

try {

p = BigInteger.ONE.shiftLeft(Integer.parseInt(args[0])).nextProbablePrime();

} catch (NumberFormatException e) {

System.out.println("Входные параметры заданы некорректно. Было передано не число.\n" + e.getMessage());

return;

}

int keyLength = 128;

EKEService service = new EKEService(p);

}

}

import javax.crypto.BadPaddingException;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.NoSuchPaddingException;

import java.math.BigInteger;

import java.security.\*;

import java.util.ArrayList;

public class EKEService {

public BigInteger p, g;

Participant alice, bob;

public Participant getBob() {

return bob;

}

public Participant getAlice() {

return alice;

}

public EKEService(BigInteger p) throws NoSuchAlgorithmException, InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException, InvalidKeyException {

this.p = p;

this.g = generatePrimitiveRoot(p);

AESService aes = new AESService();

this.alice = new Participant("Алиса", p, g, aes);

this.bob = new Participant("Боб", p, g, aes);

System.out.printf("%s и %s имеют общий пароль P = %s\n", alice.getName(), bob.getName(), AESService.convertSecretKeyToString(aes.getPublicKey()));

System.out.printf("По схеме Эль-Гамаля были выбраны p = %s и g = %s\n", p, g);

this.step1();

}

private void step1() throws NoSuchAlgorithmException, InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException, InvalidKeyException {

System.out.println("\n--- Шаг 1 ---");

System.out.printf("%s выбирает закрытый ключ r = %d\n", alice.getName(), alice.getPrivateElGamalKey());

System.out.printf("""

%s посылает пользователю %s следующее сообщение:

\t- %s

\t- Открытый ключ yA = %s\n""", alice.getName(), bob.getName(), alice.getName(), alice.getY());

this.step2(alice.getName(), alice.getY());

}

private void step2(String alice, BigInteger y) throws NoSuchAlgorithmException, InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException, InvalidKeyException {

System.out.println("\n--- Шаг 2 ---");

bob.generateSessionKey(p);

System.out.printf("%s делает следующее:\n", bob.getName());

System.out.printf("\tгенерирует сеансовый ключ К = %s\n", bob.getSessionKeyInt());

System.out.printf("\tвыбирает закрытый ключ R = %d\n", bob.getPrivateElGamalKey());

BigInteger kyR = bob.getSessionKeyInt().multiply(y.modPow(bob.getPrivateElGamalKey(), p)).mod(p);

System.out.printf("\tвычисляет K \* yA^R mod p = %d\n", kyR);

String encBobPublicKey = bob.getEncrypted(bob.getY().toString(), bob.getService().getPublicKey());

System.out.printf("\tзашифровывает свой открытый ключ %d и получает %s\n", bob.getY(), encBobPublicKey);

String encKYR = bob.getEncrypted(kyR.toString(), bob.getService().getPublicKey());

System.out.printf("\tзашифровывает K \* yA^R mod p и получает %s\n", encKYR);

System.out.printf("""

%s посылает пользователю %s следующее сообщение:

\t- %s

\t- %s\n""", bob.getName(), this.alice.getName(), encBobPublicKey, encKYR);

this.step3(encBobPublicKey, encKYR);

}

private void step3(String encBobPublicKey, String encKYR) throws InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException, InvalidKeyException {

System.out.println("\n--- Шаг 3 ---");

BigInteger decBobPublicKey = new BigInteger(alice.getDecrypted(encBobPublicKey, alice.getService().getPublicKey()));

BigInteger decKYR = new BigInteger(alice.getDecrypted(encKYR, alice.getService().getPublicKey()));

System.out.printf("%s делает следующее:\n", alice.getName());

System.out.printf("""

\tрасшировывает сообщение и получает:

\t\t- %d

\t\t- %d\n""", decBobPublicKey, decKYR);

BigInteger k = decKYR.multiply((decBobPublicKey.modPow(alice.getPrivateElGamalKey(), p)).modInverse(p)).mod(p);

alice.setSessionKey(k);

System.out.printf("\tвычисляет ключ К и получает %d\n", k);

alice.generateString();

System.out.printf("\tгенерирует случайную строку Ra = %s\n", alice.getRandomString());

String encRandomString = alice.getEncrypted(alice.getRandomString(), alice.getSessionKey());

System.out.printf("\tзашифровывает случайную строку Ra и получает %s\n", encRandomString);

System.out.printf("""

%s посылает пользователю %s следующее сообщение:

\t- %s\n""", alice.getName(), bob.getName(), encRandomString);

step4(encRandomString);

}

private void step4(String encRandomAliceString) throws InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException, InvalidKeyException {

System.out.println("\n--- Шаг 4 ---");

String decRandomAliceString = bob.getDecrypted(encRandomAliceString, bob.getSessionKey());

bob.generateString();

String encBobString = bob.getEncrypted(bob.getRandomString(), bob.getSessionKey());

String encAliceString = bob.getEncrypted(decRandomAliceString, bob.getSessionKey());

System.out.printf("""

%s делает следующее:

\tрасшифровывает сообщение и получает Ra = %s

\tгенерирует случайную строку Rb = %s

\tзашифровывает Rb и получает %s

\tзашифровывает Ra и получает %s\n""", bob.getName(), decRandomAliceString, bob.getRandomString(), encBobString, encAliceString);

System.out.printf("""

%s посылает пользователю %s следующее сообщение:

\t- %s

\t- %s\n""", bob.getName(), alice.getName(), encAliceString, encBobString);

step5(encAliceString, encBobString);

}

private void step5(String encAliceString, String encBobString) throws InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException, InvalidKeyException {

System.out.println("\n--- Шаг 5 ---");

String decAliceString = alice.getDecrypted(encAliceString, alice.getSessionKey());

String decBobString = alice.getDecrypted(encBobString, alice.getSessionKey());

System.out.printf("""

%s расшифровывает сообщение и получает:

\t- Ra = %s

\t- Rb = %s\n""", alice.getName(), decAliceString, decBobString);

if (!decAliceString.equals(alice.getRandomString())) {

System.out.println("Полученная строка Ra отличается от изначальной. Выход из алгоритма.");

return;

}

String encBobStringByAlice = alice.getEncrypted(decBobString, alice.getSessionKey());

System.out.printf("""

%s зашифровывает Rb и посылает пользователю %s следующее сообщение:

\t- %s\n""", alice.getName(), bob.getName(), encBobStringByAlice);

step6(encBobStringByAlice);

}

private void step6(String encBobStringByAlice) throws InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException, InvalidKeyException {

System.out.println("\n--- Шаг 6 ---");

String decBobString = bob.getDecrypted(encBobStringByAlice, bob.getSessionKey());

System.out.printf("""

%s расшифровывает сообщение и получает:

\t- Rb = %s\n""", bob.getName(), decBobString);

if (!decBobString.equals(bob.getRandomString())) {

System.out.println("Полученная строка Rb отличается от изначальной. Выход из алгоритма.");

return;

}

System.out.printf("Протокол завершен успешно. Установлен сеансовый ключ K = %d\n\n", alice.getSessionKeyInt());

}

private BigInteger generatePrimitiveRoot(BigInteger n) {

if (!n.isProbablePrime(100)) {

throw new IllegalArgumentException("Переданное число не является простым");

}

SecureRandom rand = new SecureRandom();

BigInteger prime = new BigInteger(n.bitLength(), rand).mod(n);

ArrayList<BigInteger> orders = getDivisors(n.subtract(BigInteger.ONE));

for (; ; ) {

if (prime.compareTo(n) >= 0) {

prime = BigInteger.ONE;

}

if (isPrimitive(prime, n, orders)) {

return prime;

}

prime = prime.add(BigInteger.ONE);

}

}

private boolean isPrimitive(BigInteger prime, BigInteger n, ArrayList<BigInteger> orders) {

if (prime.compareTo(BigInteger.ONE) < 0 || prime.compareTo(n) >= 0) {

return false;

}

for (BigInteger each : orders) {

if (prime.modPow(each, n).compareTo(BigInteger.ONE) == 0) {

if (each.compareTo(n.subtract(BigInteger.ONE)) == 0) {

return true;

}

break;

}

}

return false;

}

private ArrayList<BigInteger> getDivisors(BigInteger num) {

ArrayList<BigInteger> divisors = new ArrayList<>();

for (BigInteger i = BigInteger.ONE; i.compareTo(num.divide(BigInteger.TWO).add(BigInteger.ONE)) < 0; i = i.add(BigInteger.ONE)) {

if (num.mod(i).compareTo(BigInteger.ZERO) == 0) {

divisors.add(i);

}

}

divisors.add(num);

return divisors;

}

}

import javax.crypto.BadPaddingException;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import java.math.BigInteger;

import java.security.InvalidAlgorithmParameterException;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.security.SecureRandom;

import java.util.Base64;

import java.util.Random;

public class Participant {

private final String name;

private final BigInteger p, g;

private BigInteger y, privateElGamalKey;

private AESService service;

private BigInteger sessionKeyInt;

private SecretKey sessionKey;

private final String algorithm = "AES/CBC/PKCS5Padding";

private String randomString;

public String getName() {

return name;

}

public BigInteger getY() {

return y;

}

public AESService getService() {

return service;

}

public BigInteger getPrivateElGamalKey() {

return privateElGamalKey;

}

public BigInteger getSessionKeyInt() {

return sessionKeyInt;

}

public SecretKey getSessionKey() {

return sessionKey;

}

public String getRandomString() {

return randomString;

}

public void generateSessionKey(BigInteger p) {

SecureRandom rnd = new SecureRandom();

do {

this.sessionKeyInt = new BigInteger(p.bitLength(), rnd).mod(p);

} while (this.sessionKeyInt.equals(BigInteger.ZERO));

setSessionKey(this.sessionKeyInt);

}

public void setSessionKey(BigInteger sessionKeyInt) {

if (this.sessionKeyInt == null) {

this.sessionKeyInt = sessionKeyInt;

}

byte[] bytesEncoded = Base64.getEncoder().encode(String.valueOf(sessionKeyInt).getBytes());

byte[] keyBytes = new byte[16];

System.arraycopy(bytesEncoded, 0, keyBytes, 0, bytesEncoded.length);

this.sessionKey = new SecretKeySpec(keyBytes, 0, keyBytes.length, "AES");

}

public Participant(String name, BigInteger p, BigInteger g, AESService service) {

this.name = name;

this.service = service;

this.p = p;

this.g = g;

this.setRandomPrivateKey();

this.y = this.g.modPow(privateElGamalKey, this.p);

}

private void setRandomPrivateKey() {

Random rand = new Random();

do {

this.privateElGamalKey = new BigInteger(this.p.bitLength(), rand).mod(p);

} while (this.privateElGamalKey.compareTo(BigInteger.ONE) <= 0

|| this.privateElGamalKey.compareTo(p.subtract(BigInteger.ONE)) >= 0);

}

public String getEncrypted(String info, SecretKey key) throws InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException, InvalidKeyException {

return AESService.encrypt(algorithm, info, key, this.service.getIv());

}

public String getDecrypted(String encInfo, SecretKey key) throws InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, NoSuchAlgorithmException, BadPaddingException, InvalidKeyException {

return AESService.decrypt(algorithm, encInfo, key, this.service.getIv());

}

public void generateString() {

int leftLimit = 48; // цифра '0'

int rightLimit = 122; // буква 'z'

int targetStringLength = 10;

Random random = new Random();

this.randomString = random.ints(leftLimit, rightLimit + 1)

.filter(i -> (i <= 57 || i >= 65) && (i <= 90 || i >= 97))

.limit(targetStringLength)

.collect(StringBuilder::new, StringBuilder::appendCodePoint, StringBuilder::append)

.toString();

}

}

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 AESService {

private final SecretKey publicKey;

private final IvParameterSpec iv;

private final int keyLength = 128;

public SecretKey getPublicKey() {

return publicKey;

}

public IvParameterSpec getIv() {

return iv;

}

public AESService() throws NoSuchAlgorithmException {

this.publicKey = generateKey(this.keyLength);

this.iv = generateIv();

}

public static SecretKey generateKey(int keyLength) throws NoSuchAlgorithmException {

KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");

keyGenerator.init(keyLength);

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 NoSuchAlgorithmException, BadPaddingException, IllegalBlockSizeException, InvalidAlgorithmParameterException, InvalidKeyException, NoSuchPaddingException {

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 convertStringToSecretKey(String encodedKey) {

byte[] decodedKey = Base64.getDecoder().decode(encodedKey);

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

}

}

import javax.crypto.BadPaddingException;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.NoSuchPaddingException;

import java.math.BigInteger;

import java.security.InvalidAlgorithmParameterException;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

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

class EKEServiceTest {

@org.junit.jupiter.api.Test

void testService() throws NoSuchAlgorithmException, InvalidAlgorithmParameterException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException, InvalidKeyException {

EKEService service = new EKEService(new BigInteger("1021"));

assertEquals(service.getAlice().getSessionKey(), service.getBob().getSessionKey());

}

@org.junit.jupiter.api.Test

void testServiceNegative() {

java.lang.IllegalArgumentException thrown = assertThrows(

java.lang.IllegalArgumentException.class,

() -> new EKEService(new BigInteger("1024")),

"Ожидалось исключение в new EKEService(1021), но его не было."

);

assertEquals("Переданное число не является простым", thrown.getMessage());

}

}

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;

class AESServiceTest {

@org.junit.jupiter.api.Test

void testEncryptDecrypt() throws InvalidAlgorithmParameterException, IllegalBlockSizeException, NoSuchPaddingException, NoSuchAlgorithmException, BadPaddingException, InvalidKeyException {

AESService aes = new AESService();

String input = "testblabla";

SecretKey key = aes.getPublicKey();

IvParameterSpec ivParameterSpec = AESService.generateIv();

String algorithm = "AES/CBC/PKCS5Padding";

String cipherText = AESService.encrypt(algorithm, input, key, ivParameterSpec);

String plainText = AESService.decrypt(algorithm, cipherText, key, ivParameterSpec);

Assert.assertEquals(input, plainText);

}

@org.junit.jupiter.api.Test

void testKeyConversion() throws NoSuchAlgorithmException {

AESService aes = new AESService();

SecretKey encodedKey = aes.getPublicKey();

String encodedString = AESService.convertSecretKeyToString(encodedKey);

SecretKey decodeKey = AESService.convertStringToSecretKey(encodedString);

Assert.assertEquals(encodedKey, decodeKey);

}

}