В ходе выполнения заданий требовалось реализовать различные методы (де)шифрования информации из интерфейса SecurityService для программы шифровальщика. Классом, который послужил для реализации всех этих методов, послужил новый класс SecurityServiceImpl.

По условию заданий требовалось реализовать шифрование на основе комбинирования алгоритмов шифрования AES+RSA(+IV), где IV – вектор инициализации, который служит для помощи более мощного шифрования при работе алгоритма AES; сам алгоритм AES служит для (де)шифрования информации при помощи одного ключа; при помощи RSA создается два ключа – приватный и публичный – благодаря которым, шифруется уже сам ключ от AES.

Так как мы используем RSA в работе шифровальщика, то были сгенерированы заранее публичный и приватный ключи при помощи консоли:

ssh-keygen -t rsa -b 4096.

**Задание 1.**

В первом задании требовалось реализовать методы encryptMessage и decryptMessage (см. Листинг 1), для (де)шифрования текста. На вход метода encryptMessage (шифрование) подаются сообщение с интерфейса ввода и файл открытого ключа. Выходным значением будет строка, состоящая из зашифрованного сообщения дополнительно закодированного в Base64 для читаемости и удобства передачи. Соответственно функция decryptMessage (дешифрование) принимает закодированную Base64 строку и файл закрытого ключа.

@Override  
public String encryptMessage(String message, File publicKeyFile) throws IOException, GeneralSecurityException {  
 *// читаем публичный ключ* PublicKey publicKey = SshKeyUtils.*getPublicKey*(publicKeyFile).getJCEPublicKey();  
  
 *// Генерация случайного вектора инициализации* byte[] iv = generateIV();  
  
 *// Генерация секретного ключа AES* SecretKey secretKey = generateSecKey();  
  
 *// Шифрование сообщения с использованием AES* Cipher aesCipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");  
 aesCipher.init(Cipher.*ENCRYPT\_MODE*, secretKey, new IvParameterSpec(iv));  
 byte[] encryptedMessage = aesCipher.doFinal(message.getBytes());  
  
 *// Шифрование ключа с использованием RSA* Cipher rsaCipher = Cipher.*getInstance*("RSA/ECB/PKCS1Padding");  
 rsaCipher.init(Cipher.*ENCRYPT\_MODE*, publicKey);  
 byte[] encryptedKey = rsaCipher.doFinal(secretKey.getEncoded());  
  
 *// Соединение зашифрованного сообщения, ключа и вектора инициализации в одну строку* byte[] result = new byte[iv.length + encryptedKey.length + encryptedMessage.length];  
  
 copy(iv, encryptedKey, encryptedMessage, result);  
   
 return Base64.*getEncoder*().encodeToString(result);  
}  
  
@Override  
public String decryptMessage(String message, File privateKeyFile) throws IOException, GeneralSecurityException, InvalidPassphraseException {  
 *// чтение приватного ключа* PrivateKey privateKey = SshKeyUtils.*getPrivateKey*(privateKeyFile,  
 "").getPrivateKey().getJCEPrivateKey();  
  
 *// Разбиение зашифрованного сообщения, ключа и вектора инициализации на отдельные строки* byte[] answer = Base64.*getDecoder*().decode(message);  
 byte[] iv = Arrays.*copyOfRange*(answer, 0, 16);  
 byte[] encryptedKey = Arrays.*copyOfRange*(answer,16, 528);  
 byte[] encryptedMessage = Arrays.*copyOfRange*(answer, 528, answer.length);  
  
 *// Дешифрование ключа с использованием RSA* Cipher rsaCipher = Cipher.*getInstance*("RSA/ECB/PKCS1Padding");  
 rsaCipher.init(Cipher.*DECRYPT\_MODE*, privateKey);  
 byte[] decryptedKey = rsaCipher.doFinal(encryptedKey);  
 SecretKey secretKey = new SecretKeySpec(decryptedKey, "AES");  
  
 *// Дешифрование сообщения с использованием AES* Cipher aesCipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");  
 aesCipher.init(Cipher.*DECRYPT\_MODE*, secretKey, new IvParameterSpec(iv));  
 byte[] decryptedMessage = aesCipher.doFinal(encryptedMessage);  
  
 *// Возврат дешифрованного сообщения* return new String(decryptedMessage);  
}

Листинг 1 – реализация методов (де)шифрования текста.

Как происходит шифрование:

1. Достаем публичный ключ из переданных параметров.
2. Генерируем вектор инициализации IV.
3. Следом генерируется ключ для алгоритма AES.
4. Шифруем переданный текст при помощи AES – получаем encyptedMessage.
5. Следом шифруем ключ AES при помощи алгоритма RSA и публичного ключа – получаем encryptedKey.
6. Создаем окончательную запись в виде IV+encryptedKey+encryptedMessage и преобразуем ее закодированную Base64 строку.

Результат работы представлен на рисунке 1.

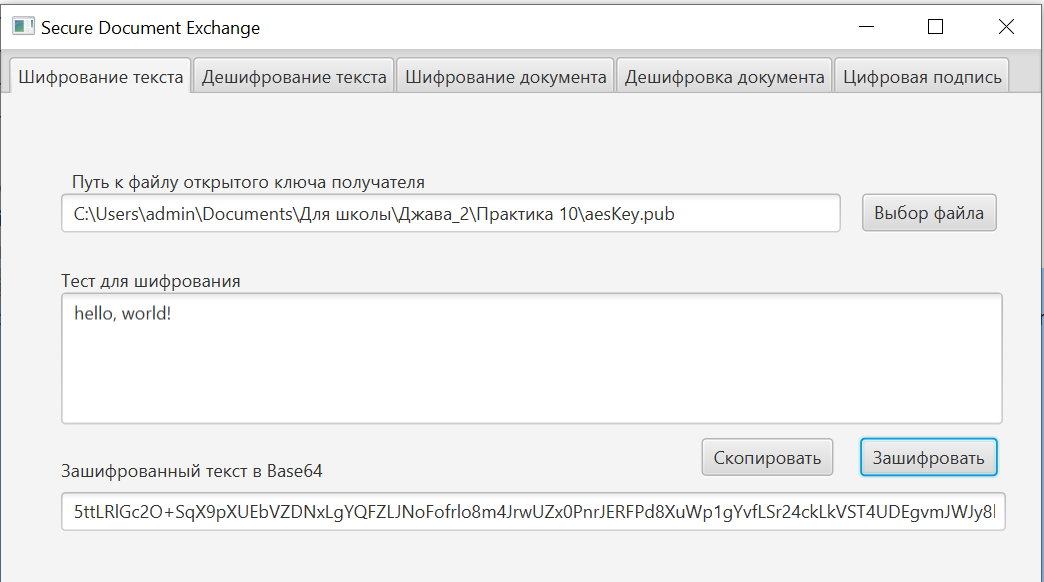


Рисунок 1 – результат шифрования текста.

Дешифрование текста происходит следущим образом:

1. Читвется приватный ключ.
2. Сообщение декодируется из Base64 в обычную зашифрованное сообщение, которое разбивается на три части в соответсвие с размерами использованных вектора инициализации в 16 байт, зашифрованного ключа AES в 512 байт и оставшигеся текса сообщения.
3. Зашифрованный ключ AES дешифруется при помощи алгоритма RSA и закрытого ключа.
4. При помощи IV и ключа AES дешифруем текст при помощи алгоритма AES.

Результат работы деширфования показа на рисунке 2.

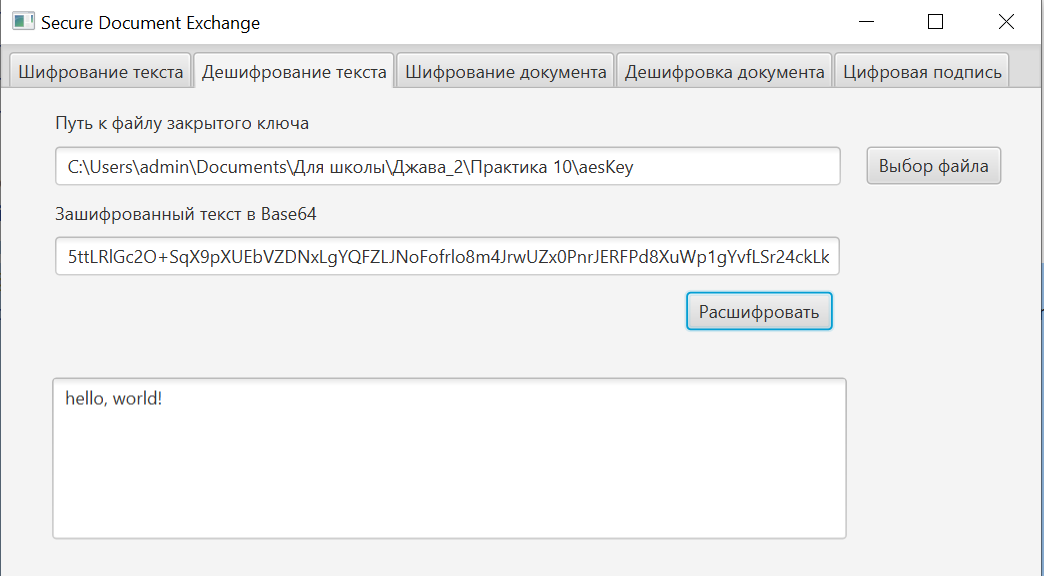


Рисунок 2 – результат дешифрования текста.

**Задание 2.**

Реализовать функцию (де)шифрования документов (и в целом файлов) при помощи методов encryptDocument и decryptDocument (см. Листинг 2).

Реализация функции encryptDocument принимает на вход файл документа и файл открытого ключа. Функция не возвращает ничего, однако должна записывать на диск зашифрованный файл с тем же названием, что и у документа и добавить расширение “.sde”.

@Override  
public void encryptDocument(File document, File openKey) throws IOException, GeneralSecurityException {  
 *// читаем публичный ключ* PublicKey publicKey = SshKeyUtils.*getPublicKey*(openKey).getJCEPublicKey();  
  
 *// Генерация случайного вектора инициализации* byte[] iv = generateIV();  
  
 *// Генерация секретного ключа AES* SecretKey secretKey = generateSecKey();  
  
 *// Инициализация AES и запись в файл iv* Cipher aesCipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");  
 aesCipher.init(Cipher.*ENCRYPT\_MODE*, secretKey, new IvParameterSpec(iv));  
 FileInputStream fisDoc = new FileInputStream(document);  
 FileOutputStream fosEncrypted = new FileOutputStream(document.getParent() + "/" + document.getName() + ".sde");  
 fosEncrypted.write(iv);  
  
 *// Шифрование ключа с использованием RSA и запись в файл* Cipher rsaCipher = Cipher.*getInstance*("RSA/ECB/PKCS1Padding");  
 rsaCipher.init(Cipher.*ENCRYPT\_MODE*, publicKey);  
 byte[] encryptedKey = rsaCipher.doFinal(secretKey.getEncoded());  
 fosEncrypted.write(encryptedKey);  
  
 *// Шифрование сообщения с использованием AES* byte[] buffer = new byte[64];  
 int bytesRead;  
 while ((bytesRead = fisDoc.read(buffer)) != -1) {  
 byte[] output = aesCipher.update(buffer, 0, bytesRead);  
 if (output != null) {  
 fosEncrypted.write(output);  
 }  
 }  
 byte[] outputBytes = aesCipher.doFinal();  
 if (outputBytes != null) {  
 fosEncrypted.write(outputBytes);  
 }  
 fisDoc.close();  
 fosEncrypted.close();  
}  
  
@Override  
public void decryptDocument(File document, File secretKey) throws IOException, GeneralSecurityException, InvalidPassphraseException {  
 PrivateKey privateKey = SshKeyUtils.*getPrivateKey*(secretKey,  
 "").getPrivateKey().getJCEPrivateKey();  
  
 *// Чтение файла с зашифрованным документом* FileInputStream fisEncrypted = new FileInputStream(document);  
 byte[] iv = new byte[16];  
 fisEncrypted.read(iv);  
  
 byte[] encryptedAesKey = new byte[512];  
 fisEncrypted.read(encryptedAesKey);  
  
 *// Расшифрование ключа AES с помощью RSA* Cipher rsaCipher = Cipher.*getInstance*("RSA/ECB/PKCS1Padding");  
 rsaCipher.init(Cipher.*DECRYPT\_MODE*, privateKey);  
 byte[] aesKeyBytes = rsaCipher.doFinal(encryptedAesKey);  
 SecretKey aesKey = new SecretKeySpec(aesKeyBytes, "AES");  
  
 *// Расшифрование файла AES* Cipher aesCipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");  
 aesCipher.init(Cipher.*DECRYPT\_MODE*, aesKey, new IvParameterSpec(iv));  
 FileOutputStream fosDecrypted = new FileOutputStream(document.getParent() + "/" + document.getName().replace(".sde", ""));  
  
 byte[] buffer = new byte[64];  
 int bytesRead;  
 while ((bytesRead = fisEncrypted.read(buffer)) != -1) {  
 byte[] output = aesCipher.update(buffer, 0, bytesRead);  
 if (output != null) {  
 fosDecrypted.write(output);  
 }  
 }  
 byte[] outputBytes = aesCipher.doFinal();  
 if (outputBytes != null) {  
 fosDecrypted.write(outputBytes);  
 }  
 fisEncrypted.close();  
 fosDecrypted.close();  
}

Листинг 2 – (де)шифрование докуметов.

Логика работы шифрования и дешифрования документов та же, что и при работе с текстом кроме двух отличий.

1. Документ читается не разом, а кусочками, которые и обрабатываются алгоритмом AES.
2. Никакого кодирования Base64 не применяется.

Результаты работы шифрование и дешифрования показы на рисунке 3 и рисунке 4 соответсвенно.

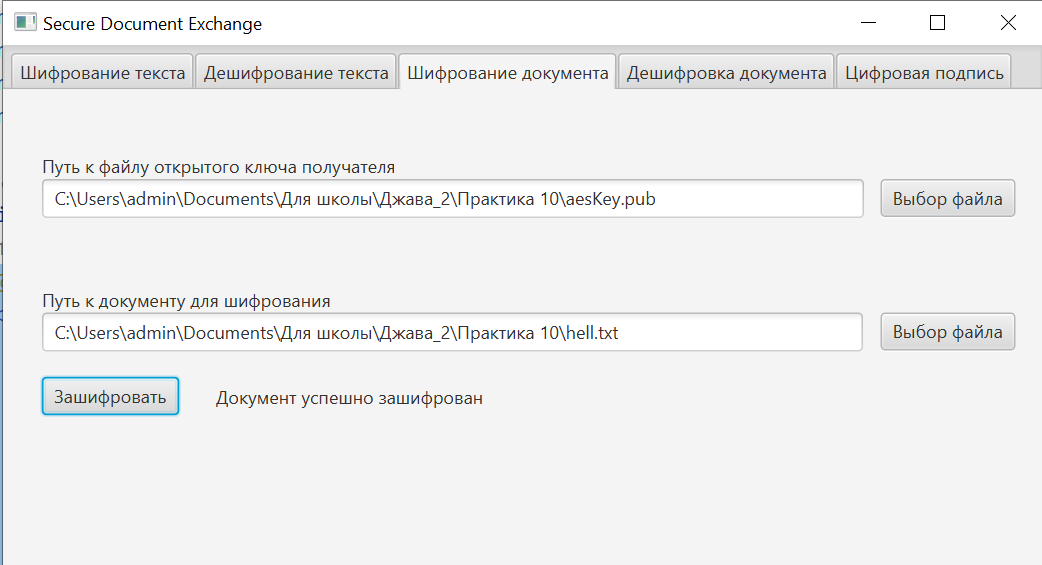


Рисунок 3

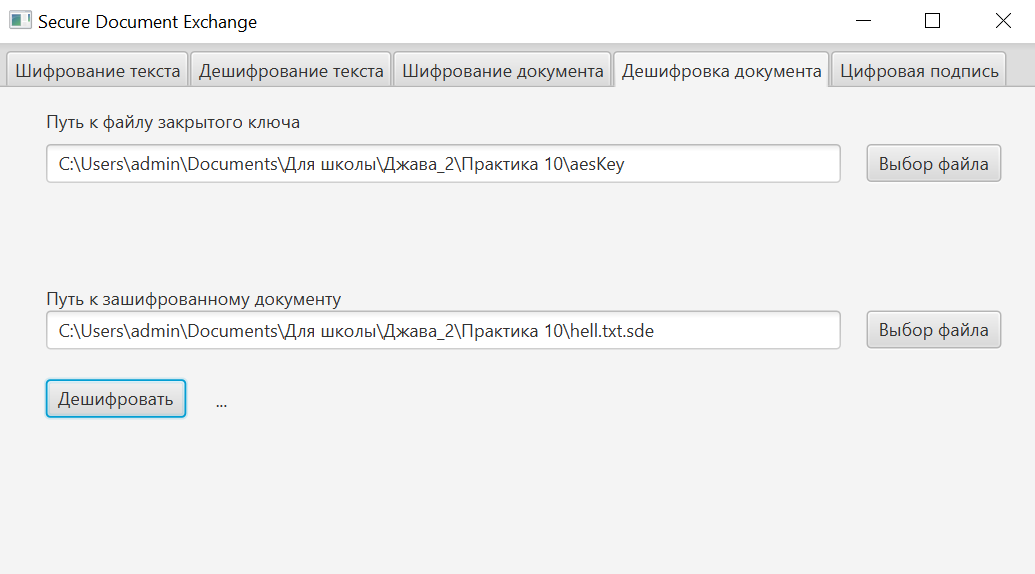


Рисунок 4

**Задание 3.**

Реализовать функцию цифровой подписи документа закрытым ключом отправителя в методе signDocument, а также проверку подписи через открытый ключ в методе verifyDocument (см. Листинг 3).

Реализация метода signDocument принимает на вход файл документа и закрытый ключ. После чего происходит логика создания цифровой подписи и далее происходит запись на диске файла цифровой подписи с тем же названием документа, но в формате “.sig”. Реализация функции verifyDocument принимает на вход файл документа, файл цифровой подписи и открытый ключ. В результате проверки возвращается true или false в зависимости от верификации документа.

@Override  
public void signDocument(File document, File privateKey) throws IOException, GeneralSecurityException, InvalidPassphraseException {  
 PrivateKey rsaPrivateKey = SshKeyUtils.*getPrivateKey*(privateKey,  
 "").getPrivateKey().getJCEPrivateKey();  
  
 *// Читаем содержимое документа* FileInputStream fis = new FileInputStream(document);  
 byte[] bytes = new byte[(int) document.length()];  
 fis.read(bytes);  
 fis.close();  
  
 *// Создаем объект для подписи* Signature signature = Signature.*getInstance*("SHA256withRSA");  
 signature.initSign(rsaPrivateKey);  
 signature.update(bytes);  
  
 *// Подписываем содержимое документа* byte[] signedBytes = signature.sign();  
  
 *// Сохраняем цифровую подпись в файл с расширением ".sig"* FileOutputStream fos = new FileOutputStream(document.getParent() + "/" + document.getName() + ".sig");  
 fos.write(signedBytes);  
 fos.close();  
}  
  
@Override  
public boolean verifyDocument(File document, File signFile, File publicKey) throws IOException, GeneralSecurityException {  
 *// читаем публичный ключ* PublicKey rsaPublicKey = SshKeyUtils.*getPublicKey*(publicKey).getJCEPublicKey();  
  
 *// Читаем содержимое документа* FileInputStream fis = new FileInputStream(document);  
 byte[] bytes = new byte[(int) document.length()];  
 fis.read(bytes);  
 fis.close();  
  
 *//Читаем цифровую подпись из файла* fis = new FileInputStream(signFile);  
 byte[] signature = new byte[(int) signFile.length()];  
 fis.read(signature);  
 fis.close();  
  
  
 *// Создаем объект для проверки подписи* Signature signatureVerifier = Signature.*getInstance*("SHA256withRSA");  
 signatureVerifier.initVerify(rsaPublicKey);  
 signatureVerifier.update(bytes);  
  
 *// Проверяем подпись* return signatureVerifier.verify(signature);  
}

Листинг 3 – реализация цифровой подписи.

При подписи (шифровании) мы используем закрытый ключ, а при проверке (дешифрованию) используем открытый ключ. Результат работы цифровой подписи можно видеть на рисунке 5.

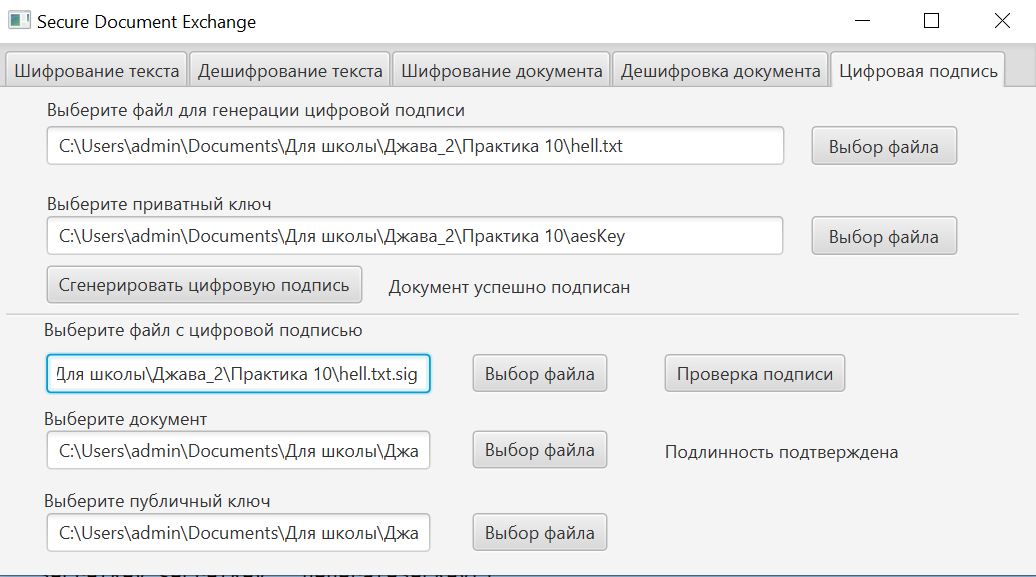


Рисунок 5 – результат цифровой подпись.

**Приложение А.**

Открытый ключ:

ssh-rsa  admin@DESKTOP-JNB51RK

**Приложение Б.**

public class SecurityServiceImpl implements SecurityService{  
 @Override  
 public String encryptMessage(String message, File publicKeyFile) throws IOException, GeneralSecurityException {  
 *// читаем публичный ключ* PublicKey publicKey = SshKeyUtils.*getPublicKey*(publicKeyFile).getJCEPublicKey();  
  
 *// Генерация случайного вектора инициализации* byte[] iv = generateIV();  
  
 *// Генерация секретного ключа AES* SecretKey secretKey = generateSecKey();  
  
 *// Шифрование сообщения с использованием AES* Cipher aesCipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");  
 aesCipher.init(Cipher.*ENCRYPT\_MODE*, secretKey, new IvParameterSpec(iv));  
 byte[] encryptedMessage = aesCipher.doFinal(message.getBytes());  
  
 *// Шифрование ключа с использованием RSA* Cipher rsaCipher = Cipher.*getInstance*("RSA/ECB/PKCS1Padding");  
 rsaCipher.init(Cipher.*ENCRYPT\_MODE*, publicKey);  
 byte[] encryptedKey = rsaCipher.doFinal(secretKey.getEncoded());  
  
 *// Соединение зашифрованного сообщения, ключа и вектора инициализации в одну строку* byte[] result = new byte[iv.length + encryptedKey.length + encryptedMessage.length];  
  
 copy(iv, encryptedKey, encryptedMessage, result);  
  
 return Base64.*getEncoder*().encodeToString(result);  
 }  
  
 @Override  
 public String decryptMessage(String message, File privateKeyFile) throws IOException, GeneralSecurityException, InvalidPassphraseException {  
 *// чтение приватного ключа* PrivateKey privateKey = SshKeyUtils.*getPrivateKey*(privateKeyFile,  
 "").getPrivateKey().getJCEPrivateKey();  
  
 *// Разбиение зашифрованного сообщения, ключа и вектора инициализации на отдельные строки* byte[] answer = Base64.*getDecoder*().decode(message);  
 byte[] iv = Arrays.*copyOfRange*(answer, 0, 16);  
 byte[] encryptedKey = Arrays.*copyOfRange*(answer,16, 528);  
 byte[] encryptedMessage = Arrays.*copyOfRange*(answer, 528, answer.length);  
  
 *// Дешифрование ключа с использованием RSA* Cipher rsaCipher = Cipher.*getInstance*("RSA/ECB/PKCS1Padding");  
 rsaCipher.init(Cipher.*DECRYPT\_MODE*, privateKey);  
 byte[] decryptedKey = rsaCipher.doFinal(encryptedKey);  
 SecretKey secretKey = new SecretKeySpec(decryptedKey, "AES");  
  
 *// Дешифрование сообщения с использованием AES* Cipher aesCipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");  
 aesCipher.init(Cipher.*DECRYPT\_MODE*, secretKey, new IvParameterSpec(iv));  
 byte[] decryptedMessage = aesCipher.doFinal(encryptedMessage);  
  
 *// Возврат дешифрованного сообщения* return new String(decryptedMessage);  
 }  
  
 @Override  
 public void encryptDocument(File document, File openKey) throws IOException, GeneralSecurityException {  
 *// читаем публичный ключ* PublicKey publicKey = SshKeyUtils.*getPublicKey*(openKey).getJCEPublicKey();  
  
 *// Генерация случайного вектора инициализации* byte[] iv = generateIV();  
  
 *// Генерация секретного ключа AES* SecretKey secretKey = generateSecKey();  
  
 *// Инициализация AES и запись в файл iv* Cipher aesCipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");  
 aesCipher.init(Cipher.*ENCRYPT\_MODE*, secretKey, new IvParameterSpec(iv));  
 FileInputStream fisDoc = new FileInputStream(document);  
 FileOutputStream fosEncrypted = new FileOutputStream(document.getParent() + "/" + document.getName() + ".sde");  
 fosEncrypted.write(iv);  
  
 *// Шифрование ключа с использованием RSA и запись в файл* Cipher rsaCipher = Cipher.*getInstance*("RSA/ECB/PKCS1Padding");  
 rsaCipher.init(Cipher.*ENCRYPT\_MODE*, publicKey);  
 byte[] encryptedKey = rsaCipher.doFinal(secretKey.getEncoded());  
 fosEncrypted.write(encryptedKey);  
  
 *// Шифрование сообщения с использованием AES* byte[] buffer = new byte[64];  
 int bytesRead;  
 while ((bytesRead = fisDoc.read(buffer)) != -1) {  
 byte[] output = aesCipher.update(buffer, 0, bytesRead);  
 if (output != null) {  
 fosEncrypted.write(output);  
 }  
 }  
 byte[] outputBytes = aesCipher.doFinal();  
 if (outputBytes != null) {  
 fosEncrypted.write(outputBytes);  
 }  
 fisDoc.close();  
 fosEncrypted.close();  
 }  
  
 @Override  
 public void decryptDocument(File document, File secretKey) throws IOException, GeneralSecurityException, InvalidPassphraseException {  
 PrivateKey privateKey = SshKeyUtils.*getPrivateKey*(secretKey,  
 "").getPrivateKey().getJCEPrivateKey();  
  
 *// Чтение файла с зашифрованным документом* FileInputStream fisEncrypted = new FileInputStream(document);  
 byte[] iv = new byte[16];  
 fisEncrypted.read(iv);  
  
 byte[] encryptedAesKey = new byte[512];  
 fisEncrypted.read(encryptedAesKey);  
  
 *// Расшифрование ключа AES с помощью RSA* Cipher rsaCipher = Cipher.*getInstance*("RSA/ECB/PKCS1Padding");  
 rsaCipher.init(Cipher.*DECRYPT\_MODE*, privateKey);  
 byte[] aesKeyBytes = rsaCipher.doFinal(encryptedAesKey);  
 SecretKey aesKey = new SecretKeySpec(aesKeyBytes, "AES");  
  
 *// Расшифрование файла AES* Cipher aesCipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");  
 aesCipher.init(Cipher.*DECRYPT\_MODE*, aesKey, new IvParameterSpec(iv));  
 FileOutputStream fosDecrypted = new FileOutputStream(document.getParent() + "/" + document.getName().replace(".sde", ""));  
  
 byte[] buffer = new byte[64];  
 int bytesRead;  
 while ((bytesRead = fisEncrypted.read(buffer)) != -1) {  
 byte[] output = aesCipher.update(buffer, 0, bytesRead);  
 if (output != null) {  
 fosDecrypted.write(output);  
 }  
 }  
 byte[] outputBytes = aesCipher.doFinal();  
 if (outputBytes != null) {  
 fosDecrypted.write(outputBytes);  
 }  
 fisEncrypted.close();  
 fosDecrypted.close();  
 }  
  
 @Override  
public void signDocument(File document, File privateKey) throws IOException, GeneralSecurityException, InvalidPassphraseException {  
 PrivateKey rsaPrivateKey = SshKeyUtils.*getPrivateKey*(privateKey,  
 "").getPrivateKey().getJCEPrivateKey();  
  
 *// Читаем содержимое документа* FileInputStream fis = new FileInputStream(document);  
 byte[] bytes = new byte[(int) document.length()];  
 fis.read(bytes);  
 fis.close();  
  
 *// Создаем объект для подписи* Signature signature = Signature.*getInstance*("SHA256withRSA");  
 signature.initSign(rsaPrivateKey);  
 signature.update(bytes);  
  
 *// Подписываем содержимое документа* byte[] signedBytes = signature.sign();  
  
 *// Сохраняем цифровую подпись в файл с расширением ".sig"* FileOutputStream fos = new FileOutputStream(document.getParent() + "/" + document.getName() + ".sig");  
 fos.write(signedBytes);  
 fos.close();  
}  
  
@Override  
public boolean verifyDocument(File document, File signFile, File publicKey) throws IOException, GeneralSecurityException {  
 *// читаем публичный ключ* PublicKey rsaPublicKey = SshKeyUtils.*getPublicKey*(publicKey).getJCEPublicKey();  
  
 *// Читаем содержимое документа* FileInputStream fis = new FileInputStream(document);  
 byte[] bytes = new byte[(int) document.length()];  
 fis.read(bytes);  
 fis.close();  
  
 *//Читаем цифровую подпись из файла* fis = new FileInputStream(signFile);  
 byte[] signature = new byte[(int) signFile.length()];  
 fis.read(signature);  
 fis.close();  
  
  
 *// Создаем объект для проверки подписи* Signature signatureVerifier = Signature.*getInstance*("SHA256withRSA");  
 signatureVerifier.initVerify(rsaPublicKey);  
 signatureVerifier.update(bytes);  
  
 *// Проверяем подпись* return signatureVerifier.verify(signature);  
}

private byte[] generateIV() {  
 *// Генерация случайного вектора инициализации* byte[] iv = new byte[16];  
 SecureRandom random = new SecureRandom();  
 random.nextBytes(iv);  
 return iv;  
 }  
  
 private SecretKey generateSecKey() throws NoSuchAlgorithmException {  
 *// Генерация секретного ключа AES* KeyGenerator keyGen = KeyGenerator.*getInstance*("AES");  
 keyGen.init(256);  
 SecretKey secretKey = keyGen.generateKey();  
 return secretKey;  
 }  
  
 private void copy(byte[] a, byte[] b, byte[] c, byte[] res) {  
 System.*arraycopy*(a, 0, res, 0, a.length);  
 System.*arraycopy*(b, 0, res, a.length, b.length);  
 System.*arraycopy*(c, 0, res, a.length + b.length, c.length);  
 }  
  
 private void copyReverse(byte[] res, byte[] a, byte[] b, byte[] c) {  
 System.*arraycopy*(res, 0, a, 0, a.length);  
 System.*arraycopy*(res, a.length, b, 0, b.length);  
 System.*arraycopy*(res, a.length + b.length, c, 0, c.length);  
 }  
}