МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КУБГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчет**

**по практическому заданию №12**

**по курсу**

**«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»**

Работу выполнил

Студент 49 группы

Лобода Д.А.

Преподаватель:

Крамаренко А.А.

Краснодар 2024

**Постановка задачи.**

Реализовать протокол подписания и проверки подписи файла для

протокола DSS. Возможно пользоваться встроенными библиотеками языков.

Протокол DSS (Digital Signature Standard) — это алгоритм цифровой подписи, предложенный Национальным институтом стандартов и технологий США (NIST), который использует хеш-функции и алгоритмы цифровой подписи на основе дискретного логарифмирования для обеспечения аутентификации и целостности данных.

Рассмотрим основные шаги протокола подписания и проверки подписи файла для DSS:

**1. Генерация ключевой пары:**

* Пользователь генерирует ключевую пару, состоящую из закрытого ключа (private key) и открытого ключа (public key).
* Закрытый ключ является случайным числом, которое выбирается секретно и используется для создания подписи.
* Открытый ключ вычисляется на основе закрытого ключа с использованием специальных алгоритмов, определенных в стандарте DSS.

**2. Подписание файла:**

* Для подписания файла пользователь вычисляет хеш (хеш-сумму) файла, который он хочет подписать, используя криптографическую хеш-функцию, такую как SHA-256.
* Затем пользователь подписывает хеш файла, используя свой закрытый ключ и алгоритм цифровой подписи, определенный в стандарте DSS. Это происходит в несколько этапов:

1 ) Выбирается случайное число (nonce), которое используется в процессе подписи.

2) Выполняются математические операции с использованием закрытого ключа и хеша файла для создания подписи.

**3. Проверка подписи:**

* Для проверки подписи получатель сначала вычисляет хеш файла, который был подписан, с использованием той же хеш-функции, что и отправитель.
* Затем получатель использует открытый ключ отправителя для проверки подписи:

1. Используя открытый ключ отправителя, получатель проверяет, соответствует ли подпись данным хеша файла.
2. Путем использования открытого ключа отправителя и координаты X точки R проверяется, что подпись была создана тем же человеком, который владеет закрытым ключом.
3. Если проверка прошла успешно, то подпись считается действительной, иначе она считается недействительной.

**Замечания:**

1. Аутентификация и целостность данных:

Протокол DSS обеспечивает аутентификацию отправителя и целостность данных. Это достигается за счет создания цифровой подписи, которая позволяет убедиться в том, что сообщение было подписано отправителем, и что оно не было изменено в процессе передачи.

2. Открытый ключ для проверки подписи:

Открытый ключ отправителя используется для проверки подписи. Получатель может свободно распространять открытый ключ, поскольку он не используется для создания подписи и предназначен исключительно для проверки подлинности данных.

3. Хранение закрытого ключа:

Закрытый ключ должен храниться в безопасном месте и никогда не должен раскрываться. Это важно для обеспечения безопасности подписи и предотвращения возможных атак на систему.

4. Независимость подписи от размера файла:

Подпись, созданная в рамках протокола DSS, не зависит от размера подписываемого файла. Вместо этого подписывается его хеш-значение, что делает протокол эффективным и позволяет подписывать файлы любого размера с минимальными затратами ресурсов.

**Текст программы:**

**Файл Lab\_12.java:**

import java.io.\*;  
import java.security.\*;  
  
  
public class Main {  
  
 // Генерация ключей для подписи  
 public static KeyPair generateKeyPair() throws NoSuchAlgorithmException {  
 KeyPairGenerator keyPairGenerator = KeyPairGenerator.*getInstance*("DSA");  
 keyPairGenerator.initialize(1024); // Размер ключа  
 return keyPairGenerator.generateKeyPair();  
 }  
  
 // Подписание данных  
 public static byte[] signData(byte[] data, PrivateKey privateKey) throws Exception {  
 Signature signature = Signature.*getInstance*("SHA1withDSA");  
 signature.initSign(privateKey);  
 signature.update(data);  
 return signature.sign();  
 }  
  
 // Проверка подписи  
 public static boolean verifySignature(byte[] data, byte[] signatureBytes, PublicKey publicKey) throws Exception {  
 Signature signature = Signature.*getInstance*("SHA1withDSA");  
 signature.initVerify(publicKey);  
 signature.update(data);  
 return signature.verify(signatureBytes);  
 }  
  
 public static void main(String[] args) {  
 try {  
 // Генерация ключей  
 KeyPair keyPair = *generateKeyPair*();  
 PrivateKey privateKey = keyPair.getPrivate();  
 PublicKey publicKey = keyPair.getPublic();  
  
 // Пример данных для подписи  
 String data = "Hello, World!";  
 byte[] dataBytes = data.getBytes();  
  
 // Подписание данных  
 byte[] signature = *signData*(dataBytes, privateKey);  
  
 // Сохранение подписи в файл  
 FileOutputStream signatureOut = new FileOutputStream("signature.txt");  
 signatureOut.write(signature);  
 signatureOut.close();  
  
 // Проверка подписи  
 FileInputStream signatureIn = new FileInputStream("signature.txt");  
 byte[] signatureBytes = signatureIn.readAllBytes();  
 signatureIn.close();  
  
 boolean verified = *verifySignature*(dataBytes, signatureBytes, publicKey);  
 System.*out*.println("Signature verified: " + verified);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}