УДК 004.223.2

ИСПОЛЬЗОВАНИЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ JAVA ДЛЯ РАЗРАБОТКИ ГИБРИДНОЙ КРИПТОСИСТЕМЫ

А.В. СУББОТИН
(Представлено: канд. физ.-мат. наук., доц. Ю.Ф. ПАСТУХОВ)

В статье рассматривается криптографические возможности в языке JAVA. В ходе анализа были рассмотрены основные классы и интерфейсы расширения. Проведены исследования по актуальности разработки с использованием данного расширения.

Ключевые слова: *информационные технологии, криптографические методы, безопасность, язык программирования Java.*

Для реализации криптоустойчивых приложений, написанных на языке программирования Java, создано расширение Java Cryptography Extension(JCE), которое является частью платформы Java.

Java Cryptography Architecture (JCA) – название внутреннего дизайна API криптографии в Java. JCA структурирован вокруг нескольких основных классов и интерфейсов общего назначения. Реальная функциональность этих интерфейсов обеспечивается поставщиками. Таким образом, можно использовать класс Cipher (Шифр) для шифрования и расшифровки некоторых данных, но конкретная реализация шифра (алгоритм шифрования) зависит от конкретного используемого поставщика.

Основные классы и интерфейсы. API криптографии Java состоит из следующих пакетов Java:

– java.security
– java.security.cert
– java.security.spec
– java.security.interfaces
– javax.crypto
– javax.crypto.spec
– javax.crypto.interfaces

Рассмотрим основные классы и интерфейсы этих пакетов.

1. *Provider* (Поставщик криптографии)

Класс Provider (java.security.Provider) является центральным классом в Java crypto API. Для того чтобы использовать Java crypto API, вам нужно установить поставщика криптографии. Java SDK поставляется с собственным поставщиком криптографии. Один из самых популярных поставщиков криптографии для Java crypto API называется Bouncy Castle.

```
import org.bouncycastle.jce.provider.BouncyCastleProvider;

import java.security.Security;

public class ProviderExample {
    public static void main(String[] args) {

        Security.addProvider(new BouncyCastleProvider());

    }
}
```

Рисунок 1. – Добавление Provider

2. *Cipher* (Шифр)

Класс Cipher (javax.crypto.Cipher) представляет криптографический алгоритм. Шифр может использоваться как для шифрования, так и для расшифровки данных. Создание экземпляра класса шифр, который использует алгоритм шифрования AES для внутреннего использования:

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
```

Рисунок 2. – Инициализация шифра

Инициализация шифра

Перед использованием экземпляра шифра его необходимо инициализировать. Экземпляр шифра инициализируется вызывом метода init(). Метод init() принимает два параметра:

– Режим – Шифрование / Расшифровка
– Ключ

Первый параметр указывает, режим работы экземпляра шифр: шифровать или расшифровывать данные. Второй параметр указывает, какой ключ они используют для шифрования или расшифровки данных.

3. *Keys* (Ключи)

Для шифрования или дешифрования данных вам нужен ключ. Существует два типа ключей в зависимости от того, какой тип алгоритма шифрования используется:

– Симметричные ключи
– Асимметричные ключи

Симметричные ключи используются для симметричных алгоритмов шифрования. Алгоритмы симметричного шифрования используют один и тот же ключ для шифрования и расшифровки. Асимметричные ключи используются для алгоритмов асимметричного шифрования. Алгоритмы асимметричного шифрования используют один ключ для шифрования, а другой для дешифрования. Алгоритмы шифрования с открытым и закрытым ключом – примеры асимметричных алгоритмов шифрования.

Каким-то образом сторона, которая должна расшифровать данные, должна знать ключ, необходимый для дешифрования данных. Если дешифрующая сторона не является стороной шифрующей данные, эти две стороны должны договориться о ключе или обменяться ключом. Это называется обменом ключами.

4. *Подпись* (Signature)

Класс Signature (java.security.Signature) используется для цифровой подписи данных. Когда данные подписаны, цифровая подпись создается из этих данных. Таким образом, подпись отделена от данных.

Цифровая подпись создается путем создания дайджеста сообщения (хеша) из данных и шифрования этого дайджеста сообщения с помощью закрытого ключа устройства, лица или организации, которая должна подписать данные. Дайджест зашифрованного сообщения называется цифровой подписью.

Чтобы подписать данные, вы должны инициализировать экземпляр подписи в режиме подписи вызывая метод initSign(...), передавая закрытый ключ для подписи данных.

После инициализации экземпляра подписи, его можно использовать для подписи данных. Это делается вызовом метода update(), передавая данные для подписи в качестве параметра. Можно вызывать метод update() несколько раз, чтобы дополнить данные для создании подписи. После передачи всех данных в метод update(), вызывается метод sign() для получения цифровой подписи.

Чтобы проверить подпись, нужно инициализировать экземпляр подписи в режиме проверки путем вызова метода initVerify(...), передавая в качестве параметра открытый ключ, который используется для проверки подписи. После инициализации в режиме проверки в метод update() передаются данные, которые подписаны подписью. Вызов метода verify(), возвращает значение true или false в зависимости от того, можно ли проверить подпись или нет.

```
SecureRandom secureRandom = new SecureRandom();
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");
KeyPair keyPair = keyPairGenerator.generateKeyPair();

Signature signature = Signature.getInstance("SHA256WithDSA");

signature.initSign(keyPair.getPrivate(), secureRandom);

byte[] data = "abcdefghijklmnopqrstuvxyz".getBytes("UTF-8");
signature.update(data);

byte[] digitalSignature = signature.sign();

Signature signature2 = Signature.getInstance("SHA256WithDSA");
signature2.initVerify(keyPair.getPublic());

signature2.update(data);

boolean verified = signature2.verify(digitalSignature);
System.out.println("verified = " + verified);
```

Рисунок 3. – Процесс подписи

Заключение. В ходе данного исследования были сделаны выводы об актуальности используемой криптографической платформы. Возможности данного расширения позволяют спроектировать криптоустойчивую систему без установки дополнительных расширений.

ЛИТЕРАТУРА

1. Java Cryptography [Электронный ресурс] – Режим доступа: http://tutorials.jenkov.com/java-cryptography/index.html – Дата доступа: 23.09.2018.
2. Java Docs [Электронный ресурс] – Режим доступа: https://docs.oracle.com/javase/7/docs/api/ – Дата доступа: 23.09.2018.