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

Студенты 46 группы

Нагалевский А.М.

Преподаватель:

Крамаренко А.А.

Краснодар 2024

**Постановка задачи.**

Реализовать протокол подписания и проверки подписи файла для протокола ECDSA. Возможно пользоваться встроенными библиотеками языков.

Протокол ECDSA (Elliptic Curve Digital Signature Algorithm) — это алгоритм цифровой подписи, который базируется на математических принципах работы с эллиптическими кривыми. Он использует преимущества эллиптической криптографии для создания эффективных и безопасных цифровых подписей.

Рассмотри шаги протокола подписания и проверки подписи файла для ECDSA:

**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 и представляет собой точку на эллиптической кривой. Он используется для проверки подписи.

**2. Подписание файла:**

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

1. Выбирается случайное число (nonce), которое используется в процессе подписи.
2. Это случайное число умножается на базовую точку эллиптической кривой, что приводит к формированию точки на кривой (R).
3. Вычисляется значение подписи с использованием закрытого ключа, хеша сообщения и координаты X компоненты точки R.

**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. Получатель извлекает координату X из подписи.
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:**

* Протокол ECDSA обеспечивает аутентификацию и целостность данных.
* Открытый ключ может быть распространен в открытом доступе, поскольку он используется только для проверки подписи.
* Закрытый ключ должен храниться в безопасном месте и никогда не должен раскрываться.
* Подпись не является чувствительной к размеру файла, так как подписывается только его хеш-значение.

Протокол ECDSA широко используется в различных системах цифровой подписи, таких как блокчейн, электронные документы и цифровые сертификаты.

**Текст программы:**

**Файл task1.py:**

from cryptography.hazmat.backends import default\_backend

from cryptography.hazmat.primitives.asymmetric import ec

from cryptography.hazmat.primitives import hashes

def generate\_key\_pair():

    private\_key = ec.generate\_private\_key(ec.SECP256R1(), default\_backend())

    public\_key = private\_key.public\_key()

    return private\_key, public\_key

def sign\_message(private\_key, message):

    signature = private\_key.sign(

        message,

        ec.ECDSA(hashes.SHA256())

    )

    return signature

def verify\_signature(public\_key, message, signature):

    try:

        public\_key.verify(

            signature,

            message,

            ec.ECDSA(hashes.SHA256())

        )

        return True

    except:

        return False

def read\_file(file\_path):

    with open(file\_path, "rb") as file:

        return file.read()

def write\_file(file\_path, data):

    with open(file\_path, "wb") as file:

        file.write(data)

# Генерация ключевой пары

private\_key, public\_key = generate\_key\_pair()

# Чтение файла, который нужно подписать

file\_path = "test.txt"

message = read\_file(file\_path)

# Подписание файла

signature = sign\_message(private\_key, message)

# Запись подписи в файл

signature\_file\_path = "signature.sig"

write\_file(signature\_file\_path, signature)

# Проверка подписи

signature\_to\_verify = read\_file(signature\_file\_path)

is\_valid\_signature = verify\_signature(public\_key, message, signature\_to\_verify)

if is\_valid\_signature:

    print("Подпись верна.")

else:

    print("Подпись неверна.")