Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Защита информации в информационных системах

ОТЧЕТ

по лабораторной работе №5

на тему

криптографическая защита данных с помощью
microsoft cryptoapi

Студент:

Проверил:

МИНСК 2024

1. ЦЕЛЬ РАБОТЫ

1. Реализовать аутентификацию сообщений на основе алгоритма HMAC средствами CryptoAPI двумя способами:

а) используя криптоинтерфейс только для хеширования значений (конкатенацию сообщения и секретного значения выполнять самостоятельно);

б) используя вызов функции CryptCreateHash с ключевым значением для вычисления кода аутентификации HMAC средствами криптоинтерфейса.

2. Реализовать программное средство постановки электронной цифровой подписи для дисковых файлов с помощью соответствующего криптоалгоритма ALG1. На вход системы при постановке подписи подаются имя подписываемого файла и имя файла цифровой подписи, при проверке подписи входными данными являются имя подписанного файла и файла подписи, а также имя пользователя, подлинность подписи которого проверяется. В качестве алгоритма хеширования используйте криптоалгоритм ALG2.

2. ПРАКТИЧЕСКОЕ ВЫПОЛНЕНИЕ

2.1 Реализация аутентификации сообщений на основе алгоритма HMAC средствами CryptoAPI двумя способами.

2.1.1 Первый способ используя криптоинтерфейс только для хеширования значений (конкатенацию сообщения и секретного значения выполнять самостоятельно).

```
from Crypto.Hash import SHA256
def hmac_sha256(message, secret):

    # Конкатенация
    message_bytes = message.encode('utf-8')
    secret_bytes = secret.encode('utf-8')
    combined = message_bytes + secret_bytes

    # Вычисление хеша
    hash_obj = SHA256.new(combined)
    return hash_obj.hexdigest()

message = input("Введите сообщение: ")
secret = input("Введите секретный ключ: ")

# Вычисление HMAC
hmac_value = hmac_sha256(message, secret)
print("Сообщение: " + message + "\nПосле хеширования: ", hmac_value)
```

2.1.2 Реализации работы программы

Для работы программы необходимо ввести сообщение, ввести секретный ключ. После чего программа выведет сообщение и сообщение после хеширования. Представлено на рисунке 2.5.2.

```
Введите сообщение: Привет Мир!
Введите секретный ключ: 3214
Сообщение: Привет Мир!
После хеширования:  aa87cf651cb40dd7def96901669cd3d5763fd32143e10fdc16e68c97c4347f85
```

Рисунок 2.5.2 – Работа программы

2.2 Второй способ используя вызов функции CryptCreateHash с ключевым значением для вычисления кода аутентификации HMAC средствами криптоинтерфейса.

```
from Crypto.Hash import HMAC, SHA256

def hmac_with_cryptoapi(message, secret):
    # Создание объекта HMAC с использованием SHA256
    hmac_obj = HMAC.new(secret.encode('utf-8'), msg=message.encode('utf-8'),
digestmod=SHA256)

    # Получение хеша
    return hmac_obj.hexdigest()

message = input("Введите сообщение: ")
secret = input("Введите секретный ключ: ")
hmac_value = hmac_with_cryptoapi(message, secret)
print("Сообщение: " + message + "\nПосле хеширования: ", hmac_value)
```

2.2.1 Реализация работы программы

Для работы программы необходимо ввести сообщение, ввести секретный ключ. После чего программа выведет сообщение и сообщение после хеширования. Представлено на рисунке 2.5.4.

```
Введите сообщение: Привет Мир!
Введите секретный ключ: 3214
Сообщение: Привет Мир!
После хеширования:  56345dd65c69cb902ada88ee3bfe963fba68533f29165df934a2231d7c2f8cb5
```

Рисунок 2.5.4 – Работа программы

2.3 Реализуйте программное средство постановки электронной цифровой подписи для дисковых файлов с помощью соответствующего криптоалгоритма CALG_RSA_SIGN. В качестве алгоритма хеширования используйте криптоалгоритм CALG_SHA256.

```
import os
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import serialization

EXPECTED_VALUE = "XYZ"  # Ожидаемое значение для проверки

def generate_keys():
    """Генерация пары ключей RSA и сохранение их в файлы."""
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    public_key = private_key.public_key()

    # Сохранение закрытого ключа в файл
    with open("private_key.pem", "wb") as f:
        f.write(private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption()
        ))

    # Сохранение открытого ключа в файл
    with open("public_key.pem", "wb") as f:
        f.write(public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        ))


def sign_file(file_name, signature_file):
    """Подписывает указанный файл и сохраняет подпись в указанный файл."""
    try:
        # Загрузка закрытого ключа
        with open("private_key.pem", "rb") as f:
            private_key = serialization.load_pem_private_key(
                f.read(),
                password=None,
                backend=default_backend()
            )

        # Чтение данных файла для подписи
        with open(file_name, "rb") as f:
            file_data = f.read()

        # Хеширование данных файла с использованием SHA-256
```

```
        hasher = hashes.Hash(hashes.SHA256(), backend=default_backend())
        hasher.update(file_data)
        digest = hasher.finalize()

        # Подпись хеша с использованием RSA и PSS
        signature = private_key.sign(
            digest,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )

        # Сохранение подписи в файл .txt
        with open(signature_file, "wb") as f:
            f.write(signature)

    except Exception as e:
        print(f"Ошибка при подписании файла: {e}")


def verify_signature(file_name, signature_file, public_key_file, user_name):
    """Проверяет подпись указанного файла с использованием открытого ключа."""
    try:
        # Загрузка открытого ключа
        with open(public_key_file, "rb") as f:
            public_key = serialization.load_pem_public_key(
                f.read(),
                backend=default_backend()
            )

        # Чтение данных файла для проверки подписи
        with open(file_name, "rb") as f:
            file_data = f.read()

        # Вывод содержимого файла
        print("Содержимое файла:", file_data.decode('utf-8'))

        # Сравнение содержимого файла с ожидаемым значением
        if file_data.decode('utf-8').strip() != EXPECTED_VALUE:
            print("Подпись недействительна: содержимое файла не соответствует
ожидаемому.")
            return

        # Хеширование данных файла с использованием SHA-256
        hasher = hashes.Hash(hashes.SHA256(), backend=default_backend())
        hasher.update(file_data)
        digest = hasher.finalize()

        # Вывод хеша в шестнадцатеричном формате
        print("Хеш:", digest.hex())

        # Загрузка подписи из файла .txt
        with open(signature_file, "rb") as f:
```

```
            signature = f.read()

        # Вывод содержимого файла подписи в шестнадцатеричном формате
        print("Содержимое файла подписи:", signature.hex())

        # Проверка подписи с использованием открытого ключа
        public_key.verify(
            signature,
            digest,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )

        print(f"Подпись действительна для пользователя: {user_name}.")

    except Exception as e:
        print("Подпись недействительна:", str(e))


if __name__ == "__main__":
    generate_keys()

    file_name = "example.txt"  # Имя файла для подписи

    # Создание тестового файла для подписи, если он не существует
    if not os.path.exists(file_name):
        with open(file_name, "w") as f:
            f.write(EXPECTED_VALUE)

    signature_file = "signature.txt"  # Имя файла для сохранения подписи

    sign_file(file_name, signature_file)  # Подпись файла

    user_name = "User1"

    # Проверка подписи
    verify_signature(file_name, signature_file, "public_key.pem", user_name)
```

2.3.1 Реализация работы программы

Код позволяет генерировать пару ключей RSA, подписывать файлы с использованием закрытого ключа RSA, проверять подписи файлов с использованием открытого ключа RSA. Таким образом, данный код реализует базовую систему цифровых подписей, обеспечивая целостность и подлинность данных. Представлено на рисунке 2.3.1. При не действительной подписи представлено на рисунке 2.3.2.

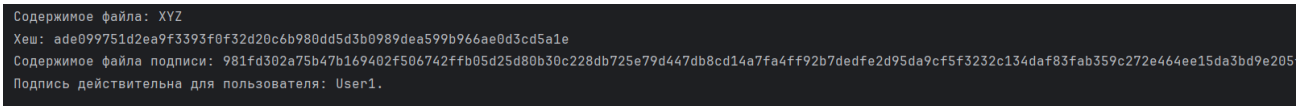

Рисунок 2.3.1 – Вывод программы

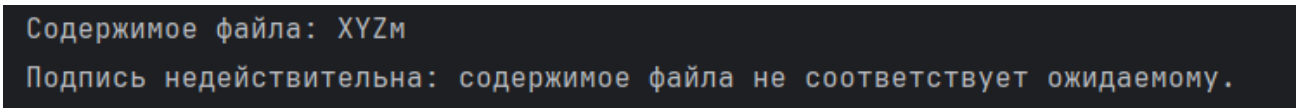

Рисунок 2.3.2 – Вывод программы

3. ЗАКЛЮЧЕНИЕ

Реализовали программное средство постановки электронной цифровой подписи для дисковых файлов с помощью соответствующего криптоалгоритма CALG_RSA_SIGN. На вход системы при постановке подписи подавались имя подписываемого файла и имя файла цифровой подписи, при проверке подписи входными данными являлись имя подписанного файла и файла подписи, а также имя пользователя, подлинность подписи которого проверяется. В качестве алгорит- ма хеширования использовался криптоалгоритм CALG_SHA256.