МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт КНТ

Факультет ФИСП

Кафедра ПИ им. Л.П. Фельдмана

Лабораторная работа №4

Тема: «Изучение стандартных средств для реализации приложений, использующих симметричное и ассиметричное шифрование.»

Курс: Безопасность программ и данных

Выполнил

ст. гр. ПИ-18б

Моргунов А.Г.

Проверил

Ногтев Е.А.

Донецк – 2021

ХОД РАБОТЫ

1) Краткое теоретическое описание выбранных алгоритмов шифрования (симметричного и ассиметричного, однонаправленной хеш-функции).

AES – это симметричный алгоритм шифрования. Он является блочным алгоритмом шифрования. Алгоритм работает в несколько раундов в зависимости от размера ключа (для 128, 192, 256 битных ключей 10, 12, 14 раундов соответственно). Каждый раунд (кроме последнего) состоит из 4 этапов SubBytes, ShiftRows, MixColumns, AddRoundKey. Последний раунд состоит из 3 этапов SubBytes, ShiftRows, AddRoundKey. Дешифрование происходит с помощью инвертированных операций, производимых в обратном порядке.

RSA – это ассиметричный алгоритм шифрования, который использует пару ключей открытый-закрытый. С помощью открытого ключа пользователи могут шифровать сообщения, а при помощи закрытого ключа можно эти сообщения расшифровать. Также этот алгоритм используется для электронной подписи документов. При электронной подписи пользователь может подписать документ с помощью закрытого ключа, а другие пользователи могут проверить подлинность документа при помощи открытого ключа.

SHA-256 – это однонаправленная хэш-функция семейства SHA-2. Особенность функции SHA-256 является то, что дайджест сообщения не зависит от длины сообщения и всегда равен 256 бит.

2) Описание структур, методов, которые использовались в программе.

В программе используется библиотека Python Cryptography, которая реализует симметричное, ассиметричное шифрование и цифровую подпись.

Также реализованы методы для записи и считывания информации из/в файл, методы генерации ключей.

3) Текст программы.

import os

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

from cryptography.hazmat.primitives.asymmetric import rsa

from cryptography.hazmat.primitives import serialization

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.asymmetric import padding

def printVariableAndSize(variableName, variable):

print(f"{variableName}: {variable}")

print(f"{variableName} length: {len(variable) \* 8} bits ({len(variable)} bytes)\n")

def alignToSize(string, size = 16):

# print(f"a: {len(string+bytes(chr(size-len(string)%size)\*(size-len(string)%size)))}")

return string+chr(size-len(string)%size).encode('utf-8')\*(size-len(string)%size)

def releaseZero(string):

return string[:-ord(string[-1])]

def encrypt(string, cipher):

encryptor = cipher.encryptor()

# printVariableAndSize("string", string)

# string = alignToSize(string)

# stringInBytes = bytes(string, 'utf-8')

# stringInBytes = alignToSize(stringInBytes)

stringInBytes = alignToSize(string)

# printVariableAndSize("stringInBytes", stringInBytes)

cryptoText = encryptor.update(stringInBytes)

encryptor.finalize()

# print(f"Encrypted text: {cryptoText}")

return cryptoText

def decrypt(string, cipher):

decryptor = cipher.decryptor()

text = decryptor.update(string)

decryptor.finalize()

text = text.decode('utf-8')

text = releaseZero(text)

# printVariableAndSize("Decrypted text", text)

return text

def readFromFile(fileName):

file = open(fileName, "br")

message = bytearray();

byte = file.read(1)

while byte:

message += byte

byte = file.read(1)

# message = message.decode('utf-8')

message = bytes(message)

file.close()

return message

def writeToFile(fileName, message):

file = open(fileName, "bw")

file.write(message)

file.close()

def generateSymmetricKey():

key = os.urandom(16)

return key

def getMessage(fileName):

message = readFromFile(fileName)

return message

def setupCipher(key):

cipher = Cipher(algorithms.AES(key), modes.ECB())

return cipher

def generatePrivateKey():

private\_key = rsa.generate\_private\_key(

public\_exponent = 65537,

key\_size = 1024,

)

return private\_key

def loadPrivateKey(fileName):

with open(fileName, "rb") as private\_key\_file:

private\_key = serialization.load\_pem\_private\_key(

private\_key\_file.read(),

password=None,

)

return private\_key

def loadPublicKey(fileName):

with open(fileName, "rb") as public\_key\_file:

public\_key = serialization.load\_pem\_public\_key(

public\_key\_file.read(),

)

return public\_key

def savePrivateKey(private\_key, fileName):

private\_pem = private\_key.private\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PrivateFormat.TraditionalOpenSSL,

encryption\_algorithm=serialization.NoEncryption()

)

writeToFile(fileName, private\_pem)

def savePublicKey(public\_key, fileName):

public\_pem = public\_key.public\_bytes(

encoding=serialization.Encoding.PEM,

format=serialization.PublicFormat.SubjectPublicKeyInfo

)

writeToFile(fileName, public\_pem)

def rsaEncrypt(message, public\_key):

cipherText = public\_key.encrypt(

message,

padding.OAEP(

mgf=padding.MGF1(algorithm=hashes.SHA256()),

algorithm=hashes.SHA256(),

label=None

)

)

return cipherText

def rsaDecrypt(cipherText, private\_key):

plaintext = private\_key.decrypt(

cipherText,

padding.OAEP(

mgf=padding.MGF1(algorithm=hashes.SHA256()),

algorithm=hashes.SHA256(),

label=None

)

)

return plaintext

def rsaSign(message, private\_key):

signature = private\_key.sign(

message,

padding.PSS(

mgf=padding.MGF1(hashes.SHA256()),

salt\_length=padding.PSS.MAX\_LENGTH

),

hashes.SHA256()

)

return signature

def rsaVerify(signature, message, public\_key):

public\_key.verify(

signature,

message,

padding.PSS(

mgf=padding.MGF1(hashes.SHA256()),

salt\_length=padding.PSS.MAX\_LENGTH

),

hashes.SHA256()

)

########################## SIPHER

print("++++++++++++SIPHER++++++++++++")

private\_key = generatePrivateKey()

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

savePublicKey(public\_key, "public\_key")

loadPublicKey("public\_key")

symmetricKey = generateSymmetricKey()

encryptedKey = rsaEncrypt(symmetricKey, public\_key)

writeToFile("encryptedKey", encryptedKey)

encryptedKey = readFromFile("encryptedKey")

symmetricKey = rsaDecrypt(encryptedKey, private\_key)

message = getMessage("test.txt")

cipher = setupCipher(symmetricKey)

encryptedMessage = encrypt(message, cipher)

writeToFile("Encrypted.txt", encryptedMessage)

decryptedMessage = decrypt(encryptedMessage, cipher)

printVariableAndSize("decryptedMessage", decryptedMessage)

decryptedMessage = bytes(decryptedMessage, 'utf-8')

printVariableAndSize("decryptedMessage in bytes", decryptedMessage)

writeToFile("Decrypted.txt", decryptedMessage)

########################## SIGN

print("++++++++++++SIGN++++++++++++")

private\_key = generatePrivateKey()

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

message = b'Sign test'

print(f"message: {message}")

signedMessage = rsaSign(message, private\_key)

writeToFile("signedMessage", signedMessage)

signedMessage = readFromFile("signedMessage")

print(f"signed Message: {signedMessage}")

rsaVerify(signedMessage, message, public\_key)

print("verification is done!")

4) Результаты работы приложения (экранные формы работы программы)

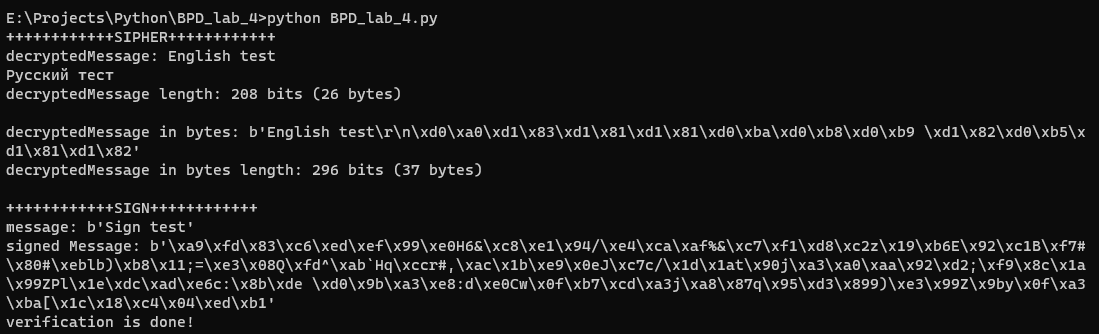


Рисунок 1 – Работа программы

Если электронная подпись была модифицирована, то вылетает исключение.

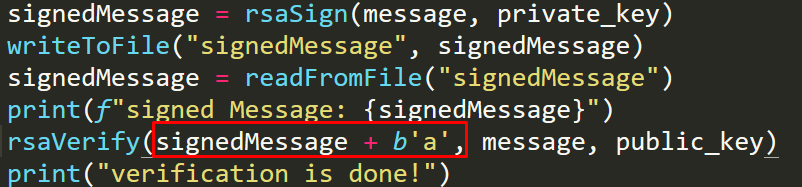


Рисунок 2 – Модификация подписанного сообщения

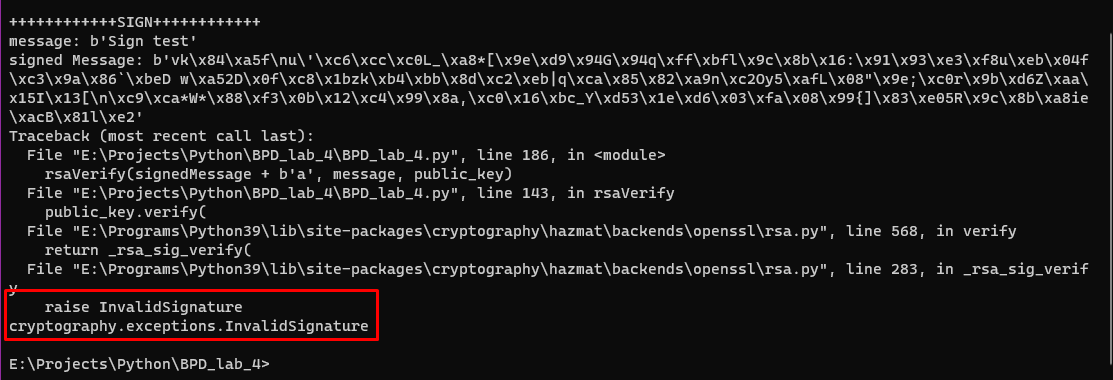


Рисунок 3 – Подписанное сообщение не прошло верификацию