МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт КНТ

Факультет ФИСП

Кафедра ПИ им. Л.П. Фельдмана

Лабораторная работа №6

Тема: «Использование стеганографических и криптографических средств защиты информации.»

Курс: Безопасность программ и данных

Выполнил

ст. гр. ПИ-18б

Моргунов А.Г.

Проверил

Ногтев Е.А.

Донецк – 2021

ХОД РАБОТЫ

ВАРИАНТ 5

Изучить существующие программные продукты для стеганографический защиты информации.

Написать программу защиты информации с использованием криптографических (ранее изученных) и стеганографических (LSB) алгоритмов. В качестве контейнера использовать графический формат BMP.

В алгоритме LSB число используемых младших бит четные варианты - 2 бита, **нечетные варианты - 1 бит.**

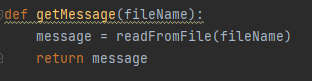
В качестве симметричного алгоритма шифрования использовать алгоритм AES. Четные варианты используют режим CBC. **Нечетные варианты используют режим CFB.**

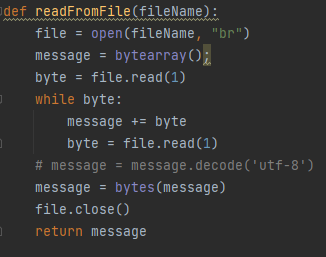
Для проверки целостности данных использовать однонаправленные хеш-функции. Четные варианты- MD5, **нечетные варианты - SHA2** В конец файла хеш добавляют четные варианты, в **начало файла - нечетные варианты**.

В рамках выполнения лабораторной работы необходимо:

1) Считать файл для дальнейшего шифрования симметричным алгоритмом (AES)







2) Получить хеш с использованием однонаправленной хеш-функции.

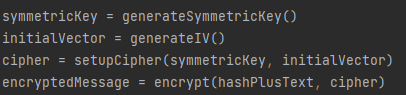




3) Добавить хеш к файлу.



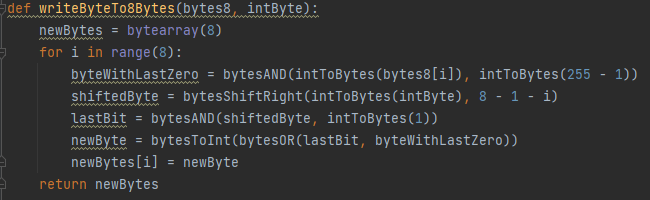
4) Зашифровать файл симметричным алгоритмом. (лаба 4)



5) Поместить зашифрованный файл в BMP-контейнер по алгоритму LSB.

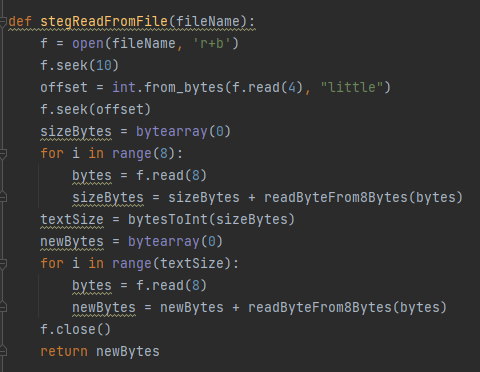


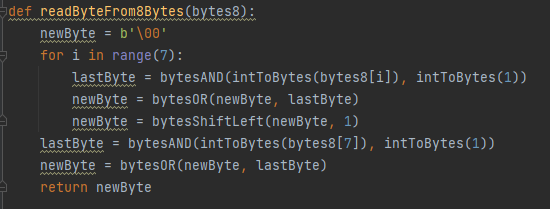




6) Извлечь данные из BMP-контейнера.





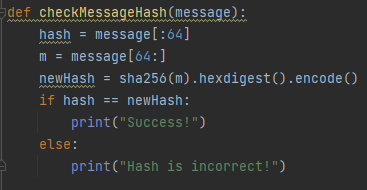


7) расшифровать с использованием ключа симметричного алгоритма.  
(лаба 4)



8) Проверить целостность данных.





**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()

)

import shutil

import sys

from BPD\_lab\_4 import \*

from hashlib import sha256

def intToBytes(x):

return x.to\_bytes(1, "little")

def bytesToInt(x):

return int.from\_bytes(x, "little")

def bytesAND(x, y):

ix = bytesToInt(x)

iy = bytesToInt(y)

return intToBytes(iy & ix)

def bytesOR(x, y):

ix = bytesToInt(x)

iy = bytesToInt(y)

return intToBytes(iy | ix)

def bytesShiftRight(x, y):

ix = bytesToInt(x)

return intToBytes(ix >> y)

def bytesShiftLeft(x, y):

ix = bytesToInt(x)

return intToBytes(ix << y)

def stegWriteToFile(fileName, text):

f = open(fileName, 'r+b')

f.seek(10)

offset = int.from\_bytes(f.read(4), "little")

f.seek(offset)

# encodedText = text.encode("utf-8")

encodedText = text

textSize = len(encodedText)

for i in range(textSize):

bytes = f.read(8)

intByte = encodedText[i]

# print("byte = " + str(bin(intByte)))

newBytes = writeByteTo8Bytes(bytearray(8), intByte)

f.seek(f.tell() - 8)

f.write(newBytes)

f.close()

def writeByteTo8Bytes(bytes8, intByte):

newBytes = bytearray(8)

for i in range(8):

byteWithLastZero = bytesAND(intToBytes(bytes8[i]), intToBytes(255 - 1))

shiftedByte = bytesShiftRight(intToBytes(intByte), 8 - 1 - i)

lastBit = bytesAND(shiftedByte, intToBytes(1))

newByte = bytesToInt(bytesOR(lastBit, byteWithLastZero))

newBytes[i] = newByte

return newBytes

def readByteFrom8Bytes(bytes8):

newByte = b'\00'

for i in range(7):

lastByte = bytesAND(intToBytes(bytes8[i]), intToBytes(1))

newByte = bytesOR(newByte, lastByte)

newByte = bytesShiftLeft(newByte, 1)

lastByte = bytesAND(intToBytes(bytes8[7]), intToBytes(1))

newByte = bytesOR(newByte, lastByte)

return newByte

def stegReadFromFile(fileName):

f = open(fileName, 'r+b')

f.seek(10)

offset = int.from\_bytes(f.read(4), "little")

f.seek(offset)

sizeBytes = bytearray(0)

for i in range(8):

bytes = f.read(8)

sizeBytes = sizeBytes + readByteFrom8Bytes(bytes)

textSize = bytesToInt(sizeBytes)

newBytes = bytearray(0)

for i in range(textSize):

bytes = f.read(8)

newBytes = newBytes + readByteFrom8Bytes(bytes)

f.close()

return newBytes

def generateIV():

iv = os.urandom(16)

return iv

def checkMessageHash(message):

hash = message[:64]

m = message[64:]

newHash = sha256(m).hexdigest().encode()

if hash == newHash:

print("Success!")

else:

print("Hash is incorrect!")

def getHash(message):

return sha256(message).hexdigest().encode()

original = "sample\_640×426.bmp"

target = "copy.bmp"

shutil.copyfile(original, target)

text = getMessage("TextFile.txt")

hash = getHash(text)

print(len(hash))

hashPlusText = hash + text

symmetricKey = generateSymmetricKey()

initialVector = generateIV()

cipher = setupCipher(symmetricKey, initialVector)

encryptedMessage = encrypt(hashPlusText, cipher)

printVariableAndSize("encryptedMessage before", encryptedMessage)

textSize = len(encryptedMessage)

newText = (textSize).to\_bytes(8, 'little') + encryptedMessage

printVariableAndSize("newText", newText)

stegWriteToFile(target, newText)

encryptedMessage = stegReadFromFile(target)

printVariableAndSize("encryptedMessage after", encryptedMessage)

decryptedMessage = decrypt(encryptedMessage, cipher)

printVariableAndSize("decryptedMessage", decryptedMessage)

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

printVariableAndSize("decryptedMessage in bytes", decryptedMessage)

checkMessageHash(decryptedMessage)

**Результаты работы приложения (экранные формы работы программы)**

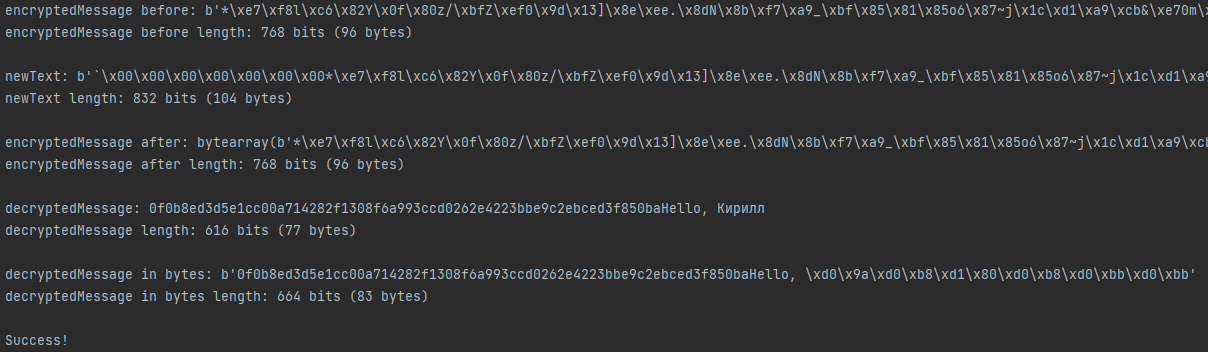


Рисунок 1 – Работа программы