Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра теоретической и прикладной информатики

Лабораторная работа №5  
“Криптографические библиотеки”

по дисциплине “Теория информации и криптографии”

Факультет: ПМИ

Группа: ПМИ-12

Бригада: 10

Студент(-ы): Курдюков И.Н.

Омельницкая Е.И.

Преподаватель: Авдеенко Т.В.

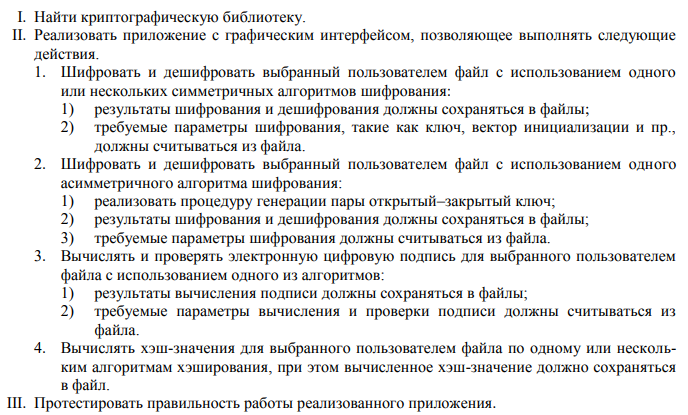
Новосибирск

2024

**1) Цель работы**

Познакомиться с существующими криптографическими библиотеками. Научиться использовать сторонние криптографические библиотеки при разработке собственных приложений.

**2) Задание**



**3) Описание метода решения заданий**

Использованные библиотеки

Hashbase – библиотека, позволяющая вычислять хэш-функции.

Cryptography – библиотека, позволяющая шифровать и дешифровать путём симметричных и асимметричных алгоритмов.

PyCrypto - библиотека, позволяющая шифровать и дешифровать путём симметричных и асимметричных алгоритмов.

**4) Описание разработанного программного средства**

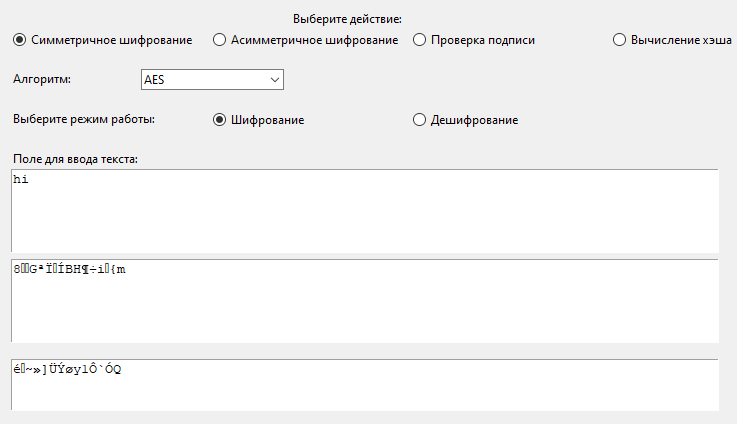
Разработанное приложение позволяет пользователю шифровать и дешифровать данные с помощью симметричных и асимметричных алгоритмов, вычислять хэш-функции, а также проводить верификацию электронной подписи.

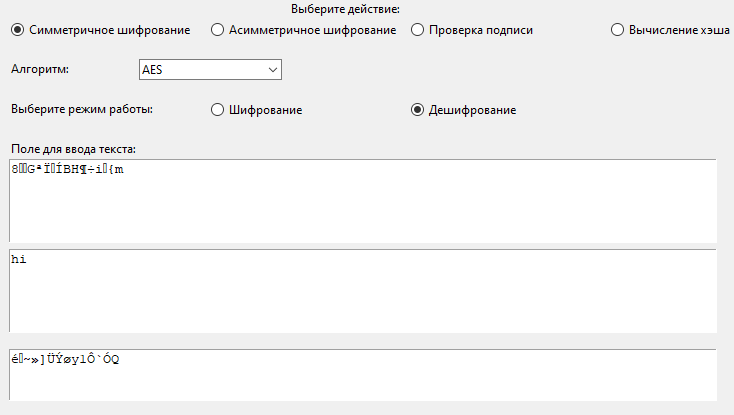
**5) Программный код**

from tkinter import \* *#GUI*from tkinter.ttk import \* *#GUI*from hashbase import \* *#HASH*from Crypto.Cipher import AES  
from Crypto.Random import get\_random\_bytes  
from Crypto.Util.Padding import pad, unpad *#simm*from cryptography.hazmat.primitives.asymmetric import rsa *#asimm*from cryptography.hazmat.primitives.asymmetric import padding  
from cryptography.hazmat.primitives import hashes  
from cryptography.hazmat.primitives import serialization  
  
from cryptography.hazmat.backends import default\_backend *#signature\_verification  
  
#--------------------------------------------1-------------------------------------------#*def symmetric\_encryption(alg, data):  
 if alg == "AES":  
 global encr\_d  
 global g\_key  
 data = data.encode()  
  
 *# Генерируем случайный ключ длиной 16 байт* key = get\_random\_bytes(16)  
 g\_key = key  
 *# Создаем объект AES с ключом* cipher = AES.new(key, AES.MODE\_ECB)  
  
 *# Выравниваем данные по границам блока* data = pad(data, AES.block\_size)  
  
 *# Шифруем данные* encrypted\_data = cipher.encrypt(data)  
 encr\_d = encrypted\_data  
 return[encrypted\_data, key]  
  
 if alg == "DES":  
 print("encryp")  
 print("encryp")  
  
def symmetric\_decryption(alg, encrypted\_data, key):  
 if alg == "AES":  
 cipher = AES.new(key, AES.MODE\_ECB)  
 decrypted\_data = unpad(cipher.decrypt(encrypted\_data), AES.block\_size)  
 return decrypted\_data  
  
 if alg == "DES":  
 print("encryp")  
 print("decryp")  
  
  
*#--------------------------------------------2-------------------------------------------#  
#асимметричное шифрование*def asymmetric\_encryption(alg, message):  
 if alg == "RSA":  
 global ciphertext\_g, private\_key\_g  
 message = message.encode()  
 *# Генерация пары ключей* private\_key = rsa.generate\_private\_key(public\_exponent=65537, key\_size=2048)  
  
 public\_key = private\_key.public\_key()  
  
 public\_key\_bytes = public\_key.public\_bytes(  
 encoding=serialization.Encoding.PEM,  
 format=serialization.PublicFormat.SubjectPublicKeyInfo  
 )  
 private\_key\_bytes = private\_key.private\_bytes(  
 encoding=serialization.Encoding.PEM,  
 format=serialization.PrivateFormat.TraditionalOpenSSL,  
 encryption\_algorithm=serialization.NoEncryption()  
 )  
 ciphertext = public\_key.encrypt(  
 message,  
 padding.OAEP(  
 mgf=padding.MGF1(algorithm=hashes.SHA256()),  
 algorithm=hashes.SHA256(),  
 label=None))  
 ciphertext\_g = ciphertext  
 private\_key\_g = private\_key  
 return[private\_key\_bytes.decode(), public\_key\_bytes.decode(), ciphertext]  
 if alg == "DSA":  
 print("encryp")  
 print("encryp")  
  
def asymmetric\_decryption(alg, ciphertext, private\_key):  
 if alg == "RSA":  
 plaintext = private\_key\_g.decrypt(  
 ciphertext\_g,  
 padding.OAEP(  
 mgf=padding.MGF1(algorithm=hashes.SHA256()),  
 algorithm=hashes.SHA256(),  
 label=None))  
 return plaintext  
 if alg == "DSA":  
 print("encryp")  
 print("decryp")  
  
*#--------------------------------------------3-------------------------------------------#*def signature\_verification(message):  
 private\_key, public\_key = generate\_key()  
 message = message.encode()  
 signature = sign\_message(private\_key, message)  
 print(signature)  
 is\_valid = verify\_signature(public\_key, message, signature)  
 public\_key\_bytes = public\_key.public\_bytes(  
 encoding=serialization.Encoding.PEM,  
 format=serialization.PublicFormat.SubjectPublicKeyInfo  
 )  
 private\_key\_bytes = private\_key.private\_bytes(  
 encoding=serialization.Encoding.PEM,  
 format=serialization.PrivateFormat.TraditionalOpenSSL,  
 encryption\_algorithm=serialization.NoEncryption()  
 )  
 return [is\_valid, private\_key\_bytes.decode(), public\_key\_bytes.decode()]  
  
def generate\_key():  
 private\_key = rsa.generate\_private\_key(  
 public\_exponent=65537,  
 key\_size=2048,  
 backend=default\_backend()  
 )  
 public\_key = private\_key.public\_key()  
 return private\_key, public\_key  
  
def sign\_message(private\_key, message):  
 signature = private\_key.sign(  
 message,  
 padding.PSS(  
 mgf=padding.MGF1(hashes.SHA256()),  
 salt\_length=padding.PSS.MAX\_LENGTH  
 ),  
 hashes.SHA256()  
 )  
 return signature  
  
def verify\_signature(public\_key, message, signature):  
 try:  
 public\_key.verify(  
 signature,  
 message,  
 padding.PSS(  
 mgf=padding.MGF1(hashes.SHA256()),  
 salt\_length=padding.PSS.MAX\_LENGTH  
 ),  
 hashes.SHA256()  
 )  
 return True  
 except InvalidSignature:  
 return False  
  
*#--------------------------------------------4-------------------------------------------#  
#вычисление хэша*def hash(alg, txt):  
 match alg:  
 case "MD5":  
 return MD5().generate\_hash(txt)  
 case "SHA-1":  
 return SHA1().generate\_hash(txt)  
 case "SHA-224":  
 return SHA512\_224().generate\_hash(txt)  
 case "SHA-256":  
 return SHA512\_256().generate\_hash(txt)  
 case "SHA-512":  
 return SHA512().generate\_hash(txt)  
 case "Ripemd-256":  
 return RIPEMD256().generate\_hash(txt)  
  
*#--------------------------------------------GUI-------------------------------------------#*def run():  
 n = method\_mode.get()  
 match n:  
 case 0:  
 text2.delete(1.0, END)  
 alg = combobox\_0.get()  
 encryp\_or\_decryp = cr\_mode.get()  
 txt = text1.get(1.0, END).rstrip()  
 if encryp\_or\_decryp == 0:  
 text3.delete(1.0, END)  
 res = symmetric\_encryption(alg, txt)  
 text3.insert(1.0, res[1])  
 text2.insert(1.0, res[0])  
 else:  
 res = symmetric\_decryption(alg, encr\_d, g\_key)  
 text2.insert(1.0, res)  
 case 1:  
 text2.delete(1.0, END)  
 alg = combobox\_1.get()  
 encryp\_or\_decryp = cr\_mode.get()  
 txt = text1.get(1.0, END).rstrip()  
 if encryp\_or\_decryp == 0:  
 pr\_key, op\_key, t = asymmetric\_encryption(alg, txt)  
 text2.insert(1.0, t)  
 text4.insert(1.0, op\_key)  
 text5.insert(1.0, pr\_key)  
 else:  
 private\_key = text5.get(1.0, END).rstrip()  
 res = asymmetric\_decryption(alg, txt, private\_key)  
 text2.insert(1.0, res)  
 case 2:  
 text2.delete(1.0, END)  
 txt = text1.get(1.0, END).rstrip()  
 res=signature\_verification(txt)  
 if res[0]:  
 text2.insert(1.0, "Верификация цифровой подписи прошла успешно")  
 else:  
 text2.insert(1.0, "Цифровая подпись верификацию не прошла")  
 text4.insert(1.0, res[2])  
 text5.insert(1.0, res[1])  
 case 3:  
 text2.delete(1.0, END)  
 alg = combobox\_3.get()  
 txt = text1.get(1.0, END).rstrip()  
 res = hash(alg, txt)  
 print(res)  
 text2.insert(1.0, res)  
  
  
def change\_method():  
 n = method\_mode.get()  
 match n:  
 case 0:  
 combobox\_0.place(x=150, y=70)  
 combobox\_1.place\_forget()  
 combobox\_2.place\_forget()  
 combobox\_3.place\_forget()  
 crypt\_mode\_0.place(x=220, y=110)  
 crypt\_mode\_1.place(x=420, y=110)  
 text3.place(x=20, y=360)  
 text4.place\_forget()  
 text5.place\_forget()  
 case 1:  
 combobox\_1.place(x=150, y=70)  
 combobox\_0.place\_forget()  
 combobox\_2.place\_forget()  
 combobox\_3.place\_forget()  
 crypt\_mode\_0.place(x=220, y=110)  
 crypt\_mode\_1.place(x=420, y=110)  
 text4.place(x=20, y=360)  
 text5.place(x=20, y=450)  
 text3.place\_forget()  
 case 2:  
 combobox\_2.place(x=150, y=70)  
 combobox\_1.place\_forget()  
 combobox\_0.place\_forget()  
 combobox\_3.place\_forget()  
 crypt\_mode\_0.place\_forget()  
 crypt\_mode\_1.place\_forget()  
 text3.place\_forget()  
 text4.place(x=20, y=360)  
 text5.place(x=20, y=450)  
 case 3:  
 combobox\_3.place(x=150, y=70)  
 combobox\_1.place\_forget()  
 combobox\_2.place\_forget()  
 combobox\_0.place\_forget()  
 crypt\_mode\_0.place\_forget()  
 crypt\_mode\_1.place\_forget()  
 text3.place\_forget()  
 text4.place\_forget()  
 text5.place\_forget()  
  
win = Tk()  
win.title('Криптография')  
win.resizable(False, False)  
w, h = 750, 600  
win.geometry(f"{w}x{h}+{(win.winfo\_screenwidth()-w)//2}+{(win.winfo\_screenheight()-h)//2}") *#центр*method\_mode = IntVar(value=0)  
cr\_mode = IntVar(value=0)  
  
algorithms\_0 = ["AES"]  
algorithm\_mode\_0 = StringVar(value=algorithms\_0[0])  
  
algorithms\_1 = ["RSA"]  
algorithm\_mode\_1 = StringVar(value=algorithms\_1[0])  
  
algorithms\_2 = ["RSA"]  
algorithm\_mode\_2 = StringVar(value=algorithms\_2[0])  
  
algorithms\_3 = ["MD5", "SHA-1", "SHA-224", "SHA-256", "SHA-512", "Ripemd-256"]  
algorithm\_mode\_3 = StringVar(value=algorithms\_3[0])  
  
lang = StringVar(value="Выберите действие:")  
lang2 = StringVar(value="Алгоритм:")  
lang3 = StringVar(value="Поле для ввода текста:")  
lang4 = StringVar(value="Выберите режим работы:")  
  
header = Label(textvariable=lang).place(x=300, y=10)  
  
method\_btn = Radiobutton(text="Симметричное шифрование", value=0, variable=method\_mode, command=change\_method).place(x=20, y=30)  
method\_btn = Radiobutton(text="Асимметричное шифрование", value=1, variable=method\_mode, command=change\_method).place(x=220, y=30)  
method\_btn = Radiobutton(text="Проверка подписи", value=2, variable=method\_mode, command=change\_method).place(x=420, y=30)  
method\_btn = Radiobutton(text="Вычисление хэша", value=3, variable=method\_mode, command=change\_method).place(x=620, y=30)  
  
crypt\_mode\_0 = Radiobutton(text="Шифрование", value=0, variable=cr\_mode)  
crypt\_mode\_0.place(x=220, y=110)  
crypt\_mode\_1 = Radiobutton(text="Дешифрование", value=1, variable=cr\_mode)  
crypt\_mode\_1.place(x=420, y=110)  
  
combobox\_0 = Combobox(textvariable=algorithm\_mode\_0, values=algorithms\_0)  
combobox\_0.place(x=150, y=70)  
combobox\_1 = Combobox(textvariable=algorithm\_mode\_1, values=algorithms\_1)  
combobox\_1.place(x=150, y=70)  
combobox\_2 = Combobox(textvariable=algorithm\_mode\_2, values=algorithms\_2)  
combobox\_2.place(x=150, y=70)  
combobox\_3 = Combobox(textvariable=algorithm\_mode\_3, values=algorithms\_3)  
combobox\_3.place(x=150, y=70)  
  
combobox\_1.place\_forget()  
combobox\_2.place\_forget()  
combobox\_3.place\_forget()  
  
header = Label(textvariable=lang2).place(x=20, y=70)  
header = Label(textvariable=lang4).place(x=20, y=110)  
header = Label(textvariable=lang3).place(x=20, y=150)  
  
text1 = Text(width=88, height=5)  
text1.place(x=20, y=170)  
text2 = Text(width=88, height=5)  
text2.place(x=20, y=260)  
text3 = Text(width=88, height=3)  
text3.place(x=20, y=360)  
  
text4 = Text(width=88, height=5)  
text5 = Text(width=88, height=5)  
  
button = Button(text="Старт", command=run, width=52).place(x=200, y=550)  
  
win.mainloop()

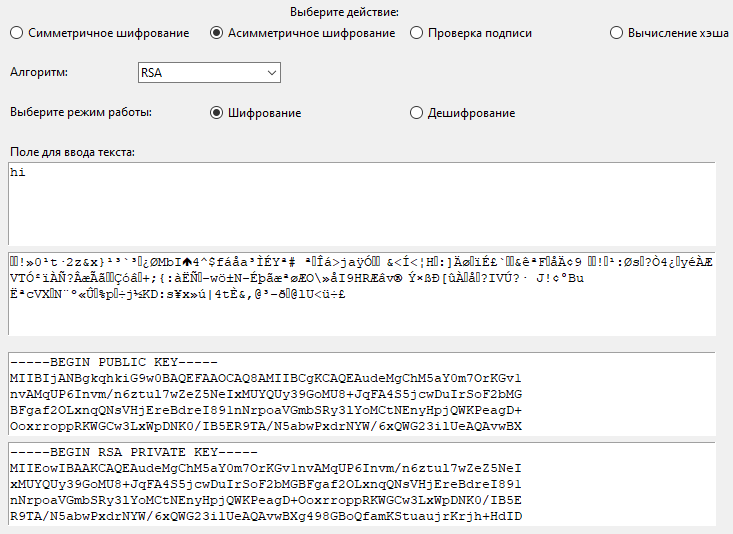
**6) Тестирование программы**

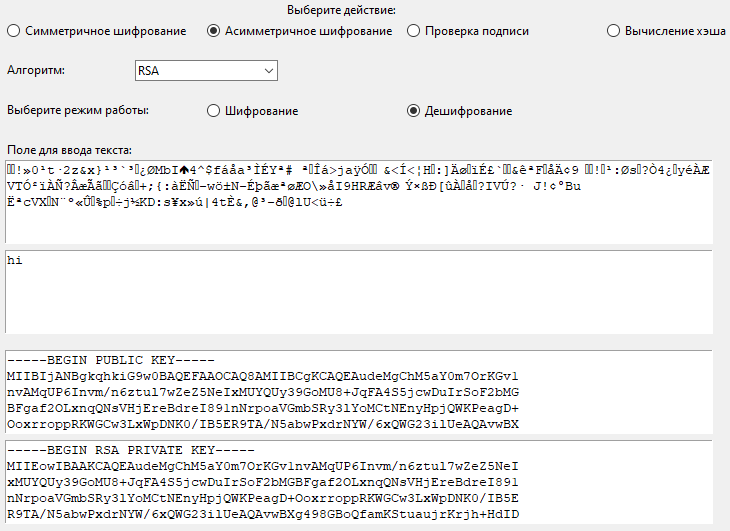
1) Симметричное шифрование и дешифрование



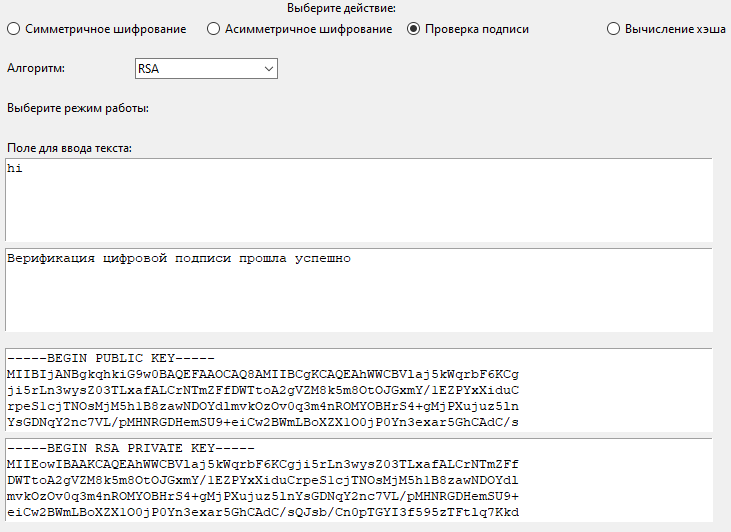


2) Асимметричное шифрование и дешифрование





3) Верификация электронной подписи



4) Вычисление хэш-функции

