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 №10**

**по курсу**

**«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»**

Работу выполнил

Студент 49 группы

Нагалевский А.М.

Преподаватель:

Крамаренко А.А.

Краснодар 2024

**Постановка задачи.**

Реализовать программный продукт, эмулирующий обмен

зашифрованными сообщениями между двумя пользователями. Шифрование

сообщения происходит любым блочным симметричным шифром в режиме

электронной книги. Ключи для шифрования и расшифрования для

симметричной системы формируются автоматически перед началом работы

системы и доступны двум пользователям. Пользователь Алиса выбирает

ключ и зашифровывает его номер произвольным ассиметричным

алгоритмом. Открытый и закрытый ключ записываются в файл.

Пользователь Боб принимает номер ключа шифрования, расшифровывает его

и выбирает нужный ключ для обмена закрытой информацией. После чего

Алиса получает открытый текст(клавиатура или файл), шифрует его

установленным симметричным алгоритмом и отправляет Бобу. Боб должен

его расшифровать. Шифрование реализовать для двух различных кодировок

текста. Для реализации криптоалгоритмов возможно пользоваться

встроенными библиотеками используемых языков.

Основная задача программного продукта состоит в эмуляции обмена зашифрованными сообщениями между двумя пользователями - Алисой и Бобом. Программа использует блочный симметричный шифр в режиме электронной книги для шифрования сообщений. Ключи для шифрования и расшифрования генерируются автоматически перед началом работы системы и доступны обоим пользователям.

Процесс работы программы следующий:

**Генерация ключей:**

Перед началом работы системы генерируются ключи для симметричного шифра.

Пользователь Алиса выбирает ключ для шифрования и зашифровывает его номер с помощью ассиметричного алгоритма. Открытый и закрытый ключ записываются в файл.

**Обмен сообщениями:**

Пользователь Боб принимает номер ключа шифрования от Алисы, расшифровывает его и выбирает нужный ключ для обмена закрытой информацией.

Алиса получает открытый текст (ввод с клавиатуры или из файла), шифрует его установленным симметричным алгоритмом и отправляет Бобу.

Боб расшифровывает полученное сообщение.

**Шифрование текста:**

Шифрование текста реализовано для двух различных кодировок текста.

Программа использует встроенные библиотеки используемого языка программирования для реализации криптоалгоритмов. Она обеспечивает безопасный обмен зашифрованными сообщениями между пользователями, используя симметричное и асимметричное шифрование.

**Текст программы:**

**Файл Lab\_10.py:**

from cryptography.hazmat.primitives import serialization, padding, hashes  
from cryptography.hazmat.primitives.asymmetric import rsa  
from cryptography.hazmat.backends import default\_backend  
from Crypto.Cipher import AES  
from Crypto.Util.Padding import pad, unpad  
import base64  
import hashlib  
import secrets  
  
# Функция для генерации случайного ключа  
def generate\_symmetric\_key():  
 return secrets.token\_bytes(32) # Генерируем ключ длиной 32 байта (256 бит)  
  
# Функция для шифрования сообщения  
def encrypt\_message(message, key):  
 cipher = AES.new(key, AES.MODE\_ECB)  
 padded\_message = pad(message.encode(), AES.block\_size)  
 encrypted\_message = cipher.encrypt(padded\_message)  
 return base64.b64encode(encrypted\_message)  
  
# Функция для дешифрования сообщения  
def decrypt\_message(encrypted\_message, key):  
 cipher = AES.new(key, AES.MODE\_ECB)  
 encrypted\_message = base64.b64decode(encrypted\_message)  
 decrypted\_message = cipher.decrypt(encrypted\_message)  
 return unpad(decrypted\_message, AES.block\_size).decode()  
  
# Функция для генерации ключей RSA и сохранения их в файл  
def generate\_and\_save\_rsa\_keys(file\_prefix):  
 # Генерация закрытого и открытого ключей RSA  
 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(f"{file\_prefix}\_private.pem", "wb") as private\_key\_file:  
 private\_key\_pem = private\_key.private\_bytes(  
 encoding=serialization.Encoding.PEM,  
 format=serialization.PrivateFormat.TraditionalOpenSSL,  
 encryption\_algorithm=serialization.NoEncryption()  
 )  
 private\_key\_file.write(private\_key\_pem)  
  
 # Сохранение открытого ключа в файл  
 with open(f"{file\_prefix}\_public.pem", "wb") as public\_key\_file:  
 public\_key\_pem = public\_key.public\_bytes(  
 encoding=serialization.Encoding.PEM,  
 format=serialization.PublicFormat.SubjectPublicKeyInfo  
 )  
 public\_key\_file.write(public\_key\_pem)  
  
 print("Ключи RSA успешно сгенерированы и сохранены в файлах.")  
  
# Функция для загрузки ключа RSA из файла  
def load\_rsa\_key\_from\_file(file\_path, key\_type):  
 with open(file\_path, "rb") as key\_file:  
 if key\_type == "public":  
 return serialization.load\_pem\_public\_key(  
 key\_file.read(),  
 backend=default\_backend()  
 )  
 elif key\_type == "private":  
 return serialization.load\_pem\_private\_key(  
 key\_file.read(),  
 password=None,  
 backend=default\_backend()  
 )  
  
# Функция для выбора симметричного ключа по номеру  
def select\_symmetric\_key(symmetric\_key\_number):  
 # Преобразование номера ключа в байтовую строку  
 symmetric\_key\_number\_bytes = str(symmetric\_key\_number).encode()  
 # Хеширование номера ключа для генерации симметричного ключа  
 hashed\_key = hashlib.sha256(symmetric\_key\_number\_bytes).digest()  
 return hashed\_key  
  
# Пример использования  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Генерируем и сохраняем ключи RSA для пользователя Алисы  
 file\_prefix = "alice\_keys"  
 generate\_and\_save\_rsa\_keys(file\_prefix)  
  
 # Генерируем общий симметричный ключ для шифрования сообщений  
 symmetric\_key = generate\_symmetric\_key()  
  
 # Пользователь 1 (Алиса) отправляет сообщение пользователю 2 (Бобу)  
 plaintext\_to\_bob = "Привет, Боб!"  
 encrypted\_message\_to\_bob = encrypt\_message(plaintext\_to\_bob, symmetric\_key)  
 print("Зашифрованное сообщение от Алисы для Боба:", encrypted\_message\_to\_bob)  
  
 # Пользователь 2 (Боб) получает и дешифрует сообщение от пользователя 1 (Алисы)  
 decrypted\_message\_from\_alice = decrypt\_message(encrypted\_message\_to\_bob, symmetric\_key)  
 print("Дешифрованное сообщение для Боба от Алисы:", decrypted\_message\_from\_alice)  
  
 # Генерируем и сохраняем ключи RSA для пользователя Боба  
 file\_prefix = "bob\_keys"  
 generate\_and\_save\_rsa\_keys(file\_prefix)  
  
 # Пользователь Боб загружает свой закрытый ключ RSA  
 bob\_private\_key = load\_rsa\_key\_from\_file("bob\_keys\_private.pem", "private")  
  
 # Пользователь Боб принимает симметричный ключ от Алисы  
 symmetric\_key\_from\_alice = b'...' # Предположим, что ключ был передан напрямую  
  
 # Пользователь Боб выбирает соответствующий симметричный ключ для обмена сообщениями с Алисой  
 symmetric\_key = select\_symmetric\_key(symmetric\_key\_from\_alice)  
  
 # Пользователь Боб принимает открытый текст от Алисы и шифрует его  
 # В качестве примера, предположим, что открытый текст - это просто строка  
 plaintext\_from\_alice = "Hello, Alice!"  
 # Шифрование для двух различных кодировок текста: ASCII и UTF-8  
 encrypted\_message\_ascii = encrypt\_message(plaintext\_from\_alice, symmetric\_key)  
 encrypted\_message\_utf8 = encrypt\_message(plaintext\_from\_alice, symmetric\_key)  
 print("Зашифрованное сообщение от Алисы для Боба (ASCII):", encrypted\_message\_ascii)  
 print("Зашифрованное сообщение от Алисы для Боба (UTF-8):", encrypted\_message\_utf8)  
  
 # Пользователь 2 (Боб) получает и дешифрует сообщение от пользователя 1 (Алисы)  
 decrypted\_message\_ascii = decrypt\_message(encrypted\_message\_ascii, symmetric\_key)  
 decrypted\_message\_utf8 = decrypt\_message(encrypted\_message\_utf8, symmetric\_key)  
 print("Дешифрованное сообщение для Боба от Алисы (ASCII):", decrypted\_message\_ascii)  
 print("Дешифрованное сообщение для Боба от Алисы (UTF-8):", decrypted\_message\_utf8)