Министерство науки и высшего образования Российской Федерации
САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
Факультет Безопасности информационных технологий

Дисциплина: Разработка систем аутентификации и криптографии

Отчет

по лабораторной работе №3

«Реализация алгоритма шифрования RSA»

Выполнил
Магистрант учебной группы N42514c
Васильев Роман Александрович

Проверил:
Федров Иван Романович

Санкт-Петербург
2020 г.

Цель работы.
Часть 1: изучить основные принципы работы алгоритма RSA и реализовать без использования криптографических библиотек.

Часть 2: изучить способы осуществления цифровой подписи приложений и подписать свое приложение так, чтобы во вкладке «Цифровые подписи» отображалась моя фамилия.

Выполнение работы

Часть 1.

Для реализации алгоритма RSA был выбрал язык программирования Python 3, так как он предоставляет возможность удобной работы с очень большими числами и математическими вычислениями.

Для GUI использовался фреймворк PyQT5.

Исходный код представлен в Приложении и по ссылке:

https://github.com/DaCentDD/Cryptography/tree/master/RSA

RSA – криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других

Алгоритм RSA состоит из следующих шагов:

1. Выбор двух больших неодинаковых простых чисел p и q.
2. Находится их произведение n = p*q, которое называется модулем.
3. Вычисляется функция Эйлера от числа n. Φ(n) = (p-1)(q-1)
4. Находится открытая экспонента e, которая является взаимно простой с Φ(n). Обычно используются числа Ферма.
5. Находится секретная экспонента d, которая мультипликативно обратная к e по модулю Φ(n).
6. Далее пара {e, n} используется в качестве открытого ключа, а пара {d, n} в качестве закрытого ключа.
7. Для зашифровки используется выражение $C = m^e mod(n)$, где m – сообщение.
8. Для расшифровки $m = C^d mod(n)$

Результаты работы программы представлены на рисунке 1 и 2.

Рисунок 1. Результат работы программы при вводе текста «Hello world!».

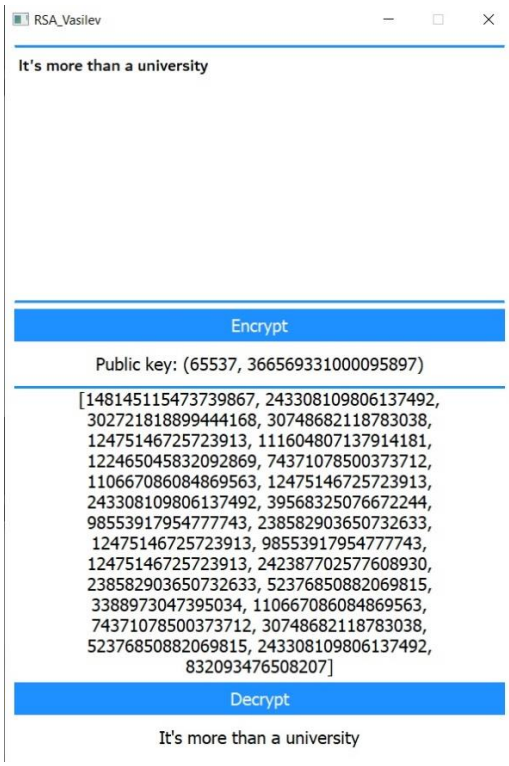

Рисунок 2. Результат работы программы при вводе текста «It’s more than a university».

Часть 2.

Для подписи приложения использовалась утилита PKI Client.

С помощью команды New-SelfSignedCertificate можно создать само подписанный сертификат.
Использовались следующие методы:

-Type – уточняет тип сертификата.

-Subject – строка-описание сертификата.

-KeyUsage – определяет использование ключа.

-FriendlyName – доверенное лицо.

-CertStoreLocation – место хранения сертификата.

Далее сертификат присвоился переменной с помощью команды: `$cert=Get-ChildItem -Path cert:\CurrentUser\my -CodeSigningCert`

И в конце с помощью следующей команды подписываем приложение:

```
Set-AuthenticodeSignature rsa.exe $cert -HashAlgorithm RSA
```

Процесс подписывания приложения можно увидеть на рисунке 3.

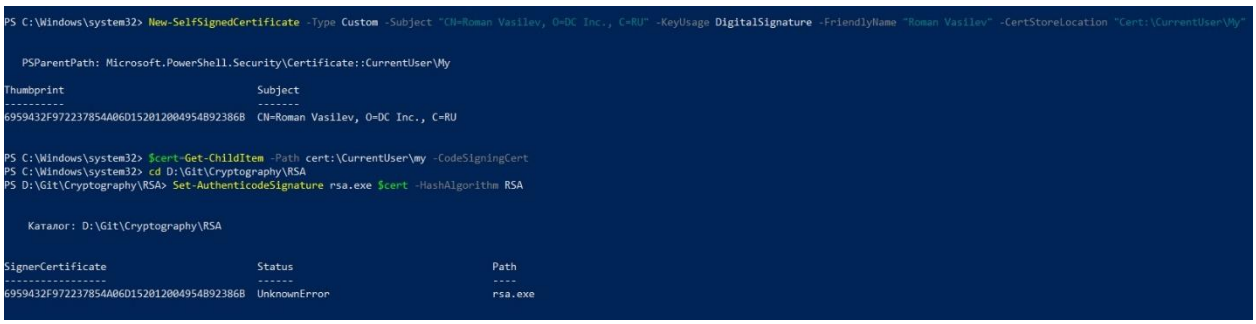

Рисунок 3. Процесс подписывания приложения.

Приложение

Файл **main.py**

```
import math
import random
import sys

from PyQt5 import QtWidgets, QtCore
from RSA_UI import Ui_MainWindow  # Импорт GUI


encrypted = []
e, d, n = 0, 0, 0
dim = 64  # Размерность генерируемого ключа в битах.


def is_prime(num, r):  # Проверка на простоту Миллера-Рабина
    for i in range(r):
        s = 2
        while True:
            t = (num - 1) / 2 ** s
            if int(t) == 0:
                s, t = 3, 1
            if t % int(t) == 0 and t % 2 == 1:
                t = int(t)
                break
            s += 1
        a = random.randint(2, num - 2)
        x = pow(a, t, num)
        if x == 1 or x == (num - 1):
            continue
        for i in range(s - 1):
            x = pow(x, 2, num)
            if x == 1:
                return False
            if x == (num - 1):
                continue
        return False
    return True
```

```
def prime_num():  # Генератор простого числа
    while True:
        num = random.randint(2 ** (dim/2 - 1), 2 ** (dim/2) - 1)  # Генерируем число указанной размерности / 2
        r = math.ceil(math.log2(num)) // 4  # Выбираем количество раундов равным порядку log2(n)//4
        if is_prime(num, r):  # Проверяем простое ли оно
            return num


def opened_exp(euler):  # Вычисление открытой экспоненты
    ferms = [3, 5, 17, 527, 65537]  # Числа Ферма
    for ferm in ferms:
        if math.gcd(euler, ferm) == 1:  # Если НОД равен единице, то числа взаимно простые
            return ferm


def bezout_recursive(a, b):  # Вычисление коэффициенов Безу
    if not b:
        return 1, 0, a
    y, x, g = bezout_recursive(b, a % b)
    return x, y - (a // b) * x, g


def closed_exp(e, euler):  # Вычисление закрытой экспоненты с помощью Расширенной теоремы Евклида и соотношения Безу
    x = bezout_recursive(e, euler)
    return x[0] + euler if x[0] < 0 else x[0]


class GenKey(QtCore.QThread):  # Генерация ключей в потоке-воркере
    set_key_text = QtCore.pyqtSignal(int, int)
    set_enc_text = QtCore.pyqtSignal(list)
    finish = QtCore.pyqtSignal()

    def __init__(self, coded_message):
        QtCore.QThread.__init__(self)
        self.coded_message = coded_message
```

```
    def run(self):
        global e, d, n, encrypted
        p = prime_num()  # Выбираются два простых числа p и q
        q = prime_num()
        n = p * q
        euler = (p - 1) * (q - 1)  # Определяется φ(n)=(p − 1)( q − 1)
        e = opened_exp(euler)  # Выбор числа e, взаимно простого с φ(n), причем e < φ(n)
        d = closed_exp(e, euler)  # Выбор числа d, отвечающего тождеству e*d = 1 (mod
φ(n))
        self.set_key_text.emit(e, n)
        encrypted = [pow(x, e, n) for x in self.coded_message]
        self.set_enc_text.emit(encrypted)
        self.finish.emit()

    @QtCore.pyqtSlot(int, int)
    def set_key(self):
        application.ui.label_key.setText(f'Public key: {e, n}')

    @QtCore.pyqtSlot(list)
    def set_enc(self):
        application.ui.label_encrypt.setText(f'{encrypted}')

    @QtCore.pyqtSlot()
    def return_but():
        application.ui.button_encrypt.setEnabled(True)
        application.gen.exit()



class MyWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super(MyWindow, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.button_encrypt.clicked.connect(self.encrypt)
        self.ui.button_decrypt.clicked.connect(self.decrypt)

    def encrypt(self):  # Шифрование
        message, coded_message = list(self.ui.text_input.toPlainText()), []  #
Считывание сообщения
        if not len(message):
            return
```

```
        for letter in message:
            coded_message.append(ord(letter))
        self.ui.label_key.setText("Generating new keys for you")
        self.ui.button_encrypt.setEnabled(False)
        self.gen = GenKey(coded_message)
        self.gen.set_key_text.connect(GenKey.set_key)
        self.gen.set_enc_text.connect(GenKey.set_enc)
        self.gen.finish.connect(GenKey.return_but)
        self.gen.start()

    def decrypt(self):  # Расшифровка
        decoded_message = []
        decrypted = [pow(x, d, n) for x in encrypted]  # Расшифрование
        for letter in decrypted:
            decoded_message.append(chr(letter))
        result = "".join(decoded_message)
        self.ui.label_decrypt.setText(result)


app = QtWidgets.QApplication([])
application = MyWindow()
application.show()
sys.exit(app.exec())
```

Файл **RSA_UI.py**

```
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(595, 849)
        MainWindow.setMinimumSize(QtCore.QSize(595, 849))
        MainWindow.setMaximumSize(QtCore.QSize(595, 849))
        font = QtGui.QFont()
```

```
font.setPointSize(10)
MainWindow.setFont(font)
MainWindow.setStyleSheet("QMainWindow {\n"
                         "    background-color: white;\n"
                         "}\n"
                         "\n"
                         "#text_input {\n"
                         "    border-top: 3px solid #1E90FF;\n"
                         "    border-bottom: 3px solid #1E90FF;\n"
                         "    background-color: white;\n"
                         "}\n"
                         "\n"
                         "QPushButton {\n"
                         "    color: white;\n"
                         "    background-color: #1E90FF;\n"
                         "    border: 0\n"
                         "}\n"
                         "\n"
                         "QPushButton:disabled {\n"
                         "    color: white;\n"
                         "    background-color: #00BFFF;\n"
                         "}\n"
                         "QPushButton:hover {\n"
                         "    color: white;\n"
                         "    background-color: #00BFFF;\n"
                         "}\n"
                         "\n"
                         "#label_encrypt {\n"
                         "    border-top: 3px solid #1E90FF;\n"
                         "}")
self.centralwidget = QtWidgets.QWidget(MainWindow)
self.centralwidget.setObjectName("centralwidget")
self.verticalLayout_2 = QtWidgets.QVBoxLayout(self.centralwidget)
```

```
self.verticalLayout_2.setObjectName("verticalLayout_2")
self.verticalLayout = QtWidgets.QVBoxLayout()
self.verticalLayout.setObjectName("verticalLayout")
self.text_input = QtWidgets.QTextEdit(self.centralwidget)
self.text_input.setEnabled(True)
self.text_input.setMaximumSize(QtCore.QSize(1000, 301))
font = QtGui.QFont()
font.setFamily("Dubai Medium")
font.setPointSize(12)
self.text_input.setFont(font)
self.text_input.setObjectName("text_input")
self.verticalLayout.addWidget(self.text_input)
self.button_encrypt = QtWidgets.QPushButton(self.centralwidget)
self.button_encrypt.setMaximumSize(QtCore.QSize(5000, 50))
font = QtGui.QFont()
font.setPointSize(12)
self.button_encrypt.setFont(font)
self.button_encrypt.setObjectName("button_encrypt")
self.verticalLayout.addWidget(self.button_encrypt)
self.label_key = QtWidgets.QLabel(self.centralwidget)
self.label_key.setMaximumSize(QtCore.QSize(16777215, 40))
font = QtGui.QFont()
font.setPointSize(12)
self.label_key.setFont(font)
self.label_key.setText("")
self.label_key.setAlignment(QtCore.Qt.AlignCenter)
self.label_key.setObjectName("label_key")
self.verticalLayout.addWidget(self.label_key)
self.label_encrypt = QtWidgets.QLabel(self.centralwidget)
font = QtGui.QFont()
font.setPointSize(12)
self.label_encrypt.setFont(font)
self.label_encrypt.setText("")
```

```
        self.label_encrypt.setTextFormat(QtCore.Qt.RichText)
        self.label_encrypt.setAlignment(QtCore.Qt.AlignCenter)
        self.label_encrypt.setObjectName("label_encrypt")
        self.label_encrypt.setWordWrap(True)
        self.verticalLayout.addWidget(self.label_encrypt)
        self.button_decrypt = QtWidgets.QPushButton(self.centralwidget)
        self.button_decrypt.setMaximumSize(QtCore.QSize(16777215, 50))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.button_decrypt.setFont(font)
        self.button_decrypt.setObjectName("button_decrypt")
        self.verticalLayout.addWidget(self.button_decrypt)
        self.label_decrypt = QtWidgets.QLabel(self.centralwidget)
        font = QtGui.QFont()
        font.setPointSize(12)
        self.label_decrypt.setFont(font)
        self.label_decrypt.setText("")
        self.label_decrypt.setWordWrap(True)
        self.label_decrypt.setAlignment(QtCore.Qt.AlignCenter)
        self.label_decrypt.setObjectName("label_decrypt")
        self.verticalLayout.addWidget(self.label_decrypt)
        self.verticalLayout_2.addLayout(self.verticalLayout)
        MainWindow.setCentralWidget(self.centralwidget)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "RSA_Vasilev"))
        self.button_encrypt.setText(_translate("MainWindow", "Encrypt"))
        self.button_decrypt.setText(_translate("MainWindow", "Decrypt"))
```