Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский Университет ИТМО»

Факультет безопасности информационных технологий

Дисциплина:

«Разработка систем аутентификации и криптографии»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1 Алгоритмы криптографии и подпись приложений**

**RSA**

**Выполнил:**

Магистрант гр. N42514c

В.В. Герасимов



**Проверил:**

И.Р. Фёдоров

Санкт-Петербург

2020 г.

**Содержание**

[1. Цель работы 3](#_Toc53942728)

[2. Описание выбранных средств реализации и обоснования выбора 3](#_Toc53942729)

[3. Описание алгоритма 4](#_Toc53942730)

[3.1. Алгоритм создания открытого и закрытого ключа 4](#_Toc53942731)

[3.2. Алгоритм шифрования 5](#_Toc53942732)

[3.3. Алгоритм дешифрования 5](#_Toc53942733)

[4. Исходный код и демонстрация работы программы 6](#_Toc53942734)

[4.1. Демонстрация работа программы 13](#_Toc53942735)

[5. Подпись exe файла с помощью PKI Client в Windows PowerShell 13](#_Toc53942736)

[6. Выводы 14](#_Toc53942737)

1. Цель работы

Лабораторная работа №1:

**Часть 1*:*** реализация алгоритма шифрования **RSA**.

**Требования к реализации:**

* необходимо реализовать сам алгоритм (процедуры генерации ключей, шифрования и дешифрования) без использования криптографических библиотек**;**
* программа должна запускаться в среде Windows, исполняемый файл программы должен иметь расширение .EXE

**Часть 2:** подпись полученного в первой части файла .EXE

**Требования к выполнению:**

* необходимо подписать полученный файл .EXE с помощью команд Windows PowerShell (лучше использовать PKI Client)
* при открытии “Свойств” файла .EXE в разделе “Цифровые подписи” мы должны будем увидеть свою подпись.

1. Описание выбранных средств реализации и обоснования выбора

Для реализации алгоритма был выбран язык Python, так как к скорости выполнения программы не предъявлялось каких-либо особых требований.

Python – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций. Этот язык имеет довольно простой синтаксис, лёгок в изучении и имеет низкий порог вхождения. В то же время язык является очень мощным средством разработки. Такой выбор также обусловлен тем, что это достаточно распространённый язык в настоящее время.

Для реализации графического интерфейса была использована библиотека Python – tkinter.

1. Описание алгоритма

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Алгоритм RSA является алгоритмом ассиметричного шифрования.

В криптографической системе с открытым ключом каждый участник располагает как открытым ключом (public key), так и закрытым ключом (англ. private key).

В криптографической системе RSA каждый ключ состоит из пары целых чисел. Каждый участник создаёт свой открытый и закрытый ключ самостоятельно. Закрытый ключ каждый из них держит в секрете, а открытые ключи можно сообщать кому угодно или даже публиковать их. Открытый и закрытый ключи каждого участника обмена сообщениями в криптосистеме RSA образуют «согласованную пару» в том смысле, что они являются взаимно обратными.

* 1. Алгоритм создания открытого и закрытого ключа

1. Выбираются 2 различных случайных простых числа p и q.

2. Вычисляется их произведение n = p \* q, которое называется модулем.

3. Вычисляется значение функции Эйлера от числа n:

4. Выбирается целое число e (для ключа шифрования) взаимно простое со значением функции φ(n) (то есть наибольший общий делитель e и φ(n) должен быть равен 1)

5. Вычисляется число d (для ключа дешифрования), которое является мультипликативной обратной величиной e:

6. (e, n) - открытый ключ RSA; (d, n) - закрытый ключ RSA.

Предположим, что Боб хочет послать открытый текст m Алисе. Чтобы зашифровать открытый текст, Боб должен знать открытый ключ Алисы (e, n) , а Алиса должна использовать свой закрытый ключ (d, n) для расшифровки информации. Алиса должна передать свой открытый ключ (e, n) Бобу. Зашифрованное сообщение Боба может быть расшифровано только с помощью закрытого ключа Алисы (d, n).

* 1. Алгоритм шифрования

1. Взять открытый ключ Алисы (e, n).

2. Взять открытый текст m.

3. Преобразовать каждый символ сообщения в его эквивалент ASCII и получить целое число.

3. Зашифровать получившееся число с использованием открытого ключа Алисы:

* 1. Алгоритм дешифрования

1. Принять зашифрованное сообщение crypt\_message

2. Взять свой закрытый ключ (d, n)

3. Применить закрытый ключ для расшифрования числа:

4. Преобразовать число в текст с помощью ASCII.

1. Исходный код и демонстрация работы программы

[Ссылка на исходный код](https://github.com/gevol69/dev_auth_crypt_system/tree/main/dev_auth_crypt_system_lab_1)

Файл – **rsa.py**:

|  |  |
| --- | --- |
|  | import random |
|  |  |
|  |  |
|  | LIMIT\_PRIME = 1000000000000 |
|  |  |
|  | """ |
|  | Функция для генерации случайного простого числа |
|  | Самого большого простого числа не существует, поэтому ограничимся LIMIT\_PRIME = триллиону для более быстрых подсчётов |
|  | Можно было взять, конечно, самое наибольшее известное простое 2 \*\* 82589933 - 1, но оно содержит 24.862.048 десятичных цифр)0))) |
|  | return: |
|  | num - случайное простое число |
|  | """ |
|  | def generate\_prime\_number(limit = LIMIT\_PRIME): |
|  | num = 0 |
|  | while True: |
|  | num = random.randint(0, limit) |
|  | if isPrime(num): |
|  | break |
|  | return num |
|  |  |
|  | """ |
|  | Функция для проверки того является ли число простым |
|  | Простое число - это такое число, которое делится только на себя и на 1 |
|  | Перебираем число n начиная с 2, пока не найдем делитель числа number |
|  | У любого составного числа есть делитель не равный 1, не превосходящий квадратного корня из числа |
|  | Чиcло number является простым, если алгоритм закончился по причине того, |
|  | что проверяемый делитель стал больше, чем корень из number. |
|  | return: |
|  | True: для простых числе |
|  | False: для составных |
|  | """ |
|  | def isPrime(number): |
|  | n = 2 |
|  | while n \* n <= number and number % n != 0: |
|  | n += 1 |
|  | return n \* n > number |
|  |  |
|  |  |
|  | """ |
|  | Функция для генерации числа n, открытого и закрытого ключа |
|  | """ |
|  | def generate(p, q, key\_size=128): |
|  | #key\_size - размер ключа (по умолчанию - 128 бит) |
|  | n = p \* q |
|  | euler\_func = (p - 1) \* (q - 1) #функция Эйлера |
|  | e = generate\_public\_key(euler\_func, key\_size) |
|  | d = generate\_private\_key(e, euler\_func) |
|  | return e, d, n |
|  |  |
|  |  |
|  | """ |
|  | Функция для генерации открытого ключа |
|  | """ |
|  | def generate\_public\_key(euler\_func, key\_size): |
|  | #генерируем случайное число длиной key\_size бит (например, для 128: начиная от 2\*\*127 (170141183460469231731687303715884105728) до 2\*128 - 1 (340282366920938463463374607431768211455) ) |
|  | e = random.randint(2 \*\* (key\_size - 1), 2\*\*key\_size - 1) |
|  | #ищем наибольший общий делитель числа e и функции Эйлера |
|  | g = gcd(e, euler\_func) |
|  | #наша цель - достичь того, чтобы НОД был равен 1 |
|  | while g != 1: |
|  | e = random.randint(2 \*\* (key\_size - 1), 2\*\*key\_size - 1) |
|  | g = gcd(e, euler\_func) |
|  | return e |
|  |  |
|  | """ |
|  | Рекурсивная функция для нахождения наибольшего общего делителя |
|  | """ |
|  | def gcd(e, euler\_func): |
|  | if e == 0: |
|  | return euler\_func |
|  | return gcd(euler\_func % e, e) |
|  |  |
|  | """ |
|  | Функция для генерации закрытого ключа |
|  | gcdex - расширенный алгоритм евклидова расстояния |
|  | """ |
|  | def generate\_private\_key(e, euler\_func): |
|  | #euler\_func - обратный к элементу e, то есть e^-1 |
|  | d = gcdex(e, euler\_func)[1] |
|  | d = d % euler\_func |
|  | if d < 0 : |
|  | d += euler\_func |
|  | return d |
|  |  |
|  | """ |
|  | Рекурсивная реализация расширенного алгоритма евклидова расстояния |
|  | Цель - подобрать такие числа x и y, чтобы ax + by = d |
|  | return: |
|  | d - наибольший общий делитель a и b |
|  | x и y - такие числа, что ax + by = d |
|  | """ |
|  | def gcdex(a, b): |
|  | if a == 0: |
|  | return (b, 0, 1) |
|  | else: |
|  | g, y, x = gcdex(b % a, a) |
|  | return (g, x - (b // a) \* y, y) |
|  |  |
|  |  |
|  | """ |
|  | Функция для шифрования текста |
|  | plain\_text: открытый текст |
|  | (e, n): открытый ключ |
|  | """ |
|  | def encrypt(plain\_text, e, n): |
|  | cipher\_text = [] #список для хранения шифротекста |
|  | for char in plain\_text: |
|  | #возведение каждого символа, переведённого в его эквивалент ASCII, в степень по модулю n |
|  | cipher\_text.append(pow(ord(char), e, n)) |
|  | return cipher\_text |
|  |  |
|  | """ |
|  | Функция для дешифрования текста |
|  | cipher\_text: зашифрованный текст |
|  | (d, n): закрытый ключ |
|  | return: открытый текст |
|  | """ |
|  | def decrypt(cipher\_text, d, n): |
|  | plain\_text = [] |
|  | try: |
|  | for char in cipher\_text: |
|  | #обратная операция: возведение каждого числа в степень по модулю n, и его перевод в эквивалент ASCII |
|  | plain\_text.append(chr(pow(char, d, n))) |
|  | return "".join(plain\_text) |
|  | except TypeError as e: |
|  | print(e) |
|  |  |
|  | """ |
|  | Функция для экспорта ключей в файлы |
|  | public.key - открытый ключ |
|  | private.key - закрытый ключ |
|  | """ |
|  | def export(e, d, n): |
|  | with open("public.key","w") as fp: |
|  | fp.write(f"RSA PUBLIC KEY:\n{e}\n{n}\nEND") |
|  | with open("private.key","w") as fp: |
|  | fp.write(f"RSA PRIVATE KEY:\n{d}\n{n}\nEND") |

Файл – **main.py**:

|  |
| --- |
|  |
|  | from rsa import \* |
|  | import os |
|  | import tkinter as tk |
|  | import tkinter.ttk as ttk |
|  | from tkinter.filedialog import askdirectory, askopenfilename |
|  | from tkinter import messagebox as mb |
|  | from tkinter import Tk, RIGHT, BOTH, RAISED |
|  | from tkinter.ttk import Frame, Button, Style |
|  | from tkinter import \* |
|  |  |
|  | """ |
|  | Создание формы tkinter |
|  | """ |
|  | class Main(tk.Frame): |
|  | def \_\_init\_\_(self, root): |
|  | tk.Frame.\_\_init\_\_(self, root) |
|  | self.root = root |
|  | self.init\_main() |
|  |  |
|  | def init\_main(self): |
|  | self.centerWindow() |
|  |  |
|  |  |
|  | def centerWindow(self): |
|  | w = 475 |
|  | h = 350 |
|  | sw = self.root.winfo\_screenwidth() |
|  | sh = self.root.winfo\_screenheight() |
|  | x = (sw - w)/2 |
|  | y = (sh - h)/2 |
|  | self.root.geometry('%dx%d+%d+%d' % (w, h, x, y)) |
|  |  |
|  | """ |
|  | Выбор директории для сохранения ключей |
|  | """ |
|  | def opening\_folder(): |
|  | try: |
|  | mb.showinfo("Информация", "Создайте директорию и выберете её, в ней будут сохранены сгенерированные ключи!") |
|  | dir\_path = askdirectory() |
|  | os.chdir(dir\_path) |
|  | p = generate\_prime\_number() |
|  | q = generate\_prime\_number() |
|  | e, d, n = generate(p, q, key\_size=int(key\_size\_text.get())) |
|  | export(e, d, n) |
|  | mb.showinfo("Информация", "Открытый и закрытый ключи созданы и успешно сохранены в файлы public.key и private.key в выбранную директорию!") |
|  | show\_public\_key.configure(state='active') |
|  | show\_private\_key .configure(state='active') |
|  | text.configure(state='normal') |
|  | encrypt\_text.configure(state='active') |
|  | clear.configure(state='active') |
|  | except OSError: |
|  | mb.showerror("Ошибка", "Директория не выбрана!") |
|  | except ValueError: |
|  | mb.showerror("Ошибка", "В поле 'Длина ключа' должно быть целое число") |
|  |  |
|  |  |
|  | def information\_key\_size(): |
|  | mb.showinfo("Информация", "По умолчанию - 128 бит. \n 31 декабря 2013 года браузеры Mozilla перестали поддерживать сертификаты удостоверяющих центров с ключами RSA меньше 2048 бит))0)0)") |
|  |  |
|  | def show\_info\_public\_key(): |
|  | with open("public.key","r") as fp: |
|  | mb.showinfo("Открытый ключ", fp.read()) |
|  |  |
|  | def show\_info\_private\_key(): |
|  | with open("private.key","r") as fp: |
|  | mb.showinfo("Закрытый ключ", fp.read()) |
|  |  |
|  | def encryption\_text(): |
|  | try: |
|  | mb.showinfo("Информация", "Выберите открытый ключ того, кому хотите передать сообщение (в случае если от Боба Алисе, то выберите открытый ключ Алисы)!") |
|  | public\_key = askopenfilename() |
|  | with open(public\_key,'r') as fp: |
|  | \_,e,n = fp.readline(),int(fp.readline()),int(fp.readline()) |
|  | message = text.get("1.0", END) |
|  | cipher\_text = encrypt(message, e, n) |
|  | text.delete(1.0, END) |
|  | text.insert("1.0", cipher\_text) |
|  | text.configure(state='disabled') |
|  | encrypt\_text.configure(state='disabled') |
|  | decrypt\_text.configure(state='active') |
|  | mb.showinfo("Успех", "Сообщение успешно зашифровано и отображается в поле!") |
|  | except OSError: |
|  | mb.showerror("Ошибка", "Директория не выбрана!") |
|  |  |
|  | def decryption\_text(): |
|  | try: |
|  | mb.showinfo("Информация", "Выберите закрытый ключ того, кому передавалось сообщение (в случае если от Боба Алисе, то выберите закрытый ключ Алисы)!") |
|  | private\_key = askopenfilename() |
|  | with open(private\_key,'r') as fp: |
|  | \_,d,n = fp.readline(),int(fp.readline()),int(fp.readline()) |
|  | message\_encrypt = [int(i) for i in text.get("1.0", END).strip().split(' ')] |
|  | plain\_text = decrypt(message\_encrypt, d, n) |
|  | text.configure(state='normal') |
|  | text.delete(1.0, END) |
|  | text.insert("1.0", plain\_text) |
|  | encrypt\_text.configure(state='active') |
|  | decrypt\_text.configure(state='disabled') |
|  | mb.showinfo("Успех", "Сообщение успешно расшифровано и отображается в поле!") |
|  | except OSError: |
|  | mb.showerror("Ошибка", "Директория не выбрана!") |
|  |  |
|  | def clear\_text(): |
|  | text.delete(1.0, END) |
|  | text.configure(state='disabled') |
|  | encrypt\_text.configure(state='disabled') |
|  | decrypt\_text.configure(state='disabled') |
|  | clear.configure(state='disabled') |
|  | show\_public\_key.configure(state='disabled') |
|  | show\_private\_key .configure(state='disabled') |
|  | mb.showinfo("Информация", "Поля очищены, данные сброшены, сгенерируйте ключи повторно!") |
|  |  |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | root = tk.Tk() |
|  | app = Main(root) |
|  | root.title("Алгоритм RSA") |
|  | root.resizable(False, False) |
|  |  |
|  | #опции для позиционирования элементов |
|  | opts = { 'ipadx': 10, 'ipady': 2, 'sticky': 'nswe' } |
|  |  |
|  | frame1 = Frame(relief=RAISED, borderwidth=1) |
|  |  |
|  | label\_key\_size = tk.Label(root, text='Длина ключа, бит', font='Arial 10 bold') |
|  | label\_key\_size.grid(row = 1, column = 1, \*\*opts) |
|  |  |
|  | key\_size\_text = tk.Entry(root, font='Arial 10') |
|  | key\_size\_text.insert(0, '128') |
|  | key\_size\_text.grid(row = 1, column = 2, \*\*opts) |
|  |  |
|  | info\_key\_size = tk.Button(root) |
|  | info\_key\_size.configure(text='Инфо', font='Arial 10', command=information\_key\_size) |
|  | info\_key\_size.grid(row = 1, column = 3, \*\*opts) |
|  |  |
|  | frame1.grid(row = 2, column = 1, columnspan = 3, \*\*opts) |
|  |  |
|  | label\_keys = tk.Label(root, text='Генерация ключей', font='Arial 10 bold') |
|  | label\_keys.grid(row = 3, column = 1, \*\*opts) |
|  |  |
|  | generate\_keys = tk.Button(root) |
|  | generate\_keys.configure(text='Сгенерировать ключи в директорию', font='Arial 10', command=opening\_folder) |
|  | generate\_keys.grid(row = 3, column = 2, columnspan = 3, \*\*opts) |
|  |  |
|  | frame2 = Frame(relief=RAISED, borderwidth=1) |
|  | frame2.grid(row = 4, column = 1, columnspan = 3, \*\*opts) |
|  |  |
|  | label\_show\_keys = tk.Label(root, text='Просмотр ключей', font='Arial 10 bold') |
|  | label\_show\_keys.grid(row = 5, column = 1, \*\*opts) |
|  |  |
|  | show\_public\_key = tk.Button(root) |
|  | show\_public\_key.configure(text='Показать открытый ключ', font='Arial 10', command=show\_info\_public\_key) |
|  | show\_public\_key.configure(state='disabled') |
|  | show\_public\_key.grid(row = 5, column = 2) |
|  |  |
|  | show\_private\_key = tk.Button(root) |
|  | show\_private\_key.configure(text='Показать закрытый ключ', font='Arial 10', command=show\_info\_private\_key) |
|  | show\_private\_key .configure(state='disabled') |
|  | show\_private\_key.grid(row = 5, column = 3) |
|  |  |
|  | frame3 = Frame(relief=RAISED, borderwidth=1) |
|  | frame3.grid(row = 6, column = 1, columnspan = 3, \*\*opts) |
|  |  |
|  | label\_keys = tk.Label(root, text='Сообщение', font='Arial 10 bold') |
|  | label\_keys.grid(row = 7, column = 1, columnspan = 3, \*\*opts) |
|  |  |
|  | text = tk.Text(root) |
|  | text.configure(width=25, height=11) |
|  | text.configure(state='disabled') |
|  | text.grid(row = 8, column = 1, columnspan = 3, \*\*opts) |
|  |  |
|  | encrypt\_text = tk.Button(root) |
|  | encrypt\_text.configure(text='Зашифровать', font='Arial 10', command=encryption\_text) |
|  | encrypt\_text.configure(state='disabled') |
|  | encrypt\_text.grid(row = 9, column = 1, \*\*opts) |
|  |  |
|  | decrypt\_text = tk.Button(root) |
|  | decrypt\_text.configure(text='Расшифровать', font='Arial 10', command=decryption\_text) |
|  | decrypt\_text.configure(state='disabled') |
|  | decrypt\_text.grid(row = 9, column = 2, \*\*opts) |
|  |  |
|  | clear = tk.Button(root) |
|  | clear.configure(text='Рестарт', font='Arial 10', command=clear\_text) |
|  | clear.configure(state='disabled') |
|  | clear.grid(row = 9, column = 3, \*\*opts) |
|  |  |
|  | root.mainloop() |

* 1. Демонстрация работа программы

Работа программы продемонстрирована в [этой](https://camo.githubusercontent.com/d00c9c760cccdb6781660a19287ebbaa20bdcc6a/687474703a2f2f697069632e73752f696d672f696d67372f66732f323032302d31302d31382d31382d30352d34322e313630333033333834382e676966) демке.

1. Подпись exe файла с помощью PKI Client в Windows PowerShell

Подписать exe файл самозаверяющимся сертификатом можно осуществить следующим образом:

1. Создать самозаверяющийся сертификат командой:

New-SelfSignedCertificate -Type Custom -Subject "CN=Vlad Gerasimov, O=Gevol Corporation, C=RU" -KeyUsage DigitalSignature -FriendlyName "Vlad Gerasimov" -CertStoreLocation "Cert:\CurrentUser\My" -TextExtension @("2.5.29.37={text}1.3.6.1.5.5.7.3.3", "2.5.29.19={text}")

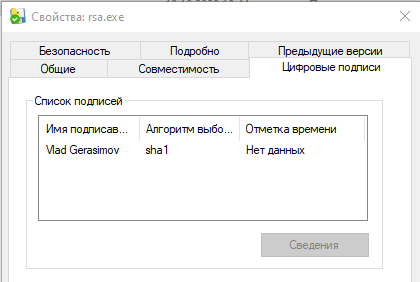
2. Задать переменной cert только что созданный сертификат:

$cert=Get-ChildItem -Path cert:\CurrentUser\my -CodeSigningCert

3. Подписать exe файл этим сертификатом командой:

Set-AuthenticodeSignature rsa.exe $cert -HashAlgorithm RSA

4. Файл подписан.



1. Выводы

В ходе лабораторной работы были изучены основные принципы работы ассиметричного алгоритма RSA, а также подпись exe файлов с помощью powershell. В частности, была написана программа на языке Python, реализующая алгоритм RSA с графическим интерфейсом, скрипт был преобразован в .exe файл и подписан с помощью PKI Client.