Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

АСИММЕТРИЧНЫЕ АЛГОРИТМЫ ШИФРОВАНИЯ

Лабораторная работа №3

по дисциплине

«Информационная безопасность»

Студент гр. 430-2

_____А.А. Лузинсан

«_____»_____2023 г.

Руководитель

Ассистент каф. АСУ

_____ Я.В. Яблонский

«____»_____2023 г.

Томск 2023

Оглавление

1 Цель работы

Цель: познакомиться и научиться работать с ассиметричными алгоритмами шифрования.

2 Задание

Задание по варианту №16: пользуясь алгоритмом RSA с параметрами p=857, q=673, e=5, напишите программу, которая позволит зашифровать произвольный открытый текст, предварительно закодировав его согласно прилагаемым таблицам 1, 2, 3 и расшифровать его. Зашифрованный текст должен сохраняться в файле для пересылки своему другу. При написании программы используйте алгоритм быстрого возведения в степень и алгоритмы Евклида.

Таблица 2.1 — Кодировка русского алфавита

А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41

Таблица 2.2 — Кодировка латинского алфавита

A	B	C	D	E	F	G	H	I	J	K	L	M
42	43	44	45	46	47	48	49	50	51	52	53	54
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
55	56	57	58	59	60	61	62	63	64	65	66	67

Таблица 2.3 — Дополнительные символы

Пробел	Запятая	Точка
68	69	70

3 Описание алгоритма шифрования

RSA (Rivest, Shamir, Adleman) — это ассиметричный криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших полупростых чисел. Криптосистема RSA стала первой системой, пригодной как для шифрования, так и для создания цифровой подписи.

В криптографической системе с открытым ключом каждый участник располагает как открытым ключом, так и закрытым ключом. 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.

3.1 Алгоритм создания согласованной пары

RSA-ключи генерируются следующим образом:

- Выбираются два различных случайных простых числа p и q заданного размера. Согласно варианту, эти значения уже даны: p=857, q=673.
- Вычисляется их произведение n=p*q, которое называется модулем.
- Вычисляется значение функции Эйлера от числа n по формуле 3.1:

$$\phi = (p-1) * (q-1) \tag{3.1}$$

- Выбирается целое число e, взаимно простое со значением функции Эйлера, которое называется открытой экспонентой. Обычно в качестве e берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, так как в этом случае время, необходимое для шифрования

с использованием быстрого возведения в степень, будет меньше. Слишком малые значения e потенциально могут ослабить безопасность схемы RSA. Однако, по варианту данное значение уже задано: e=3.

- Вычисляется число d, мультипликативно обратное к числу e по модулю функции Эйлера. Число d называется секретной экспонентой. Для реализации этой задачи внутри метода использовался расширенный алгоритм Евклида, суть которого заключается в том, что, помимо вычисления непосредственно НОД, находятся коэффициенты Безу. В результате мультипликативное обратное с переданными аргументами e и phi возвращает значение (x mod phi + phi) % phi в том случае, если НОД = 1. Определение мультипликативного обратного в модульной арифметрике представлено в формуле 4.2.

$$\mathrm{d} * \mathrm{e} \equiv 1 \mod (\phi(n)) \tag{4.2}$$

- Пара (e, n) публикуется в качестве открытого ключа RSA.
- Пара (d, n) играет роль закрытого ключа RSA и держится в секрете.

3.2 Шифрование и расшифрование

Алгоритм шифрования заключается в следующем:

- Берётся открытый ключ получателя (e, n).
- Берется открытый текст m.
- Сообщение *m* шифруется с использованием открытого ключа получателя по формуле 4.3. При этом используется алгоритм быстрого возведения в степень по модулю в варианте «справа-налево».

$$\mathrm{c} = \mathrm{E(m)} = \mathrm{m}^e \mod n \tag{4.3}$$

Алгоритм расшифрования заключается в следующем:

- Принимается зашифрованное сообщение *c*.
- Получатель берёт закрытый ключ (d, n).
- К зашифрованному сообщению *c* применяется закрытый ключ для

расшифрования сообщения по формуле 4.4. При этом также применяется алгоритм быстрого возведения в степень по модулю в варианте «справа-налево».

$$\mathrm{c} = \mathrm{D(c)} = \mathrm{c}^d \mod n \tag{4.4}$$

4 Листинг программы

Содержимое файла представлено листинге 4.1.

Листинг 4.1 — Содержимое файла по лабораторной работе

```
from typing import Union
from initialize import *

def get_input_data():
    if dpg.get_value('input_method') == 'File':
        file_path = dpg.get_value('file')
        file = open(file_path, 'r')
        input_data = file.read()
        file.close()
    else:
        input_data = dpg.get_value('Manually')
    return [ord(c) for c in input_data], input_data

class RSA:
    __public_key__: tuple
    __private_key__: tuple

    def __init__(self, _p: int, _q: int, _e: int):
        self.__make_keys__(_p, _q, _e)

    @staticmethod
    def bezout_recursive(a, b) -> tuple[int, int, int]:
        """
        A recursive implementation of extended Euclidean algorithm.
        Returns integer x, y and gcd(a, b) for Bezout equation:
            ax + by = gcd(a, b).
        """
        if not b:
            return 1, 0, a
        y, x, g = RSA.bezout_recursive(b, a % b)
        return x, y - (a // b) * x, g
```

```
    @staticmethod
    def mod_inverse(a, m):
        x, y, gcd = RSA.bezout_recursive(a, m)
        return (x % m + m) % m if gcd == 1 else -1

    @staticmethod
    def fast_pow_module(base, degree, module):
        degree = list(map(int, bin(degree)[:1:-1]))
        r = 1
        for i in degree:
            if i:  r = (r * base) % module
            base = pow(base, 2, module)
        return r

    def __make_keys__(self, _p: int, _q: int, _e: int):
        p, q, e = _p, _q, _e
        phi = (p - 1) * (q - 1)
        d = RSA.mod_inverse(e, phi)
        N = p * q
        print(f"\t\tPublic key: [{e}], [{N}]")
        self.__public_key__, self.__private_key__ = (e, N), (d, N)

    def encrypt(self, input_bytes: list) -> tuple[list, str]:
        e, N = self.__public_key__
        __cipher_bytes__ = [RSA.fast_pow_module(c, e, N) for c in input_bytes]
        return __cipher_bytes__, ''.join(map(lambda x: str(x), __cipher_bytes__))

    def decrypt(self, cipher_bytes: list) -> tuple[list, str]:
        d, N = self.__private_key__
        decipher_bytes = [RSA.fast_pow_module(c, d, N) for c in cipher_bytes]
        return decipher_bytes, ''.join([chr(c) for c in decipher_bytes])

def preparing(sender, app_data, user_data):
    dpg.show_item('Cipher method')
    input_bytes, input_data = get_input_data()
    dpg.set_value('input data', value=input_data)
    p, q, e = dpg.get_value('key_p'), \
```

```
            dpg.get_value('key_q'), \
            dpg.get_value('key_e')
        rce = RSA(p, q, e)
        fout = open('plaintext.txt', 'w')
        fout.writelines("\n\n\tText:\n" + input_data)
        fout.close()
        cipher_bytes, cipher_text = rce.encrypt(input_bytes)
        dpg.set_value('encrypted', value=cipher_text)
        fout = open('ciphertext.txt', 'w')
        fout.writelines(cipher_text)
        fout.close()
        # region test
        decrypted_bytes, decrypted_data = rce.decrypt(cipher_bytes)
        dpg.set_value('test', value=decrypted_data)
        fout = open('decrypted_text.txt', 'w')
        fout.writelines("\n\tDecrypted:\n" + decrypted_data)
        fout.close()
        if ''.join(input_data) == ''.join(decrypted_data):
            dpg.configure_item('dots', default_value='True', color=(0, 255, 0, 255))
        else:
            dpg.configure_item('dots', default_value='False', color=(255, 0, 0, 255))
        # endregion

    def initialize_lr3():
        with dpg.window(label="Лабораторная работа #3", tag='lr3',
                        show=True, width=500, height=700, pos=(100, 100),
                        on_close=lambda: dpg.delete_item('lr3')):
            initialize()
            dpg.add_input_int(tag='key_p', label='Key: p',
                              default_value=857, show=True, before='Manually')
            dpg.add_input_int(tag='key_q', label='Key: q',
                              default_value=673, show=True, before='Manually')
            dpg.add_input_int(tag='key_e', label='Key: e',
                              default_value=5, show=True, before='Manually')
            dpg.add_button(label="Continue: RCA", callback=preparing,
                           show=False, tag='continue')
```

5 Пример работы программы

Программа поддерживает файловый ввод исходного текста, либо же ввод вручную, а также вывод результата в выходной файл output.txt. Результат работы для файлового ввода представлен на рисунке 5.1.

Пример работы программы на данных, введённых вручную, представлен на рисунке 5.2.

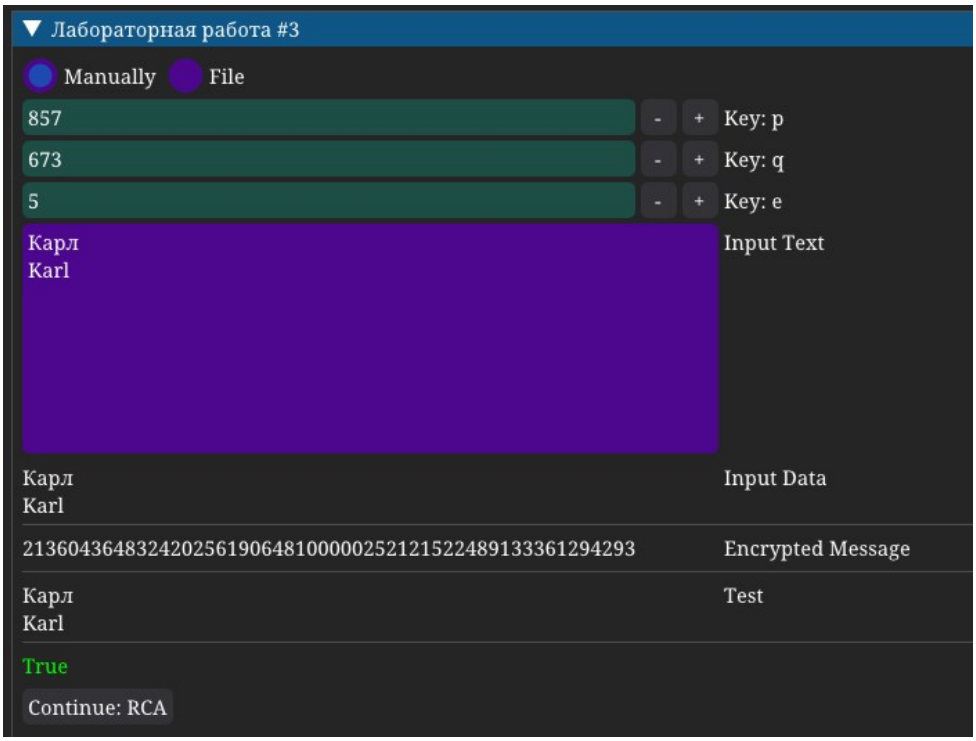

Рисунок 5.1 — Кодирование текста из файла

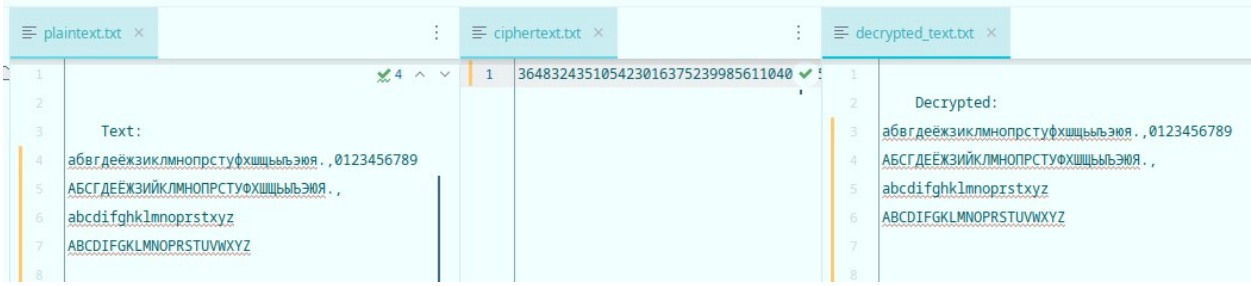

Рисунок 5.2 — Кодирование текста, введённого вручную

6 Вывод о проделанной работе

В результате выполнения лабораторной работы я познакомилась и научилась работать с ассиметричными алгоритмами шифрования.