ОТЧЕТ К ЛАБОРАТОРНОЙ РАБОТЕ № 4

Лабораторная работа № 4

Шифрование с открытым ключом. Алгоритм RSA

Вариант №5

Ф. И. О. студента: Гюнтер Тимофей Вячеславович

Группа: ФИТ-221

Проверил: Дата:

Основные сведения

Формула шифрования: $C = M^e \ mod \ n$

Формула расшифрования: $M = C^d \ mod \ n$

Формула, связывающая открытый ключ и закрытый ключ: $e^{-1} = d \ mod \ \varphi(n)$,

где $\varphi(n) -$ функция Эйлера

Результаты

Параметр n = 107*241 = 25787

1) Открытый ключ шифрования e = 10901

Закрытый ключ шифрования d = 11741

2) Открытый ключ шифрования e = 5213

Закрытый ключ шифрования d = 9077

3) Открытый ключ шифрования e = 17753

Закрытый ключ шифрования d = 7817

Открытый текст: А РОЗА УПАЛА НА ЛАПУ АЗОРА

Блоки открытого текста:

$$M_1 = 10992, M_2 = 6241, M_3 = 7109, M_4 = 9292, M_5 = 5102, M_6 = 11099,$$

$$M_7 = 23109, M_8 = 921, M_9 = 10252, M_{10} = 23109, M_{11} = 921, M_{12} = 10252,$$

$$M_{13} = 999, M_{14} = 10172, M_{15} = 426, M_{16} = 10$$

Шифр-текст: 15482 3831 19167 6348 11045 21260 16174 1306 18913 5715 17670 21222 1527

Код программы

```
import numpy as np

import random

from lab2 import euclid_extended



mapping_dict = {

    'А': 10, 'Б': 11, 'В': 12, 'Г': 13, 'Д': 14, 'Е': 15, 'Ж': 16, 'З': 17, 'И': 18, 'Й': 19,

    'К': 20, 'Л': 21, 'М': 22, 'Н': 23, 'О': 24, 'П': 25, 'Р': 26, 'С': 27, 'Т': 28, 'У': 29,

    'Ф': 30, 'Х': 31, 'Ц': 32, 'Ч': 33, 'Ш': 34, 'Щ': 35, 'Ъ': 36, 'Ы': 37, 'Ь': 38, 'Э': 39,

    'Ю': 40, 'Я': 41, ' ': 99

}



reversed_mapping_dict = {v:k for k,v in mapping_dict.items()}



def binary_modular_power(a, power, modulus):

    res = 1

    t = a % modulus
```

```
    while power > 0:
        if power % 2 == 1:
            res = (res*t) % modulus
        power = power >> 1
        t = (t*t) % modulus
    return res


def generate_rsa_keys(p, q):
    phi = (p-1)*(q-1)
    n = p*q
    e = random.randrange(2, phi)


    gcd = 0
    max_iterations = 100
    iteration = 0
    while gcd != 1 and iteration <= max_iterations:
        e = random.randrange(2, phi)
        gcd, _, d  = euclid_extended(phi, e)
        iteration += 1


    if gcd != 1:
        raise ValueError('Максимальное кол-во итераций достигнуто и всё же не найдено обратного к e по mod phi')
```

```
    return (e, n), (d%phi, n)


def encrypt(open_text: str, public_key):
    e, n = public_key
    num_message = ''.join([str(mapping_dict[letter]) for letter in open_text.upper() if letter in mapping_dict])
    print(num_message)
    block_size = n
    blocks = []
    current_block = ""

    for char in num_message:
        current_block += char
        if current_block[0] == "0":
            if not blocks:
                current_block = current_block[1:]
            else:
                last_value = blocks[-1][-1]
                blocks[-1] = blocks[-1][:-1]
                current_block = last_value + current_block
        if int(current_block) > block_size:
            blocks.append(current_block[:-1])
            current_block = current_block[-1:]
```

```
    if current_block:
        if current_block[0] == "0":
            current_block = blocks[-1][-1] + current_block
            blocks[-1] = blocks[-1][:-1]
        blocks.append(current_block)

    print(blocks)
     encrypted_message = [binary_modular_power(int(block), e, n) for block in blocks]

    return encrypted_message

def decrypt(cipher_text_code, private_key):
    d, n = private_key
     numeric_decrypted = ''.join([str(binary_modular_power(digit, d, n)) for digit in cipher_text_code])
    decrypted_message = ''

    for i in range(0, len(numeric_decrypted), 2):
        number = int(numeric_decrypted[i:i+2])
        if number in reversed_mapping_dict:
            decrypted_message += reversed_mapping_dict[number]
```

```
        else:
                print(f'list: {numeric_decrypted}, block_number: {number}, number: {number}')
            raise ValueError("Должны быть русские буквы и пробелы")
    return decrypted_message


# p = 103
# q = 239
from lab5 import is_prime


def p_q_input():
    print("Введите два простых числа p и q:")
    try:
        p, q = map(int, input().split())
        if not (is_prime(p) and is_prime(q)):
            raise ValueError("Оба числа должны быть простыми!")
    except ValueError as e:
        print(f"Ошибка: {e}")
        exit()
    return p, q


# p, q = p_q_input()
```

```
keys = []
def automatic_key_generation(p, q):
    try:
        public_key, private_key = generate_rsa_keys(p, q)
        keys.append((public_key, private_key))
        print(f"Сгенерированная пара ключей: {len(keys)}")
    except ValueError as e:
        print(f"Ошибка: {e}")

def open_keys_input(p, q):
    try:
        e = int(input("Введите значение e: "))
        gcd, d, _  = euclid_extended(e, (p - 1) * (q - 1))

        if gcd != 1:
            raise ValueError("e и φ(n) должны быть взаимно простыми!")

        private_key = (d, p * q)
        keys.append(((e, p * q), private_key))
        print(f"Кол-во сгенерированных пар ключей: {len(keys)}")
    except ValueError as e:
            print(f"Ошибка: {e}")
```

```
def display_keys_table():
    print("\nТаблица ключей:")
    for i, (public_key, private_key) in enumerate(keys, 1):
        print(f"{i}. Публичный ключ: {public_key}, Приватный ключ: {private_key}")

def encryption():
    try:
        message = input("Введите сообщение для шифрования (только буквы русского алфавита): ")
        key_number = int(input("Выберите номер ключа для шифрования: "))
        public_key = keys[key_number - 1][0]
        encrypted_message = encrypt(message, public_key)
        print(f"Зашифрованное сообщение: {encrypted_message}")
    except (IndexError, ValueError):
        print("Неверный выбор ключа или некорректный ввод!")

def decryption():
    try:
        ciphertext = list(map(int, input("Введите зашифрованное сообщение: ").split()))
        key_number = int(input("Выберите номер ключа для дешифрования: "))
        private_key = keys[key_number - 1][1]
```

```
        decrypted_message = decrypt(ciphertext, private_key)
        print(f"Расшифрованное сообщение: {decrypted_message}")
except (IndexError, ValueError) as e:
        print(f'{e}!')
```