ОТЧЕТ К ЛАБОРАТОРНОЙ РАБОТЕ № 5

Лабораторная работа № 5

Криптоанализ шифра RSA

Вариант №5

Ф. И. О. студента: Гюнтер Тимофей Вячеславович

Группа: ФИТ-221

Проверил: Дата:

Основные сведения

Факторизация числа n = p*q

Функция Эйлера: $\varphi(n) = (p-1)(q-1) = 31552$

Формула для нахождения закрытого ключа по известному открытому ключу:

$$d = e^{-1} \, mod \, \varphi(n)$$

Результаты

Параметр n = 31921

Открытый ключ шифрования e = 259

Факторизация числа n = 31921 = 137*233

Закрытый ключ шифрования d = 13035

Шифр-текст: C = 28368

Код программы

```
import numpy as np

from lab2 import euclid_extended

from lab4 import binary_modular_power
```

```
def is_prime(number):
    number = abs(number)
    if number <= 1:
        return False
    if number == 2 or number == 3:
        return True
    for x in range(2, int(np.sqrt(number))+1):
        if number % x == 0:
            return False
    return True


def private_key_search(e, p, q):
    phi = (p-1)*(q-1)
    gcd, _, d = euclid_extended(phi, e)
    assert gcd == 1
    return d % phi


# 36 -> 18 -> 9 -> 3 -> 1
# 2 2 3 3


def generate_prime_products() -> dict[str]:
    primes = []
```

```
    for num in range(2, 1000):
        if is_prime(num):
            primes.append(num)

    prime_products = {} # []
    for i in range(0, len(primes), 3):
        if i + 2 < len(primes):
            prime_products[f'{primes[i]} {primes[i+1]} {primes[i+2]}'] = primes[i] * primes[i + 1] * primes[i + 2]
    return prime_products

# print(generate_prime_products())

# for k in dict: print(dict[k])

from itertools import combinations
from math import prod
# факторизация методом ПРОБНОГО ДЕЛЕНИЯ
def trial_division(n: int):
    factors = []
    prime_products = generate_prime_products()

    for keys, product in prime_products.items():
```

```
    gcd, _, _ = euclid_extended(product, n)
    while (gcd != 1):
        if is_prime(n):
            factors.append(n)
            return factors
        if is_prime(gcd):
            factors.append(gcd)
        else:
            prime_fact = list(map(int, keys.split()))
            quit = False
            for k in range(3, 0, -1):
                for x in list(combinations(prime_fact, k)):
                    if prod(x) == gcd:
                        [factors.append(h) for h in x]
                        quit = True
                        break
                if quit:
                    break
        n //= gcd
        gcd, _, _ = euclid_extended(product, n)

if n > 1:
    factors.append(n)
```

```
    return factors


# print(trial_division(1687788))


def decrypt_message(C, d, n):
    return binary_modular_power(C, d, n)


def trial_division_interface():
    n = int(input(f'Введите число для факторизации(n):\n'))


    factors = trial_division(n)
    print(f'Найденные простые множители числа: {factors}')


# trial_division_interface()
def induvidual_var():
    e, n, C = 59, 29999, 21959


    p, q = trial_division(n)
    d = private_key_search(e, p, q)


    print(f'Расшифрованное сообщение: {decrypt_message(C, d, n)}')


def private_key_search_interface():
```

```
    input_split = input('Введите последовательно через пробел открытый ключ(e), два множителя n (p,q):\n')
    e, p, q = map(int, input_split.split())
    d = private_key_search(e, p, q)
    print(f'Найденный закрытый ключ: {d}')


def decrypt_message_interface():
        input_split = input('Введите последовательно через пробел блок зашифрованного сообщения(C), закрытый ключ(d) и модуль шифрования(n):\n')
    C, d, n = map(int, input_split.split())
    M = decrypt_message(C, d, n)
    print(f'Расшифрованное сообщение: {M}')


# обычный метод факторизации----------
# def factorize_simple(n):
#     factors = []
#     # iter = 0
#     for x in range(2, int(np.sqrt(n)+1)):
#         while n%x==0:
#             factors.append(x)
#             n //= x
#         if is_prime(n):
```

```
#             factors.append(n)
#             n //= n
#         if n == 1:
#             break
#         # 356 / 4 = 89
#         # iter += 1
#     return factors
# --------------------------
```