**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

Факультет **Инфокоммуникационных технологий**

Образовательная программа **Мобильные и сетевые технологии**

Направление подготовки(специальность) **09.03.03 Прикладная информатика**

**ПРАКТИЧЕСКАЯ РАБОТА №2**

**По дисциплине «Программирование»**

**Тема: Алгоритмы шифрования.**

**Выполнил** Проскуряков Р. В. K3139

**Проверил** Терещенко В. В.

**Дата** 26.10.2023

**Санкт-Петербург 2023**

**Цель работы**

Познакомиться базовым конструкциям языка Python. Также с общим подход к выполнению заданий и разработке.

**Ход работы**

В ходе выполнения работы были написаны три алгоритма шифрования:

Шифр Цезаря, Шифр Виженера, RSA шифрование.

1. Шифрование с помощью “алгоритма Цезаря”

def encrypt\_caesar(plaintext: str, shift: int = 3) -> str:

    ciphertext = ""

    for el in plaintext:

        added\_ch = ''

        if ord('a') <= ord(el) <= ord('z'):

            added\_ch = chr(ord('a') + (ord(el) - ord('a') + shift) % 26)

        elif ord('A') <= ord(el) <= ord('Z'):

            added\_ch = chr(ord('A') + (ord(el) - ord('A') + shift) % 26)

        else:

            added\_ch = el

        ciphertext += added\_ch

    return ciphertext

1. Усовершенствуем алгоритм до “Шифра Виженера

def encrypt\_vigenere(plaintext: str, keyword: str) -> str:

    keyword = keyword.lower()

    ciphertext = ""

    for i, el in enumerate(plaintext):

        added\_ch = ''

        if ord('a') <= ord(el) <= ord('z'):

            added\_ch = chr(ord('a') + (ord(el) - ord('a') + ord(keyword[i % len(keyword)]) - ord('a')) % 26)

        elif ord('A') <= ord(el) <= ord('Z'):

            added\_ch = chr(ord('A') + (ord(el) - ord('A') + ord(keyword[i % len(keyword)]) - ord('a')) % 26)

        else:

            added\_ch = el

        ciphertext += added\_ch

    return ciphertext

1. Пишем реализацию функций:
   1. Проверка числа на простоту
   2. Алгоритм Евклида
   3. Расширенный алгоритм Евклида

def is\_prime(n: int) -> bool:

    for i in range(2, int(n\*\*0.5) + 1):

        if(n % i == 0):

            return False

    return True

def gcd(a: int, b: int) -> int:

    while(b != 0):

        a, b = b, a % b

    return a

def multiplicative\_inverse(e: int, phi: int) -> int:

    a = e

    b = phi

    if(a < b):

        a, b = b, a

    a\_div\_b = []

    while(b != 0):

        a\_div\_b.append(a // b)

        a, b = b, a % b

    x = 0

    y = 1

    for i in range(1, len(a\_div\_b)):

        x, y = y, x - y \* a\_div\_b[-i - 1]

    return (y + phi) % phi

И с помощью этих функций реализуем генерацию ключей для RSA шифрования.

def generate\_keypair(p: int, q: int) -> tp.Tuple[tp.Tuple[int, int], tp.Tuple[int, int]]:

    if not (is\_prime(p) and is\_prime(q)):

        raise ValueError("Both numbers must be prime.")

    elif p == q:

        raise ValueError("p and q cannot be equal")

    # n = pq

    n = p \* q

    # phi = (p-1)(q-1)

    phi = (p-1) \* (q-1)

    # Choose an integer e such that e and phi(n) are coprime

    e = random.randrange(1, phi)

    # Use Euclid's Algorithm to verify that e and phi(n) are coprime

    g = gcd(e, phi)

    while g != 1:

        e = random.randrange(1, phi)

        g = gcd(e, phi)

    # Use Extended Euclid's Algorithm to generate the private key

    d = multiplicative\_inverse(e, phi)

    # Return public and private keypair

    # Public key is (e, n) and private key is (d, n)

    return ((e, n), (d, n))

**Вывод**

В результате проделанной работы мы реализовали алгоритмы шифрования, которые можно использовать в реальной жизни.