МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет безопасности информационных технологий

Лабораторная работа

Тема: Реализация и анализ криптосистемы RSA

Студент:

Группа № 3146 *М.А. Лопарева*

Преподаватель:

В.М. Коржук

Санкт-Петербург

2023

Цель: Ознакомление с основами асимметричного шифрования на примере криптосистемы RSA. Разработка программы для шифрования и дешифрования сооьщения с использованием RSA

Задачи:

1. Изучить принципы работы криптосистемы RSA, генерацию ключей, процесс шифрования и дешифрования
2. Посредством языка Python реализовать программу для генерации открытого и закрытого ключей в криптосистеме RSA
3. Реализовать функцию шифрования, которая принимает на вход открытый текст и публичный ключ и возвращается зашифрованное сообщение
4. Реализовать функцию дешифрования, которая принимает на вход зашифрованное сообщение и закрытый ключ и возвращает исходный текст

При шифровании RSA сообщения шифруются с помощью кода, называемого открытый ключ, которыми можно поделиться открыто. Из-за некоторых четких математических свойств алгоритма RSA, если сообщение было зашифровано открытым ключом, оно может быть расшифровано только другим ключом, известным как закрытый ключ. У каждого пользователя RSA есть пара ключей, состоящая из их открытого и закрытого ключей. Как следует из названия, закрытый ключ должен храниться в секрете.

Схемы шифрования с открытым ключом отличаются от шифрование с симметричным ключом, где и в процессе шифрования, и в дешифровании используется один и тот же закрытый ключ. Эти различия делают шифрование с открытым ключом, такое как RSA, полезным для связи в ситуациях, когда не было возможности безопасно распространять ключи заранее.

RSA-шифрование часто используется в комбинация с другими схемами шифрования, или для цифровые подписи который может доказать подлинность и целостность сообщения. Обычно он не используется для шифрования целых сообщений или файлов, потому что он менее эффективен и требует больше ресурсов, чем шифрование с симметричным ключом.

Шифрование RSA может использоваться в ряде различных систем. Это может быть реализовано в OpenSSL, wolfCrypt, cryptlib и ряде других криптографических библиотек.

RSA также часто используется для создания безопасных соединений между VPN-клиентами и VPN-серверами. В таких протоколах, как OpenVPN, рукопожатия TLS могут использовать алгоритм RSA для обмена ключами и установления безопасного канала.

Принцип работы криптосистемы и генерация ключа

Асимметричные криптографические системы основаны на так называемых односторонних функциях с секретом. Под односторонней понимается такая функция $y = f(x)$, которая легко вычисляется при имеющемся x, но аргумент x при заданном значении функции вычислить сложно. Аналогично, односторонней функцией с секретом называется функция $y = f(x, k)$, которая легко вычисляется при заданном x, причём при заданном секрете k аргумент x по заданному у восстановить просто, а при неизвестном k – сложно

Подобным свойством обладает операция возведения числа в степень по модулю:

$$c \equiv f(m) \equiv m^e \, mod \, n$$
$$m \equiv f^{-1}(c) \equiv c^d \, mod \, n$$
$$d \equiv e^{-1} \, mod \, \phi(n)$$

где $\phi(n)$ - функция Эйлера

Давайте посмотрим на первое выражение. Здесь число с получено в результате возведения в степень по модулю числа m. Назовём это действие шифрованием. Тогда становится очевидно, что m выступает в роли открытого текста, а c – шифртекста. Результат с зависит от степени е, в которую мы возводим m, и от модуля n, по которому мы получаем результат шифрования. Эту пару чисел (e, n) мы будем называть публичным ключом.

Смотрим на вторoе действие. Здесь d является параметром, с помощью которого мы получаем исходный текст m из шифртекста c. Этот параметр мы назовём приватным ключом.

Осталось подобрать параметры для ключей. Выберем n такое, что $n = p \cdot q$, где p и q некоторые разные простые числа. Для такого n функция Эйлера будет иметь следующий вид:

$$\phi(n) = (p - 1) \cdot (q - 1)$$

Возвращаемся к генерации ключей. Выберем целое число e:

$$e \in [3, \phi(n) - 1]$$
$$GCD(e, \phi(n) - 1) = 1$$

Для него вычислим число d:

$$d \equiv e^{-1} \, mod \, \phi(n)$$

Тогда для шифрования сообщения нужно возвести число в степень e и взять остаток по модулю. Чтобы дешифровать сообщение нужно возвести зашифрованное сообщение в степень d и взять остаток по модулю n.

Листинг разработанной программы на языке Python

```python
# список первых простых чисел для последующей проверки
import random
first_primes_list = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
                                        31, 37, 41, 43, 47, 53, 59, 61, 67,
                                        71, 73, 79, 83, 89, 97, 101, 103,
                                        107, 109, 113, 127, 131, 137, 139,
                                        149, 151, 157, 163, 167, 173, 179,
                                        181, 191, 193, 197, 199, 211, 223,
                                        227, 229, 233, 239, 241, 251, 257,
                                        263, 269, 271, 277, 281, 283, 293,
                                        307, 311, 313, 317, 331, 337, 347]

# выбираем p из такого промежутка, который определяет длину числа в битах. Например,
# при n = 3 будет выбрано число из промежутка чисел от 5 (101 в двоичной СС) до 7 (111 в двоичной СС)

def n_bit_random(n):
        return random.randrange(2**(n-1)+1, 2**n - 1)

# первая проверка на простоту
def low_level_prime(n):
        while True:
                pc = n_bit_random(n)

                for i in first_primes_list:
                        if pc % i == 0 and i**2 <= pc:
                                break
                else:
                        return pc
```

```
# проверка на простоту по методу Миллера Рабина
def Miller_Rabin_passed(mrc):
        max_Divisions_Two = 0
        ec = mrc-1
        while ec % 2 == 0:
                ec >>= 1
                max_Divisions_Two += 1
        assert(2**max_Divisions_Two * ec == mrc-1)

        def trialComposite(round_tester):
                if pow(round_tester, ec, mrc) == 1:
                        return False
                for i in range(max_Divisions_Two):
                        if pow(round_tester, 2**i * ec, mrc) == mrc-1:
                                return False
                return True

        number_Rabin_Trials = 20
        for i in range(number_Rabin_Trials):
                round_tester = random.randrange(2, mrc)
                if trialComposite(round_tester):
                        return False
        return True

# генерируем p с длиной 8 бит
while True:
        bit = 8
        prime_candidate = low_level_prime(bit)
        if not Miller_Rabin_passed(prime_candidate):
                continue
        else:
                p = prime_candidate
                break

#  генерируем q с длиной 8 бит
while True:
        bit = 8
        prime_candidate = low_level_prime(bit)
        if not Miller_Rabin_passed(prime_candidate):
                continue
        else:
                q = prime_candidate
                break
```

```
# находим функцию Эйлера, для нашего n будет: phi(n) = phi(p*q) = (p-1)*(q-1)
n = p*q
def euler(p, q):
        return (p-1)*(q-1)




# теперь есть возможность сгенерировать открытый ключ e, который должен удовлетворять 2 условиям:
# 1) взаимно простое с phi(n)
# 2) меньше phi(n)

def GCD(n, m):
        if m == 0:
                return n
        else:
                return GCD(m, n % m)


for i in range(phi_n-1, 3, -1):
        if GCD(i, phi_n) == 1:
                e = i
                break
print('Открытый ключ: ', (e, n))

# генерируем закрытый ключ, зная, что он обратен нашему e по модулю phi(n)

for i in range(100000, 3, -1):
        if (e*i) % phi_n == 1:
                d = i
                break
print('Закрытый ключ: ', (d, n))

# ключи получены, можно написать функции шифрования и дешифрования сообщений.
# чтобы зашифровать сообщение понадобится представить буквы в виде чисел,
# с этим может помочь таблица символов Unicode




# сами функци написаны очевидным способом
```

```python
def encryption(text, key):
        e, n = key
        text_a = []
        for i in range(len(text)):
                text_a.append(ord(text[i]))
        for i in range(len(text_a)):
                text_a[i] = ((text_a[i]**e) % n)
        return text_a


def decryption(text_encryption, key):
        d, n = key
        text_b = []
        for i in range(len(text_encryption)):
                text_b.append((text_encryption[i]**e) % n)
        text_decryption = ''
        for i in range(len(text_b)):
                text_decryption += chr(text_b[i])
        return text_decryption

# проверим работу на сообщении, содержащее буквы разного регистра, цифры и иные симоволы.
# выведем зашифрованное сообщение, затем расшифрованно, должны совпасть
x = encryption('LaTeX-BIT2023', (e, n))
print(x)
print(decryption(x, (d, n)))
```

Вывод программы:

```
Открытый ключ:  (43511, 43931)
Закрытый ключ:  (87023, 43931)
[14451, 30797, 523, 33057, 18471, 3905, 24628, 23470, 523, 25480, 11898, 25480, 24119]
LaTeX-BIT2023
```

Nota bene: *при каждом запуске генерируются новые ключи*