Лабораторная работа 4

Криптосистема RSA

Вариант 18

Раконяц Даниела

7 группа

2023

**RSA** - криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших полупростых чисел.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

Формально как и для любой ассиметричной криптосистемы, для работы RSA необходимо реализовать 3 основные процедуры: генерацию ключей, функцию зашифрования и функцию расшифрования.

Генерация ключей:

1. Выбираются два больших простых числа *p* и *q*.

2. Вычисляется число *n* = *pq*; *n* – открытый параметр.

3. Открытый ключ *e* выбирается случайным образом из Zϕ(*n*) = {0, ..., ϕ(*n*)-1} таким образом, чтобы НОД(*e*, ϕ(*n*)) = 1. (Зная *p* и *q,* можно легко вычислить ϕ(*n*) = (*p*-1)(*q*-1)).

4. Закрытый ключ *d* из Zϕ(*n*); находится по формуле *d* = *e*-1 (mod ϕ(*n*)).

Замечание: Проверить, что случайно выбранное число *e* удовлетворяет условию НОД(*e*, ϕ(*n*)) = 1, и, если условие выполнено, найти закрытый ключ *d* = *e*-1 (mod ϕ(*n*)) можно с помощью расширенного алгоритма Евклида.

Сообщение *X*, которое необходимо зашифровать, представляется в виде некоторого числа из Z*n*.

Функция зашифрования: E*e*(*X*) = *Xe* (mod *n*), где *n* – открытый параметр, *e* – открытый ключ.

Замечание: Для возведения в степень рекомендуется использовать алгоритм быстрого возведения в степень. Вычислять остаток от деления необходимо после каждой операции умножения.

Функция расшифрования: D*d*(*Y*) = *Yd* (mod *n*), где *n* – открытый параметр, *d* – закрытый ключ.

| № Вар. | *p* | *q* | *e* | *X*1 | *Y*2 |
| --- | --- | --- | --- | --- | --- |
| 18 | 950226133300007 | 973747816218557 | 272205786540380931859823391349 | 487590396324873679144487947752 | 371209390170967767404608751313 |

Код:

class RSA:

def \_\_init\_\_(self, p: int, q: int, e: int):

self.n = p \* q

self.e = e

self.d = self.\_generate\_private\_key(p, q)

def euclid\_ext(self, a: int, b: int):

if a == 0:

return b, 0, 1

gcd, x, y = self.euclid\_ext(b % a, a)

return gcd, y - (b // a) \* x, x

def mod\_inverse(self, a: int, n: int):

g, x, \_ = self.euclid\_ext(a, n)

if g == 1:

return x % n

return None

def \_generate\_private\_key(self, p: int, q: int) -> int:

phi = (p - 1) \* (q - 1)

return self.mod\_inverse(self.e, phi)

def encrypt(self, message: int) -> int:

return pow(message, self.e, self.n)

def decrypt(self, message: int) -> int:

return pow(message, self.d, self.n)

if \_\_name\_\_ == '\_\_main\_\_':

p = 950226133300007

q = 973747816218557

e = 272205786540380931859823391349

X1 = 487590396324873679144487947752

Y2 = 371209390170967767404608751313

rsa = RSA(p, q, e)

print(f'Исходное сообщение X1: {X1}')

Y1 = rsa.encrypt(X1)

print(f'Зашифрованное сообщение Y1: {Y1}')

X1\_dec = rsa.decrypt(Y1)

print(f'После расшифровки: {X1\_dec}')

print(f'Зашифрованное сообщение Y2: {Y2}')

X2\_dec = rsa.decrypt(Y2)

print(f'После расшифровки: {X2\_dec}')

Результат:

