Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчёт

по лабораторной работе №3

**Ассиметричная криптография. RSA**

Выполнил:

Александров А.А.

Проверил:

Артемьев В.С.

Минск 2019

Содержание

[1. Постановка задачиError: Reference source not found](#_Toc506485975)

2

2. Краткие теоретические сведения

3[. Результаты выполненияError: Reference source not found](#_Toc506485977)

4[. ВыводыError: Reference source not found](#_Toc506485978)

[ПРИЛОЖЕНИЕ 1. Исходный код программы.Error: Reference source not foundError: Reference source not foundError: Reference source not found](#_Toc506485979)

# 1. Постановка задачи

1. Изучить теоретические сведения.

2. Создать программу, генерирующую ключи, а также шифрующую (дешифрующую) сообщения алгоритмом RSA

**2. Краткие теоретические сведения**

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) - криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых 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.

Опубликованная в ноябре 1976 года статья Уитфилда Диффи и Мартина Хеллмана «Новые направления в криптографии» (англ. New Directions in Cryptography) перевернула представление о криптографических системах, заложив основы криптографии с открытым ключом. Разработанный впоследствии алгоритм Диффи — Хеллмана позволял двум сторонам получить общий секретный ключ, используя незащищенный канал связи. Однако этот алгоритм не решал проблему аутентификации. Без дополнительных средств пользователи не могли быть уверены, с кем именно они сгенерировали общий секретный ключ.

Изучив эту статью, трое учёных Рональд Ривест, Ади Шамир и Леонард Адлеман из Массачусетского технологического института (MIT) приступили к поискам математической функции, которая бы позволяла реализовать сформулированную Уитфилдом Диффи и Мартином Хеллманом модель криптографической системы с открытым ключом. После работы над более чем 40 возможными вариантами им удалось найти алгоритм, основанный на различии в том, насколько легко находить большие простые числа и насколько сложно раскладывать на множители произведение двух больших простых чисел, получивший впоследствии название RSA. Система была названа по первым буквам фамилий её создателей.

**Алгоритм создания открытого и секретного ключей**

RSA-ключи генерируются следующим образом:

1. Выбираются два различных случайных простых числа p и q заданного размера (например, 1024 бита каждое).

2. Вычисляется их произведение n = p\*q, которое называется модулем.

3/ Вычисляется значение функции Эйлера от числа n:

phi(n) = phi(pq) = (p-1)(q-1)

4.Выбирается целое число e (1<e<n), взаимно простое со значением функции phi(n)

Число e называется открытой экспонентой (англ. public exponent)

Обычно в качестве e берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, например, простые из чисел Ферма: 17, 257 или 65537, так как в этом случае время, необходимое для шифрования с использованием быстрого возведения в степень будет меньше.

Слишком малые значения e, например 3, потенциально могут ослабить безопасность схемы RSA.[17]

5. Вычисляется число d, мультипликативно обратное к числу e по модулю phi(n)

Число d называется секретной экспонентой. Обычно оно вычисляется при помощи расширенного алгоритма Евклида.

6. Пара (e,n) публикуется в качестве открытого ключа RSA (англ. RSA public key).

7. Пара (d,n)играет роль закрытого ключа RSA (англ. RSA private key) и держится в секрете.

**Алгоритм шифрования:**

1.Взять открытый ключ (e,n)

2. Взять открытый текст m

3. Зашифровать сообщение с использованием открытого ключа:

c=E(m)=m^e (mod n)

**Алгоритм расшифрования:**

1. Принять зашифрованное сообщение c

2. Взять свой закрытый ключ (d,n)

3.Применить закрытый ключ для расшифрования сообщения:

m=D(c)=c^d(mod n)

**Схема алгоритма**



# Результаты выполнения

# 

# Выводы

# В результате выполнения лаюораторной работы была получена реализация алгоритма RSA, а также алгоритм генерации ключей для данного алгоритма. Данный алгоритм относится к семейству асимметричных алгоритмов.

# ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

import random

def get\_primes(start, stop):

print("start:", start, ", stop:", stop)

if start >= stop:

return []

primes = [2]

for n in range(3, stop + 1, 2):

for p in primes:

if n % p == 0:

break

else:

primes.append(n)

while primes and primes[0] < start:

del primes[0]

return primes

def are\_relatively\_prime(a, b):

for n in range(2, min(a, b) + 1):

if a % n == b % n == 0:

return False

return True

def make\_key\_pair(length):

if length < 4:

raise ValueError('cannot generate a key of length less '

'than 4 (got {!r})'.format(length))

n\_min = 1 << (length - 1)

n\_max = (1 << length) - 1

print("n\_min", n\_min, ", n\_max:", n\_max)

start = 1 << (length // 2 - 1)

stop = 1 << (length // 2 + 1)

primes = get\_primes(start, stop)

while primes:

print("Primes:", primes)

p = random.choice(primes)

primes.remove(p)

q\_candidates = [q for q in primes

if n\_min <= p \* q <= n\_max]

if q\_candidates:

q = random.choice(q\_candidates)

break

else:

raise AssertionError("cannot find 'p' and 'q' for a key of "

"length={!r}".format(length))

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

for e in range(3, stop, 2):

if are\_relatively\_prime(e, stop):

break

else:

raise AssertionError("cannot find 'e' with p={!r} "

"and q={!r}".format(p, q))

for d in range(3, stop, 2):

if d \* e % stop == 1:

break

else:

raise AssertionError("cannot find 'd' with p={!r}, q={!r} "

"and e={!r}".format(p, q, e))

print("p:", p, ", q:", q)

# That's all. We can build and return the public and private keys.

return p \* q, e, d

def encrypt(data, e, n):

if isinstance(data, str):

data = data.encode('ascii')

encrypted\_data = []

for byte in data:

encrypted\_data.append(pow(byte, e, n))

return encrypted\_data

def decrypt(data, d, n):

decrypted\_data = []

for byte in data:

decrypted\_data.append(pow(byte, d, n))

return bytes(decrypted\_data).decode("utf8")