Министерство образования Республики Беларусь
Белорусский государственный университет информатики и радиоэлектроники
Кафедра информатики

ОТЧЕТ
по лабораторной работе №3

Ассиметричная криптография. RSA

Выполнил:
студент гр. 653501
Шинкевич Г. С.

Проверил:
Артемьев В. С.

Минск 2019

ЗАДАНИЕ:

Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма RSA.

Ассиметричное шифрование

В системах с открытым ключом используются два ключа — открытый и закрытый, связанные определённым математическим образом друг с другом. Открытый ключ передаётся по открытому (то есть незащищённому, доступному для наблюдения) каналу и используется для шифрования сообщения и для проверки ЭЦП. Для расшифровки сообщения и для генерации ЭЦП используется секретный ключ.

Данная схема решает проблему симметричных схем, связанную с начальной передачей ключа другой стороне. Если в симметричных схемах злоумышленник перехватит ключ, то он сможет как «слушать», так и вносить правки в передаваемую информацию. В асимметричных системах другой стороне передается открытый ключ, который позволяет шифровать, но не расшифровывать информацию. Таким образом решается проблема симметричных систем, связанная с синхронизацией ключей.

Рисунок 1. Схема ассимтеричного шифрования

Алгоритм RSA

Криптографические системы с открытым ключом используют так называемые односторонние функции, которые обладают следующим свойством:

- если известно x, то f(x) вычислить относительно просто;
- если известно y=f(x), то для вычисления x нет простого (эффективного) пути.

Под односторонностью понимается не теоретическая однонаправленность, а практическая невозможность вычислить обратное значение, используя современные вычислительные средства, за обозримый интервал времени.

В основу криптографической системы с открытым ключом RSA положена сложность задачи факторизации произведения двух больших простых чисел. Для шифрования используется операция возведения в степень по модулю большого числа. Для дешифрования за разумное время необходимо уметь вычислять функцию Эйлера от данного большого числа, для чего необходимо знать разложение числа на простые множители.

В криптографической системе с открытым ключом каждый участник располагает как открытым ключом, так и закрытым ключом. В криптографической системе RSA каждый ключ состоит из пары целых чисел. Каждый участник создаёт свой открытый и закрытый ключ самостоятельно. Закрытый ключ каждый из них держит в секрете, а открытые ключи можно сообщать кому угодно или даже публиковать их.

Генерация ключей

RSA-ключи генерируются следующим образом:

1. Выбираются два различных случайных простых числа p и q заданного размера.
2. Вычисляется их произведение n = p · q, которое называется модулем.
3. Вычисляется значение функции Эйлера от числа n:

$$\varphi(n) = (p-1) \cdot (q-1).$$

4. Выбирается целое число e (1 < e < φ (n)), взаимно простое со значением функции φ (n).
5. Вычисляется число d, мультипликативно обратное к числу e по модулю φ (n), то есть число, удовлетворяющее сравнению:

$$d \cdot e \equiv 1 \pmod{\varphi(n)}.$$

6. Пара (e, n) публикуется в качестве открытого ключа.
7. Пара (d, n) – закрытый ключ.

Шифрование

Сообщение шифруется открытым ключом того человека, кому мы хотим передать сообщение:

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

Расшифрование

Сообщение дешифруется закрытым ключом:

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

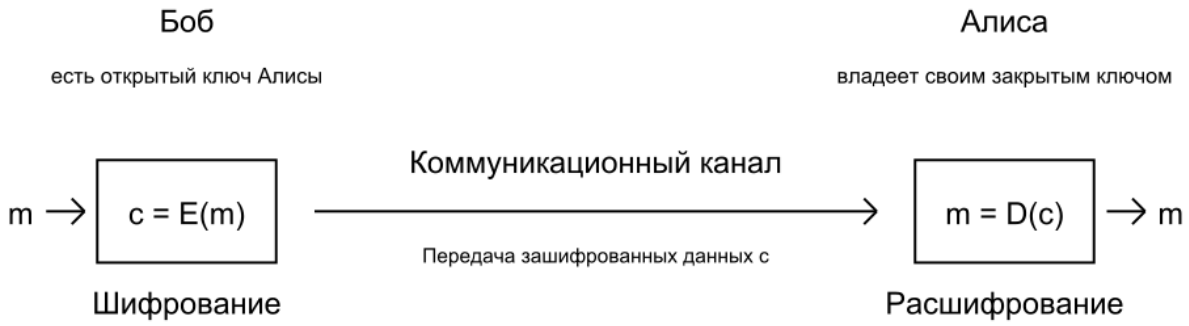

Рисунок 2. Алгоритм шифрования/дешифрования RSA

ДЕМОНСТРАЦИЯ РАБОТЫ

```
Public key: (13005878136298965981313074923955234510962995068045968510678891026484669399385 7,
  19890515792612900094534133566173350890551869476305375108065794262764253430476 9)
Initial text: bla-bla
Encrypted array: [17722237199130991401536361043975729969800017692200392819907941599492280694305 7,
  11137121497013117669864836646621529252626716505495073894973556947955928755350 3,
  19573452164715417277785867389612974736419342129409551473969414840778873827512 3,
  18192395595937454101233602629108602120682527487091313501805447117814037806417 3,
  17722237199130991401536361043975729969800017692200392819907941599492280694305 7,
  11137121497013117669864836646621529252626716505495073894973556947955928755350 3,
  19573452164715417277785867389612974736419342129409551473969414840778873827512 3]
Decrypted text: bla-bla
```

ВЫВОД

В результате лабораторной работы была написана программа для шифрования и дешифрования текстовых файлов с помощью алгоритма RSA.

ЛИСТИНГ ПРОГРАММЫ

```
import itertools
import math
import random
import csv

FIRST_PRIMES_10K = []


def get_first_primes():
    primes = []
    with open("10000.txt") as file:
        reader = csv.reader(file, delimiter=' ',
quoting=csv.QUOTE_NONNUMERIC)
        for row in reader:
            for elem in row:
                if isinstance(elem, float):
                    primes.append(int(elem))
    return primes


def generate_random_prime(size):
    p = (random.getrandbits(size) | (1 << size)) | 1
    for i in itertools.count(1):
        if is_prime(p):
            return p
        else:
            if i % (size * 2) == 0:
                p = (random.getrandbits(size) | (1 << size)) | 1
            else:
                p += 2


def is_prime(n):
    is_prime = is_prime_simple(n, 256)
    if is_prime is not None:
        return is_prime
```

```
    return is_prime_rabin_miller(n)


def is_prime_simple(number, first_primes_number):
    for p in FIRST_PRIMES_10K[:first_primes_number]:
        if number % p == 0:
            return number == p
    return None


def is_prime_rabin_miller(number):
    rounds = int(math.log2(number))
    t = number - 1
    s = 0
    while t % 2 == 0:
        s += 1
        t //= 2
    generated_numbers = set()
    for _ in range(rounds):
        a = random.randint(2, number - 2)
        while a in generated_numbers:
            a = random.randint(2, number - 2)
        generated_numbers.add(a)
        x = pow(a, t, number)
        if x == 1 or x == number - 1:
            continue
        for _ in range(s - 1):
            x = pow(x, 2, number)
            if x == 1:
                return False
            elif x == number - 1:
                break
        else:
            return False
        continue
    return True


def get_bezout_coeffs(a, b):
```

```
    x, x_, y, y_ = 1, 0, 0, 1
    while b:
        q = a // b
        a, b = b, a % b
        x, x_ = x_, x - x_ * q
        y, y_ = y_, y - y_ * q
    return x, y


def multiplicative_inverse(a, b):
    x, y = get_bezout_coeffs(a, b)
    if x < 0:
        return b + x
    return x


def generate_keys(size):
    p = generate_random_prime(size // 2)
    q = generate_random_prime(size // 2)
    while q == p:
        q = generate_random_prime(size // 2)
    n = p * q
    phi = (p - 1) * (q - 1)
    while True:
        e = random.randint(17, phi - 1)
        if math.gcd(e, phi) == 1:
            break
    d = multiplicative_inverse(e, phi)
    return (e, n), (d, n)


def rsa_encrypt(message, key):
    e, n = key
    x = [pow(ord(a), e, n) for a in message]
    return x


def rsa_decrypt(message, key):
    d, n = key
```

```
    x = [pow(a, d, n) for a in message]
    return ''.join(chr(a) for a in x)


if __name__ == '__main__':
    FIRST_PRIMES_10K = get_first_primes()

    public_key, private_key = generate_keys(256)
    print("Public key: {}".format(public_key))

    with open("data.txt") as file:
        text = file.read()

    print("Initial text: {}".format(text))
    encrypted = rsa_encrypt(text, public_key)
    print("Encrypted array: {}".format(encrypted))
    decrypted = rsa_decrypt(encrypted, private_key)
    print("Decrypted text: {}".format(decrypted))
```