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

ОТЧЕТ
по лабораторной работе №3

Асимметричная криптография. RSA

Выполнил:
студент гр. 653503
Юревич А.А.

Проверил:
Артемьев В.С.

Минск 2019

Введение

Постановка задачи:

1) Изучить теоретические сведения.
2) Создать программу, читающую данные из файла и шифрующую исходного текста (дешифрующую) их с помощью с помощью RSA.

Краткие теоретические сведения:

Целью данной лабораторной работы является изучение и написание алгоритма шифрования RSA, в ходе которого мы познакомимся с концепцией криптографического алгоритма с открытым ключом, основывающегося на вычислительной сложности задачи факторизации больших целых чисел. Необходимо реализовать алгоритм шифрования с открытым ключом.

Теоретические сведения

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

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 (обратной операции) 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. Открытый и закрытый ключи каждого участника обмена сообщениями в криптосистеме RSA образуют «согласованную пару» в том смысле, что они являются *взаимно обратными.*

Тест Миллера — Рабина

Тест Миллера — Рабина — вероятностный полиномиальный тест простоты. Тест Миллера — Рабина, наряду с тестом Ферма и тестом Соловея — Штрассена, позволяет эффективно определить, является ли данное число составным. Однако, с его помощью нельзя строго доказать простоту числа. Тем не менее тест Миллера — Рабина часто используется в криптографии для получения больших случайных простых чисел.

Так как криптостойкость многих алгоритмов шифрования основывается на секретных ключах, для создания которых необходимы простые числа (например, так работает шифр RSA), то при создании таких ключей важно уметь достаточно быстро проверять большие числа на простоту. Вероятностные тесты простоты, такие как тест Миллера-Рабина и Тест Соловея — Штрассена, показывают большую эффективность использования и простоту выражения по сравнению с детерминированными тестами. Алгоритм Миллера-Рабина позволяет выполнять проверку за малое время и давать при этом достаточно малую вероятность того, что число на самом деле является составным.

Алгоритм Миллера — Рабина параметризуется количеством раундов r. Рекомендуется брать r порядка величины log2(m), где m — проверяемое число. Для данного m находятся такие целое число s и целое нечётное число t, что $m-1=2st$. Выбирается случайное число a,$1<a<m$. Если a не является свидетелем простоты числа m, то выдается ответ «m составное», и алгоритм завершается. Иначе, выбирается новое случайное число a и процедура проверки повторяется. После нахождения r свидетелей простоты, выдается ответ «m, вероятно, простое», и алгоритм завершается.

Алгоритм создания открытого и секретного ключей

RSA-ключи генерируются следующим образом:

1. для начала мы берём два случайных простых числа (pp и qq) стандартного размера (например, 1024 бит - то есть - достаточно большие)
2. затем вычисляем функцию Эйлера от числа n, таким образом:

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

   (правая часть здесь рассчитывается моментально, а потому из данного уравнения мы можем получить само значение n)

3. Далее вы подбираем e - открытую экспоненту, так чтобы:

    - её значение было взаимно простым со значением функции $\varphi(n)$
    - удовлетворяло неравенству: $1 < e < \varphi(n)$

4. И теперь вычисляем закрытую экспоненту d, значение которой должно соответствовать условию:

    $d \cdot e \equiv 1 \bmod \varphi(n)$

5. Пара {e, n} публикуется в качестве *открытого ключа*.
6. Пара $\{d, \varphi(n)\}$ играет роль *закрытого ключа RSA* и держится в секрете.

Практическая часть

Создадим файл с начальным текстом (рис. 1).

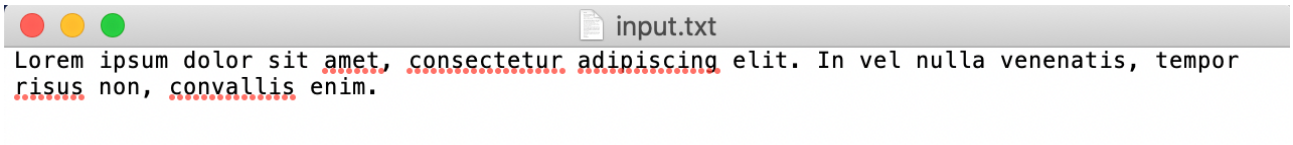

Рисунок 1 – Начальный текст

Далее запустим программу и получим 2 файла: с зашифрованным и расшифрованным сообщениями (рис. 2 и рис. 3).

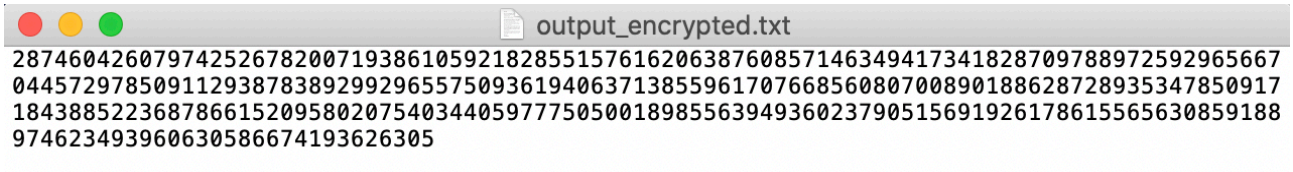

Рисунок 2 – Зашифрованный текст

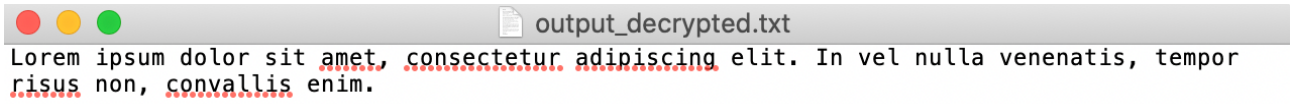

Рисунок 3 –Расшифрованный текст

Выводы

Система RSA используется для защиты программного обеспечения и в схемах цифровой подписи. Надёжность шифрования обеспечивается тем, что третьему лицу (старающемуся взломать шифр) очень трудно вычислить закрытый ключ по открытому. Оба ключа вычисляются из одной пары простых чисел (p и q). То есть ключи связаны между собой. Но установить эту связь очень сложно. Основной загвоздкой является декомпозиция модуля n на простые сомножители p и q. Если число является произведением двух очень больших простых чисел, то его очень трудно разложить на множители.

На самом деле, RSA способ шифрования очень слаб и никогда не используется. Причина проста — шифрование по буквам. Одна и та же буква будет шифроваться одним и тем же числом. Если злоумышленник перехватит достаточно большое сообщение, он сможет догадаться о его содержимом. Сперва он обратит внимание на частые коды пробелов и разделит шифровку на слова. Потом он заметит однобуквенные слова и догадается, как кодируются буквы «а», «и», «о», «в», «к»… Путём недолгого перебора, он вычислит дополнительные буквы по коротким словам, типа «но», «не», «по». И по более длинным словам без труда восстановит все оставшиеся буквы.

Код программы

```
import random


MILLER_RABIN_CERTAINTY = 50
PRIME_MIN = 10 ** 100
PRIME_MAX = 10 ** 150


def are_coprime(a, b):
    while b:
        a, b = b, a % b

    if a == 1:
        return True

    return False


def mod_inverse(a, b):
    x = 0
    ox = 1
    ob = b

    while b:
        a, q, b = b, a // b, a % b
        x, ox = ox - (q * x), x

    return ox % ob


def miller_rabin(n):
    if n % 2 == 0 or n % 3 == 0:
        return False

    d = n - 1
    r = 0

    while d % 2 == 0:
        d //= 2
        r += 1

    for _ in range(MILLER_RABIN_CERTAINTY):
        a = random.randrange(2, n - 1)
        v = pow(a, d, n)
        if v != 1:
            i = 0
            while v != (n - 1):
                if i == r - 1:
                    return False
                else:
                    i += 1
                    v = pow(v, 2, n)
    return True


def generate_keys():
    while 1:
        p = random.randint(PRIME_MIN, PRIME_MAX)
        if miller_rabin(p):
            break

    while 1:
        q = random.randint(PRIME_MIN, PRIME_MAX)
        if miller_rabin(q) and p != q:
            break
```

```
    n = p * q
    phi_n = (p - 1) * (q - 1)

    while 1:
        e = random.randint(2, phi_n - 1)
        if are_coprime(e, phi_n):
            break

    d = mod_inverse(e, phi_n)

    return e, d, n


def encrypt(string, e, n):
    msg_bin = "".join(bin(ord(c))[2:].zfill(8) for c in string)

    for i in range(256):
        new_msg_bin = msg_bin + bin(i)[2:].zfill(8)
        if are_coprime(int(new_msg_bin, 2), n):
            m = int(new_msg_bin, 2)
            break

    c = pow(m, e, n)

    return c


def decrypt(c, d, n):
    m = pow(c, d, n)
    binstr = bin(m)[2:]

    while len(binstr) % 8 != 0:
        binstr = "0" + binstr

    msg = ""
    binstr = binstr[:-8]

    for i in range(0, len(binstr), 8):
        msg += chr(int(binstr[i:i + 8], 2))

    return msg


if __name__ == '__main__':
    e, d, n = generate_keys()

    with open('input.txt') as f:
        text = f.read()

    msg = encrypt(text, e, n)
    with open('output_encrypted.txt', "w") as f:
        f.write(str(msg))

    with open("output_decrypted.txt", "w") as f:
        f.write(decrypt(msg, d, n))
```