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

ОТЧЕТ
по лабораторной работе №4

Асимметричная криптография. Алгоритм Эль-Гамаля

Выполнил:
студент гр. 653503
Юревич А.А.

Проверил:
Артемьев В.С.

Минск 2019

Введение

Целью данной лабораторной работы является изучение и написание алгоритма шифрования Эль-Гамаля, в ходе которого мы познакомимся с концепцией криптографического алгоритма с открытым ключом, основанный на трудности вычисления дискретных логарифмов в конечном поле. Необходимо реализовать алгоритм шифрования с открытым ключом.

Теоретические сведения

Схема Эль-Гамаля (Elgamal) — криптосистема с открытым ключом, основанная на трудности вычисления дискретных логарифмов в конечном поле. Криптосистема включает в себя алгоритм шифрования и алгоритм цифровой подписи. Схема Эль-Гамаля лежит в основе бывших стандартов электронной цифровой подписи в США (DSA) и России (ГОСТ Р 34.10-94).

Генерация ключей:

- Генерируется случайное простое число p.
- Выбирается целое число g — первообразный корень p.
- Выбирается случайное целое число x такое, что 1 < x < p - 1.
- Вычисляется y = g^x mod p.
- Открытым ключом является y, закрытым ключом — число x.

Шифрование

Сообщение M должно быть меньше числа p. Сообщение шифруется следующим образом:

1. Выбирается сессионный ключ — случайное целое число K такое, что

   1 < k < p - 1

2. Вычисляются числа a = g^k mod p и b = y^k M mod p.
3. Пара чисел a b является шифротекстом.

Нетрудно увидеть, что длина шифротекста в схеме Эль-Гамаля длиннее исходного сообщения M вдвое.

Расшифрование

Зная закрытый ключ x, исходное сообщение можно вычислить из шифротекста a b по формуле:

$$M = b(a^x)^{-1} mod p.$$

При этом нетрудно проверить, что

$$(a^x)^{-1} = g^{-kx} mod p$$

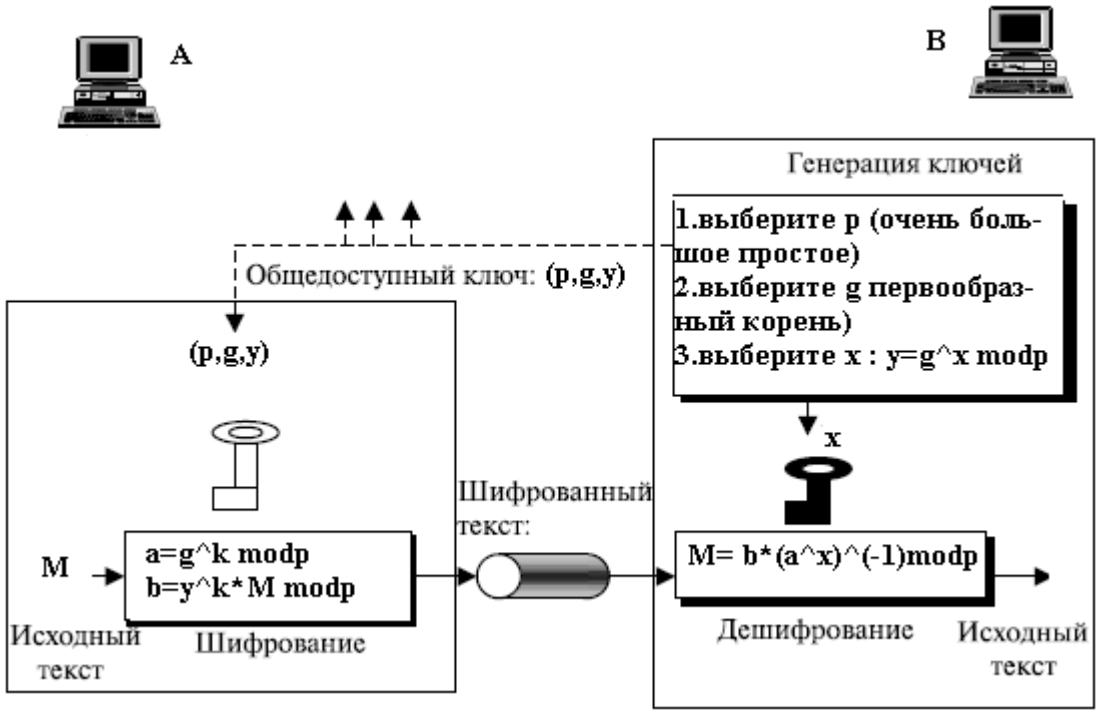
и поэтому

$$b(a^x)^{-1} = (y^k M)g^{-xk} \equiv (g^{xk} M)g^{-xk} \equiv M \pmod{p} .$$

Для практических вычислений больше подходит следующая формула:

$$M = b(a^x)^{-1} \mod p = b \cdot a^{(p-1-x)} \mod p$$

Схема шифрования



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).

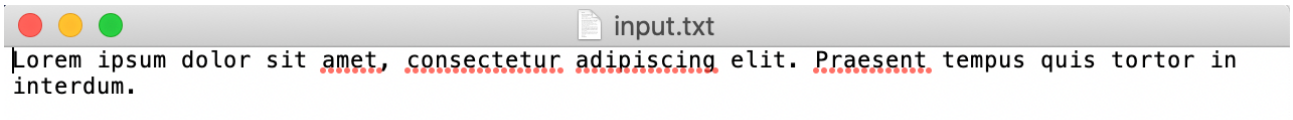


Рисунок 1 – Начальный текст

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

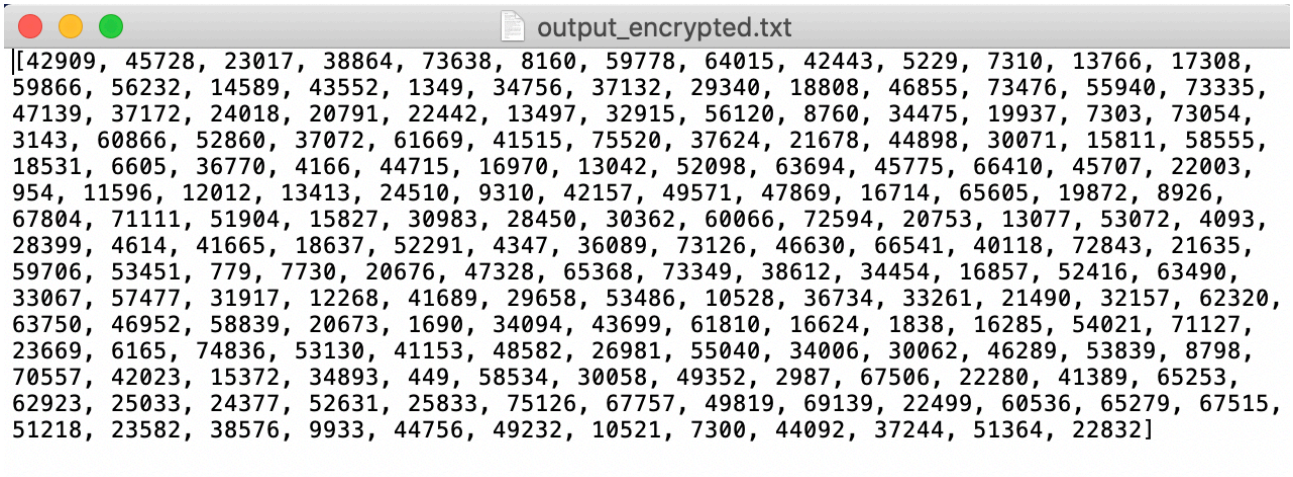


Рисунок 2 – Зашифрованный текст

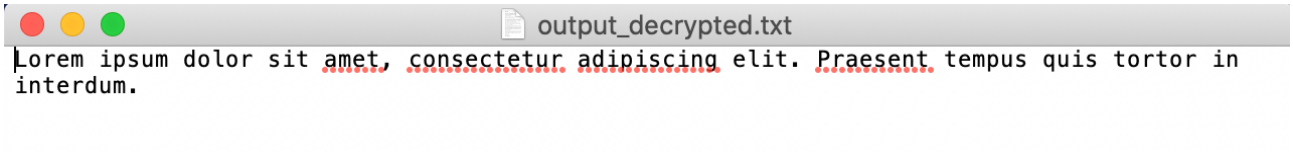


Рисунок 3 –Расшифрованный текст

Вывод

Так как в схему Эль-Гамаля вводится случайная величина k, то шифр Эль-Гамаля можно назвать шифром многозначной замены. Из-за случайности выбора числа k такую схему еще называют схемой вероятностного шифрования. Вероятностный характер шифрования является преимуществом для схемы Эль-Гамаля, так как у схем вероятностного шифрования наблюдается большая стойкость по сравнению со схемами с определенным процессом шифрования.

Недостатком схемы шифрования Эль-Гамаля является удвоение длины зашифрованного текста по сравнению с начальным текстом.

Код программы

```
import random
import math
import primes


def exp(base, power, m):
    result = 1
    for _ in range(power):
        result = (result * base) % m
    return result


def get_p():
    a = primes.a

    rand = random.randint(100, len(a))

    return a[rand]


def get_fact(p):
    fact = []
    n = p - 1

    for i in range(2, int(math.sqrt(n))):
        if n % i == 0:
            fact.append(i)
            while n % i == 0:
                n /= i

    if n > 1:
        fact.append(n)

    return fact


def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)


def get_antiderivative_root(p, fact):
    g = random.randint(2, p - 1)

    for i in range(len(fact)):
        if exp(g, int((p - 1) / fact[i]), p) == 1:
            return get_antiderivative_root(p, fact)

    return g


def get_k(p):
    while True:
        k = random.randint(1, p - 1)
        if gcd(k, p - 1) == 1:
            return k


def generate_keys():
    p = get_p()

    g = get_antiderivative_root(p, get_fact(p))

    x = random.randint(2, p - 1)
```

```
    y = exp(g, x, p)

    print()

    print("  Открытые ключи:")
    print(f"     p = {p}")
    print(f"     g = {g}")
    print(f"     y = {y}")

    print()

    print("  Закрытый ключ")
    print(f"     x = {x}")

    print()

    return {"p": p, "g": g, "y": y}, x


def encrypt(message, p, g, y):
    encrypted = []

    for m in message:
        k = get_k(p)

        a = exp(g, k, p)
        b = exp(y, k, p)
        b = (b * m) % p

        encrypted.append(a)
        encrypted.append(b)

    return encrypted


def decrypt(message, p, x):
    decrypted = []

    for i in range(0, len(message) - 1, 2):
        a = message[i]
        b = message[i + 1]

        m = (b * a ** (p - 1 - x)) % p

        decrypted.append(chr(m))

    return decrypted


if __name__ == "__main__":
    public_keys, private_key = generate_keys()

    with open("input.txt", "r") as f:
        message = f.readline()

    message_chars = []
    for m in message:
        message_chars.append(ord(m))

    encrypted = encrypt(message_chars, public_keys["p"], public_keys["g"], pub-
lic_keys["y"])
    with open('output_encrypted.txt', "w") as f:
        f.write(str(encrypted))

    decrypted = decrypt(encrypted, public_keys["p"], private_key)
    with open("output_decrypted.txt", "w") as f:
        f.write("".join(decrypted))
```