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

ОТЧЕТ
по лабораторной работе №4

Ассиметричная криптография. Алгоритм Эль-Гамаля

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 Эль-Гамаля.

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.

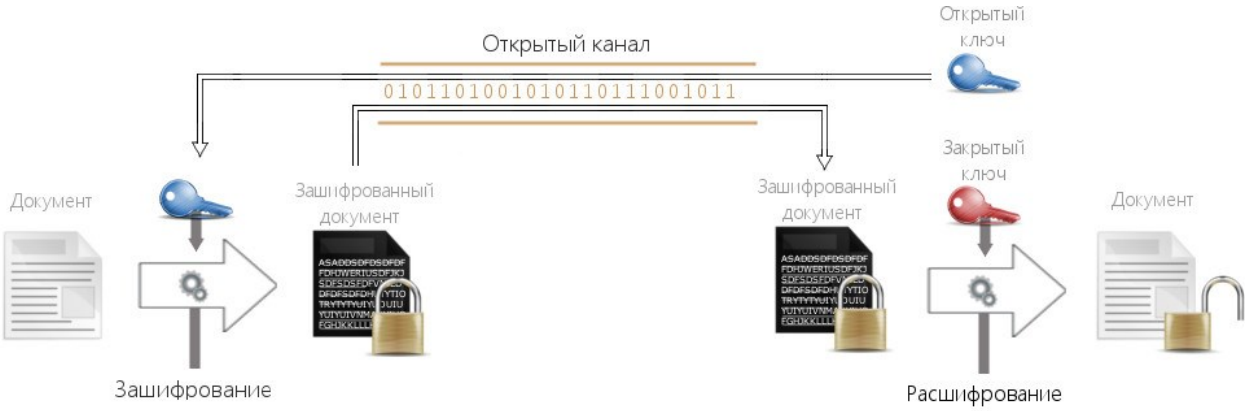

Рисунок 1. Схема ассимтеричного шифрования

Алгоритм Эль-Гамаля

Генерация ключей

1. Генерируется случайное простое число p.
2. Выбирается целое число g — первообразный корень p.
3. Выбирается случайное целое число x такое, что $1 < x < p - 1$.
4. Вычисляется $y = g^x \bmod p$.

5. Открытым ключом является (g, p, y), закрытым ключом — число x.

Шифрование

1. Выбирается сессионный ключ — случайное целое число k такое, что $1 < k < p - 1$.
2. Вычисляются числа $a = g^k \bmod p$ и $b = y^k M \bmod p$.
3. (a, b) – шифротекст.

Расшифрование

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

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

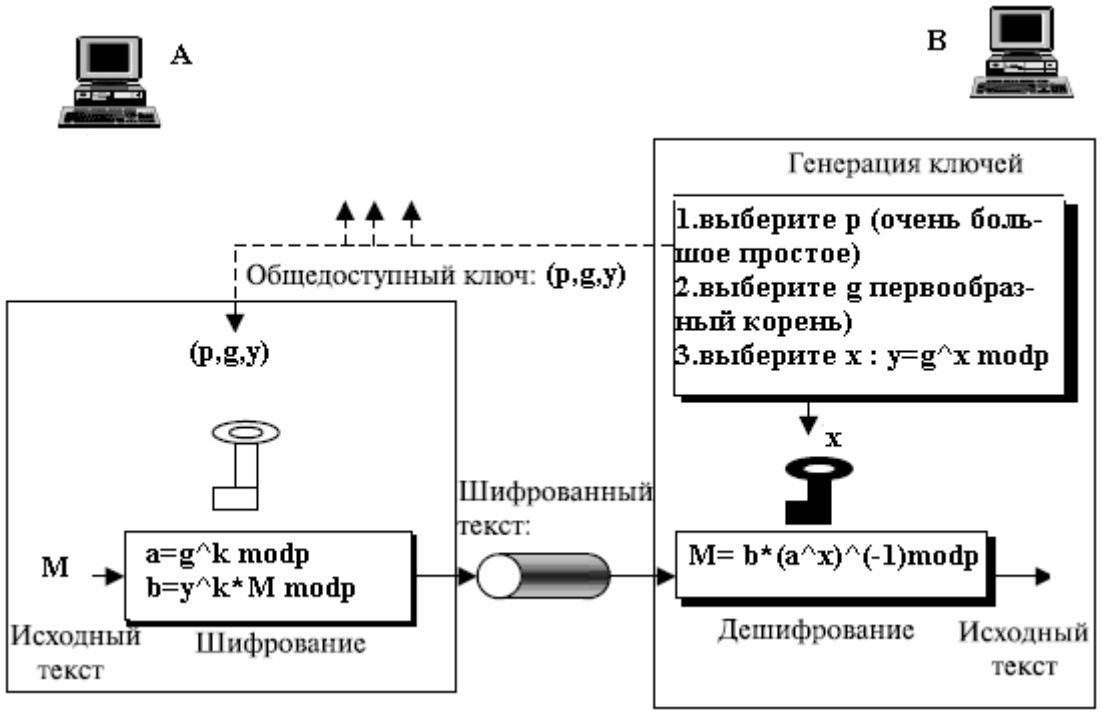

Рисунок 2. Шифрование/дешифрование схемой Эль-Гамаля

ДЕМОНСТРАЦИЯ РАБОТЫ

```
Public key: (5, 14778204539051763212160816375410218960002499998690802342135753058445369810823 3,
 32912662955846223462516217538385447121364807095432059889334052375858627051945)
Initial text: bla-bla
Encrypted array: [(17013043629340705428068473737838424369881447044171138066417141527456170667662,
 66213718955321927034645042553995392756263895863140277982212662032370147882382 44),
 (13195923551447328560512341749461731926058633467808383432775692153434076530197 0,
 51796248969491639587919372292507541979147579909441038892981872075907695297040 64),
 (75807888727009705968774004625998467568192918635092480834800363694289221649318,
 11987896416997861562532410218208082853595933326680163221648559287566144541399259),
 (72184136623181831744287752407653872593512935796092469555422067793571863088933,
 34480821733698364083736478359591848678477878189000983465509054591873969522090 60),
 (12779114180797692577376999129288308736681173669870721838893959766379375521476 6,
 85330723744924309454772571655484193386624919193316167933285391661030906701959 84),
 (14172403971506665303334877148653973086363262450224269244116476595270625379889 4,
 20025043619897297204391660584458317814982076655119654402039288816209964973987 44),
 (73878943269356894603905265032885342966137366374759007858205177153362369100381,
 12447919595228042441473136702632110137269056246340477613393469176231791803277737)]
Decrypted text: bla-bla
```

ВЫВОД

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

ЛИСТИНГ ПРОГРАММЫ

```
import csv
import itertools
import math
import random
from sympy.ntheory.residue_ntheory import primitive_root

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_primitive_root(modulo):
    return primitive_root(modulo)


def generate_keys(size):
    p = generate_random_prime(size)
    g = get_primitive_root(p)
    while True:
        x = random.randint(1, p - 1)
        if math.gcd(x, p - 1) == 1:
            break
    y = pow(g, x, p)
    return (g, p, y), x


def encrypt(number, key):
    g, p, y = key
    while True:
        k = random.randint(1, p - 1)
        if math.gcd(k, p - 1) == 1:
            break
    a = pow(g, k, p)
    b = number * pow(y, k, p)
    return a, b


def decrypt(number, key, p):
    a, b = number
    x = key
    return b * pow(pow(a, x, p), p - 2, p) % p


def encrypt_text(message, key):
    x = [encrypt(ord(a), key) for a in message]
    return x


def decrypt_text(message, key, p):
    x = [decrypt(a, key, p) 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 = encrypt_text(text, public_key)
    print("Encrypted array: {}".format(encrypted))
    decrypted = decrypt_text(encrypted, private_key, public_key[1])
    print("Decrypted text: {}".format(decrypted))
```