МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №10 НА ТЕМУ:

Исследование асимметричных шифров RSA и Эль-Гамаля

Выполнила студентка 3 курса 4 группы

Коржова Валерия Сергеевна

Минск 2023

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации асимметричных шифров RSA и Эль-Гамаля.

**Теоретические сведения**

Практически все системы асимметричного зашифрования/расшифрования основаны либо на проблеме факторизации (среди них – RSA), либо на проблеме дискретного логарифмирования (среди них – Эль-Гамаля).

**Алгоритм RSA**

Для генерации двух ключей: тайного и открытого используются два больших случайных простых числа *p* и *q*. Для максимальной большей криптостойкости нужно выбирать *p* и *q* равной длины. Рассчитывается произведение: *n* = *pq*. Это есть один из трех компонент ключа, состоящего из чисел *n*, *e*, *d*.

Затем случайным образом выбирается второй компонент ключа (открытый ключ или ключ зашифрования, *e*, такой что *e* и (*p* – 1)(*q* – 1) являются взаимно простыми числами; вспомним, что (*p* – 1)(*q* – 1) = φ(*n*) – функция Эйлера).

Наконец, расширенный алгоритм Евклида используется для вычисления третьего компонента ключа: ключа расшифрования *d* такого, что выполняется условие:

*ed* ≡ 1 (mod φ(*n*)).

Другими словами:

*d*–1 ≡ *e* (mod φ(*n*)).

Таким образом, сформирован ключ, состоящий из трех чисел, которые в свою очередь образуют две вышеупомянутые взаимосвязанные части: открытый (публичный) ключ (*e*, *n*) и тайный ключ (*d*, *n*; на самом деле, как видим, тайным здесь является лишь первое из пары чисел).

Для зашифрования/расшифрования используется ключ получателя: отправитель шифрует сообщение открытым ключом, а получатель расшифровывает шифртекст своим тайным ключом.

Зашифрование. Если шифруется сообщение *М*, состоящее из *r* блоков: *m*1, *m*2, …, *mi*, …, *mr*, то шифртекст *С* будет состоять из такого же числа (*r*) блоков, представляемых числами:

*ci*≡ (*mi*)*e* mod *n*.

Расшифрование. Для расшифрования каждого зашифрованного блока производится вычисление вида:

*mi* ≡ (*ci*)*d* mod *n*.

Размер ключа в алгоритме RSA связан с размером модуля *n*. Два числа *p* и *q*, произведение которых равно *n*, должны иметь приблизительно одинаковую длину, поскольку в этом случае найти сомножители (факторы) сложнее, чем в случае, когда длина чисел значительно различается.

**Алгоритм Эль-Гамаля**

Рассматриваемый алгоритм отличается от алгоритма RSA несколькими параметрами и особенностями:

1) генерацией ключевой информации и числом компонент, составляющих ключ;

2) каждому блоку (символу) открытого сообщения в шифртексте на основе алгоритма Эль-Гамаля соответствуют 2 блока (в RSA – один-один);

3) в алгоритме Эль-Гамаля при зашифровании используется число (обозначим его k), которое практически никак не связано с ключевой информацией получателя и которое принимает (по определению) различные значения при зашифровании различных блоков сообщения.

Генерация ключевой информации. Выбирается простое число *р*. Выбирается число (*g*, *g* < *p*), являющееся первообразным корнем числа *р* – очень важный элемент с точки зрения безопасности алгоритма (см. ниже).

Далее выбирается число *х* (*х* < *p*) и вычисляется последний компонент ключевой информации:

*y* ≡ *gх* mod *р*.

Владельцу сформированной ключевой информации, состоящей из 4 чисел, может посылаться некоторый шифртекст, созданный с использованием открытого ключа получателя: *p*, *g*, *y*. Расшифрование шифртекста получатель производит своим тайным ключом: *p*, *g*, *х*.

Зашифрование сообщения. Как ранее, предположим, что сообщение *М* = {*mi*}, где *mi* – i-й блок сообщения. Зашифрование отправителем (каждого отдельного блока *mi* исходного сообщения) предусматривает использование, как это особо подчеркивалось выше, некоторого случайного числа *k* (1 < *k* < *p* – 1).

Блок шифртекста (*ci*) состоит из двух чисел – *аi*и *bi*:

*ai* ≡ *gk* mod *p*;

*bi*≡ (*yk* *mi*) mod *p*.

Здесь стал очевидным упомянутый недостаток алгоритма шифрования Эль-Гамаля: удвоение (реально – примерно в 1,5 раза) длины зашифрованного текста по сравнению с начальным текстом. Случайное число *k* должно сразу после вычисления уничтожаться.

Расшифрование *ci*. Выполняется по следующей формуле:

*mi*≡ (*bi*(*ai*)*x*)–1) mod *p* .

Нетрудно проверить, что (((*ai*)*x*)–1) ≡ *gkх* mod *p*.

**Практическое задание**

1. С помощью простого консольного приложения составить табличную или графическую форму зависимости времени вычисления параметра *у*, функционально заданного выражением вида:

у ≡ *ax* mod *n*,

от параметров: *а* (десятичные числа от 5 до 35; можно взять 1 или 2 числа), *х* (числа, желательно простые, из диапазона от 103 до 10100; для примера взять 5–10 чисел, равномерно распределенных в указанном диапазоне), *n* (для примера взять числа, в двоичном виде состоящие из 1024 и 2048 битов).

Для *a* = 5, *х* – 5 простых из диапазона от 103 до 105, зависимость времени вычисления от параметров будет иметь следующий вид:

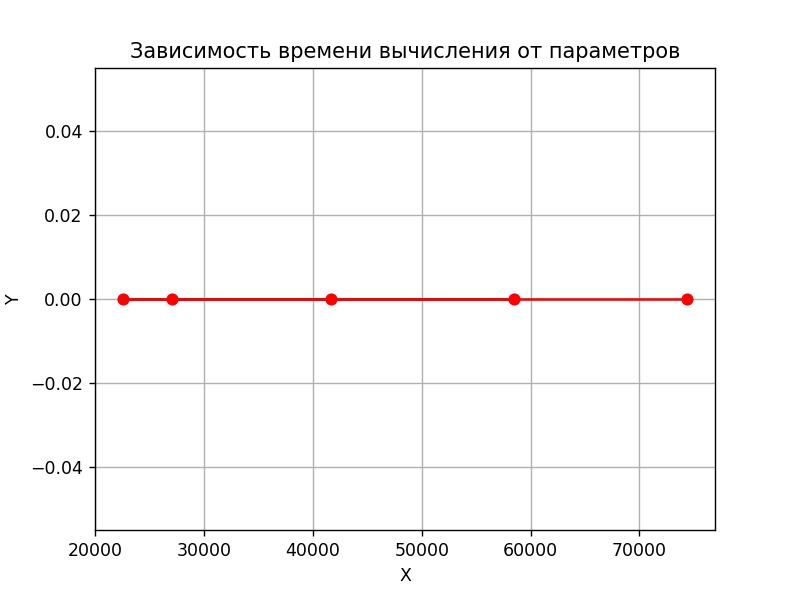


Рисунок 1 – Зависимость времени вычисления от параметров

1. Разработать авторское оконное приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться доступными библиотеками либо программными кодами.

В основе вычислений – кодировочные таблицы Base64 и ASCII.

Приложение должно реализовывать следующие операции:

• зашифрование и расшифрование текстовых документов на основе алгоритмов RSA и Эль-Гамаля;

Для зашифрования по алгоритму RSA необходимо сгенерировать открытый и закрытый ключ. Для этого генерируются случайные простые числа *p* и *q*, вычисляется их произведение *n*. Выбирается случайное число *e*, взаимно простое с (*p* – 1)(*q* – 1). Вычисляется число *d*, обратное к *e* по модулю (*p* – 1)(*q* – 1).

def generate\_key(bit\_num: int):

    p = Crypto.Util.number.getPrime(bit\_num, randfunc = Crypto.Random.get\_random\_bytes)

    q = Crypto.Util.number.getPrime(bit\_num, randfunc = Crypto.Random.get\_random\_bytes)

    print('p: ', p)

    print('q: ', q)

    # Вычисление произведения p и q

    n = p \* q

    print('n: ', n)

    # Выбор открытого ключа e

    phi = (p-1)\*(q-1)

    e = generate\_coprime(phi)

    print ('e: ', e)

    \_, d, \_ = extended\_gcd(e, phi)

    if d < 0:

        d = phi + d

    print('d: ', d)

    return ((e, n), (d, n))

Листинг 1 – Генерация ключей

Далее *e*, *n* будут использоваться для зашифрования сообщения, *d*,*n* – для дешифрования.

Для зашифрования используются значения символов ASCII в десятичной системе исчисления.

Сгенерируем 8-битные случайные числа *p* и *q*. Получаем p = 139, q = 173, n = 24047, e =21335, d = 5111. С помощью полученных значений зашифруем букву ‘p’ английского алфавита. Десятичное значение p = 112, тогда

*ci* = (*mi*)*e* mod *n* = 112^21335 mod 24047 = 11450.

def encode(message : str, package : tuple[int, int]):

    e, n = package

    ciphertext = [(ord(c) \*\* e) % n for c in message]

    return ciphertext

Листинг 2 – Зашифрование

Дешифрование каждого блока происходит по формуле:

*mi* ≡ (*ci*)*d* mod *n*.

Для полученного значения *ci* = 17393, d = 5111, n = 24047

*mi* ≡ 11450^ 5111mod 24047 = 112.

112 соответствует зашифрованному символу ‘p’.

Проведем дешифрование с помощью следующего кода:

def decode(ciphertext : str, package):

    d, n = package

    plaintext = [chr((c \*\* d) % n) for c in ciphertext]

    return (''.join(plaintext))

Листинг 3 – Дешифрование

Для зашифрования по алгоритму Эль-Гамаля необходимо сгенерировать ключевую информацию. Для этого генерируется случайное простое число *p*, генерируется первообразный корень *g* числа *p*, генерируется случайное число *х* (*х* < *p*), вычисляется *y*.

Сгенерируем ключевую информацию с помощью следующего кода:

def generate\_key(bit\_num: int):

    p = Crypto.Util.number.getPrime(bit\_num, randfunc = Crypto.Random.get\_random\_bytes)

    print('p:', p)

    g = primitive\_root(p)

    print('g:', g)

    x = random.randint(1, p-1)

    print('x:', x)

    y = g\*\*x % p

    print('y:', y)

    return ((p,g,y), (p,g,x))

Листинг 4 – Генерация ключевой информации

Получим значения *p* = 241, *g* = 7, *x* = 88.

*y* = *gx* mod *p* = 788 mod 241 = 10.

Далее значения *p, g, y* будут использованы для шифрования, а *p, g, x* ­– для дешифрования.

Зашифрование отправителем предусматривает использование некоторого случайного числа *k* (1 < *k* < p – 1). Пусть *k* = 84.

С помощью полученных значений зашифруем букву ‘p’ английского алфавита. Десятичное значение p = 112, тогда

*ai* ≡ *gk* mod *p* ≡ 784 mod 241 = 106;

*bi*≡ (*yk* *mi*) mod *p* ≡ 1084∙112 mod 241 = 131.

Проведем шифрование с помощью следующего кода:

def encode(message, package):

    p, g, y = package

    a = []

    b = []

    for m in message:

        k = random.randint(1, p-1)

        print('k', k)

        a.append(g \*\* k % p)

        b.append(y\*\*k \* ord(m) % p)

    ciphertext = [(a[i], b[i]) for i in range(0, len(b))]

    return ciphertext

Листинг 5 – Шифрование

Дешифрование выполняется по следующей формуле:

*mi*≡ (*bi*(*ai*)*x*)–1) mod *p*.

Для полученных значений *ai* = 106, *bi* = 131:

*mi*= 106((131)*x*)-1 mod 241 = 112.

112 соответствует зашифрованному символу ‘p’.

Проведем дешифрование с помощью следующего кода:

def decode(ciphertext, package):

    p, g, x = package

    plaintext = []

    for c in ciphertext:

        a\_x = c[0] \*\* x

        \_, a\_r, \_ = extended\_gcd(a\_x, p)

        if a\_r < 0:

            a\_r = p + a\_r

        plaintext.append(chr((c[1]\*a\_r) % p))

    return (''.join(plaintext))

Листинг 6 – Дешифрование

• определение времени выполнения операций

Исходный текст для зашифрования – собственные фамилия, имя, отчество. Для численного представления блоков текста можно в том числе пользоваться указанными выше кодировочными таблицами.

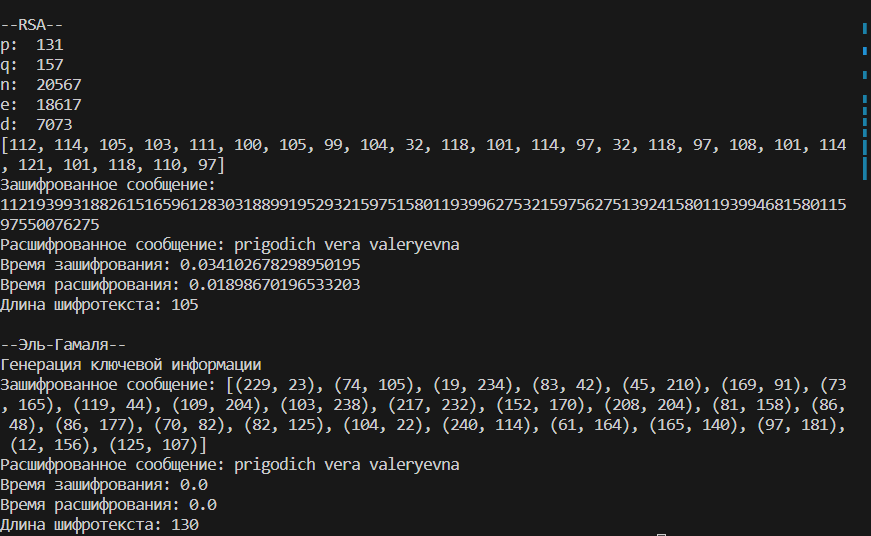


Рисунок 2 – Результат работы приложения

1. Используя примерно одинаковый порядок ключевой информации, оценить производительность обоих алгоритмов и относительное изменение объемов криптотекстов (по отношению к объемам открытых текстов).

Используем отрывок длиной 200 слов (1069 символов) из «Превращение» Франца Кафки. Для генерации ключевой информации используются случайные 8-битные числа. Результатом работы программы в таком случае будет:

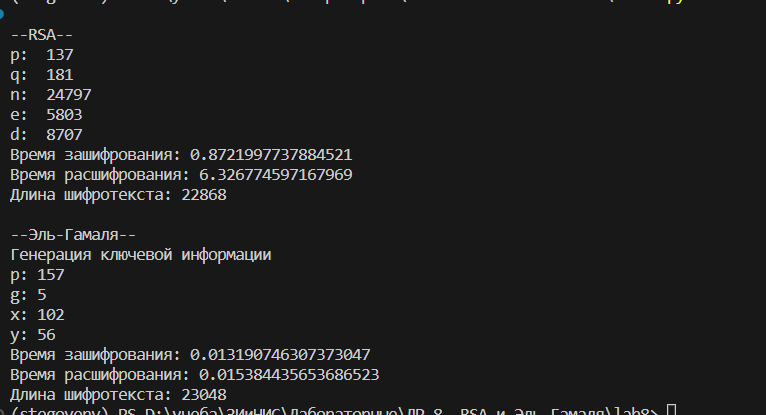


Рисунок 3 – Работа приложения для длинного текста

Как можно заметить, алгоритм Эль-Гамаля работает в разы быстрее алгоритма RSA. Длина шифротекста в алгоритмах зависит от длины используемых чисел. При меньших числах, порядок зашифрованных символов также будет ниже, но в таком случае криптостойкость алгоритма будет тоже ниже. Для алгоритма Эль-Гамаля учитывается, что для шифрования используется 2 значения – *a* и *b*. Для обоих алгоритмов длина шифротекста значительно превышает длину открытого текста.

Вывод: в ходе лабораторной работы были исследованы два асимметричных шифра: RSA и Эль-Гамаля.

Было показано, что RSA обладает высокой стойкостью к взлому при правильном выборе параметров. Однако, его размер ключа должен быть большим для достижения высокого уровня безопасности.

Эль-Гамаля, в свою очередь, требует меньше вычислительных ресурсов для шифрования и расшифрования, при этом может обеспечить более высокую стойкость к атакам по сравнению с RSA при равной длине ключа.

Таким образом, в зависимости от конкретных требований и условий использования, можно выбрать тот шифр, который наилучшим образом подходит для конкретной задачи.

**Контрольные вопросы**

1. Охарактеризовать алгоритмы RSA и Эль-Гамаля. Для каких целей они могут применяться?

RSA и Эль-Гамаля - это два широко используемых алгоритма шифрования в криптографии. Они оба основаны на математических задачах, которые трудно решить с точки зрения вычислительной сложности.

Оба алгоритма могут использоваться для защиты конфиденциальной информации и для создания цифровых подписей. RSA широко используется для шифрования данных в Интернете и для обеспечения безопасности электронной почты. Алгоритм Эль-Гамаля часто используется для создания цифровых подписей в блокчейн-технологиях и других приложениях, где необходима проверка подлинности данных.

1. На чем основана криптостойкость алгоритмов RSA и Эль-Гамаля?

Криптостойкость алгоритмов RSA и Эль-Гамаля основана на трудности решения определенных математических задач.

Алгоритм RSA основан на трудности факторизации больших целых чисел на простые множители. Если некоторое большое число состоит из двух простых множителей, то найти эти множители в общем случае очень трудно. В случае RSA, ключом является большое число, которое является произведением двух простых чисел. Шифрование и расшифрование сообщений в алгоритме RSA требует знания этих простых чисел, которые являются закрытыми ключами, и их факторизации. Трудность факторизации больших чисел является основой криптостойкости RSA.

Алгоритм Эль-Гамаля основан на задаче вычисления логарифма в конечном поле. Эта задача также является трудной для решения, особенно при использовании больших конечных полей. Ключи в алгоритме Эль-Гамаля также основаны на вычислениях в конечных полях, которые необходимы для шифрования и расшифрования сообщений. Криптостойкость Эль-Гамаля основана на трудности вычисления логарифмов в конечных полях.

1. Что такое первообразный корень?

Первообразный корень по модулю *р* является таким числом, что его степени (*gi* , 1 ≤ *i* ≤ *p* – 1) дают все возможные по модулю *р* вычеты (остатки), которые взаимно просты с *p*.

1. Найти первообразные корни (если они существуют) чисел (*р*): 13, 19, 23, 27, 31, 37, 39, 43.

Для числа 13 первообразный корень равен 2.

Для числа 19 первообразный корень равен 2.

Для числа 23 первообразный корень равен 5.

Для числа 27 первообразных корней равен 2.

Для числа 31 первообразный корень равен 3.

Для числа 37 первообразный корень равен 2.

Для числа 39 первообразных корней нет.

Для числа 43 первообразный корень равен 3.

1. Пусть пользователь А хочет передать пользователю В сообщение М, которое в некоторой кодировке соответствует числу 17 и зашифровано с помощью алгоритма RSA. Пользователь В имеет следующие ключевые параметры: *p* = 7, *q* = 11, *d* = 47. Описать процесс зашифрования сообщения пользователем А.

Необходимо сгенерировать ключевую информацию:

*n* = *p*\**q* = 77,

φ(*n*) = 10\*6 = 60.

*e* = *d– 1*mod 60 = 23.

Далее используя эти параметры необходимо зашифровать сообщение, используя формулу:

*ci* = (*mi*)*e* mod *n*

1. Пользователю системы RSA с ключевыми параметрами *n* = 33, *d* = 3 передано зашифрованное сообщение С, состоящее из блока цифр: 13. Расшифровать это сообщение (взломав систему RSA пользователя).

Находим закрытый ключ *e*: ищем значение функции Эйлера от *n*.

n = 33 => функция Эйлера от *n* будет равна φ(33) = (11 - 1) \* (3 - 1) = 20.

Находим такое значение *e*, что *d* ≡ e-1 (mod (*n*)).

Используя расширенный алгоритм Евклида, можно найти решение этого уравнения:

3 \* 7 + (-4) \* 20 = 1

Получаем e = 7

Для расшифровки сообщения С необходимо возвести его в степень *d*, по модулю *n*:

133 mod 33 = 13

7. В системе связи, применяющей шифр Эль-Гамаля, пользователь А желает передать сообщение М пользователю В. Найти недостающие параметры системы при следующих заданных параметрах: *p* = 19, *g* = 2, *х* = 3, *k* = 5, *М* = 10. Описать по шагам зашифрование сообщения и расшифрование шифртекста.

По данным значениям найдем *y*.

*y* = *gx* mod *p* = 23 mod 19 = 8.

Далее значения *p, g, y* будут использованы для шифрования, а *p, g, x* ­– для дешифрования.

Зашифрование отправителем предусматривает использование некоторого случайного числа *k* (1 < *k* < p – 1). Дано *k* = 5.

*ai* ≡ *gk* mod *p* ≡ 25 mod 19 = 13;

*bi*≡ (*yk* *mi*) mod *p* ≡8 5∙10 mod 19 =6.

Дешифрование выполняется по следующей формуле:

*mi*≡ (*bi*(*ai*)*x*)–1) mod *p*.

Для полученных значений *ai* = 13, *bi* = 6:

*mi*= (6(13)*3*)-1 mod 19 = 10.

1. Положим, что в системе применяется алгоритм шифрования/расшифрования Эль-Гамаля. Известны некоторые параметры системы: *р* = 167, *g* = 5, *y* = *g*29 ≡ 55 mod *p*. Используя указанные и недостающие (выбрать самостоятельно) параметры, зашифровать свое имя (в любом языке) в предположении: а) первая буква алфавита соответствует числу 0 и т. д.; б) первая буква алфавита соответствует числу 1. Проанализировать результат.

*x* = 29.

а) ‘vera’ = [21 4 17 0]

1) *k =* 5

*ai* ≡ *gk* mod *p* ≡ 55 mod 167 = 119;

*bi*≡ (*yk* *mi*) mod *p* ≡ 555∙21 mod 167 = 123.

2) *k =* 3

*ai* ≡ 53 mod 167 = 125;

*bi*≡ 553∙4 mod 167 = 5.

3) *k =* 4

*ai* ≡ 54 mod 167 = 124;

*bi*≡ 554∙17 mod 167 = 125.

4) *k =* 7

*ai* ≡ 57 mod 167 = 136;

*bi*≡ 557∙0 mod 167 = 0.

б) ‘vera’ = 22 5 18 1

1) *k =* 5

*ai* ≡ *gk* mod *p* ≡ 55 mod 167 = 119;

*bi*≡ (*yk* *mi*) mod *p* ≡ 555∙22 mod 167 = 105.

2) *k =* 3

*ai* ≡ 53 mod 167 = 125;

*bi*≡ 553∙5 mod 167 = 48.

3) *k =* 4

*ai* ≡ 54 mod 167 = 124;

*bi*≡ 554∙18 mod 167 = 152.

4) *k =* 7

*ai* ≡ 57 mod 167 = 136;

*bi*≡ 557∙1 mod 167 = 159.