МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9 НА ТЕМУ:

Исследование асимметричных шифров

Выполнила студентка 3 курса 4 группы

Сятковская Е. Д.

Минск 2023

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

**Теоретические сведения**

В основу асимметричной криптографии положена идея использовать ключи парами: один – для зашифрования (открытый, или публичный, ключ), другой – для расшифрования (тайный ключ). Отметим, что указанная пара ключей принадлежит получателю зашифрованного сообщения. Все алгоритмы шифрования с открытым ключом основаны на использовании односторонних функций, к числу которых, как известно, относится вычисление дискретного логарифма.

Алгоритмы шифрования с открытым ключом можно использовать для решения следующих задач:

• зашифрования/расшифрования передаваемых и хранимых данных в целях их защиты от несанкционированного доступа;

• формирования цифровой подписи под электронными документами;

• распределения секретных ключей, используемых далее при шифровании документов симметричными методами. В данной работе мы будем работать над аспектами решения первой из указанных задач.

По мнению Диффи и Хеллмана, алгоритм шифрования с открытым ключом должен:

• вычислительно легко создавать пару (открытый ключ *e* – закрытый ключ *d*);

• вычислительно легко зашифровывать сообщение *Mi* открытым ключом;

• вычислительно легко расшифровывать сообщение *Ci*, используя закрытый ключ;

• обеспечивать непреодолимую вычислительную сложность определения соответствующего закрытого ключа при известном открытом ключе;

• обеспечивать непреодолимую вычислительную сложность восстановления исходного (открытого сообщения *Mi*) зная только открытый ключ и зашифрованное сообщение *Ci*.

**Криптоалгоритм на основе задачи об укладке ранца** разработан Р. Мерклом и М. Хеллманом. Это первый алгоритм шифрования с открытым ключом широкого назначения.

Ранцевый (рюкзачный) вектор ***S*** = (*s*1, ..., *sz*) – это упорядоченный набор из *z*, *z* ≥ 3, различных натуральных чисел *si*. Входом задачи о ранце (рюкзаке) называем пару (***S***, *S*), где ***S*** – рюкзачный вектор, а *S* – натуральное число.

Решением для входа (***S***, *S*) будет такое подмножество из ***S***, сумма элементов которого равняется *S*.

В наиболее известном варианте задачи о ранце требуется выяснить, обладает или нет данный вход (***S***, *S*) решением. В варианте, используемом в криптографии, нужно для данного входа (***S***, *S*) построить решение, зная, что такое решение существует. Оба эти варианта являются NP-полными.

Имеются также варианты этой задачи, которые не лежат даже в классе NP.

Как видим, проблема укладки ранца формулируется просто. Дано множество предметов общим числом *z* различного веса. Спрашивается, можно ли положить некоторые из этих предметов в ранец так, чтобы его вес стал равен определенному значению *S*? Более формально задача формулируется так: дан набор значений *k*1, *k*2, …, *kz* и суммарное значение *S*. Требуется вычислить значения *bz* такие, что

*S* = *b*1*k*1 + *b*2*k*2 + ... + *bzkz*. (7.1)

Здесь *bi* может быть либо нулем, либо единицей. Значение *bi* = 1 означает, что предмет *mi* кладут в рюкзак, а *bi*= 0 – не кладут.

Суть метода для шифрования состоит в том, что существуют две различные задачи укладки ранца: одна из них решается легко и характеризуется линейным ростом трудоемкости, а другая решается трудно. Легкий для укладки ранец можно трансформировать в трудный. Трудный для укладки ранец применяется в качестве открытого ключа, который легко использовать для зашифрования, но невозможно – для расшифрования. В качестве закрытого ключа применяется легкий для укладки ранец, который предоставляет простой способ расшифрования сообщения.

В качестве закрытого ключа *d* (легкого для укладки ранца) используется сверхвозрастающая последовательность, состоящая из *z* элементов: *d*1, *d*2, …, *dz*: *d* = {*di*}, *i* = 1, …, *z*.

В основе операции зашифрования сообщения лежит соотношение (7.1). Для зашифрования сообщения (*М*) оно сначала разбивается на блоки, по размерам равные числу (*z*) элементов последовательности в ранце. Затем, считая, что 1 указывает на присутствие элемента последовательности в ранце, а 0 – на его отсутствие, вычисляются полные веса рюкзаков (*Si*, i = 1, …, *z*): по одному ранцу для каждого блока сообщения с использованием открытого ключа получателя ***e.***

**Практическое задание**

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:

* генерация сверхвозрастающей последовательности (тайного ключа); старший член последовательности – 100-битное число; в простейшем случае принимается z = 6 (для кодировки Base64) и z = 8 (для кодировки ASCII);

Сверхвозрастающей называется последовательность, в которой каждый последующий член больше суммы всех предыдущих.

Мы можем сгенерировать простую сверхвозрастающую последовательность следующим образом: член последовательности образуется путем добавления 1 к сумме ее предыдущих членов.

def generate\_superincreasing\_sequence(z : int) -> list:

    seq = [1] # начинаем с единицы

    while len(seq) < z:

        next\_num = sum(seq) + 1 # следующее число в последовательности

        seq.append(next\_num)

    return seq

Листинг 1 – Генерация простой сверхвозрастающей последовательности

Таким образом мы сгенерируем последовательность {1, 2, 4, 8, 16, 32, 64, 128}.

* вычисление нормальной последовательности (открытого ключа);

Для получения открытого ключа *e* (*e* = {*ei*}, *i* = 1, …, *z*) все значения закрытого ключа умножаются на некоторое число a по модулю n:

*ei*≡ *dia* mod *n.*

Значение модуля *n* должно быть больше суммы всех чисел последовательности; кроме того, НОД (*а*, *n*) = 1.

seq = generate\_superincreasing\_sequence(z)

n = sum(seq) + 1

a = generate\_coprime(n)

public\_key = get\_public\_key(seq, a, n)

def generate\_coprime(phi):

    while True:

        e = random.randrange(2, phi)

        gcd, x, y = extended\_gcd(e, phi)

        if gcd == 1:

            return e

def get\_public\_key(seq : list, a : list, n : list) -> list:

    public\_key = []

    for num in seq:

        public\_key.append((a \* num) % n)

    return public\_key

Листинг 2 – Генерация простой нормальной последовательности

Мы можем взять *n* большее суммы всех членов последовательности на 1, тогда *n* = 256, число *a* обратное ему *a* = 51.

Тогда нормальная последовательность примет вид {51, 102, 204, 152, 48, 96, 192, 128}.

* зашифрование сообщения, состоящего из собственных фамилии, имени и отчества;

Для зашифрования сообщения, состоящего из ФИО, его необходимо перевести в бинарный вид и разбить на блоки, по размерам равные числу (*z*) элементов последовательности в ранце. Затем, считая, что 1 указывает на присутствие элемента последовательности в ранце, а 0 – на его отсутствие, вычисляются полные веса рюкзаков (*Si*, *i* = 1, …, *z*): по одному ранцу для каждого блока сообщения с использованием открытого ключа получателя *e*.

    binary\_str = ''.join(format(ord(x), 'b') for x in message)

    if len(binary\_str) % z != 0:

        binary\_str += '0' \* (z - len(binary\_str) % z)

    binary\_list = [binary\_str[i:i+z] for i in range(0, len(binary\_str), z)]

    print('Сообщение:')

    for bin in binary\_list:

        print(bin)

    print()

    cipher = []

    for bin in binary\_list:

        s = 0

        for i in range(len(bin)):

            if bin[i] == '1':

                s += public\_key[i]

        cipher.append(s)

Листинг 3 – Зашифрование сообщения

Используя сгенерированные ранее данные, зашифруем сообщение «VERA». *e* = {51, 102, 204, 152, 48, 96, 192, 128}.

|  |  |  |
| --- | --- | --- |
| Бинарный код символа | Укладка ранца | Вес ранца |
| 01010110 | 102 + 152 + 96 + 192 | 542 |
| 01000101 | 102 + 96 + 128 | 326 |
| 01010010 | 102 + 152 + 192 | 446 |
| 01000001 | 102 + 128 | 230 |

Таким образом, зашифрованное сообщение *С* = 542 326 445 230.

* расшифрование сообщения;

Для расшифрования сообщения получатель (используя свой тайный ключ *d*: сверхвозрастающую последовательность) должен сначала определить такое обратное к *а* число *а*–1, что

*аа*–1 mod *n* ≡ 1.

Для вычисления обратных чисел по модулю можно использовать известный нам расширенный алгоритм Евклида.

Число обратное к 51 по модулю 256 равно 251.

После определения обратного числа каждое значение шифрограммы (*ci*) преобразуется в соответствии со следующим соотношением:

*Si*≡ *ci­а*–1 mod *n*.

Полученное на основании последней формулы для каждого блока число далее рассматривается как заданный вес ранца, который следует упаковать по изложенному выше алгоритму, используя сверхвозрастающую последовательность (тайный ключ получателя).

*d* = {1, 2, 4, 8, 16, 32, 64, 128}.

*С* = 542 326 445 230.

*S*1 = *с*1*a*–1 mod *n* = 542 · 251 mod 256 = 106.

Используем *S*1 = 106 и с помощью сверхвозрастающей последовательности ({1, 2, 4, 8, 16, 32, 64, 128}) по алгоритму упаковки ранца найдем *m*1.

Необходимо по очереди анализировать некоторый «текущий вес» *S* предметов, составляющих сверхвозрастающую последовательность; в результате анализа нужно упаковать (доупаковать) ранец.

1. В качестве текущего выбирается число S, которое сравнивается с «весом» самого тяжелого предмета (*dz*); если текущий вес меньше веса данного предмета, то его в ранец не кладут (0), в противном случае его укладывают (1) в ранец и переходят к анализу очередного (в общем случае – *i*-го предмета).

2. Если на предыдущем (*i*-м шаге) предмет пополнил ранец, то текущий вес уменьшают на вес положенного предмета (*S* = *S* – *di*); переходят к следующему по весу предмету в последовательности: *di* – 1. Шаги повторяются до тех пор, пока процесс не закончится. Если текущий вес уменьшится до нуля (*S* = 0), то решение найдено. В противном случае – нет.

1. *S* = 106, *dz* = 128 > 106, *k* = 0.

2. *S* = 106, *dz* = 64 < 106, *k* = 1.

3. *S* = 106 – 64 = 42, *dz* = 32 < 42, *k* = 1.

4. *S* = 42 – 32 = 10, *dz* = 16 > 10, *k* = 0.

5. *S* = 10, *dz* = 8 < 10, *k* = 1.

6. *S* = 10 – 8 = 2, *dz* = 4 > 2, *k* = 0.

7. *S* = 2, *dz* = 2 = 2, *k* = 1.

8. *S* = 0. Рюкзак упакован

*m*1 = 01010110 = V.

Расшифрование остальных блоков шифртекста производится аналогично.

Результат выполнения программы:

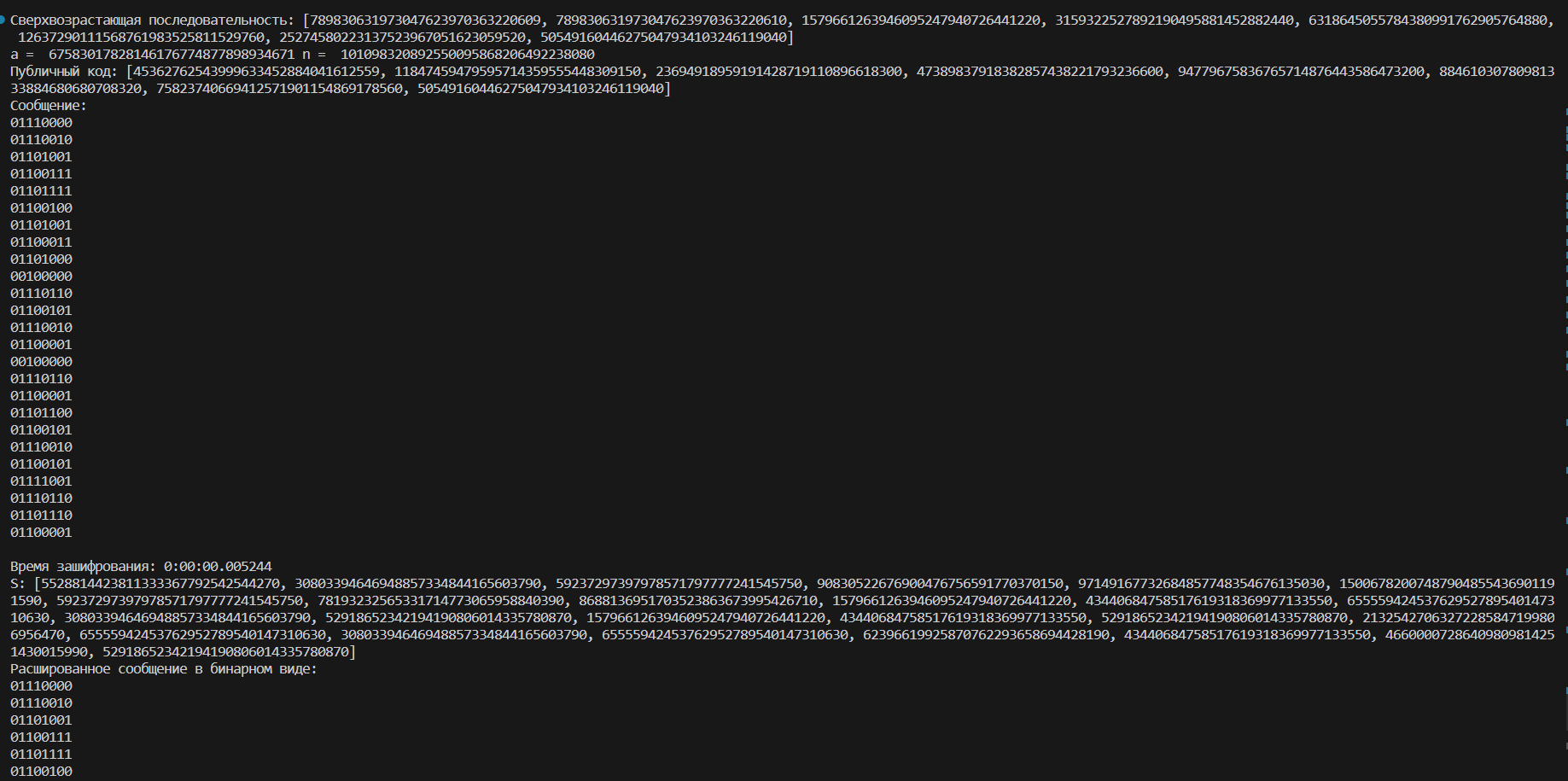


Рисунок 1 – Результат выполнения программы

* оценка времени выполнения операций зашифрования и расшифрования.

Время выполнения операций зависит от длины сообщения, размеров последовательности, длины блока. Операция дешифрования занимает больше времени, чем шифрования.

**Вывод:** в ходе лабораторной работы мы ознакомились с криптоалгоритмом на основе задачи об укладке ранца, научились формировать сверхвозрастающие последовательности, отличая их нормальных последовательностей; разработали приложение, выполняющее зашифрование и расшифрование сообщения и проанализировали время выполнения данных процессов.

**Контрольные вопросы**

**1. Что такое «ранцевый (рюкзачный) вектор»? Дать определение.**

"Ранцевый вектор" или "рюкзачный вектор" - это вектор, состоящий из целочисленных значений, которые представляют вес каждого из предметов в задаче о рюкзаке.

**2. Сформулировать задачу укладки ранца.**

Задача укладки ранца (или задача о рюкзаке) заключается в выборе оптимального набора предметов из заданного множества, чтобы заполнить рюкзак до максимально возможного веса, не превышая его грузоподъемность.

**3. Если вектор рюкзака имеет вид (14, 28, 56, 82, 90, 132, 197, 284, 341, 455), то какими следует принять коэффициенты bi из выражения (7.1), чтобы получить S = 517? Каким будет решение задачи для S = 516?**

Задачу можно попытаться решить с помощью жадного алгоритма:

def get\_knapsack\_set(value: int, seq: list):

    result = ''

    seq.sort(reverse=True)

    for num in seq:

        if num <= value:

            result = '1' + result

            value -= num

        else: result = '0' + result

    return result

sequence = [14, 28, 56, 82, 90, 132, 197, 284, 341, 455]

print(f'For 517: {get\_knapsack\_set(517, sequence)}')

print(f'For 516: {get\_knapsack\_set(516, sequence)}')

Листинг 1 – Решение задачи о рюкзаке с помощью жадного алгоритма



Рисунок 5 – Вывод консоли

Если сложить числа последовательность с помощью полученных векторов, выяснится, что задача не имеет чистого решения для жадного алгоритма, значит решать её придётся простым перебором всех возможных вариантов.

**4. Что такое сверхвозрастающая последовательность? Привести примеры.**

Сверхвозрастающей называется последовательность, в которой каждый последующий член больше суммы всех предыдущих. Последовательность [14, 28, 56, 82, 90, 132, 197, 284, 341, 455] является сверхвозрастающей.

**5. Можно ли последовательности чисел: {89, 3, 11, 2, 45, 6, 22}, {3, 41, 5, 1, 21, 10}, {2, 3, 11, 29, 45, 6, 39} преобразовать в сверхвозрастающие?**

{89, 3, 11, 2, 45, 6, 22} => {2, 3, 6, 11, 22, 45, 89}

Так как 2+3+6 ≤ 11, данная последовательность не может быть преобразована в сверхвозрастающую.

{3, 41, 5, 1, 21, 10} => {1, 3, 5, 10, 21, 41}

Эта последовательность может быть преобразована в сверхвозрастающую.

{2, 3, 11, 29, 45, 6, 39}  => {2, 3, 6, 11, 29, 39, 45}

Так как 2+3+6 ≤ 11, данная последовательность не может быть преобразована в сверхвозрастающую.

**6. Используя некоторый вектор *S* = (103, 107, 211, 430, 863, 1716, 3449, 6907, 13807, 27610), вычислить ключи для зашифрования и расшифрования сообщений.**

def get\_public\_key(seq: list, a: int, n: int) -> list:

    key = []

    for num in seq:

        key.append((num \* a) % n)

    return key

def gcd(x, y):

    if x < y:

        return gcd(y, x)

    while y != 0:

        (x, y) = (y, x % y)

    return x

seq = [103, 107, 211, 430, 863, 1716, 3449, 6907, 13807, 27610]

print(f'\nPrivate key: {seq}\n')

a = 7

n = sum(seq) + 1

while gcd(n, a) != 1:

    n += 1

print(f'a: {a}')

print(f'n: {n}\n')

print(f'Public key: {get\_public\_key(seq, a, n)}\n')

Листинг 2 – Вычисление публичного ключа для ранцевого алгоритма

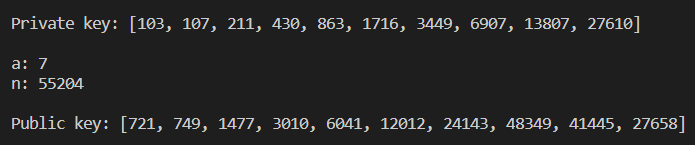


Рисунок 6 – Вывод консоли

**7. Что такое «секретная лазейка»?**

Секретная лазейка – это уязвимость в криптографической системе, которая позволяет злоумышленнику получить доступ к зашифрованным данным или нарушить их конфиденциальность без знания ключа шифрования или пароля. Такая уязвимость может быть специально встроена в криптографическую систему ее разработчиком, чтобы иметь возможность получения доступа к зашифрованным данным в случае необходимости. Но также секретные лазейки могут появляться случайно в результате ошибок в разработке криптографической системы или в результате действий злоумышленника, который специально создал уязвимость в системе.

**8. Охарактеризовать криптостойкость алгоритма на основе задачи об укладке ранца.**

Криптостойкость алгоритма во многом определяется скоростью (временем) поиска нужного варианта укладки ранца. Для последовательности из шестидесяти или немногим более того элементов нетрудно решить задачу укладки ранца, даже если последовательность не является сверхвозрастающей. При практической же реализации алгоритма ранец должен содержать не менее нескольких сотен элементов. Длина каждого члена сверхвозрастающей последовательности должна быть несколько сотен бит, а длина числа n – от 100 до 200 бит, что значительно усложнит криптоанализ.